

## THE WOSP MODEL: BALANCED INFORMATION SYSTEM DESIGN AND EVALUATION

**BRIAN WHITWORTH**

**MICHAEL ZAIC**

*Department Of Information Systems*

*New Jersey Institute Of Technology*

[bwhitworth@acm.org](mailto:bwhitworth@acm.org)

### ABSTRACT

IS researchers may wonder why computing advances often confound expectations. The leisure society, the paperless office, and programmer decline are just a few predictions that didn't happen. If current expectations are to be any more accurate, we must learn from experience. As we create e-systems to rival ourselves, and networks to match society, the problem may be a limited view of system performance. The Technology Acceptance Model (TAM) sees performance as functionality plus usability. The Web Of System Performance (WOSP) model presented in this paper extends TAM to include other requirements common in performance research literature. WOSP is a theoretical framework for the balanced design and evaluation of advanced information systems, as now being created on the Internet and elsewhere. It analyses performance via four fundamental system elements: boundary, internal structure, effectors and receptors. Each can be designed for opportunity or risk, giving eight performance goals, whose design tension comprises a "web" of performance. It explains why advances sometimes "bite back", and need creative system requirements integration. The theory applies to hardware, software, cognitive, or social system levels, but as each level is a different "world", it cannot be instantiated across levels. The WOSP model should interest those designing or evaluating advanced software systems.

**KEYWORDS:** system performance, functionality, usability, reliability, security, flexibility, extendibility, connectivity, confidentiality, privacy, openness, non-functional requirements, qualitative requirements, technology acceptance model, TAM

### INTRODUCTION

We all like to crystal ball gaze. After all, if we knew where the future was going to be, we could be there when it arrived, as "fore-warned is fore-armed". It seems sensible to divine the future from the trends of the present. Yet often in computing, expected trends do not occur, or at least not in the way expected. For example, over twenty years ago it was predicted that paper would soon be dead, replaced by an electronic "paperless office" [Toffler, 1989]. Today, despite the electronic revolution, paper is alive and well. Due to electronic communication, more paper is being produced than ever before, not less.

Shortly after Toffler's forecast, a New Zealand company wrote an application generator called "The Last One", claiming, as its name implies, to be the last program that programmer's would write. They followed James Martin, the guru of programmerless programming, who predicted that users would generate applications, making programmers redundant. Many of us then agreed that perhaps programming was a dying art - but it never happened. The demand for programmers has never faltered, and today is still rising. As *Software Development* notes: "What happened to the

promise of programming without programmers?" [Keuffel, 2001]. User application generation gave more, not less, programmer work. Possibly the earliest prediction failure was the "leisure society", that was to arise as machines took over human work. The reality is that modern workers are busier, and more stressed, than ever before [Schor, 1991].

Why did these expectations, and many others like them, miss the mark? How can the informed opinion of the day be so wrong? More important, are we repeating the same errors today? If so, what is the cause? The story of blind men describing an elephant seems to summarize the problem – one grabbed its tail and declared it like a rope and bendy, one took a leg and said it was like a pillar and fixed, one felt an ear and thought it like a rug and floppy, while yet another, holding the trunk, said it was like pipe, and very strong [Sanai, 1968]. The focus varied with the viewer, but no-one saw the elephant. We propose that the "elephant" confounding our predictions is an advanced system [Casti, 1997], and our error is to analyze system aspects in isolation. If all remained fixed except work production, computers might have created a leisure society. But they changed the whole of society, not just work, so the effect was not as predicted. Equally, computers altered application development as a whole, and again the total effect denied the local prediction of programmers becoming extinct. We propose a simple principle: To design and evaluate an advanced information system it must be viewed "in toto".

A simple tool's performance is mainly its functionality, but software is no longer a simple tool. Software performance now involves multiple aspects that may contradict [Alter, 1999]. Over a decade ago, Goodwin argued that if people find software unusable, it performs poorly [Goodwin, 1987]. The Technology Acceptance Model (TAM) proposes that both usefulness (functionality) and ease of use (usability) affect user acceptance based on perceived software performance [Davis, 1989]. But are functionality (what a system does), and usability (how it does it), the sum total of performance? For example, security seems also important [OECD, 1996]. On the World Wide Web, scalability seems equally important [Berners-Lee, 2000]. Alter adds reliability, responsiveness, and conformance to standards to the list [Alter, 1999]. Non-functional IS requirements can be system critical [Nixon, 1998], but what they are and how they interact is unclear. Yet awareness of the range of demands on software design is increasing. Usability is a movement in its own right [Nielsen, 1993]. A study by Department of Commerce's National Institute of Standards and Technology estimated that software errors cost the U.S. economy \$59.5 billion annually [NIST, 2002]. The success of the web has led to a world wide effort in open standards [Gargaro et al., 1993]. Adaptive middleware is now considered a key ingredient for a new generation of open, flexible and connected web software [Agha, 2002]. This paper extends the TAM duality into a multi-attribute model of performance, representing the variety and complexity of software today.

## **A SYSTEMS APPROACH TO SOFTWARE PERFORMANCE**

Over thirty years ago Bertalanffy was struck by the surprising isomorphies of concepts, laws, and models, in various fields like physics, biology, psychology, sociology and chemistry [Bertalanffy, 1968]. For example, the first author derived a formula for social disagreement [Whitworth and Felton, 1999] of the same form as one for ecological diversity in biological habitats [Pielou, 1969]. General systems theory proposes such commonalities arise from the nature of systems, and we now consider system performance in this sense.

## **PERFORMANCE AS THE SYSTEM-ENVIRONMENT INTERACTION**

If every system exists in an environment, its performance can be defined.

### **Performance**

A system's performance is defined as:

*How successfully it interacts with its environment,*

especially that it continues to interact with the environment. If users do not use an information system, it fails as a product, and we say it did not perform well. So performance depends on whatever affects the system-environment interaction.

### **World**

To interact with its environment, a system must be of the same type. It must exist in a given type of "world" that defines its nature as the nature of that world. Examples are the physical world, the world of information, and the social world, giving physical systems, information systems and

social systems. To define system performance, a world type must first be defined. This axiom is basic to a systems approach.

### **Environment**

The environment is that around a system which interacts with it. What “succeeds” in a system-environment interaction depends on how the environment responds. In Darwinian evolution, the environment determines what adaptations “work”, i.e. perform well. System performance is thus relative to the environment, not absolute. Three things seem relevant:

- the number of opportunities,
- the number of threats, and
- the rate these change.

In an opportunistic environment, right action can lead to great benefit. In a risky environment, wrong action can lead to great loss. In a turbulent environment, risk and opportunity change quickly. An environment can be any combination, e.g. it could be opportunistic, risky, and turbulent.

### **Reproduction**

Every system must somehow be created. Software systems are built, while organic systems are born, but both creations involve cost or effort. Reproductive systems can create other systems like themselves – one system operation is creation (i.e. reproduction). The issue of reproduction is outside our scope as, other than small virus programs, few computer systems reproduce.

## **FOUR ELEMENTS OF SYSTEM PERFORMANCE**

To interact with an environment requires four basic system elements.

### **Boundary**

If it exists in some world, a system must separate what is system from what is not. It needs a boundary to divide itself from its surroundings, e.g. a social community’s boundary defines who is and is not in that community. The first element affecting a system’s interaction with its environment is its boundary.

### **Internal Structure**

Given a system existing within a boundary, either it is one indivisible thing (as the atom was once thought to be), or it contains constituent parts [Esfeld, 1998]. How these parts form and interact to give system activity, the system’s internal structure, is the second proposed element affecting system performance.

### **Effectors/Receptors**

Any system interacts with its environment in a feedback loop over time. Feedback favors some parts specializing in acting upon the environment (effectors), while others specialize in gathering environment information (receptors). For example, we have eyes and ears to sense the environment, and hands and feet to act upon it.

### **EXAMPLE**

A cell, the simplest biological system, began by forming a boundary (the cell membrane) [Alberts et al., 1994]. While simple procaryote cells are internally undifferentiated, distinct organelles soon appear with different cell functions, and complex chemical control mechanisms evolve to manage and support the system (e.g. the nucleus). An amoeba has no specialized parts for acting and sensing, but simple eucaryotic cells like *Giardia* have flagella (effectors), which act to move it about, and complex cells like protozoa include light sensitive photo-receptors. We ourselves, though highly advanced, still have a skin boundary, senses to receive input, muscles to act externally, and structured internal sub-systems to manage and support.

We suggest four basic system elements:

1. *Boundary*: To separate the system from its environment.
2. *Internal structure*: To support and coordinate system activity.
3. *Effectors*: To act upon the environment.
4. *Receptors*: To analyze environment information.

## SYSTEM LEVELS

Computer systems are recognized as systems in the general sense [Churchman, 1979]. At the lowest level, as hardware, a computer is a mechanical system with a physical boundary (its casing), an internal architecture, keyboard and mouse “receptors”, and printer and screen “effectors”. But a computer is not only chips and circuits, it is also a software system with a boundary separating system from non-system memory, across which input/output data enter and leave the system. Software contains an internal structure, with parts calling and comprising others in complex ways. Some subroutines process input, like biological receptors, while driver programs, like biological effectors, act on the system’s environment.

Hardware and software systems together are often called information technology (IT), but higher system levels exist [Boulding, 1956], as “the system” can also be the human-computer combination [Alter, 1999; Littlewood et al., 1993], e.g. a plane is a mechanical system, but the plane plus the pilot is also a system. This combination is the basis of human-factors design, where “the system” now includes human cognitive processing [Sanders and McCormick, 1993]. Just as mechanical systems act on physical objects, and software systems act on information objects, so cognitive systems create and respond to meaning, which derives from, but is not the same as, information. For example, Shannon and Weaver noted that while transmission errors, or “noise”, decrease meaning, they actually add information to a message (by increasing signal choices) [Shannon and Weaver, 1949, p109]. Human meaning distinguishes “relevant” from “irrelevant” information, and, as Weaver notes: “The concept of information developed in this theory at first seems disappointing ... because it has nothing to do with meaning.” [Shannon and Weaver, 1949, p116]. Hence computer-mediated group interaction can be analyzed in terms of cognitive meaning generating processes [Whitworth et al., 2001]. Finally a computer-mediated community can be considered a socio-technical system, with social processes that can be modeled [Herrmann and Loser, 1999; Whitworth and Felton, 1999].

### Four System Levels.

The foregoing results in four system levels. based on IS literature, as shown in Figure 1.

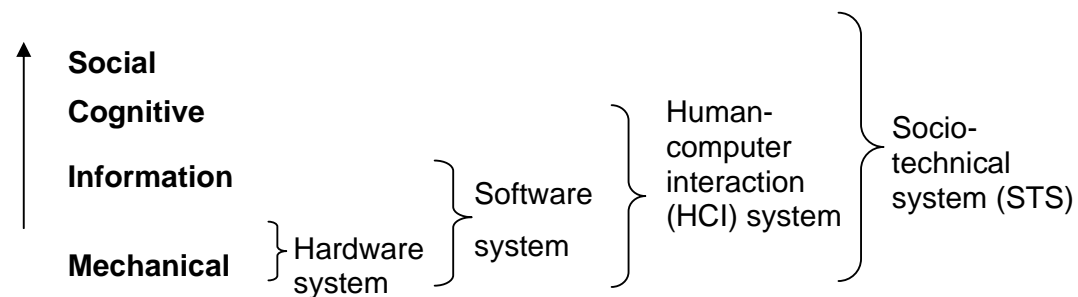


Figure 1. Information System Levels

Each level “emerges” from the previous, and adds a new level of causality. Information derives from mechanics, cognitions from information, and society from common individual cognitions. The term “information system” includes people and business processes as well as hardware and software, i.e. it includes cognitive and social levels. When Alter defines a work system as hardware, information, people and business processes [Alter, 2001], he is defining an information system which at its highest level is a socio-technical system. Its performance can be analyzed on four distinct levels. IS failure can occur at the:

- hardware level (chip overheating),
- software level (program infinite loop),
- HCI level (user misunderstanding), or
- social level (users feel the system opposes their culture).

System failure at any level affects those above them (if the hardware fails, the software must too), but success at one level does not imply success at a higher one. Each level adds requirements. Social requirements add to HCI requirements, which add to software requirements, which add to hardware requirements. For example, a virtual community IS could perform well as hardware, use

bug-free software, offer excellent user interaction, but fail as a social system if users do not trust it to be fair.

The WOSP model applies to any level system, but cannot be instantiated at more than one level at a time, because each is a different world. Each level is a total “world view” in itself. A full mechanical description of a computer system describes every voltage and circuit change. Likewise a full information description describes every bit and variable value, and a cognitive description covers every percept and concept. Each description is entire in that “world”, with nothing left out. A theoretic that combined these levels would have to do so in a “world of worlds”, which is hard to imagine. To regard hardware, software, people, and social groups as micro-theoretic “interacting entities”, as Barnard proposes, is confusing [Barnard et al., 2000, p226]. If mechanical, informational, psychological and social entities interact, within what world does that occur?

Higher levels offer more powerful descriptions, e.g. a social description of a bulletin board can be more useful than a record of bits and bytes. They are both, however, just different “views” of the same base entity. The WOSP model manages each level of a multi-level system separately. It applies to any level, but not multiple levels at once, because its axiomatic question is “What world is the system in?”. The following analysis mainly uses IT and HCI examples, but mechanical system examples are also given for clarity. For a discussion of virtual social system design see Whitworth and de Moor [2003].

### III. INFORMATION SYSTEM PERFORMANCE

Modern information systems face performance demands like those of biological systems, which explains the variety of software available today [David et al., 2003]. But the variety of life suggests that success is not only to the strong. Some animals, like tigers, possess powerful claws and teeth, but others, like plants, can barely move, yet can survive on air, water and sunlight. Some, like turtles, develop strong defensive shells, while others, like bacteria, focus on parasitism – using others for their ends. Performance, it seems, involves multiple aspects, which we now link to the basic system elements of boundary, internal structure, effectors and receptors. If systems must both gain value and avoid loss, each element has a dual role in system performance. It can be designed to maximize opportunity or minimize risk, giving eight system design goals:

1. *Effector* purposes:
  - a. *Functionality*. To act on the environment.
  - b. *Usability*. To reduce the cost of action.
2. *Boundary* purposes:
  - a. *Security*. To prevent unwelcome entry.
  - b. *Extendibility*. To use outside objects or material.
3. *Structure* purposes:
  - a. *Reliability*. To perform the same despite internal change.
  - b. *Flexibility*. To perform differently given external change.
4. *Receptor* purposes:
  - a. *Connectivity*. To exchange social meaning.
  - b. *Confidentiality*. To control or limit social meaning exchange.

Figure 2 shows the web of system performance (WOSP) model. Giving performance values to each goal creates a “web”, whose total area, it is proposed, represents overall system performance better than any point alone. The WOSP model involves eight elements:

- Four “active” properties (functionality, flexibility, extendibility, connectivity), that could make a system succeed.
- Four “passive” properties (usability, security, reliability, confidentiality), whose absence can cause failure.

The WOSP model imagines these system properties as connected by elastic, so that increasing any one property puts tension on its opposite, and to a lesser degree all the others. It shows seemingly opposing goals (such as flexibility and reliability) opposite one another. It is a web because the points are not rigidly connected, but merely in “tension”. Superficially, opposing pairs seem rigid opposites, where one feature is traded against another, but this is not always so. While opposing corners impose different design demands, the “web” can be expanded by “pulling” two corners at once, i.e. opposing goals to genuinely expands the web. For example,

increasing functionality tends to decrease usability, but it is not inevitable that great results require great effort. Mathematicians call the combination of logical power and simplicity elegance, a much admired property. The WOSP model of system performance is now discussed in detail.

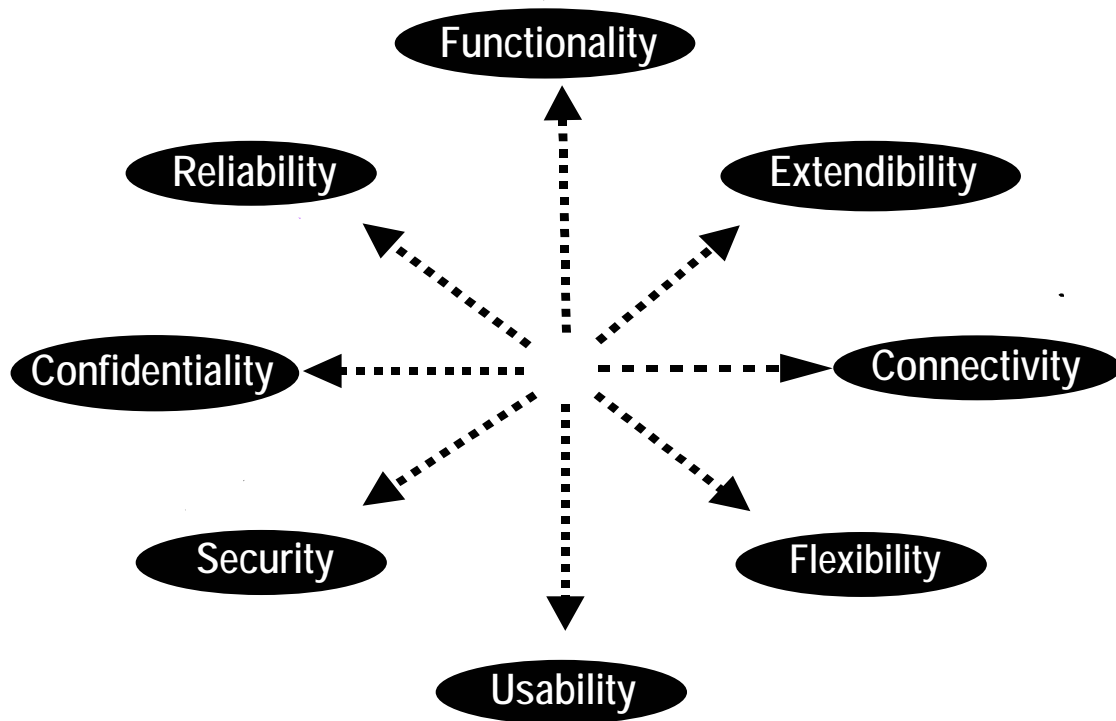


Figure 2. The Web of System Performance

External factors, like system creation cost, politics, distribution and advertising, lie outside its scope.

#### **FUNCTIONALITY AND USABILITY**

System effectors acting on the external environment, can be designed to maximize action effects (functionality), or to minimize action costs (usability).

##### **Functionality**

Functionality is a system's ability to act upon its environment. It is not just what the system does, but what it does to its environment. A car's functionality includes its speed, acceleration, and cornering. A system's functionality can be given as a set of desired environment changes. The greater the changes, the greater the functionality. An information system's functionality is its ability to change information, e.g. a word processor's can change a document. The functionality of an HCI system is the user tasks it can do, where a task is now a cognitive purpose [Zigurs et al., 1999]. The advantage of highly functional software is that it gets the job done - it is task *effectual*. A functionality focus gives "feature" laden software, like Microsoft Office, where each new version adds or extends functions, assuming more is better. The unkind term "bloatware" describes software with too many features for users to cope with.

##### **Usability**

If every system action uses resources, usability is minimizing those resource costs. If every action costs, it pays to use the minimum effort for a given function, i.e. not using a sledge hammer to crack a nut. For a car, lower petrol consumption (miles per gallon) helps performance by increasing efficiency. For a mobile phone, lower power use and longer battery life also help performance. For software, the resources are CPU cycles and memory. "Light" programs, that use little memory or CPU cycles, can easily run as background utilities. Usability implies a

minimalist approach to software development, removing “bells and whistles” and reducing waste, i.e. “simpler is better”. Reducing action costs can increase speed or agility. For example, reduced instruction set computing (RISC) uses less code to do the same work as complex instruction set computing (CISC), and therefore acts faster. In cognitive systems, the resource of value is human effort, especially for new users easily overwhelmed by detail. Modern screen design aims to reduce cognitive effort, for example, by using existing mental models, structuring information in chunks, managing attention, and using feedback loops to show information only as required, e.g. Windows drag and drop copying uses an existing mental model of spatial movement, and auto-hide only shows the taskbar when the mouse moves to the bottom of the screen, i.e. when needed. Minimizing effort is not always easy, e.g. babies reach for an object with their whole body, not just the reaching arm. Only with time does reaching occur without concomitant leg waving – the child learns what effort is necessary for a result. Usability improved markedly over the last decade, as designers learned to reduce user effort, with graphical user interfaces (GUIs) replacing command user interfaces (CUIs). Easy-to-use software is easier to learn, implement, operate, and maintain, reducing costs for training, installation, user manuals, and help support. In a web setting, usability means a higher percentage of users stay to “click on”. Effort spent on help screens, wizards, interface simplification and user feedback does not increase functionality, but does increase IS performance, and is now an integral part of software engineering [Dray, 1995].

### **Functionality and Usability**

Usability can be taken to include both ease of use and usefulness (or functionality) [Davis, 1989]. Others consider functionality (what the system does) a dimension apart from usability (how it does it) [Hix and Schulman, 1991], as does the WOSP model. If new functions add cost (in user learning or effort), increasing functionality will tend to decrease usability. More functionality means more choice, and more choice requires more cognitive effort. An easy way to increase usability is to reduce functionality. Functionality and usability are in a natural state of “tension”. When under 50% of what a system does is actually used, users may question the cognitive cost of that functionality, in long menus and crowded toolbars. In a survey of computer users by Information Week, 90% of respondents said the complexity of IT is greater than it ever was [Chabrow, 2001]. Yet a critical mass of functionality seems a key factor in software selection – perhaps because one never knows if a function *might* be needed. Given a choice between usability and functionality, users seem to prefer functionality, but given equal functionality, usability can be critical [Davis, 1989]. It seems to be the reason why users preferred Word to Word Perfect – Word offered largely the same functionality, but considerably more usability<sup>1</sup>.

One way to add functionality without usability cost is to re-use old learning. For example, the same clip and paste interface works for text and graphics, though they are internally different functions. Interface consistency reduces user effort, but is not always easy to achieve. For example in Windows 98, dragging a file between drives copied it, within a drive moved it, and dragging an .exe file created a shortcut. Users did not like the inconsistency, and Windows Me dropped the drag an .exe to create a shortcut, though it still applies to files named install.exe and setup.exe. Functionality based on existing cognitive resources does not incur usability cost, e.g. adding 3D to a web site can increase cognitive functionality by adding depth, but incur no usability increase because users (unless their vision is depth-deprived) process images for depth anyway. Usability and functionality require different system design efforts, and combining them successfully requires additional effort.

### **SECURITY AND EXTENDIBILITY**

A system’s boundary controls whether objects or materials in the external environment enter and become part of the system or not. It can be designed to repel external entities (security), or make use of them (extendibility).

#### **Security**

Security is a system’s ability to protect itself against unauthorized entry, misuse, or takeover. A powerful car does not perform well if it has no locks and is stolen. A security breach is equivalent to system failure [Littlewood et al., 1993], and a system failure is a performance failure. Secure hardware is sealed and tamper-proof for this reason. Software distributors prefer compiled to interpreted software because users cannot alter it. The principle of denying entry is the same for

---

<sup>1</sup> One could learn Word in a week, but learning Word Perfect took months.

hardware and software. The threat of hackers and virus programs makes security important for software, involving boundary strengthening (firewalls) or checks (login identification and passwords). Security is, and always was, a critical aspect of IS design.

The WOSP definition of security is

*“the ability of a system to resist attack”* [Littlewood et al., 1993].

This definition is narrower than the traditional concept of security [DSS, 2001; OECD, 1996], which aims to prevent all types of system risk by supporting:

- *Availability*, that data and programs are easy for people to access.
- *Confidentiality*, or control over information disclosure.
- *Integrity*, preserving data and programs from damage.

The WOSP model recognizes availability as ease of use, or usability, but sees both it and confidentiality as distinct from security. It also distinguishes dealing with internal failure (reliability) from defending against hostile entry (security). For example, at an operational level the role of the security forces who protect society differs from that of the engineers who maintain it. In the body, the immune system defends, while damaged tissue repair involves another system. A system can be reliable but insecure (easily penetrated), or conversely secure but unreliable (easily breaks down). The WOSP model also distinguishes security from confidentiality. Unauthorized system entry does not necessarily reduce confidentiality, and loss of confidentiality may occur without system penetration (e.g. an impenetrable but transparent plexi-glass room). While security protects the integrity of internal data, confidentiality concerns its display [Ware, 1984].

### **Extendibility**

Extendibility is a system's ability to incorporate or use outside elements in system operation. For example, a car with a tow bar can attach a trailer, which then becomes, in effect, a part of the car. Tool use, a key factor in human evolutionary success, is the use of outside objects to extend performance. The tool becomes an extension of the person. Extendibility is sometimes called openness. Our hands are *open* to use tools that extend their functionality, but our feet, while capable at what they do, are not open to other functionality. Viruses, organic and computer, show how systems with little functionality can use other insecure systems for their own ends. Extendibility has been a powerful but surprising critical success factor in modern IS. Many thought the early Apple Macintosh a better system than the early IBM PC. It appeared more capable, reliable, secure, and usable. But it was a sealed unit, a propriety black box, unavailable to third party developers. It was not extendable. The openness of the IBM PC design, with slots for expansion cards, may have been a critical success factor in its design.

For a car to attach a trailer, its tow ball must match the trailer attachment size, or it cannot be attached. To be extendable, systems and components must match. One way is to publish “standards” that all can conform to. Another is for a system to make its points of extension known. The benefits of openness explain the value of Netscape making its browser source code publicly available – it increased extendibility. Who would have thought, in the early days of software copy protection, that broadcasting source code would be a means of business success? Open software can now be extended with downloaded components, such as Internet Java applets, that extend the system [McCarty and Cassady-Dorion, 1999]. Object linking and embedding (OLE) allows software systems to use other applications as tools to process data. Just as a program can extend what it does, so data objects can be extended. For example, a document can import an external graphic if it is of a known standard. A clipboard, that allows clip and paste between applications, provides such a standard, allowing data to cross document boundaries.

Extendibility is about the benefits of opening rather than closing the boundary, letting things in not keeping them out, about opportunity not risk. Third party “plug-ins”, and open systems architecture, are now an established feature of IS performance.

### **Security and extendibility**

Increasing security means a “thicker” system boundary, whether for a network or a society. Ultimately the system is like a fortress, with extensive outer defenses, and a small active center. Increasing system security seems to go with centralization of system control, and decreased movement of material and goods across the boundary. Closed societies that focus on security, by



restricting borders and centralizing control, find the reduced trade depresses economic performance [Popper, 1945]. Just as functionality and usability are in a state of tension, so are extendibility and security. Here the tension is between using and denying what is outside. While security is paramount in dangerous environments, it excludes good influences as well as bad. Too much security, like too little, can reduce system performance. Every download, every e-mail attachment, could be a Trojan horse or a gift horse, a danger or an opportunity. Eric Raymond's analogy of the cathedral versus the bazaar styles of software development aptly expresses the contrast between security (the cathedral) and openness (the bazaar) [Raymond, 1997]. The Internet illustrates the power of extendibility. Its security is weak, but it was designed to be highly scalable (easy to extend) [Berners-Lee, 2000]. Some predicted it would soon collapse, being so insecure, but by extending and decentralizing itself, it became strangely strong. The Internet's strength is that were it destroyed tomorrow, people would simply rebuild it – its autonomous parts would recreate the system.

Combining security and extendibility means recognizing the natural tension between letting things in and keeping them out. Security is not simply building a fortress, nor is extendibility giving everyone the key. Both are part of system performance, and software designers must find creative ways to combine them.

### **RELIABILITY AND FLEXIBILITY**

A system's internal structure manages and supports the system. It can be designed to perform the same over time despite internal changes (reliability), or to perform differently to fit environmental changes (flexibility).

#### **Reliability**

Reliability is a system's ability to continue or recover performance despite internal variations, like part failure. It can be measured as the probability of failure-free operation for a specified length of time [Littlewood et al., 1993]. A car that is slow, but always starts reliably, is indeed a wonderful system. Reliable systems continue to operate under high stress or load. If affected, their performance degrades "gracefully", rather than crashing catastrophically, and if they do fail, can be repaired quickly. Error code is essentially code to recover functionality that failed, i.e. to increase reliability. Reliable systems can be trusted to perform, and as planning requires predictability, users may trade functionality for reliability. Unfortunately computer systems are not well known for reliability [Weiner, 1993], and "If builders built buildings the way programmer's wrote programs, then the first woodpecker that came along would destroy society." [Weinberg's law as quoted in Dickson, 1999, p184]. If appliances were sold like software, e.g. a refrigerator with a list of "known faults" and recommended "work arounds", there would be a public outcry. While experienced users never buy version 1.0 of any software, a trend to reliability is emerging, as users demand it, e.g. Windows XP marketing stresses its greater reliability. Companies like Dell guarantee reliability through expensive after-service support, and their customers recognize that value. Computer warranties increased significantly in the last decade, so the cost of replacement and repair is a good reason to produce reliable systems. Reliability is, and always was, an important aspect of computer performance.

#### **Flexibility**

Flexibility is a system's ability to "fit" different environments, and not be rendered ineffective by changing circumstances, e.g. a tracked vehicle trades speed for the ability to operate in any terrain. In IS, flexibility is also called *portability* or *adaptability*. A portability review for groupware reported how a University's plan to meet in a nearby IBM Decision Support Center was thwarted by the weather: one participant proclaimed "What good is all this technology when a little snow can stop the whole thing?" [Wheatley and Flexner, 1991]. Distributed e-mail systems, by contrast, perform regardless of weather or place. Flexibility enjoys a long history of relevance in IS. Acrobat documents are now marketed on their platform flexibility (PC, Mac and Palm). CSMA/CD (Ethernet) protocols, which give nodes flexible network access as required, largely replaced polling protocols, which inflexibly check every node regardless of demand, even though polling is more reliable. Likewise, in database design, the more flexible relational model largely displaced hierarchical and network models. Most software now includes a "preferences" module to configure it, e.g., for different users, countries, and networks. For example, the Windows control panel can adapt Windows for disabled users. Language conversion modules change software into any language, and cultural conversion modules could follow. Flexibility is especially important in social software, as the first author discovered when designing a groupware system with over 150 changeable design parameters for such issues as who could comment, mail availability, vote

visibility, and more. It was expected that user preferences would soon define the best group interaction configuration. This forecast didn't happen. Every new group situation demanded a new configuration. After some years, it became apparent that no configuration is "best" for human social interaction because variability is the rule not the exception [Whitworth et al., 2001]. System flexibility, designed initially as a research luxury, became in the end a practical necessity. In IT management, the need for flexibility is linked to the turbulent business computing environment [Knoll and Jarvenpaa, 1994], and is an important critical success factor in the fast changing world of computers.

### **Reliability and Flexibility**

Reliability and flexibility impose contradictory design requirements on internal structure. A system's base reliability depends on component reliability. Perfectly reliable components should result in a perfectly reliable system – but this outcome is rarely the case. Emergent reliability arises from how system parts interact. If each part's output is another's input, system reliability is the product of the part reliabilities, e.g. a system of 100 sequential components each with 99.9% reliability has only a 90% reliability, and one with a thousand components has only 37% probability of correct performance. If the system fails when one part fails, reliability drops drastically as the number of parts increases, i.e. as systems develop. Reliability thus favors reducing part inter-dependencies. Therefore, programming texts advise minimizing subroutine interactions (or coupling) to reduce error. Another way to increase reliability is duplication or redundancy. A plane may duplicate critical control cables, so if one fails the other can take over. In information exchange, redundancy (information duplication) protects against transmission failure, perhaps why English is about 50% redundant [Shannon and Weaver, 1949, p104]. In ASCII transmission the redundant check digit may signal an error, requiring re-transmission (recovery). The military cross-train personnel, so if some die, the rest can continue. An army of perfect specialists would be maximally efficient, but also maximally fragile. In sum, *de-coupling* and *redundancy* are ways to increase reliability.

Now consider the demands of flexibility - to fit functionality to environment change and be change receptive. Base flexibility again derives directly from part flexibility. A tracked vehicle, such as a military tank, is flexible because its tracks operate on a variety of terrain. But often this is not so. Although one can replace parts entirely, like a "flexible" business that simply fires and hires as times change, adding or removing system parts is expensive. Nature prefers emergent flexibility, which derives from how parts interact, like a company that re-structures jobs internally when times change. Emergent flexibility is the variety of adaptive configurations a system can adopt, like a flexible person who can "flex" into a variety of yoga positions. For example, a car can attach chains to travel in snow, giving a new configuration, and then remove them for highway driving. Because more parts to configure means more effort, an 8-wheel vehicle is harder to convert than a four wheel one. Flexibility thus favors less redundancy, not more (as reliability does). Hence programmers are taught modularity – to place each logic in a single place, and make all other uses sub-routine calls. Then, if needs change, the programmer must change only one place. However such modularity increases part interaction, which as shown earlier, reduces reliability. Reliability seeks to contain ripple effects, where one error spreads and crashes the system, but coupling can also increase adaptive variability. Millions of colors are generated by the eye from the interaction of just three retinal frequency receptors (red, blue and green). The astounding information power of genetic code, where a zygote the size of the period at the end of this sentence contains enough information to create a human being, arises from gene interaction. Rather than one gene affecting one trait, many genes combine to affect each trait, and most genes affect many traits. More information can be stored in the interactions than in the elements themselves. In sum, *interactivity* and *modularity* are ways to increase flexibility.

Reliability and flexibility are in a natural design tension. Increasing flexibility tends to decrease reliability because every change is also an error opportunity. For example, Windows users can flexibly change their file type associations. The first author accidentally associated icon files (\*.ico) and Word, so every desktop icon became a Word file, effectively disabling the desktop. If reliability and flexibility are in design tension, to increase performance designers must address both at once, e.g. early Windows users could flexibly change their screen settings, but changing to an unsupported screen level meant the user could not recover, as they could no longer see the screen. Now, unless the user confirms screen changes within a few seconds, the system recovers the previous settings, i.e. more flexibility also required more reliability. Many options modules now include a "Restore Defaults" button, so users can recover from changes. Likewise increasing reliability may require more flexibility. For example, Windows Me's System File

Protection (SFP) scans system files and replaces suspicious ones with the originals. It seems a great idea to avoid file corruption errors. However flexibility was ignored, as it uses fixed criteria, set when Windows Me was released. Therefore, if you upgrade an application to a better driver, SFP will instantly undo the upgrade next time it runs. SFP design ignored the fact that systems must change as well as not change; hence, experienced users often disable it.

Perfect reliability, like perfect security, seems a myth. While theoretically error code covers all contingencies, in modern event driven software, the combinations increase factorially with the number of messaging objects. Perfect reliability soon becomes practically impossible, as every combination cannot be checked. In a changing environment, perfect reliability may be a fault not a virtue. Hence designers of computer agents searching the Web are exploring the benefits of serendipity, or chance system variability (i.e. unreliability!) [Campos and Figuerido, 2002]. As software advances, the WOSP model suggests that combining reliability and flexibility is more beneficial than seeking either exclusively, giving what is called internal and external robustness [El Sawy and Nanus, 1989].

### **CONNECTIVITY AND CONFIDENTIALITY**

The world wide web introduced a new social dimension to software performance, involving communities, conversations and networks. Social interaction is a performance multiplier, critical in human biological success [Maturana and Varela, 1998]. People in societies create non-zero-sum benefits from the synergy of cooperation using communication [Wright, 2001]. While previous purposes are individualistic, communication requires two or more systems. Systems can be designed to enable information and meaning exchange (connectivity), or to limit it (confidentiality).

#### **Connectivity**

Connectivity is the ability of a system to support information and meaning exchange, about the external world, personal states, and group action [Whitworth et al., 2000]. Functional actions were earlier attributed to effectors, because effectors create actions. Even so, actions occur in a feedback loop, where perception plays a guiding role. Communication also involves a feedback loop, but the end result, *meaning*, is now created by receptor activity. Meaning is created by system receptors, just as actions are created by effectors. A car's horn works because other drivers hear and attach meaning to it. For cars to be connected, they would need to directly sense each other (e.g. to avoid crashes using radar receptors). The signals of communication are arbitrary. For example, people can communicate by voice, sign language, facial expression, smoke signals, and in many languages. Communication requires no particular signal, only common processing. When children learn a language, they learn common ways to process arbitrary signals. This requires not strong muscles, claws or teeth, but "strong" eyes, ears and sensory processing. Computer communication also requires common processing. Protocols like ASCII, RS232 and TCP/IP are essentially "social" agreements on how arbitrary signals are processed. Connected software allows users to interact socially with others, or can itself connect to stay current by downloading updates. Connectivity includes how signals are routed, for example circuit vs. packet switching. It also includes the number of channels, channel bandwidth, immediacy, symbol variety, and rehearsability [Dennis and Valacich, 1999].

Connectivity is distinct from functionality, as its goal is meaning exchange, not environment change. Without a listener, speaking is functionally just the creation of air turbulence. An e-mail system's functionality is its ability to act as a word processor, and create message content. By contrast, its connectivity is its ability to send and receive messages. A system could create graphical messages, but only connect by an ASCII text channel, or vice-versa. Connectivity is also distinct from extendibility. Communicating parties have mutual choice, but when a virus program uses another system, it is not a mutual affair. Importing information into a document is not communication, which requires an independent and autonomous other. In today's online world, connectivity is an important new aspect of system performance.

#### **Confidentiality**

Confidentiality is a system's ability to limit the release of information about itself. Not releasing information can be as important as communicating. Car anti-radar equipment tries to confound radar detectors, and tinted windows keep out prying eyes. Camouflage in animals is an attempt to keep confidentiality, as are military "stealth" capabilities. In a social setting, confidentiality is usually called privacy, and is important because how one is viewed affects how one is treated socially. Being viewed significantly energizes the viewed party, an effect psychology calls social

facilitation [Geen and Gange, 1983]. Confidentiality is important for social beings, whose well being depends heavily on how the group sees them. Most people do not wish to be viewed asleep for example, as unconscious words or acts could lead to ridicule or contempt. In software, subroutines can declare variables private (not released) or public (released). Good reasons exist not to make all variables public. Encryption is one way software can protect information from display.

Privacy seems based on the idea that a system should own information about itself [Whitworth and deMoor, 2003], and hence have the choice to release it or not. US Supreme Court Justice Louis Brandeis described privacy as the 'right to be left alone', i.e. to not communicate. Confidentiality is as important as connectivity in social interaction. The widespread selling and storing of personal information on the web for profit, with effects from identity theft to electronic "spam", led to public complaint [Wright and Kakalik, 1997]. Online actions can be recorded electronically, limited only by a storage capacity that is doubling every eighteen months. It may soon be simple to record all actions of all people on the Internet. If so, everything said and done in cyber-space could, as is currently the case for public media figures, be "on a record". Without privacy, something said in a chat room by a person at 18 could be publicly re-quoted twenty years later. For example, statements Billy Graham made about Jews, in a supposed private telephone conversation with Richard Nixon over twenty years ago, were recently released, much to his chagrin, when Nixon's tapes were made public. Few private citizens would like everything they said on a permanent, searchable electronic record. Computers could make cyber-space an unforgiving place, where everyone is a public figure, always "in view" and "on record" [Brin, 1998]. It does not have to be this way. Johnson [2000] makes the case that privacy is a social good, allowing people to be themselves, free from social pressure. Confidentiality seems a "sleeper" issue whose importance will increase as the Internet becomes more social [Agre and Rotenberg, 1997].

#### **Connectivity and Confidentiality**

While connectivity increases information and meaning exchange, confidentiality limits it. Both are natural requirements of social interaction, and naturally opposed. Yet privacy is not just about holding back personal information, but rather about user choice over their own information [Nissenbaum, 1997]. People want to communicate by choice, and be private by choice. For example, when you enter a public space you can see others, but equally they can see you. You choose to be viewed so that you can also view. To be like the invisible man, able to see others who cannot see you, seems an unfair social arrangement. Fairness – that if you see another, they should be able to see you – seems a way to reconcile connectivity and confidentiality [Whitworth and deMoor, 2003].

#### **IV. USING THE WOSP MODEL**

The WOSP model can be used as a:

- *Process* oriented system design framework for designers developing systems, or
- *Product* oriented evaluation framework for users making system purchases.

#### **WOSP REQUIREMENTS SPECIFICATION**

While most system design models, like the waterfall model, define *how* design should proceed, the WOSP model defines design *goals*. It is a multi-goal orientated approach to requirements engineering, as suggested by Chung et al.[1999] As goals may contradict, design becomes a balancing act in the space of system performance. The overall goal is system performance, i.e. successful environment interaction. It is a more dynamic approach to software engineering than the traditional one-dimensional production line path to "excellence" via fixed specifications [Clegg et al., 1997]. It suggests the following for advanced system design:

1. *Environment fit*. Performance profiles must "fit" the environment .
2. *One-dimensional development*. Can be of limited value.
3. *Balanced development*. May give the most benefit.
4. *Combination breakthroughs*. May be necessary for system success.
5. *Development strategy*. Need both more specialization and integration.

#### **Environment Fit**

The web of performance is a multi-dimensional performance space, in which different system designs position themselves. Each design has not only a capacity, but also a profile. Which profile aspects are valued depends on the environment or situation. Some require systems with

simple functionality but high security, others need high usability but ignore confidentiality. While excellence in all areas is ideal, one may deliberately choose efficiency of code size over run-time flexibility, if this approach “fits” better. In a rapidly changing world, flexibility and extendibility may be important, while in a threat environment, security and privacy may need more focus. For a mobile workforce, the time to learn a system (usability) may be critical - a system that takes three months to learn is of little use if workers tend to leave after 6 months. Designers could estimate the resource allocated to each goal, as shown in Table 1, to get a system development profile.

Table 1. WOSP Goal-Environment Fit

<b>Goal</b>	<b>Detail</b>	<b>Resource%</b>
<i>Functionality</i>	To act effectively upon the environment.	
<i>Usability</i>	To operate efficiently, easily or quickly.	
<i>Security</i>	To resist or avoid outside attack or take-over.	
<i>Extendibility</i>	To use outside components or data.	
<i>Reliability</i>	To avoid or recover from internal failure.	
<i>Flexibility</i>	To change to fit outer circumstances.	
<i>Connectivity</i>	To communicate with other systems.	
<i>Confidentiality</i>	To control internal information release.	
<b>Performance</b>	<b>To interact successfully with environment.</b>	<b>100%</b>

### One-Dimensional Development

The WOSP model suggests that improving any aspect of system performance can create tension in others. For example, the connectivity of the Internet raises design problems for security [James et al., 2001] and privacy [Lessig, 1999]. Early developments may be easy, because “slack” is available, but as performance increases so, it is suggested, does the tension. The more a system advances, the harder it is to advance further by single feature progress. If the web of performance is elastic, pulling the web at a single corner only may reduce the value at other corners, so one-dimensional technology gains can “bite back”:

*“The more powerful systems have become, the more human time it takes to maintain them, to develop the software, to resolve bugs and conflicts, to learn new versions, to fiddle with options.” [Tenner, 1997, p266].*

The designer’s paradox is that adding a feature can even reduce system performance. For example, consider adding video/multi-media functionality to e-mail. Video e-mail could then become less usable (need extra training). Video-viruses may be easier to conceal in multi-channel data, reducing security. Extendibility could be less, if video e-mail cannot be printed. It could be less reliable (more likely to fail), and less flexible if versions do not exist for Macintosh or Unix users. It could provide less connectivity if it overloads network channels. Finally, if users cannot control where the camera points, they may release unwanted information (such as about the room behind). Who wants to tidy their room before sending an e-mail to Mother? A gain in functionality could reduce usability, security, extendibility, reliability, flexibility, connectivity, and confidentiality. Depending on the value of video, the result could be a net performance loss. For video e-mail to out-perform current e-mail, and supercede it, developers cannot just add video functionality, but must “weave” it in with other performance goals.

### Balanced Development

Unintuitive as it seems, improving a system’s best aspect, which probably contributed most to its success, can be the *worst* way to increase performance, even though this aspect is the current design focus, and where pundits look to predict the future. If performance is the web’s area, it is most increased by improving the weakest (and least obvious) aspect. This concept seems why technology trends tend, after a while, to go in unobvious directions. For example, virtual reality goggles were touted a few years ago as the future of gaming (based on previous multi-media gaming advances). However it did not take off, while interactive gaming, connecting in virtual social worlds, became popular (e.g. The Sims). Games also became more extendable, as users produced plug-in files to add new creatures, weapons, and even rules, e.g. Doom’s WAD files. Game editors allow users to create scenarios or maps that others can download and play, and expansion packs that extend games sell well. Similarly in business, while more of the same media richness was seen as the future of IS, recent advances are in anytime, anywhere flexible

computing. That balance, rather than power, is the key to success may explain why broadly successful “killer” applications, like browsers, e-mail, and chat rooms, are surprisingly simple in their functionality.

**Combination Breakthroughs**

Application advances often do not come easily. In 1992 Apple CEO John Sculley introduced the portable, hand held Newton, saying portability was the wave of the future. We now know he was right, but developing flexibility alone was not enough. The Newton’s handwriting recognition was poor, making it hard to use, and in 1998 Apple dropped the line. The flexibility advance was neutralized by reduced usability. Only when Palm solved the usability problem, with its Graffiti language, did the market for PDAs revive. The WOSP model implies that as systems advance, simply increasing one system aspect alone may be insufficient. Advances may require combination breakthroughs, i.e. simultaneous development at more than one corner of the web.

**Development Strategy**

The WOSP model implies that as systems advance, development will require a multi-focus effort, with both more specialization and more integration. But why not just create one list of “system requirements”? Although possible, designing a user interface and a database update require different skill sets. Designing for plug-in extensibility needs knowledge of data standards, while security design needs knowledge of virus threats, suggesting specialized teams of various sizes for each goal, e.g., a security team, a usability team, a standards team, and a privacy team. Table 2 suggests how each team could design a different system layer, with different specifications, code, and testing. for example, error code can be designed, written and tested apart from functional code.

Table 2. WOSP Layers for Analysis, Design and Testing

<b>Goal</b>	<b>Analysis</b>	<b>Design</b>	<b>Testing</b>
<i>Functionality, capability, usefulness, effectiveness</i>	<i>Functional analysis:</i> Analyze desired environment effects	<i>Main application layer:</i> What the system does	<i>Validity testing:</i> input/output test bed
<i>Usability, simplicity, parsimony, efficiency, ease</i>	<i>Usability analysis:</i> Analyze user processing costs, feedback cycle time, cognitive effort, simplicity	<i>Interface layer:</i> Help system, tutorials, wizards, advice pop-ups, status bar feedback	<i>Usability testing:</i> cognitive walk through, protocol analysis, user heuristics
<i>Security, defense, protectiveness</i>	<i>Threat analysis:</i> Analyze external threats and denial methods, identify weak points	<i>Security layer:</i> Logon and system entry, authorization and security levels	<i>Penetration testing:</i> Unauthorized entry probes, attack tests, hacker simulation
<i>Extensibility, openness, scalability, compatability</i>	<i>Interoperability analysis:</i> Analyze third party data, standards and tools	<i>Portal layer,</i> plug-in manager, import & export data, clipboard	<i>Compatability testing:</i> Plug-in testing, import/export testing
<i>Reliability, stability, dependability</i>	<i>Contingency analysis:</i> Analyze possible failures & recovery	<i>Recovery layer,</i> error recognition, backup, and recovery	<i>Failure testing:</i> Load tests, disaster recovery tests
<i>Flexibility, adaptability, portability, customizability, plasticity</i>	<i>Adaptability analysis:</i> Analyze likely environment changes and possible responses. Includes trends analysis	<i>Configuration layer,</i> control panel options, user preferences, system parameters, change management	<i>Situation testing:</i> Test different hardware platforms, languages, users, operating systems,
<i>Connectivity, interactivity, sociability</i>	<i>Communication analysis:</i> Analyze ability to exchange information	<i>Interaction layer,</i> protocols, networking, channel transmissions	<i>Transmission testing:</i> Test for sending/receiving information.
<i>Confidentiality privacy, secrecy, rights</i>	<i>Legitimacy analysis:</i> Analyze who owns what information.	<i>Rights layer,</i> System permissions, transfer and delegation of rights	<i>Social testing:</i> Test the community accepts the system

Such specialization creates a need for integration, because developing for one purpose may “cut across” another [Moreira et al., 2002], giving design conflicts that must be traded-off, prioritized, or creatively resolved. For example, a logon sub-system (security) could make a system less usable, but equally is a chance to welcome the user by name, and remember their preferences,

i.e. also make the system more usable. Specialists must work together to achieve such synergies. Though the WOSP design purposes are distinct, there is only one system. Fitting one purpose while denying another is like taking from housekeeping to pay the gas bill. Each team could flag areas for discussion with other teams, or opposing teams could be combined, e.g., if functionality and usability specialists were on the same team, they would tend to reconcile differences. This would result in four teams, considering actions (functionality and usability), interactions (security and extendibility), contingencies (reliability and flexibility), and communications (connectivity and confidentiality).

### WOSP SYSTEM EVALUATION




The WOSP model can help evaluate one or more system's performance.

#### Measuring WOSP Aspects

System aspects like security, usability, and reliability are often considered "quality" features, that constrain or modify functionality [Nixon, 1998]. The WOSP model does not make this distinction. Every aspect, it is proposed, can constrain or modify another, e.g. usability can compromise security, and security can diminish usability. The functionality goal in Table 1 differs from others only in being more obvious and easily specified. It can be detailed as a set of desired environment effects, e.g. to authorize sales, maintain accounts and print statements. These "tasks" are a qualitative measure of functionality. They can be decomposed further into sub-tasks of any detail. Functionality can be measured at the ordinal level by asking people to rate their satisfaction with these tasks, and at the interval level by the percentage of desired base functions the system supports [Hix and Schulman, 1991]. It is less obvious that the usability goal of Table 1 can similarly be decomposed and measured, but as user rather than environment effects. For example, the "three click rule" (any task must be doable in three clicks) is a usability "task". Gediga et al reduced 651 items from usability inventories to 90 items for the seven usability "principles" of ISO9241-10, like self-descriptiveness, and conformity to expectations [Gediga et al., 1999]. This goal decomposition allowed system experts to rate usability, giving scores that reliably and validly discriminated between systems [Gediga et al., 1999]. Just as one can develop a prototype user interface with no functionality, so usability can be rated independently of functionality. Naturally there is no usability without functionality, but equally unusable software effectively provides no functionality. The same logic applies to reliability, security and flexibility, as system failure, system attack, and environment change can render functionality null. The detail is beyond the scope of this paper, but the same approach, of decomposing goals, can be applied to the other WOSP dimensions of Table 1, giving cardinal measures (desired effects), ordinal measures (rating scales), or interval measures (summary scores). For example the TRUSTe trademark system is a qualitative rating of web site privacy [Benassi, 1999], and mean time between failure (MTBF) is an interval measure for reliability. The WOSP constructs are conceptually distinct, and so measurable.

#### Combined measurements

WOSP dimensions can be combined, e.g. Hix and Schulman found evaluators could rate the usability of various system functions (Figure 3).

		Usability			
		N/A			
Functionality	Function 1		✓		
	Function 2				✓
	Function 3			✓	
	Function 4	✓			
	Function 5, etc		✓		

Adapted from [Hix and Schulman, 1991]

Figure 3. Usability by Functionality

Note that in combining the two dimensions, functionality is measured qualitatively, while usability is ordinal. If both were interval, one could divide them, as in the traditional performance metric for database performance, throughput (or workload) divided by cost or time per transaction, i.e. functionality divided by ease of action [Bitton et al., 1983]. Both measures are equally part of performance – to prefer one to the other is like taking the numerator of a fraction as more important than the denominator. The approach could be applied to other pairs, for example measuring openness relative to security, or connectivity against confidentiality. Equally Figure 3 could show reliability against functionality, or rate the security risk of different extendibility operations. Measuring such intersections shows where meeting one requirement reduces another, an issue ignored when measuring performance aspects in isolation [Moreira et al., 2002].

### Comparing Webs

If system aspects can be rated, then experts can compare systems in a common environment without developing common measures. For example, compared to DOS, the early Windows operating system seemed to have at least equal functionality but more usability (GUI interface), more flexibility (control panel), more extendibility (drivers), and more connectivity (network ability). Whether it was more secure, reliable, or confidential than DOS is debatable, but it seemed not markedly less so. The resulting Windows web of system performance overlays that of DOS completely (Figure 4), suggesting it performs better, and should replace DOS, given marketing and change management to overcome the inertia of an entrenched system.

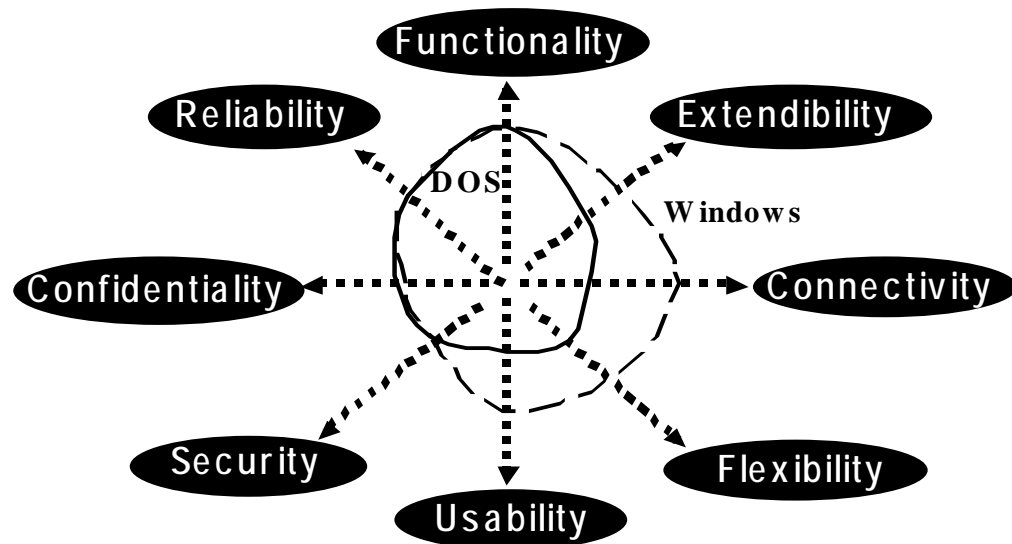


Figure 4. WOSP comparison of DOS and Windows

Comparing electronic and paper communication systems however yields a different picture. E-mail provides an electronic form with similar capabilities to paper, but better editing. Usability is also high in replying to a message, as the reply address, date sent, sender address, and sender title/text is added automatically, so a reply like "OK. Do it" costs literally only a few words of typing. Users perceive e-mail as more spontaneous than letter writing because it is easier to use [Lea, 1991, p169]. E-mail, like paper mail, can contain attachments. Security is a problem, because e-mail is easy to forge, and does not have the status of a signed legal document. Paper is also more reliable. It never "crashes" and seems more permanent, while network servers can fail and lose e-mail. Paper also seems more flexible - a note can be read almost anywhere, while e-mail requires an online computer to be read initially. E-mail's lack of privacy is a concern, because e-mail is viewable by anyone with file access, whether a network supervisor or hacker.

By contrast, postal messages have a covering envelope, which leaves visible signs if tampered with. As succinctly said "The e-mail of the PC is more deadly than the mail [postal]" [Neumann, 2000]. President Bush's 2001 decision not to use e-mail seems based on its lack of privacy. Yet e-mail offers extraordinary connectivity, e.g. Kiesler reported that a new idea posted on



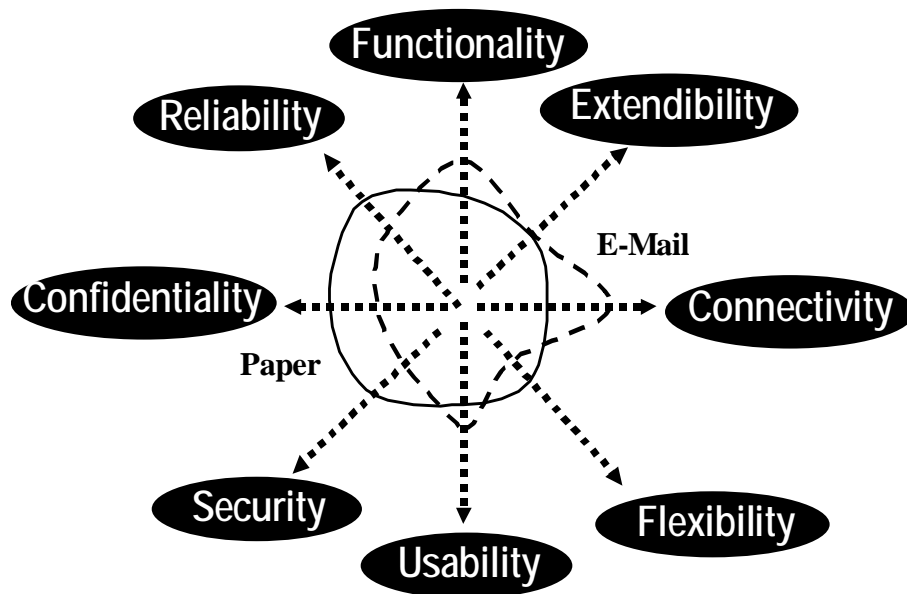


Figure 5. WOSP Comparison of E-Mail and Paper Communication

a computer network was directly sent to 300 colleagues in branches across the country, and in two days generated enough replies to launch a new project [Kiesler et al., 1984]. Figure 5 summarizes these conclusions, showing e-mail with more functionality, usability, and connectivity, similar extendibility, but less reliability, flexibility, security, and confidentiality than letter mail. The e-mail web of performance only partially overlays that of paper communication, suggesting it will only partially replace it, as it has. For e-mail to replace paper entirely (i.e. the paperless office), the WOSP model suggests it must become more reliable, flexible, secure and confidential. These changes may already be happening. Computer network reliability is improving every year, pervasive wireless networks now allow anywhere access to e-mail via small portable systems, electronic bio-identity devices could make e-mail as secure as a written signature, and public encryption could provide a much needed privacy “envelope” for e-mail. But until electronic mail’s web of performance does expand, we can expect paper to remain in use.

#### NEW TECHNOLOGY

The WOSP model provides a useful framework for new technology. Wireless devices make people’s work more flexible (or environment independent), but other WOSP factors still apply. On a hardware level, cell phones can be more or less functional (voice message quality), usable (keys not too small), portable (easily carried), reliable (if dropped), secure (if stolen), extendable (earphones, phone covers), connected (to communication channels) and private (others cannot overhear conversations). On a software level, the same goals provide challenges for a ubiquitous computing application [Banavar and Berstein, 2002], which may have to work with different devices, in different networks and for different users, i.e. move seamlessly from one environment to another. It must be resilient to data entry errors, network unavailability, or device power failure, and recover graciously from them. The software must be scalable, to add support for new products, and secure to protect against virus attack. Lack of both standards and security are currently major obstacles to mobile wireless application development [Smith et al., 2002, p472]. Advancing in one area can cause problems in another, e.g. upgrading connectivity from one-to-one 1G circuit switching to 2.5G “always on” packet switching created security problems. On cognitive and social levels, privacy may be the Achilles heel of mobile computing. Anyone can pick up broadcast wireless signals with the right equipment. Not just cell phones, but also home wireless networks, could be monitored. The issue is not just message content (as when reporters

overheard Princess Diana using her cell phone for a private meeting), but also message properties:

*“A key feature of these devices [wireless devices] is their ability to identify a user’s location, if the user is connected, and, in the future, who the user is.” [Smith et al., 2002, p469].*

General Motor’s developed the OnStar system to provide navigation support for its cars. But who owns that information, which includes user identity, location, and time? Could it, for example, be subpoenaed in a messy divorce case? The potential privacy problems of mobile computing may challenge its effectiveness. In a social world, people do not want to live in glass houses. It is not an insoluble problem, because a system could be designed to not store personal data, and so avoid privacy issues. Balance implies that while success requires many causes, failure needs only one. The WOSP model is a useful checklist for weaknesses.

## IV. DISCUSSION

### THEORY INTEGRATION

The WOSP model extends and integrates previous theories, including TAM, the general security model, and non-functional requirements research.

#### TAM Theory

The Technology Acceptance Model (TAM) proposes that perceived usefulness (functionality) and perceived ease of use (usability) affect the user decision to accept software [Davis, 1989]. The WOSP model extends TAM, by proposing that other factors, like reliability and security, also factor into user acceptance, i.e. that users base system acceptance on a general estimate of system performance.

#### General Security Model

The general security model developed in the context of a hostile environment [Jonsson, 1998], and “security” came to cover all properties that reduce the risk of failure and increase “survivability” [Lipson & Fisher, 2000]. The WOSP model suggests that to succeed in an environment, a system must thrive as well as survive, so survival is not equivalent to successful performance. Success requires systems also take opportunities, via functionality, extendibility, flexibility and connectivity. The WOSP model is thus also an extension of the general security model.

#### Non-Functional/Quality Requirements

Requirements like security and reliability are often considered “non-functional” requirements (NFRs). Though specified as functional requirements [Rosa et al., 2001], they are considered different because [Nixon, 1998, p132]:

1. They can conflict with each other.
2. Specification techniques are varied.
3. Their impact is global.
4. Their relevance varies from system to system.

However these differences are not absolute.

1. Goals like security can conflict with functionality, as well as other NFRs.
2. Functional specification methods vary.
3. A system’s functionality also impacts globally.
4. While the relevance of NFRs varies among systems, the relevance of functionality also varies (relatively). Some systems contain more error code, or interface code, than functional code.

The WOSP model does not distinguish “functional” from “non-functional”, but sees them all as “requirements” which modify one another. Their existence and interaction are considered system realities that must be addressed. Many software failures can be attributed to treating NFRs as “a second or even third class type of requirement ... frequently neglected or forgotten.” [Cysneiros and Leita, 2002, p699]. All requirements should be considered from the beginning of software development, as the cost of adding them later increases rapidly with time. If performance is more than just functionality, the priority of performance goals should be assigned, not assumed. In dynamic, mixed opportunity/threat environments, like the word wide web, users want systems with functionality and usability, reliability and flexibility, security and extendibility, connectivity and

confidentiality, i.e. balanced systems. Anything less may tend to be less successful. The WOSP model integrates and extends a variety of NFR specialist research.

### Theory Expansion

Each WOSP aspect of system performance is valid but incomplete. As a 1997 security review notes:

*“The face of security is changing. In the past, systems were often grouped into two broad categories: those that placed security above all other requirements, and those for which security was not a significant concern.” [Kienzle and Wulf, 1997].*

If security must co-exist with system needs outside its specialty, one response is to broaden the concept. Perhaps integrity, reliability, availability, and confidentiality evolved as the global security concept grew. While this broadening fills a theoretical vacuum, it creates conceptual confusion, e.g. mechanisms that increase fault-tolerance (reliability) can actually reduce system security, which is illogical if reliability is an aspect of security [Jonsson, 1998]. While security could be part of a unifying “dependability” concept [Laprie and Costes, 1982], recent models reclaim the reliability/security distinction, as the first is based on provision of service, while the second is based on denial of service [Jonsson, 1998].

Other specialties seem also subject to theoretical expansion. The usability measure of Gediga et al. [1999] includes “suitability for the task” and “error tolerance” as aspects of usability. Usability, like security before it, is in danger of becoming a confusing catch-all term for performance. A review of flexibility in IT management includes “scalability” and “connectivity” as potential aspects of flexibility [Knoll and Jarvenpaa, 1994, p6]. Again a specialist term is conceptually expanded to fill the theory space available. The WOSP model suggests it is better to retain modular constructs, but recognize their intersection. Security then is as much an aspect of reliability as reliability is of security, or reliability of usability. The unifying concept is system performance.

### The Technology Advance Cycle

That performance is not uni-dimensional explains why progress is not uni-directional. Progress is not a train on a single set of railway tracks. Rather it seems on many tracks simultaneously, switching between them in unexpected ways. Hence neither the explosion of the Internet, nor the cell phone expansion was predicted by experts [Smith et al., 2002]. Berners-Lee’s World Wide Web project was rejected by CERN, the Hypertext community, and Microsoft, before becoming the way of the future [Berners-Lee, 2000]. But once new advances are recognized, suddenly only their benefits are seen, while the problems they will create are not [Standage, 1998]. Cars, first seen as avoiding pollution (from horse dung), are now major air polluters, an entirely unpredicted problem. Today’s computer hype also occurred with the telephone a hundred years ago. The most recent hype, mobile computing, is reminiscent of the multi-media hype of a decade ago. The promise that multi-media interaction will solve everything is being replaced by the promise that mobile computing will solve everything. The WOSP model supports neither expectation. Many who over-invested in bandwidth alone discovered to their cost that progress is not uni-dimensional. More media power now seems like just the nearest hill on a journey. Four stages of technology advance seem to exist:

1. *Naivety*: The advance is unexpected to current experts.
2. *Resistance*: The advance is resisted by stakeholders unfamiliar with it.
3. *Hype*: The advance is expected to change everything for the better.
4. *Realism*: The advance “settles”, with both benefits and problems.

The middle “uncertainty” stages of the cycle often involve considerable avoidable loss, either as opportunity loss from under-investing in progress (stage 2), or as risk from over-investing in hype (stage 3).

The WOSP model suggests a realistic approach, minimizing stages 2 & 3.

- Prior advances may not be the best guide to future benefit. Experts are, by the nature of things, experts in the past. In progress, we must expect the unexpected, for it is sure to occur in some form.
- If developing one aspect can reduce others, there is no one “answer”, whose improvement will solve all problems.

Performance is always a balancing act, and advances seem like vitamin C - too much can be as much a problem as too little. The WOSP model is a realistic multi-attribute approach espousing balance. It encourages wiser system design, system evaluation, and system investment.

### REQUIREMENTS INTEGRATION.

The issue of integrating performance requirements is still in its infancy. The WOSP model proposes multiple goals that pull system design in different directions. Fortunately the web is not rigid – it can be expanded, e.g. functionality is not the inevitable enemy of usability [Borenstein, 1990], and Internet connectivity need not mean the end of privacy. What at first seems a zero-sum allocation need not be so. But creating performance synergies by reconciling two or more performance dimensions almost always requires creative system design. The challenge of IS to devise creative ways to reconcile apparently contradictory performance goals, and genuinely expand the web of system performance.

#### Flexibility and Reliability

For example, how can a system be both flexible and reliable, both de-coupled/redundant *and* interactive/modular? A maximum reliability system would contain many duplicated parts, operating in parallel, with minimum interaction. Then if one part fails, the others can continue – performance degrades but does not collapse. Plants illustrate a reliability focus – many cells are equi-functional, with cellulose walls that prevent interaction, hence plants cannot get cancer (the cell walls prevent it spreading). However a maximum flexibility system would contain specialized non-redundant parts, so that changes in one place can alter the entire system. The human cortex seems designed for such flexibility, with massive interconnectivity, each neuron connecting to up to ten thousand others. Rather than storing memories in single locations, as hard drives do, the brain seems to store them in the part interconnections, so one memory involves many neurons, and one neuron is involved in many memories [Copeland, 1993]. Hence human memory never reports a “disk full” error. But while extraordinarily flexible, like a database indexed on every data field, human memory is not particularly reliable. Presumably natural selection preferred memory flexibility to reliability. But the brain in general is extraordinarily reliable. Most computers “hang” or malfunction at least once a month, while brains do so more rarely.

#### EXAMPLE

Consider the case of Phineas Gage, a railway worker. A flying iron rod smashed the middle and left lobes of his cerebrum. Within minutes he was conscious and speaking, and while showing disturbed behavior, lived for 13 years, and died of unknown causes [Copeland, 1993]. It is difficult to imagine pushing even a pin through a computer without total collapse.

How can a brain, built from a thousand billion highly coupled and unreliable neurons, be more reliable than a computer with fewer, more reliable, components? One answer is its design, which *overlays advanced flexible sub-systems over primitive reliable ones*, like a plane that develops computer controls, but retains the “fail-safe” manual ones, in case the onboard computer crashes. In the brain, new sub-systems suppress old ones, but leave them intact (and available if the new ones fail). This “overlay it” contrasts with the usual “replace it” approach to software updates, and may be a better alternative [Meadows, 1997]. It increases both reliability and flexibility, because it provides two potential processing sources. For example, retaining a keyboard interface while adding a mouse interface seems inefficient, but it adds both flexibility (keys can be faster for experienced users) and reliability (keys can be used if the mouse fails). Operating systems are built in a layered fashion, and their reliability depends on their “kernel”. Hence for Microsoft to increase Windows reliability, it had to change its base layer from DOS to NT, at great cost.

Modern error systems also illustrate sub-system layering. While early error code operated at the mainline functional level, it resulted in “spaghetti” code. It was found better to separate error code and functional code. Now an error control unit “calls” the main application, so if it fails, control “falls back” to error recovery routines. One problem with a master control unit is that it could also fail, causing the whole system to fail, another is how to update it. Multiple processing, however, raises subtle issues of how control is transferred.

#### Specialization and Holism

System evolution seems to require both greater part specialization and greater interaction among parts. A system’s differentiation is the degree of difference between its constituent parts, e.g. in embryonic development, each cell begins as equivalent - any cell can develop any function. As the body develops, cells specialize into different forms and functions (hair cell, blood cell, brain

cell, etc). Likewise in simple social groups, most members can perform most functions, but as societies evolve the division of labor increases. [Wright, 2001]. As systems advance, their parts become more specialized, but more specialization means more complex part interaction. Therefore, advanced systems tend to be "holistic". Holism is how much changing one system part changes the properties of others [Bertalanffy, 1968]. Holistic system parts may gain key properties only when in relation to others [Esfeld, 1998]. Thus most biological sub-systems behave differently when isolated than when within a body [Bertalanffy, 1968, p68]. In holistic systems, a small change can cascade, creatively or catastrophically, giving what chaos theory colorfully calls the "butterfly effect" ("The flap of a butterfly's wings in Brazil can cause a tornado in Texas") [Gleick, 1987]. Since changing any part, even slightly, can change the whole drastically, holistic systems are individualistic [Bertalanffy, 1968]; for example, one rarely finds two people who look exactly the same. Complex programs, like the Windows operating system, are becoming more holistic, e.g. small differences can have cascading effects that make each user's Windows installation different. Ignoring holism in technology design gives what Tenner calls the "Frankenstein effect" – a cause of low system performance [Tenner, 1997]<sup>2</sup>. The issue of managing both more specialization and more internal interaction resolves into one of transfer of control, of what sub-system should act when. Natural selection has created an unexpected solution: autonomy.

### **Autonomy**

For a system to advance, its parts must specialize, which means only they know what they do, and hence when to act. If a master control mechanism directs action, specialty knowledge must be duplicated in the control mechanism, which defeats the point of specialization. For specialization benefits to be realized, system parts need autonomy, the ability to direct their own actions. This principle applies to societies, where specialized workers are more productive if given freedom. It also seems to apply to software. While early software used a single main control module to direct all actions, object orientated systems (OOS) grant "objects" limited autonomy to initiate interactions by messages to other objects. It distributes rather than centralizes control. Computer system part autonomy is increasing, e.g. screens that turn themselves off, printers that perform self-maintenance as required, and operating systems that decide to check the hard drive if you shut down incorrectly. Systems designed following the principle of autonomy would not contain a master control center. Each part would act according to its nature, and system actions would arise from the part interaction. The human brain seems to work in this fashion, as a collection of sub-systems with no CPU [Whitworth, 1975]. For example, "split-brain" studies show that the highest level of brain operation (the cortical hemisphere) is duplicated, and each hemisphere can operate autonomously, as a "brain" in itself [Sperry and Gazzaniga, 1967]. Autonomy seems the key to advanced system development. IBM is exploring how increasing autonomy can improve IS performance [Mowbray, 2002].

### **Final Comment**

In designing systems to satisfy multiple purposes, nature seems an excellent guide, and we ourselves a good example. Our imperfection on any single scale is as manifest as our success overall. Naïve ideas of linear progress must be abandoned for a realistic view of disparate system goals in design tension in a changing environment. The goal of advanced systems requires balance, and advocates of single point "excellence", whether it be reliability, privacy, security or any other, should recognize each other. The situation we are in is ironic, being ourselves advanced biological systems, with millions of years of system development, trying to create advanced computer systems. To do so we must understand the general requirements of advanced systems. It is a rare system that comprehends the principles of its own design. However, since the systems we create must work with the systems we are, based on the same world, the best guide to advanced computer system design may be ourselves.

### **ACKNOWLEDGEMENTS**

Thanks to Beryl Plimmer, Manukau Institute of Technology, New Zealand, for first suggesting the various system properties could be shown as a "web".

*Editor's Note:* This article was received on August 26, 2002. It was with the authors for approximately 5 and a half months for 2 revisions and was published on September 2, 2003.

---

<sup>2</sup> Dr Frankenstein made a human being by stitching together the best of each individual body part he could find in the graveyard. The result was a monster.

## REFERENCES

EDITOR'S NOTE: The following reference list contains the address of World Wide Web pages. Readers who have the ability to access the Web directly from their computer or are reading the paper on the Web, can gain direct access to these references. Readers are warned, however, that

1. these links existed as of the date of publication but are not guaranteed to be working thereafter.
2. the contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.
3. the authors of the Web pages, not CAIS, are responsible for the accuracy of their content.
4. the authors of this article, not CAIS, are responsible for the accuracy of the URL and version information.

Agha, G. A. (2002). "Adaptive Middleware". *Communications of the ACM*, 45(6), 31-32.

Agre, P., and Rotenberg, M. (Eds.). (1997). *Technology and Privacy: The New Landscape*.: MIT Press.

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D. (1994). *Molecular Biology of the Cell* ( Third ed.). New York: Garland Publishing Inc.

Alter, S. (1999). "A General, Yet Useful Theory of Information Systems". *Communications of the AIS*, 1(March), article 13.

Alter, S. (2001). "Which Life Cycle --- Work System, Information System, or Software?". *Communications of the AIS*, 7(17), 1-52.

Banavar, G., and Berstein, A. (2002). "Software Infrastructure and Design Challenges for Ubiquitous Computing Applications". *Communications of the ACM*, 45(12), 92-96.

Barnard, P., May, J., Duke, D., and Duce, D. (2000). "Systems, Interactions, and Macrotheory". *ACM Transactions on Computer-Human Interaction*, 7(2).

Benassi, P. (1999). "TRUSTe: An Online Privacy Seal Program". *Communications of the ACM*, 42(2).

Berners-Lee, T. (2000). *Weaving The Web: The Original Design And Ultimate Destiny of the World Wide Web*. New York: Harper-Collins.

Bertalanffy, L. v. (1968). *General System Theory*. New York: George Braziller Inc.

Bitton, D., DeWitt, D., and Turbyfill, C. (1983). *Benchmarking Database Systems - A Systematic Approach*. Paper presented at the Proc. VLDB Conference.

Borenstein, N. S. (1990). "Power, Ease of Use and Cooperative Work in a Practical Multimedia Message System". *International Journal of Man-Machine Studies*, 34(2), 229-259.

Boulding, K. E. (1956). "General Systems Theory - the Skeleton Of A Science". *Management Science*, 2(3), 197-208.

Brin, D. (1998). *The Transparent Society*. Reading, Mass: Perseus.

Campos, J., and Figuerido, A. D. (2002). *Programming for Serendipity*. Paper presented at the Chance Discovery: The Discovery and Management of Chance Events, North Falmouth, MA.

Casti, J. (1997). *Would-be Worlds: How Simulation is Changing the Frontiers of Science*. New York: John-Wiley and Sons.

Chabrow, E. (2001). Uncomplicating IT: Simpler Said Than Done. *InformationWeek*, 831(04/02/01), 44

Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (1999). *Non-functional Requirements in Software Engineering*.: Kluwer Academic.

Churchman, C. W. (1979). *The Systems Approach*. New York: Dell Publishing.

- Clegg, C. W., Waterson, P. E., and Axtell, C. M. (1997). "Software Development: Some Critical Views". *Behaviour & Information Technology*, 16(6), 359-362.
- Copeland, J. (1993). *Artificial Intelligence*. Oxford: Blackwell Publishers.
- Cysneiros, L. M., and Leita, J. (2002). *Non-functional Requirements: From Elicitation to Modelling Languages*. Paper presented at the ICSE, Orlando, Florida.
- David, J. S., McCarthy, W. E., and Sommer, B. S. (2003). "Agility - The Key to Survival of the Fittest". *Communications of the ACM*, 46(5), 65-69.
- Davis, F. D. (1989). "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology". *Management Information Systems Quarterly*, Sep.
- Dennis, A. R. and Valacich, J. S. (1999). *Rethinking Media Richness: Towards a Theory of Media Synchronicity*. Paper presented at the Proceedings of the 32nd Hawaii International Conference on System Sciences, Hawaii.
- Dickson, P. (1999). *The Official Rules and Explanations*. Springfield, Massachusetts: Federal Street Press.
- Dray, S. M. (1995). "The Importance of Designing Usable Systems". *Interactions*, 2(1).
- DSS. (2001, Nov-14-2001 12:57). *Information System Security*. Defence Security Service. Available: [http://www.dss.mil/isec/change\\_ch8.htm](http://www.dss.mil/isec/change_ch8.htm) [2002].
- El Sawy, O. A., and Nanus, B. (1989). "Towards the Design of Robust Information Systems". *Journal of Management Information Systems*, 5(4), 33-54.
- Esfeld, M. (1998). "Holism and Analytic Philosophy". *Mind*, 107, 365-380.
- Gargaro, A., Rada, R., Moore, J., Carson, G. S., DeBlasi, J., Emery, D., Haynes, C., Klensin, J., Montanez, I., and Spafford, E. (1993). "The Power of Standards." *Communications of the ACM*, 36(8), 11-12.
- Gediga, G., Hamborg, K., and Duntsch, I. (1999). "The IsoMetrics Usability Inventory: an Operationalization of ISO9241-10 Supporting Summative and Formative Evaluation of Software Systems". *Behaviour & Information Technology*, 18(3), 151-164.
- Geen, R. G., and Gange, J. J. (1983). Social Facilitation: Drive Theory and Beyond. In A. P. V. K. M. D. H. H. Blumberg; Hare (Ed.), *Small Groups and Social Interaction* (1) pp. 141-153.
- Gleick, J. (1987). *Chaos: Making a New Science*. London: Penguin Books.
- Goodwin, N. C. (1987). "Functionality and Usability". *Communications of the ACM*, March, 229-233.
- Herrmann, T., and Loser, K. (1999). "Vagueness in Models of Socio-technical Systems". *Behaviour & Information Technology*, 1999, 18(5), 313-323.
- Hix, D., and Schulman, R. S. (1991). "Human-Computer Interface Development Tools: A Methodology for their Evaluation". *Communications of the ACM*, 34(3), 75-87.
- James, B., Joshi, D., Aref, W. G., Ghafoor, A., and Spafford, E. H. (2001). "Security Models for Web-Based Applications". *Communications of the ACM*, 44(2), 38-44.
- Johnson, D. G. (2001). *Computer Ethics*. Upper Saddle River, New Jersey: Prentice-Hall.
- Jonsson, E. (1998). *An Integrated Framework for Security and Dependability*. Paper presented at the NSPW, Charlottesville, VA, USA.
- Keuffel, W. (2001). "Playing by the Rules". *Software Development*, 9(8), 64.
- Kienzle, D. M., and Wulf, W. A. (1997). *A Practical Approach to Security Assessment*. Paper presented at the New Security Paradigms Workshop, Langdale, Cumbria, U.K.
- Kiesler, S., Siegel, J., and McGuire, T. (1984). "Social Psychological Aspects of Computer-Mediated Communication". *American Psychologist*, Oct, 39(10), 1123-1134.
- Knoll, K., and Jarvenpaa, S. L. (1994). *Information Technology Alignment or "Fit" in highly turbulent environments: The Concept of Flexibility*. Paper presented at the SIGCPR 94, Alexandria, Virginia, USA.

- Laprie, J. C., and Costes, A. (1982). *Dependability: A Unifying Concept for Reliable Computing*. Paper presented at the Proc 12th International Symposium on Fault-Tolerant Computing, FTCS-12.
- Lea, M. (1991). "Rationalist Assumptions in Cross-Media Comparisons of Computer-Mediated Communication". *Behaviour and Information Technology*, 10(2), 153-172.
- Lessig, L. (1999). *Code and Other Laws of Cyberspace*. New York: Basic Books.
- Lipson, H., and Fisher, D. A. (2000). "Survivability: A New Technical and Business Perspective". *CERT Coordination Center, reprinted by the ACM*.
- Littlewood, B., Brockhurst, S., Fenton, N., Mellor, P., Page, S., Wright, D., Dobson, J., McDermid, J., and Gollman, D. (1993). "Towards Operational Measures of Computer Security". *Journal of Computer Security*, 2(3), 211-229.
- Maturana, H. R., & Varela, F. J. (1998). *The Tree of Knowledge*. Boston: Shambala.
- McCarty, B., & Cassady-Dorion, L. (1999). *Java Distributed Objects: The Authoritative Solution*.: Sams.
- Meadows, C. (1997). *Three Paradigms in Computer Security*. Paper presented at the New Security Paradigms Workshop, Langdale, Cumbria, United Kingdom.
- Moreira, A., Araujo, J., and Brita, I. (2002). *Crosscutting Quality Attributes for Requirements Engineering*. Paper presented at the SEKE, Ischia, Italy.
- Mowbray, S. E. (2002). Big Blue's Big Horn. *Popular Science*, March, 34.
- Neumann, P. G. (2000). "Risks in Our Information Infrastructures.". *Ubiquity: An ACM IT Magazine and Forum*, 1(13).
- Nielsen, J. (1993). *Usability Engineering*. San Diego, CA: Academic Press Prof.
- Nissenbaum, H. (1997). "Toward an Approach to Privacy in Public: Challenges of Information Technology". *Ethics and Behavior*, 7(3), 207-219.
- NIST. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing* ( NIST Planning Report 02-3): National Institute of Standards and Technology (NIST).
- Nixon, B. A. (1998). *Managing Performance Requirements for Information Systems*. Paper presented at the Proceedings of the First International Workshop on Software and Performance, Santa Fe, New Mexico, United States.
- OECD. (1996). *Guidelines for the Security of Information Systems*.: OECD.
- Pielou, E. C. (1969). *An Introduction to Mathematical Ecology*.: Wiley.
- Popper, K. (1945). *The Open Society and its Enemies*. London.
- Raymond, E. S. (1997). The Cathedral and the Bazaar. <http://tuxedo.org/~esr/writings/cathedral-bazaar/>.
- Rosa, N. S., Justo, G. R. R., and Cunha, P. R. F. (2001). *A Framework for Building Non-Functional Software Architectures*. Paper presented at the SAC, Las Vegas, NV.
- Sanai, H. (1968). *The Enclosed Garden of Truth* (M. J. Stephenson, Trans.). New York: Samuel Weiser.
- Sanders, M. S., and McCormick, E. J. (1993). *Human Factors in Engineering and Design*. NY: McGraw-Hill.
- Schor, J. B. (1991). *The Overworked American: The Unexpected Decline of Leisure*. New York: Basic Books.
- Shannon, C. E., and Weaver, W. (1949). *The Mathematical Theory of Communication*. Urbana: University of Illinois Press.
- Smith, H. A., Kulatilaka, N., and Venkatramen, N. (2002). "Developments in IS practice III: Riding the Wave: Extracting Value from Mobile Technology". *Communications of Association of Information Systems*, 8, 467-481.



Sperry, R. W., and Gazzaniga, M. S. (1967). Language Following Surgical Disconnexion of the Hemispheres. In C. H. Millikan and F. L. Darley (Eds.), *Brain Mechanisms Underlying Speech and Language*. USA: Grune and Stratton.

Standage, T. (1998). *The Victorian Internet*. New York: Berkely Books.

Tenner, E. (1997). *Why Things Bite Back*. New York: Vintage Books, Random House.

Toffler, A. (1989). *The Third Wave*, Bantam, NY (re-issue of 1980).

Ware, W. H. (1984). "Information Systems Security and Privacy". *Communications of the ACM*, 27(4), 315-321.

Weiner, R. (1993). *Digital Woes: Why We Should Not Depend on Software*. Reading, Mass: Addison-Wesley.

Wheatley, K. L., and Flexner, W. A. (1991). *The Pitfalls of Portability ... or ... Why More is Not Better*. Paper presented at the Proceedings of the 24th Hawaii International Conference on System Sciences.

Whitworth, B. (1975). *Brain Systems and the Concept of Self*. Unpublished MA dissertation, Auckland, NZ:University of Auckland.

Whitworth, B., and deMoor, A. (2003). "Legitimate by Design: Towards Trusted Virtual Community Environments". *Behaviour & Information Technology*, 22(1), 31-51.

Whitworth, B., and Felton, R. (1999). "Measuring Disagreement in Groups Facing Limited Choice Problems". *THE DATABASE for Advances in Information Systems*, 30(3 & 4), 22-33.

Whitworth, B., Gallupe, B., and McQueen, R. (2001). "Generating Agreement in Computer-Mediated Groups". *Small Group Research*, 32(5), 621-661.

Whitworth, B., Gallupe, B., and McQueen, R. J. (2000). "A Cognitive Three Process Model of Computer-Mediated Groups: Theoretical Foundations for Groupware Design". *Group Decision and Negotiation*, 9(5), 431-456.

Wright, M. A., and Kakalik, J. S. (1997). "The Erosion of Privacy". *ACM Computers and Society*, Dec, 22-25.

Wright, R. (2001). *Nonzero: The Logic of Human Destiny*. New York: Vintage Books.

Zigurs, I., Buckland, B., Connolly, J. R., and Wilson, V. (1999). "A Test of Task-Technology Fit Theory for Group Support Systems". *The Data Base for Advances in Information Systems*, 30(3,4), 34-50.

## ABOUT THE AUTHORS

**Brian Whitworth** is Assistant Professor in the Department of Information Systems, New Jersey Institute of Technology. His Masters thesis in neuro-psychology was on the implications of split-brain research for the concept of self. He was Senior Army Psychologist in the New Zealand Army, then moved into computing, developing operational software and simulations. For the last decade he studied how human and social factors, like agreement, relationships, norms, roles, legitimacy and meaning, affect the design of computer-mediated interaction systems. His publications appear in *Small Group Research*, *Group Decision and Negotiation*, *DATABASE and Behavior and Information Technology*. <http://web.njit.edu/~whitwort/bwrefs.html>

**Michael Zaic** is a student in the Department of Information Systems, New Jersey Institute of Technology. He is interested in the design of complex information systems.

Copyright © 2003 by the Association for Information Systems. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than the Association for Information Systems must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or fee. Request permission to publish from: AIS Administrative Office, P.O. Box 2712 Atlanta, GA, 30301-2712 Attn: Reprints or via e-mail from [ais@gsu.edu](mailto:ais@gsu.edu)



# Communications of the Association for Information Systems

ISSN: 1529-3181

## EDITOR-IN-CHIEF

Paul Gray  
Claremont Graduate University

### CAIS SENIOR EDITORIAL BOARD

Detmar Straub Vice President Publications Georgia State University	Paul Gray Editor, CAIS Claremont Graduate University	Sirkka Jarvenpaa Editor, JAIS University of Texas at Austin
Edward A. Stohr Editor-at-Large Stevens Inst. of Technology	Blake Ives Editor, Electronic Publications University of Houston	Reagan Ramsower Editor, ISWorld Net Baylor University

### CAIS ADVISORY BOARD

Gordon Davis University of Minnesota	Ken Kraemer Univ. of California at Irvine	Richard Mason Southern Methodist University
Jay Nunamaker University of Arizona	Henk Sol Delft University	Ralph Sprague University of Hawaii

### CAIS SENIOR EDITORS

Steve Alter U. of San Francisco	Chris Holland Manchester Business School, UK	Jaak Jurison Fordham University	Jerry Luftman Stevens Institute of Technology
------------------------------------	--	------------------------------------	---

### CAIS EDITORIAL BOARD

Tung Bui University of Hawaii	H. Michael Chung California State Univ.	Candace Deans University of Richmond	Donna Dufner U. of Nebraska -Omaha
Omar El Sawy University of Southern California	Ali Farhoomand The University of Hong Kong, China	Jane Fedorowicz Bentley College	Brent Gallupe Queens University, Canada
Robert L. Glass Computing Trends	Sy Goodman Georgia Institute of Technology	Joze Gricar University of Maribor Slovenia	Ake Gronlund University of Umea, Sweden
Ruth Guthrie California State Univ.	Juhani Iivari Univ. of Oulu, Finland	Munir Mandviwalla Temple University	M. Lynne Markus Bentley College
Don McCubbrey University of Denver	John Mooney Pepperdine University	Michael Myers University of Auckland, New Zealand	Seev Neumann Tel Aviv University, Israel
Hung Kook Park Sangmyung University, Korea	Dan Power University of Northern Iowa	Ram Ramesh SUNY-Buffalo	Nicolau Reinhardt University of Sao Paulo, Brazil
Maung Sein Agder University College, Norway	Carol Saunders University of Central Florida	Peter Seddon University of Melbourne Australia	Upkar Varshney Georgia State University
Doug Vogel City University of Hong Kong, China	Hugh Watson University of Georgia	Rolf Wigand University of Arkansas at Little Rock	Peter Wolcott University of Nebraska- Omaha

### ADMINISTRATIVE PERSONNEL

Eph McLean AIS, Executive Director Georgia State University	Samantha Spears Subscriptions Manager Georgia State University	Reagan Ramsower Publisher, CAIS Baylor University
---	--	---