A Brief History of Software Technology

Christof Ebert

Good car drivers assess situations—past, present, and future—with a mix of skills and qualities. They make unconscious decisions and meld impressions, experiences, and skills into appropriate real-time actions. The same holds for assessing software technology. When reflecting on which technologies have had the most impact in the past 25 years, we can assess it quantitatively, by looking at research papers or "hype-cycle" duration, for example. Alternatively, we might judge it like the expert driver, intuitively evaluating what was achieved compared to what was promised from a user perspective.

For this 25th-year issue of *IEEE Software*, I wish to reflect on when some key software technologies reached their respective markets during this period. Of course, many major technology breakthroughs happened before 1984: Milestones such as the IBM OS/360 and the microprocessor, and even many still-relevant software engineering practices, had been developed much earlier.[1,2] So what makes the recent 25 years unique? First, software moved from a few company desks to the lives of practically everyone on the planet. The PC, the Internet, and mobile phones showcase this tremendous evolution. Second, empirical evaluations overcame opinions. Mary Shaw described the eighties by stating, "Software engineering is not yet a true discipline, but it has the potential to become one."[3] In those early days, a lot of technologies were just assembled and delivered, but from the '80s onward, engineers evaluated and empirically assessed new technologies to judge their impact.

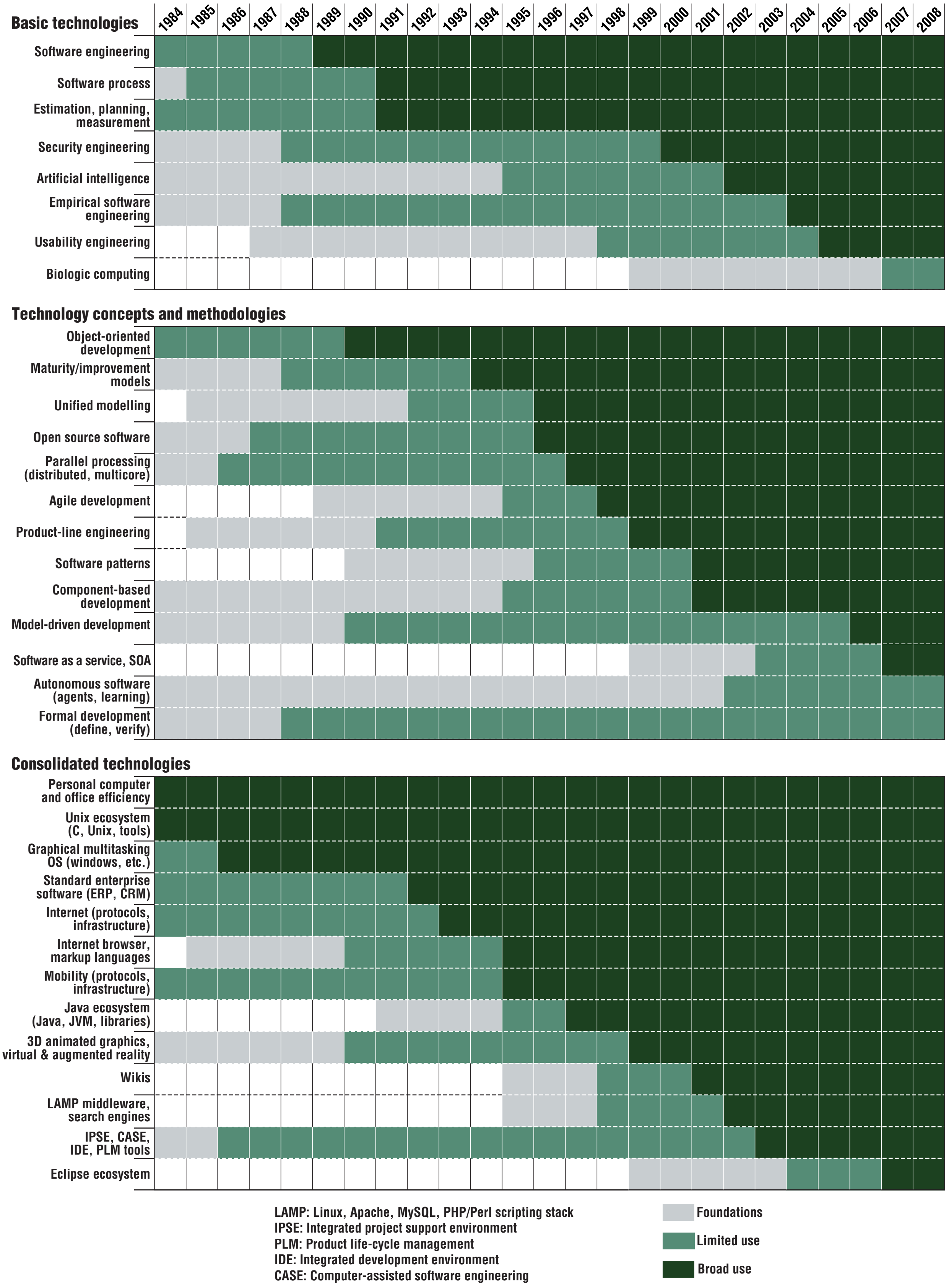
Through the Rear Mirror

> *If the automobile followed the same development as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year killing everyone inside. —Robert Cringely*

Just as we need the rear-view mirror to see what's around us, what just happened, and what might pass us by, we must evaluate past technologies in order to better and more quickly propel new topics forward.

Many interesting technologies have clearly "made it." Figure 1 shows relevant software technologies and when they reached major maturation points. It builds on a layout that Sam Redwine and William Riddle introduced.[1] For simplicity, I distinguish only three phases on the learning curve—namely, *foundations* (when basic research and concepts were created), *limited use* (when concepts reached a few companies and users), and *broad use* (when the technology reached roughly a third of its then-addressable market).

But where to start? Journals and Internet resources have changed dramatically in the past 25 years. Until the early 1980s, *Datamation* was practitioners' primary source on software technology. Today, several such sources are available for the practitioner. *IEEE Software* clearly has this focus. Online resources such as Slashdot also provide insight in the latest technology evolution. So, I compiled technologies from a lot of single data points that I found in the many technology reviews and wikis of the software world. For balance, I also asked my colleagues on *IEEE Software*'s boards for their insights on technologies during the past



Basic technologies	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008
Software engineering	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Software process	░	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Estimation, planning, measurement	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Security engineering	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█
Artificial intelligence	░	░	░	░	░	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█
Empirical software engineering	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█
Usability engineering				░	░	░	░	░	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	█	█	█	█
Biologic computing																░	░	░	░	░	░	░	░	▒	▒
Technology concepts and methodologies																									
Object-oriented development	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Maturity/improvement models	░	░	░	░	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Unified modelling		░	░	░	░	░	░	░	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█
Open source software	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█
Parallel processing (distributed, multicore)	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█
Agile development						░	░	░	░	░	░	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█
Product-line engineering		░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█
Software patterns							░	░	░	░	░	░	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█
Component-based development	░	░	░	░	░	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█
Model-driven development	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█
Software as a service, SOA																░	░	░	░	▒	▒	▒	▒	█	█
Autonomous software (agents, learning)	░	░	░	░	░	░	░	░	░	░	░	░	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒
Formal development (define, verify)	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
Consolidated technologies																									
Personal computer and office efficiency	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Unix ecosystem (C, Unix, tools)	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Graphical multitasking OS (windows, etc.)	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Standard enterprise software (ERP, CRM)	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Internet (protocols, infrastructure)	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Internet browser, markup languages		░	░	░	░	░	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Mobility (protocols, infrastructure)	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█	█	█
Java ecosystem (Java, JVM, libraries)								░	░	░	░	▒	▒	█	█	█	█	█	█	█	█	█	█	█	█
3D animated graphics, virtual & augmented reality	░	░	░	░	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█	█	█	█	█
Wikis												░	░	░	▒	▒	▒	█	█	█	█	█	█	█	█
LAMP middleware, search engines												░	░	░	▒	▒	▒	▒	█	█	█	█	█	█	█
IPSE, CASE, IDE, PLM tools	░	░	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	█	█	█	█	█	█
Eclipse ecosystem																░	░	░	░	░	▒	▒	▒	█	█

LAMP: Linux, Apache, MySQL, PHP/Perl scripting stack
IPSE: Integrated project support environment
PLM: Product life-cycle management
IDE: Integrated development environment
CASE: Computer-assisted software engineering

░ Foundations
▒ Limited use
█ Broad use

Figure 1. Software technology maturation during the past 25 years. The colors show how a specific technology matured and gradually reached its market.

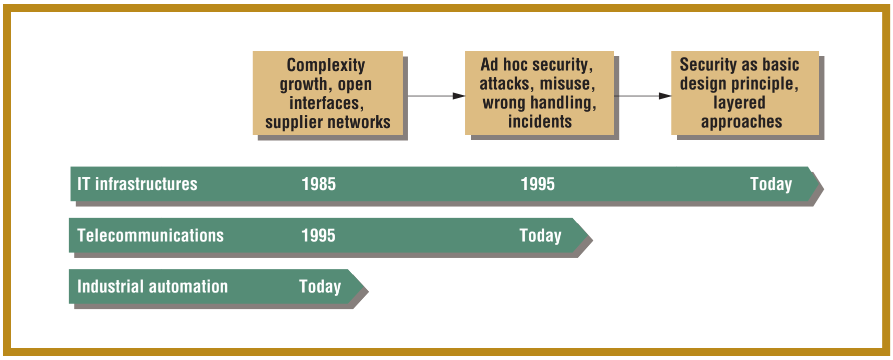

Complexity growth, open interfaces, supplier networks
Ad hoc security, attacks, misuse, wrong handling, incidents
Security as basic design principle, layered approaches
IT infrastructures
1985
1995
Today
Telecommunications
1995
Today
Industrial automation
Today

Figure 2. IT security and its industry-specific maturation points. Technologies face different challenges and are adopted at different rates depending on the application domain.

25 years. They've been involved in technology long enough to assess it from a variety of angles. Clearly, providing precise time stamps is impossible. Just think about object-oriented development, which has been used extensively since the 1990s but is still unappreciated in some industries.

Figure 1 shows three clusters of software technologies. *Basic technologies* contribute to broad trends and disciplines as they evolve, and they apply to all industries and across software development. Most of those we know today have been around the past 25 years. *Technology concepts and methodologies* combine underlying techniques that are used in many different industries and products. *Consolidated technologies* build on concepts and provide ready-to-use technical solutions. In cases where a certain technology should appear in two such clusters, I ranked it in the more general cluster.

Major Trends

What exactly does it mean that a software technology has "impact"? Asking different people the same question will yield a variety of perspectives. A professor will look at reputation and research grants and how a technology will help achieve such targets. A researcher will answer on the basis of innovation potential. An industry manager will look at profitability, image, and innovative products. A software engineer will look at usability and effectiveness to solve a problem at hand. The typical consumer would probably judge the technology on the basis of how ubiquitously it weaves into the fabric of everyday life and supports getting a job done, and kids would look at keeping up with their peers. Not only do the two consumer groups—everyday users of software technology—far outnumber the other stakeholders I listed, but they also judge software technology and products very differently. They consider how invisible, easy to use, and embedded the software is—in other words, how calmly yet effectively it supports them in getting real things done.

Looking at Figure 1, we can see several trends that characterize software technology evolution during the past 25 years:

- Ecosystems of researchers, suppliers, customers, and users rather than individual companies drive software technologies.
- Technologies need several trials with different focuses before they succeed.
- A particular technology is adopted in different industries with varying delays.
- Domain-specific focus lets users adjust technologies to their specific needs.
- Working with processes has replaced ad hoc trial-and-error design and delivery.
- Technologies that used to be fragmented and isolated are now integrated.

Each of these trends left strong footprints in engineering products and in shaping the software industry.

Impacts on Products and Industries

Microsoft with Windows or Sun with Java are major technology drivers as individual companies, but their technologies succeed because they're created and propagated through industries. We can't even imagine Windows without Intel and an entire ecosystem of suppliers and service providers. Similarly, banking created ATMs and developed many software technologies, such as distributed and secure transaction processing, around them. Retailers stimulated the development of point-of-sale terminals and the necessary supply chain software, including bar codes and RFID.

Some technologies have overly long maturation periods—or never fully develop. Their transition to broad usage follows S-shaped innovation patterns that flow from initial research and trials to wide industry usage and then repeat over and over again.[4,5] This explains why successful companies can fail practically overnight just because they didn't introduce a certain technology in a timely way. Software managers are too often biased toward conservation rather than growth. They focus on efficiency and undervalue experimentation and innovation. As a senior engineering vice president said, "After being rewarded for many years for doing things right, it might take them a while to accept that you've got a new way and they should bank their career on it."[6]

Software technologies are useful if they're broadly used. However, any particular technology reaches some industries much faster than others. A good example is the long and winding road toward useful code-generation and engineering tool suites. These tool sets started out with technology not being ready; later, the market wasn't ready. AI and expert systems faced the same fate. Today they're almost ubiquitous because industry realized that an expert system is not a stand-alone technology but rather must be embedded into products. Figure 2 shows this effect in some detail for information security.

Security was first recognized as a key technology in IT infrastructures during the late '80s when the Jerusalem virus and Morris worm effectively brought early Internet traffic to a halt. Incidents continued throughout the '90s as technology was applied only ad hoc and without thorough architecture considerations. Today, after 20 years, basic security design principles are finally being used and deployed with new IT products. The same story is repeating in telecommunications, as voice-over-IP attacks show, where we're again seeing ad hoc patches but no real control. Industrial automation and other domains are even more delayed in security engineering, as incidents such as the Slammer worm showed.

Domain-specific focus replaced the one-size-fits-all approach during the '90s. Early CASE and distributed component models were trapped in trying to solve too many problems. When industry realized that different domains have their own specific needs and speeds, it was much easier to optimize a technology and introduce it to a specific market. Modeling tools immediately became successful when they were adapted to specific domain needs, such as embedded controllers or telecommunication protocols.

Software processes, both for engineering and for management, boosted technology evolution from the '80s onward. Software system complexity grows faster than we can control it. We had already seen this roadblock in the '60s, but it began to shrink when major industries moved their attention to the process of engineering software. As a consequence, software development has changed dramatically over the past 25 years, from an often-individual creative activity to a mostly collaborative engineering discipline.

Integration of processes, tools, and people speeds technology introduction, as I've learned from many of the companies I work with. Today, it's hard to believe that 25 years ago most software and its developers and users acted in isolation. Software integration is best visible with the Internet's advent and huge growth, owing to the interaction and integration it provides. Component frameworks and open standards further stimulate this trend. Successful adoption and integration is not trivial. In order to deliver value to engineers, new technologies, processes, and engineering tools need profound change management.

Assess and Anticipate

> *640K ought to be enough for anybody.* —*Bill Gates, 1981*

The ability of companies to rapidly assess new technologies and effectively integrate and blend them into innovative processes and products will determine the winners of tomorrow. There are more than enough good ideas, hypes, and unproven technologies around.[7] But they need profound and sound assessment. So, here are 10 guidelines that will help you assess and anticipate new software technologies:

- Don't get trapped in hype. Most software technologies never make it. Life is too short and budgets too restricted to jump on everything you hear about at a conference or read about in articles.
- Don't fall in love with your technology. Continuously question how you can do things better. Think outside the box for appropriate solutions. Allow your customers to replace your products with your newer technologies. If you don't, your competition will do it for you.
- Think first, then leap to a new technology. Understand concrete needs and specify priorities that should be addressed. Identify the relevant stakeholders in decision making and get to a shared vision. Keep these stakeholders onboard to avoid sudden attacks or refusals.
- Consider value and set concrete, measurable objectives and milestones. Tolerate small losses in evaluating technologies in order to win big occasionally. Typical criteria are efficiency, cash flow, and time to profit. Not all innovations should have a precise return on investment up front, as this will kill creativity. However, at a given point, they must deliver value—or disappear.
- Avoid big-bang technology introductions. Don't risk big, but risk often. Introduce technologies in increments, and consider how they would reach the market through your products and services.
- Separate functionality (that is, customer value) from software technology. If you split the function from its implementation, you can think about how to deliver the function in radically different ways.
- Train engineers and managers on new technologies, free from immediate product usage. Software technology knowledge has a half-life of less than two years, so you'll inevitably have to look beyond what you know.
- Never ever assume that your team or your colleagues have the technologies and skills you need. These are all from the past. Hire fresh minds and rotate people so that they're pushed to throw away complacency.
- Consider change management. New technologies impact products, processes, and people. Prepare a road map for how the technology will be introduced. Have an exit strategy in case those promises aren't fulfilled.
- Periodically align your product portfolios with your technology road map. Set milestones for new technologies when they ought to deliver, and synchronize with market needs and product development. Dare to kill products and technologies if they don't deliver according to expectations.

Naturally, not all these hints apply to all settings. For instance, a company would not expect its engineers to question all their legacy technologies when products are in maintenance mode. However, the opposite is equally true. Just take a look at companies such as SAP and Microsoft, which have survived this long only because they continuously and heavily challenge what they're doing.

Modern society with globalized trade, communication, and collaboration would have been impossible without steady innovation of appropriate technologies and engineers that drive this evolution. Like the expert car driver, we benefit from looking in the rear-view mirror, being aware of the road in front of us, and being alert to what's around us.

Acknowledgments

Special thanks to David Blaine, John Favaro, Robert Glass, Simon Helsen, Dieter Lederer, Diomidis Spinellis, and Rebecca Wirfs-Brock for sharing their thoughts.

References

1. S. Redwine and W. Riddle, "Software Technology Maturation," *Proc. 8th Int'l Conf. Software Eng.* (ICSE 85), IEEE CS Press, 1985, pp. 189–200.
2. B. Boehm, "A View of 20th and 21st Century Software Engineering," *Proc. 28th Int'l Conf. Software Eng.* (ICSE 06), ACM Press, 2006, pp. 12–29.
3. M. Shaw, "Prospects for an Engineering Discipline of Software," *IEEE Software*, vol. 7, no. 6, 1990, pp. 15–24.
4. G. Hamel, *Leading the Revolution*, Harvard Business School Press, 2000.
5. C. Ebert and R. Dumke, *Software Measurement*, Springer, 2007.
6. H. Krasner, "Bottlenecks in the Transfer of Engineering Technology," *Proc. 28th Ann. Hawaii Int'l Conf. Systems Sciences*, IEEE CS Press, 1995, pp. 635–641.
7. *Gartner Hype Cycles*, Gartner Group, 2008; www.gartner.com/it/products/hc/hc.jsp.

Christof Ebert is managing director at Vector Consulting Services. Contact him at christof.ebert@vector-consulting.de.