Packaged software development teams: What makes them different?

Erran Carmel
American University
Kogod College of Business Administration
4400 Massachusetts Ave., NW
Washington DC 20016-8044 USA
(o) 202-885-1928
(f) 202-885-1992
(e) carmel@american.edu

Steve Sawyer
Syracuse University
School of Information Studies
4-291 Center for Science and Technology
Syracuse NY 13244-4100 USA
(o) 315-443-4473
(f) 315-443-5806
(e) ssawyer@cat.syr.edu

Carmel, E. and Sawyer, S., (1998) APackaged Software Teams: What Makes Them So Special?,@ *Information Technology & People, 11*(1), 6-17.

Packaged software development teams: What makes them different?

ABSTRACT

An increasingly large number of software development teams develop packaged software rather than custom information systems (IS). We discuss the characteristics of packaged software versus IS development environments that capture the differences between the teams that develop software in these respective industries. Our analysis spans four levels: the industry, the dynamics of software development, the cultural milieu, and the teams themselves. Relative to IS the packaged software industry is characterized by: intense time pressures, less attention to costs, and different measures of success. Relative to IS the packaged software development environment is characterized by: being a Aline@ rather than Astaff@ unit, having a greater distance from the actual users/customers, a less mature development process, an integrated approach to design and development, and coordinated approach to design control rather consensus-building. Relative to IS the packaged software cultural milieu is characterized as individualistic and entrepreneurial. Relative to IS the packaged software team is characterized as: less likely to be matrix managed, having members involved in more phases of the development cycle, more cohesive, "jelled," having more opportunities for financial rewards, smaller, more co-located, with a greater shared vision.

Packaged software development teams: What makes them different?

Packaged software is now a powerful global industry: sales of its 500 largest vendors amounted to $72 billion in 1996 (Frye, 1997). In the US the industry accounts for 2.1 million jobs and is the US=s fifth largest manufacturing industry (Business Software Alliance, 1997). And, while the US economy grew 30% in the decade from 1984 to 1994, the packaged software market grew 269% (Leebaert, 1995, p.5). The largest firms in this global industry-- Microsoft, Oracle, Corel and others -- are household names.

A relatively new stream of literature is emerging that examines the unique work cultures and production (software development) characteristics of the firms in this industry (Cringely, 1992; Zachary, 1994; Cusumano and Selby, 1995; Yourdon, 1996; Carmel, 1997; Carmel and Bird 1997). Given the recency of this industry=s ascendance, and the diverse nature of the firms in and around this industry, such research is still in its formative stages. Our objective in this article is to begin to fill this void by examining software development *teams* within the packaged software industry.

Our interest is the development of *packaged* software (i.e., software sold as a product). This is contrasted with the more established domain of software development for custom applications, particularly information systems applications in business and government. Unlike packaged software development, information systems development is typically done by internal-to-the-organization information technology departments or consulting/ service firms (e.g., EDS, IBM=s ISSC, or Anderson Consulting) who build custom systems on a contract basis. Collectively, we refer to these as "custom IS development" or AIS@ for short. Packaged software companies are also differentiated from consulting or contracting companies in that packaged software firms build products for sale to a market, not to specific customers.[1]

Clearly teams developing either IS or packaged software share many common characteristics since their central preoccupation is the development of a software artifact. However, our objective in this article is to highlight the ways we believe these two types of team *differ*.

Software Development Teams

We deal in this article with the concept of a *software development team*. By team, we mean a group of two or more people who are a distinct managerial and social unit embedded in a larger organizational structure, engaged toward achieving a common objective (Goodman et al, 1986; McGrath, 1990). This means that much of what characterizes software development are issues of social dynamics in a technical setting (DeMarco and Lister, 1987). The individual members of software development teams possess skill sets ranging from programming to quality assurance, from technical sophistication to knowledge of the application domain for which the product is intended, from management experience to interpersonal communication.

The broader topic of IS teams has been studied intensely for many years. For example, Weinberg (1971) pointed to difficulties of team-based software development, writing: A...these are teams

which have grown up in an environment pervaded by the myth that programming is the last bastion of individuality" (p. 35). Brooks (1974), writing about the development of IBM=s Systems 360 operating system, articulated how team-behaviors actually drive software development. He codified the Amythical man-month,@ explaining how adding people to a project team running late would make it run later still. Curtis, Krasner and Iscoe (1988) highlighted several themes that pervade team-based software development efforts. These include the thin spread of application domain knowledge, conflicting and changing requirements, and coordination and communication breakdowns among team members.

DeMarco and Lister (1987) describe the skills and interactions of the software development team members as Apeopleware@ and argue that this aspect of software development is the most critical, and least understood. Over the years, a variety of technologies (e.g., CASE) and methodologies have been introduced in order to address these and other concerns. However, they have not been successful at making it significantly easier for software development team members to work together (i.e., Guinan, Cooprider and Sawyer, 1997). As Brooks (1987) states @there is no silver bullet@ to make team-based software development easy.

Place near here Figure 1: Analytic Framework

Our Analytic Framework

Our analytic framework (Figure 1) is derived in part from the works of Curtis, Krasner and Iscoe (1988) and Carmel (1997) who recognized that the work units we call teams are social units. This suggests that they should be analyzed as such -- as groups of people embedded in larger social units. This leads us to four levels of analysis and the comparison of IS to packaged software development teams is organized by first looking at broad social units (e.g., the industry as a whole) and then moving to the smaller unitsCthe teams themselves. Table 1 summarizes the principle components of this article.

Place near here Table 1: Summary of Differences

The Packaged Software Industry

We believe that packaged software firms function in an environment of intense time-to-market pressure relative to IS development efforts. This is due, in part, to packaged software being such a high-profile industry today B under continual scrutiny by investors, stock-holders, Wall Street analysts, and hundreds of technology media sources. Packaged software firms are under constant pressure to innovate and beat out the competition in delivering their products to market. When a firm releases some critical functionality "A" in the latest release, all other firms developing for this marketplace are compelled to modify their development to include "A" in the next release, even if it is relatively late in their development cycle. In one of our interviews with packaged software developers, the team lead told us that she woke up late at night A... worried someone would bring a similar product to the market before I could.@ If, until the 1990s, packaged software firms had some safety in the typical 12-24 month release cycles (Carmel, 1995)-- in the age of *Internet time*, this rhythm has been broken and time pressures are more intense. Finally, packaged software products, much more so than IS development, are motivated by technology push-- rather than demand-pull-- intensifying the time pressure.

Another observable difference between the IS and packaged environment is the level of available resources. Some packaged software firms operate in very resource-poor environments, giving rise to the popular Amade-in-my-garage@ ethos that permeates the packaged software industry. These low-budget efforts are financed with sweat equity in the hopes that the developers product will strike it big (i.e., Maglitta, 1997).

But, many software developers operate in a resource-rich organization **B** where development cost is a secondary factor to delivery. Successful or promising packaged software firms may be cash-rich from an IPO or well supported by venture capital. Individual programmers are made very wealthy from these financial riches. For example, the Silicon Valley computing industry produces, on average, 64 new millionaires each day (Harpers, 1997). Microsoft=s success alone creates 17 new employee millionaires each week (Zachary, 1994). Further, successful packaged software firms operate under conditions of increasing returns to scale. The practical implication of this is that they have ample financial resources.

We can find no parallel to this in the custom IS development milieu since cost is almost always paramount. In IS the trade-offs between schedule (time) and resource/ budget are seen as key aspects of project management (Boehm, 1987) and cost is considered a major risk and a critical measure of project success (Boehm, 1991). Certainly there are projects where cost becomes secondary, such as altering a core system=s functionality to comply with mandated tax code changes. However, this is a cost of doing business, not an opportunity to exploit. Further, this cost is typically absorbed elsewhere in the corporate budget (by funding fewer IS initiatives, for example).

Finally, it appears that the high level measures of success are different in the packaged software industry and in IS development. For example, packaged software products are typically measured by profits and revenues, market share, and good product reviews (Keil and Carmel, 1995). However, IS development efforts are typically measured by user satisfaction, user acceptance, quality, and cost.

Software Development

At the second level of analysis **B** that of the software development environment **B** the first observable difference is that the developers in packaged software firms are in Aline@ positions while the typical IS developers are in "staff" positions. In packaged software shops developers are the primary producers of revenue for the company. Thus, developer needs are central to production. This helps to explain why packaged software companies create Acampuses@ and provide perks such as skateboarding ramps, free sodas, and late-night transport from office to home **B** they are catering to their key producers. Developers in IS shops, however, are service-oriented staff. They produce software to support line operations and are, for example, often targeted for cost containment. This structural difference may have helped spur the enormous growth in systems consulting and outsourcing.

A second difference between packaged and IS development that we observe is the relationship between developers and their product=s users/customers (Grudin, 1991). Most packaged software developers are physically and organizationally separated from product users and rely on a number

of intermediaries such as customer support lines and sales staff to link to these users (Keil and Carmel, 1995). Custom IS development, however, relies on methods and techniques such as Joint Applications Development which are based on a joint user/developer commitment to sharing in the construction of a system. Contemporary and traditional systems analysis and requirements engineering approaches are premised on users delineating system requirements (e.g., Kozar, 1989; Guimaraes, 1985). Many of the assumptions implicit in these approaches are inapplicable for packaged software development.

Another example of the difference in user/developer relations between packaged software developers and IS developers is manifested in how the products from these two domains are implemented. Custom IS developers are concerned with implementation-- the introduction, roll-out, and host-organization acceptance of the system (e.g., Markus, 1983; Kling and Iacono, 1984). Implementation stands separate from the work of the packaged software developers. For instance, while Peoplesoft and other enterprise-wide package developers sell their complex products, they largely leave implementation support to third party consultants.

We contend a third difference is that the custom IS software development *processes* tend to be both more central and more mature. That is, they are closer to the engineering paradigm or to what is called a structured process (Carmel & Becker, 1995). This relative maturity is exemplified by the work of the Software Engineering Institute (SEI) and their software capability maturity model (CMM) (Humphrey, 1988; Paulk, 1995). The underlying premise of this perspective is that attention to process leads to better products. Thus, there is explicit guidance regarding the roles development personnel play -- including their roles in implementing appropriate development methods and techniques.

In contrast, in packaged software development, the product itself is the focal point and the process-- with its engineering orientation-- is both secondary and less mature. For example, most packaged software organizations, until very recently, relied on loose software development guidelines. In fact, the very industrial-engineering-oriented notions of software development, now widely embraced in the IS community, are rejected in most packaged software organizations as notions that are bureaucratic, boring, and stifling of critical innovation. Bach (1995) argues that the only basis for any kind of software development success are the heroic efforts of a dedicated team of individuals-- where a hero is defined as one who takes initiative to solve ambiguous problems. He argues that the SEI=s CMM discourages acts of heroism. As we discuss below, this concept of programmer-as-hero also permeates the packaged software development industry cultural milieu.

An example of these differences is reflected in data from one of the author=s research studies at a small ($10 million annual sales) packaged software firm that develops network products. The firm secured a special contract with AT&T that called for some custom modification of their marketed product. AT&T required that the firm follow ISO 9000 standards in development. The package developers felt a strong cultural gulf when they wanted to prototype various design ideas before implementation and had to battle their large customer who wanted the entire product to be built from specifications.

At Microsoft, a compromise between these two world views was recently documented by Cusumano and Selby (1995; 1997) and dubbed Asynch and stabilize.@ In this approach there are

frequent points where team members meet to both stabilize and synchronize their individual work. The moniker reflects the loose, and mostly informal, development practices of the firm's software development teams. The subtext to this is remarkable in that just a few years previous, Cusumano (1991) wrote favorably about the extremely rigorous development processes at Japanese software factories.

We believe a fourth difference between development efforts in these two environments concerns the way products are conceived. Most IS development approaches attempt to separate design from production. However, in packaged software the acts of creation/design and production are often combined. The creation/design of a packaged software product is typically driven by the vision of a small group -- perhaps even one person -- indispensable to its production. From these key individuals arise many innovations that define a product. For example, David Cutler, the team leader at Microsoft who championed the development of NT, also coded important modules himself (Zachary, 1994). Many of the small, niche companies rely on this type of creator. This implies that even an excellent development process cannot replace key people! The development process is subordinate to the creation and innovation needed to produce the product (Heckman and Sawyer, 1996).

A fifth difference between packaged software development efforts and IS approaches is the way design control is exerted. As we argued above, in packaged software development much of the product vision is driven by one person or a small, core, team. This visionary, or small group of visionaries, controls the design of the product. During the ensuing software development efforts, team-members work on individual modules and meet frequently. The goal of these frequent meetings is to internally coordinate their work and to fit into the overall vision of the key designer(s).

This differs from IS development which relies on consensus-building communications between team members and important external players. This is because the ultimate design of this product does not lay in the head of one person (or even a small number of people). Instead, the typical IS product=s final design revolves around building a shared, consensus-oriented, view of needs and functions (Walz, Elam and Curtis, 1993). So, control of design in most IS products is actually done by constant attention to building a shared consensus among all designers and users (Crowston and Cammerer, forthcoming). This consensus-based approach also relies on formal @sign-offs@ and documentation to assist those involved in the development effort to maintain a shared view. This reliance on consensus underscores both the difficulties, and pervasiveness, of the communication and coordination breakdowns that plague IS development (Curtis, et. al., 1988). This consensus focus also places extensive pressure on project management to keep the communication levels open (Adams and Kirchof, 1982).

The Packaged Software Cultural Milieu

Culture refers to the collection of patterns of behavior and artifacts that represent the ideas, values and norms of a social unit (Schein,1992, p.9). A work culture reflects the behaviors, artifacts and shared norms of a social unit that shares similar work (such as farmers, or packaged software programmers). Work cultures also inherit behaviors and shared norms from the larger social units in which they are embedded. For example, an IS development team at a large insurance company may be closely aligned with the business culture of the insurance industry.

We contend that the work culture of packaged software firms is both entrepreneurial and highly individualistic. These two cultural traits stem, in part, from the industry=s youth, but are actively perpetuated even in larger and more established firms (Zachary, 1994; this issue).

With low entry barriers, and the potential for huge rewards, packaged software development work reflects many attributes of the entrepreneurial legend: long hours, grit and determination, and high risk. We believe this is one reason why packaged software firms attract more aggressive, risk-seeking individuals, while IS organizations are more likely to attract individuals who seek greater stability and structure.

We note that the work culture of packaged software development organizations is highly individualistic (Carmel, 1997; Weinberg, 1971). Constantine (1995) discusses one of the software myths A...that glorifies the brilliant genius who single-handedly conceives and codes clever new systems in sweaty and sleepless weekends of nonstop programming@ (p. 48). Rugged individuals--software cowboys -- are respected, even revered, in most packaged software organizations. These individuals represent the hacker sub-culture of programming (Levy, 1987). Hacking, the pursuit and display of coding expertise, epitomizes the relationship between a programmer and his code. The individualistic, rule-abhorring developer is not as attractive to most IS development shops. They tend to seek process-oriented people who are more inclined to function well in traditional, mechanistic, organizational settings.

Packaged Software Teams

Given the differences between IS and packaged software development industries, development norms, and work cultures (as described above), it should not be surprising that the IS and packaged software *teams* are themselves different. For example, Ed Yourdon (1996) states:

> AIf you=re a veteran with 20 years of experience under your belt, imagine what would you do if you could start all over again today. Would you go work for the MIS department of XYZ Mega-Bank, where you=re one of 2000 anonymous COBOL programmers -- or would you go to work for a 10-person software startup building interactive multimedia virtual-reality applications to be deployed on the Internet? Would you even think twice about such a choice? (p. 32)

Typically, IS teams are structured as "project teams." (i.e., Larson & Gobeli, 1988). One dimension is of particular interest to our comparative analysis: project teams often do not work together full-time. The IS project team members are typically involved in multiple projects on a temporary basis as determined by the proper mix of staff skills. These IS teams are more likely to work on a project basis in which staff, reporting in a matrix relationship, rotate into and out of the project at various stages of the life cycle. DeMarco & Lister (1987) argue that, in software development, the key difference between a Aproject@ and a Ateam@ is that people can be on multiple projects simultaneously, but can be on only one team at a time. In short, IS project teams are actually more like transient groups of people.

In contrast, packaged software development teams are far less likely to be managed in a project/matrix structure. With their entrepreneurial roots, one is more likely to find packaged software teams that are self-directed. We define a self-directed work team to be a group of interdependent

employees who share many of the roles traditionally thought of as defining a supervisor. Closely related, in many such firms, one finds instances of what Carmel (1995) calls a "core team." The core development team drives key business, design, and technical decisions.

The implications of these differences on the team=s cohesion, motivation levels, sense of purpose, and "jelling" are profound. DeMarco and Lister introduce the term jelling to describe a team that functions well (DeMarco & Lister 1987, DeMarco 1995). The software development community accepts as axiomatic that a team that is more cohesive, motivated, and can "jell" is more likely to produce a better product than a team that is weak in these characteristics. Cohesion is derived from common goals and objectives-- an *esprit de corps*, working for the good of the team rather than of the individual. DeMarco, who generally studies IS teams, notes that most software teams do not jell well.

In part this cohesion leads to dedication and hard work. For example, Carmel (1995) found that packaged software development teams regularly work more than 50 hours per week -- even more during crunch periods. Another factor leading to hard work and dedication is the possibility of very large financial rewards that are available to many developers of package software. This financial incentive seems rare for those developing custom IS. Moreover, when teams have "jelled" non-monetary rewards also take on more importance. That is, while monetary rewards serve as one important incentive, developers in most any software development environment are often motivated by non-financial rewards such as solving a tough problem or showing-off an elegantly crafted piece of code to their peers. At one packaged software development firm which we visited, we were told: AWe=ve got the best [programmers] on the planet-- [they want to excel] not for their wives or for the money, but so that their peers will go >wow=.@

We contend that two other reasons for team cohesion (not directly related to team structure) are team size and co-location. Packaged software development teams tend to be smaller (Carmel and Bird, 1997). Again, an obvious reason for small size is that package firms themselves are often small. But, the reasons go beyond that. Packaged software development teams exist in an entrepreneurial and individualistic work culture that pervades the industry. So, developers make an effort to stay small. In contrast, IS teams commonly allocate more and more human resources to a project (Brooks, 1974).

The two types of teams often make use of their physical space differently. DeMarco & Lister (1987) first pointed to the problems of physical space in software development. Custom IS teams, because of their project-orientation, temporary structures, and matrix-management, are more likely to forgo rearranging locations and offices and rely on more formalized meetings and other forms of communications. Packaged software developers, however, seem to work best when they share common spaces (Sawyer, Farber, and Spillers, 1997). Packaged software development teams are far more likely to be co-located physically than IS teams. Small teams are often housed in the same room, or in adjacent offices.

Teams that stay together for a longer time begin to share and nourish the vision of the product. In packaged software it is not uncommon to find teams that are involved not only in one development cycle, but in multiple development cycles (or Areleases@ in the industry's parlance). For example, at a firm visited by one of the authors B a $40-million-per-year niche firm developing business applications B the team=s members were committed to one product for multiple development cycles and became masters of the application domain. Importantly, this team came to share a

vision of the product. This Avision sharing@ is a concept voiced by many developers in packaged software teams. Sharing a vision can only be done over time -- it is not something that can be written in system specifications and effectively passed on to the programmers.

The Microsoft model of teams

The differences between IS teams and packaged software development teams are epitomized by Microsoft. Says Ed Yourdon (1996):

> AIndeed the contrast between Microsoft=s peopleware culture and the culture I see in most in-house IT departments is staggering. In one case, the organizational culture is like the Army: unmotivated, semiliterate people whose behavior is augmented with Asmart@ weapons and rigidly controlled with military discipline. In Microsoft=s case, its more like the Marines or the Green Berets: highly motivated, highly skilled, and highly empowered to use whatever means necessary to achieve an objective. (p. 266)@

The "Microsoft model," with its management structure, its team practices, and its work culture, is becoming well known through articles (Cusumano and Selby, 1997) books (McConnell, 1993; Zachary, 1994; McGuire, 1994; Cusumano & Selby, 1995), workshops, and peer networks. This model is evolving and, like Microsoft=s products, is largely borrowed from the outside and refined from within. The Microsoft model is assimilated and emulated by many companies in the software industry. Having visited about one hundred firms between the two authors, we know that this technology transfer is taking place: we have noticed the above-mentioned books on shelves of software managers. Furthermore, in interviews, the exemplar of Microsoft's practices frequently comes up (though admittedly, not always positively). The key question is how much and how fast are Microsoft's practices diffusing within the industry?

Microsoft tries to keep its teams small-- even in large-scale projects-- in which the parallel teams are allowed some measure of independence. The company has worked to preserve many characteristics of both the entrepreneurial and hacker cultures from which it arose. And, the firms seeks to provide just enough structure to build adequate large scale software products in a highly iterative development process (Cusumano & Selby, 1997). The team structure includes the key position of team leads B Atechnical lead@ as well as Aproduct lead@ and Aprogram lead@ (the latter two representing some boundary spanning roles), who together share key authority roles.

Conclusions and Implications

We have argued that there are many factors, at several levels of analysis, that distinguish software development teams that produce packaged software from those that produce customized IS. Our motivation for presenting this is twofold. First, we believe the research community must be better aware of these distinctions when studying the various teams that produce software. Second, the professional community -- those who do this work -- must keep these distinctions in mind as they devise and refine methods and techniques for software teams.

Limitations of this study B and our contentions and observations B should be tested through additional research. Our observations are supported by both existing literature and our ongoing

empirical research. However, these comparisons are provided as a means to generate thinking and research. For those engaged in research we highlight two issues raised through this comparison.

Firstly, we cannot properly reconcile two contradictory dimensions of the packaged software team: that it is made up of individualists, yet has many positive characteristics of a successful team, such as cohesion. Is this a problem in our observations and analysis? For example, perhaps we have actually come across two types of work groups over the years: the first is composed of those so-called rugged individualists, is loosely functioning, yet is successful because of other factors. Another work group has less individualistic members and actually does have high levels of cohesion **B** and is also successful. Alternatively, is the problem that we have assumed as axiomatic that a cohesive team cannot be made up of a group of individualists? Perhaps we will discover that software firms have actually been able to reconcile these conflicting attributes successfully.

A second issue for researchers is implied in the title of this paper. That is, differences exist among software development domains. These differences affect the relationship between software development team characteristics to performance (see Krishnan, this issue) and the relationship between software vendors and software users (i.e., Kjaer and Madsen, 1997). Further, these differences call into question the applicability of the current research on software team processes (i.e., Zachary, 1998, this issue).

For those doing packaged software development we have argued that this work is different from custom IS development in several ways. This leads to two broad implications. The first broad implication is to be very thoughtful about what to borrow from the existing body of literature and practice on software development. For example, Keil and Carmel (1995) show that greater attention to how the existing literature on how users and developers interact will help improve this aspect of development in packaged software development. However, both Carmel and Becker (1995) and Cusumano and Selby (1995, 1997) suggest that packaged software development processes are much different from the more common, and more formalized, software engineering processes. That is, while the ethos of the packaged software industry has been to establish it=s own ways with little regard for traditional software development practices, the discerning developer can learn from the existing literature -- if they are wise about the differences between these domains. The work of Dube (this issue) supports this assertion.

The second broad implication raised by the assertions in this article for those doing packaged software development is to manage these team members differently. For example, packaged software developers are more attuned to financial rewards, expect more out of their employers, and seek flexible and open work arrangements. This puts pressure on development leads to balance control while encouraging creativity, a theme that plays out in Zachary=s (1994) book. After all, this is the world of computing gurus and hacking kings.

[the end]

Note 1: Packaged software is also referred to as Acommercial,@ Ashrink-wrapped,@ Acommercial-off-the-shelf (COTS)@ and software products. Another major segment of software development that is not discussed in this paper is the embedded software (e.g., software for automobiles, washing machines, avionics, etc).

References

Adams, J. and Kirchof, N. (1982), *Conflict Management for Project Managers*, Drexel Hill, PA: Project Management Institute, 1-13.

Bach, J. (1995), Enough about process: what we need are heroes. IEEE Software, March, 12(2), pp. 96-98.

Brooks, F. (1987), ›No Silver Bullet: Essence and Accidents of Software Engineering,‹ *Computer, 19*(5), 10-19.

Brooks, F. (1974), *The Mythical Man-Month*, Reading, MA: Addison-Wesley.

Boehm, B. (1987), ›Improving Software Productivity,‹ *Computer, 19*(9), 43-57.

Boehm, B. (1991), ›Software Risk Management: Principles and Practices,‹ *IEEE Software, 8*(1), 426-435.

Business Software Alliance (1997), *World Trends*, Washington DC: BSA.

Carmel, E. (1997), ›American hegemony in packaged software trade and the "culture of software",‹ *The Information Society,13*(1), 125-142.

Carmel, E. (1995), ›Cycle-time in packaged software firms,‹ *Journal of Product Innovation Management, 12*(2), 110-123.

Carmel, E. and Bird, B. (1997), ›Small is beautiful: a study of packaged software development teams,‹ *Journal of High Technology Management Research, 8*(1), 129-148.

Carmel, E. and Becker, S. (1995), ›A process model for packaged software development,‹ *IEEE Transactions on Engineering Management, 41*(5), pp. 50-61

Cringely, R. (1992), *Accidental Empires.* Reading, Mass: Addison-Wesley.

Crowston, K. and Cammerer, E. (forthcoming), ›Software Development as Collective Mind,‹ *IBM Systems Journal.*

Cusumano, M.A. 1991. *Japan's software factories: a challenge to US management.* New York: Oxford University Press.

Cusumano, M. and Selby, R. (1997), ›How Microsoft Builds Software,‹ *Communications of the ACM, 40*(6), 53-61.

Cusumano, M. and Selby, R. (1995), *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, New York: Free Press/Simon & Schuster.

Curtis, B., Krasner, H. and Iscoe, N. (1988), ›A Field Study of the Software Design Process for Large Systems,‹ *Communications of the ACM, 31(*11), 1268-1287.

Constantine, L. (1995), *Constantine on Peopleware*, Englewood Cliffs, NJ: Yourdon Press.

DeMarco, T. (1995), *Why does software cost so much and other puzzles of the information age*. NY: Dorsett House Publishing.

DeMarco T. and Lister, T. (1987), *Peopleware: productive projects and teams*, New York: Dorsett House.

Dube, L. (1998, this issue), ‹The Difficult Journey from a Sequential to a Team-based Development Process: The Story of Software Co.,› *Information Technology & People 11*(1).

Frye, C. (1997). The Software 500, *Software Magazine,17(*8), 41- 52.

Goodman, P., Ravlin, E. and Argote, L. (1986), ‹Current Thinking About Groups: Setting the Stage for New Ideas. *Designing Effective Work Groups,*› In P. Goodman and Associates (Eds), San Francisco: Jossey-Bass. 227-242.

Grudin, J. (1991), ‹Interactive Systems: Bridging the gap between developers and users,› *IEEE Computer, 24* (5), 59-69.

Guimaraes, T. (1985), ‹A Study of Application Program Development Techniques,› *Communications of the ACM, 28*(5), 494-499.

Guinan, P. and Cooprider, J., and Sawyer, S. (1997), "The Effective Use of Automated Application Development Tools," *IBM Systems Journal, 36*(1), 124-139.

Harpers, (1997), *The Harpers Index, 295*(1770), 17, 92.

Heckman, R. and Sawyer, S. (1996), "The Information Resource Acquisition Process, "*Proceedings of the 1996 AIS Americas Conference*, J. Carey, (Ed), New York: ACM Press, 207-209.

Humphrey, W. (1989), *Managing the Software Process*, Reading, MA: Addison-Wesley.

Keil, M. and Carmel, E. (1995), ‹Customer-developer links in software development,› *Communications of the ACM, 38*(5), 33-44.

Kjaer, A. and Madsen, K. (1997), ‹Customer-Vendor Co-Operation,› *Information Technology & People 10*(3), 205-223.

Kling, R. and Iacono, S. (1984), ‹The Control of Information Systems Developments after Implementation,› *Communications of the ACM*, 27(12), 1218-1226.

Krishnan, M. (1998, this issue), ‹The Role of Team-Factors in Packaged Software Quality,› *Information Technology & People 11*(1).

Kozar, K. (1989), ‹Adopting Systems Development Methods: An Exploratory Study,› *Journal of Management Information Systems, 5*(4), 73-86.

Larson, E. & Gobeli, D. (1988), ‹Organizing for product development projects.› *Journal of Product Innovation Management*, *5*, 180-190.

Leebaert, D. (1995), ‹News From the Frontiers,› in Leebaert, D. (Ed), *The Future of Software*, Cambridge, MA: MIT Press, 1-28.

Levy, L. (1987), *Taming the Tiger: Software Engineering and Software Economics*, New York: Springer-Verlag.
Maglitta, J. (1997), ASuper Programmers,@ *Computerworld*, 11/24/97, 95-96.
Markus, M. (1983), APower, Politics, and MIS Implementation,@ *Communications of the ACM*, *26*(6), 430-444.
McGrath, J. (1990), ATime Matters in Groups,@ In J. Galeghar, R. Kraut, & C. Egido (Eds.), *Intellectual Teamwork, Social and Technological Foundations of Cooperative Work* . Hillsdale, NJ: Lawrence Earlbaum Associates.
McConnell, S. (1993), *Code Complete*. Redmond, Washington: Microsoft Press.
McGuire, S. (1994), *Debugging the development process: practical strategies for staying focused, hitting ship dates, and building solid teams*. Redmond: Microsoft Press.
Paulk, M. (1995), AThe Evolution of the SEI=s Capability Maturity Model for Software,@ *Software Process, 1*(1), 3-16.
Sawyer, S., Farber, J. and Spillers, R. (1997), ASupporting the Social Processes of Software Development Teams,@ *Information Technology & People (10)* 1, 46-62.
Schein, E. (1992), *Organizational Culture and Leadership, 2cnd Edition*, San Francisco: Jossey-Bass.
Weinberg, G. (1971), *The Psychology of Computer Programming*, New York: Van-Nostrand Rheinhold.
Yourdon, E. (1996), *Rise and resurrection of the American Programmer*, Upper Saddle River, NJ: Prentice Hall.
Yourdon, E. (1993), *Decline and Fall of the American Programmer*, Upper Saddle River, NJ: Prentice Hall.
Zachary, G. (1994), *Showstopper: The breakneck race to create Windows-NT and the next generation at Microsoft*. New York: The Free Press.
Zachary (1998, this issue), AArmed Truce: Software in the Age of Teams,@ *Information Technology & People, 11*(1).
Walz, D., Elam, J. and Curtis, B. (1993), "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration,@ *Communications of the ACM*, *36*(10), 63-77.

Figure 1: Analytic Framework

Table 1: Summary Table of differences between IS and packaged software

	Packaged Software	Information Systems
INDUSTRY		
	Time to market pressures	Cost pressures
	Success measure: profit, market share	Success measures: satisfaction, acceptance
SOFTWARE DEVELOPMENT		
	Line positions	Staff positions
	User is distant and less involved	User is closer and more involved
	Process is immature	Process is more mature
	Somewhat integrated design and development	Separated design and development
	Design control via coordination	Design control via consensus-building
CULTURAL MILIEU		
	Entrepreneurial	Bureaucratic
	Individualistic	Less individualistic
TEAMS		
	Less likely to have matrix/ project structure. More likely to be self-managed	Matrix managed and project focused
	Involved in entire development cycle	People assigned to multiple projects
	More cohesive, motivated, jelled	Work together as needed
	Opportunities for large financial rewards	Salary-based
	Likelier to be small, collocated	Grow larger over time and tend to disperse
	Share a vision for their product(s)	Rely on formal specifications/ documents