Software Development Worldwide: The State of the Practice

Michael Cusumano, *Massachusetts Institute of Technology*
Alan MacCormack, *Harvard University*
Chris F. Kemerer, *University of Pittsburgh*
Bill Crandall, *Hewlett-Packard*

Over the past decade, researchers have studied the wide range of practices available to software developers[1,2] and the differences in practices and performance levels around the world.[3] This article reports early descriptive results from a new global survey of completed software projects that attempts to shed light on international differences in the adoption of different development practices. Our findings are particularly relevant to firms that are considering the potential advantages or a greater use of outsourcing.

A survey of 104 projects in India, Japan, Europe, and the US is yielding quantitative data about these countries' software development practices and performance, adoption of competing development models, and the practices' impact on performance.

Research motivations

We decided to conduct a new study for three reasons. First, we were interested in how practices and performance levels vary around the world. In particular, one issue that remains unresolved is the accuracy of anecdotal reports of high levels of quality, productivity, and on-time performance in Japanese and, more recently, Indian projects. Many people have speculated that Indian software companies emphasize formal practices and processes (such as the Software Engineering Institute's Capability Maturity Model Level 5 certification) for use as a signal of quality when bidding on outsourcing contracts. However, previous studies of this topic used only small sample sizes[3] or were based mainly on older data from a few companies.[4–6]

Second, we wanted to ascertain the degree to which a large sample of projects, regardless of location, used different types of development practices associated with particular development "models." We wanted to assess the penetration of a range of practices—from those associated with more traditional waterfall approaches, which tend to emphasize control and discipline in development,[7] to those

0740-7459/03/$17.00 © 2003 IEEE

that underpin more flexible, iterative development models, which emphasize speed and flexibility in adapting to potentially uncertain and evolving customer requirements.[8–10]

Finally, we were interested in contributing to the stream of studies that have examined the impact of different development practices on various dimensions of project performance. In particular, we wanted to contrast the practices associated with more conventional waterfall-style development models with newer development styles based on iterative methods, including "synch-and-stabilize" approaches. Although previous studies found support for these newer practices' predictive ability, they tended to be limited to a small sample of projects carried out in a specific software context (for example, Internet software). We decided to broaden our analysis so that we could examine potential dependencies on other contextual factors, such as software type (for example, systems versus applications) and target hardware platform (for example, workstation versus PC).

The survey

Following our pilot survey of approximately 30 Hewlett-Packard and Agilent projects,[11,12] we conducted a global survey of software development practices during 2001 and 2002.[13,14] Participating companies included Motorola India Electronics, Infosys, Tata Consulting, and Patni Computing from India; Hitachi, NEC, IBM Japan, NTT Data, SRA, Matsushita Electronics, Omron, Fuji Xerox, and Olympus from Japan; IBM, Hewlett-Packard, Sun Microsystems, Microsoft, Siebel Systems, AT&T, Fidelity Investments, Merrill Lynch, Lockheed Martin, TRW, and Micron Technology from the US; and Siemens, Business Objects, and Nokia from Europe.

Because of the way we identified survey respondents, our sample isn't a true random sample. To initially solicit responses, we emailed a worldwide list of industry specialists and researchers and asked them to forward our request for participation to their colleagues and industry contacts. We also approached software associations and research organizations (such as the Fraunhofer Institute) and software-focused trade journals (such as *InformationWeek*) to promote participation among their members and readers. We collected the survey data using a secure Web server hosted at MIT. The university affiliation was designed to give respondents greater confidence that we would treat their responses confidentially.

Given this general approach, which reflects the difficulty in gaining global participation in such surveys, our results must be interpreted

Prior Global Surveys

Two surveys in particular helped inspire our survey. In 1990, two of us (Michael Cusumano and Chris Kemerer) published a survey of 40 projects in the US and Japan.[1] Our main purpose was to add quantitative analysis to the discussion about whether software development practices and performance levels at major Japanese firms were comparable, better, or inferior to major US firms. The results showed that the countries were similar in the types and sizes of products developed; the development tools, programming languages, and hardware platforms used; and the developers' prior work experience. In terms of differences, the Japanese projects spent more time on product design, the American teams on actual coding. The Japanese projects also exhibited higher levels of reuse, which was particularly interesting because we found a statistically significant relationship between reuse levels and lines-of-code productivity. However, although the Japanese completed projects with fewer defects and higher LOC productivity, the differences compared to US projects weren't statistically significant, perhaps due to the relatively small sample size.

In a later and broader study published in 1996, Joseph Blackburn, Gary Scudder, and Luk Van Wassenhove surveyed 40 projects from the US and Japan and 98 from Western Europe.[2] They wanted not to make regional comparisons so much as to identify factors that contributed to development speed and LOC productivity. Several factors proved to be statistically significant: for example, the use of prototypes, customer specifications, computer-aided software engineering tools, parallel development, recoding, project team management, testing strategies, code reuse, module size, communication between team members, and quality of software engineers. In particular, the researchers found that

- Spending more time and effort on customer specifications improved both development speed and productivity.
- Prototyping, better software engineers, smaller teams, and less code rework contributed to faster development.
- More time and effort spent on testing and integration negatively affected overall development time.

Their results suggested that early planning and customer specifications are crucial to productivity, whereas "doing it right the first time" is essential for reducing development time.

References

1. M. Cusumano and C.F. Kemerer, "A Quantitative Analysis of US and Japanese Practice and Performance in Software Development," *Management Science*, vol. 36, no. 11, Nov. 1990, pp. 1384–1406.
2. J.D. Blackburn, G.D. Scudder, and L.N. Van Wassenhove, "Improving Speed and Productivity of Software Development: A Global Survey of Software Developers," *IEEE Trans. Software Eng.*, vol. 22, no. 12, Dec. 1996, pp. 875–885.

Table 1

Project descriptions (percentages are in parentheses)

	India	Japan	US	Europe & other	Total
Number of projects	24	27	31	22	104
Software type					
System	7 (29.2)	5 (18.5)	4 (12.9)	4 (18.2)	20
Applications	4 (16.7)	4 (14.8)	7 (22.6)	5 (22.7)	20
Custom or semicustom	11 (45.8)	16 (59.2)	19 (61.3)	10 (45.5)	56
Embedded	2 (8.3)	2 (7.4)	1 (3.2)	3 (13.6)	8
Level of reliability					
High	8 (33.3)	12 (44.4)	8 (25.8)	4 (18.2)	32
Medium	14 (58.3)	14 (51.9)	20 (64.5)	18 (81.8)	66
Low	2 (8.3)	1 (3.7)	3 (9.7)	0 (0.0)	6
Hardware platform[1]					
Mainframe	2 (8.3)	6 (22.2)	3 (10.0)	1 (4.5)	12
Workstation	16 (66.7)	16 (59.2)	19 (63.3)	15 (68.2)	66
PC	3 (12.5)	4 (14.8)	7 (23.3)	1 (4.5)	15
Other	3 (12.5)	1 (3.7)	1 (3.3)	5 (22.7)	10
Customer type[2]					
Individual	0 (0.0)	1 (3.7)	2 (6.7)	2 (9.1)	5
Enterprise	23 (95.8)	23 (85.2)	21 (70.0)	17 (77.3)	84
In-house	1 (4.2)	3 (11.1)	7 (23.3)	3 (13.6)	14

1. One project didn't provide data on hardware platform.
2. One project didn't provide data on customer type.

cautiously. To the degree that this sample doesn't represent the broader population of projects in each region, our results might not generalize to these broader populations. For example, because the survey required a significant amount of detailed project data, projects under greater managerial control (that is, better-managed projects) are more likely to be included in the sample. So, for example, our results might represent better, rather than average, project performance.

After removing duplicate responses, responses providing insufficient data, and responses for incomplete projects, the sample comprised 104 projects. Table 1 lists descriptive data on these projects, broken down by region of origin.

Projects

Table 1 shows regional differences by various project descriptors, including software type, required reliability level, target hardware platform, and main customer type. The 104 projects were relatively evenly divided across India, Japan, the US, and the remainder, which include primarily European projects. (From here on, for simplicity we refer to the last category as "Europe" even though it includes countries outside Europe—for example, Israel. Also, in the following discussion, all comments reflect the data as presented in the tables, not a statistical analysis. So, for example, references to "significant differences" are informal, not to be interpreted in the statistical sense of the word. In fact, because of large variances in samples of this type, many seemingly large differences in averages turn out not to be statistically significant at a 1 percent or even 5 percent confidence level.) The Indian projects showed a greater proportion of system software projects, whereas the majority of Japanese and US projects said they built custom or semicustom software. Europe claimed the highest percentage of embedded software projects.

In terms of required reliability, few respondents identified their project as "low," as we might expect in such a survey based on self-reported data. Japanese projects showed the relatively greatest difference between high and low reliability.

Most of the surveyed projects used workstations as their hardware platform. Japanese projects reported the highest percentage of mainframe projects, the US the highest percentage of PC projects, and Europe the highest percentage of projects using other platforms.

Table 2

The practices used

	India	Japan	US	Europe and other	Total
Number of projects	24	27	31	22	104
Architectural specifications (%)	83.3	70.4	54.8	72.7	69.2
Functional specifications (%)	95.8	92.6	74.2	81.8	85.6
Detailed designs (%)	100.0	85.2	32.3	68.2	69.2
Code generation (%)	62.5	40.7	51.6	54.5	51.9
Design reviews (%)	100.0	100.0	77.4	77.3	88.5
Code reviews (%)	95.8	74.1	71.0	81.8	79.8
Subcycles (%)	79.2	44.4	54.8	86.4	64.4
Beta testing (% ≥ 1)	66.7	66.7	77.4	81.8	73.1
Pair testing (%)	54.2	44.4	35.5	31.8	41.3
Pair programming (%)	58.3	22.2	35.5	27.2	35.3
Daily builds (%)					
At the start	16.7	22.2	35.5	9.1	22.1
In the middle	12.5	25.9	29.0	27.3	24.0
At the end	29.2	37.0	35.5	40.9	35.6
Regression testing on each build (%)	91.7	96.3	71.0	77.3	83.7

The vast majority of projects were described as largely for enterprise use. The US sample, however, included relatively more projects for in-house use, which is consistent with the high percentage of custom projects in the US sample.

Practices

The survey first asked about conventional practices that followed more of a waterfall-style model and relied on tools and techniques once popular among large-scale software systems developers such as IBM and the Japanese computer manufacturers. For example, we asked whether the projects wrote architectural and functional specifications as well as developed detailed designs before coding. Did it use code generation tools? And did it go through formal design and code reviews? Then we asked about a set of newer techniques geared to making projects more flexible. For example, in the "synch-and-stabilize" model, projects are broken down into multiple milestones or subcycles, each geared to delivering a subset of a product's final functionality.[9] At the end of each milestone, the team "stabilizes" or debugs the code and might also release an early beta version for feedback from selected customers.[8–10,15] At a more micro level, developers "synchronize" their work using techniques such as daily or weekly code builds (and tests) to ensure that they find and correct problematic component interactions at the earliest opportunity. These techniques are often used in conjunction with other approaches for gaining early feedback on a design—for example, pairing developers with testers to check code prior to submission. We therefore report data on how many projects divided development into milestones, used early beta tests, paired programmers with testers, followed daily builds, and conducted detailed regression tests (as opposed to simple compile-and-link tests) on each build (see Table 2).

Most of the sample used architectural, functional, and design specification documents rather than just writing code with minimal planning and documentation. These conventionally well-regarded practices were especially popular in India, Japan, and Europe. The major difference was in the US, where specifications were used less often across the board. This is most striking with regard to detailed design specifications, which reportedly were only used in 32 percent of US projects (in contrast to 100 percent in India). Interestingly, Cusumano and Selby observed a decade ago that Microsoft programmers generally didn't write detailed designs but went straight from a functional specification to coding; they did so to save time and not waste effort writing specifications for features that teams might later delete.[9,15] Our results suggest this might be becoming a more widely followed practice in the US.

With regard to other more conventional

Table 3

Performance data

	India	Japan	US	Europe & other	Total
Number of projects	24	27	31	22	104
Median output[1]	209	469	270	436	374
Median defect rate[2]	.263	.020	.400	.225	.150

1. No. of new lines of code / (avg. no. of staff × no. of programmer-months).
2. No. of defects reported by customers in 12 months after implementation / total source LOC. We adjusted this ratio for projects with less than 12 months of data.

practices, all the projects we sampled also used design and code reviews extensively. In fact, 100 percent of Indian projects reported doing design reviews, and all but one reported code reviews. This suggests that there is some truth to the recent speculation that Indian software companies are increasingly adopting more formal quality management techniques.

The newer, more flexible development practices were also popular around the world, with some variations. Over 64 percent of all projects broke their work into subcycles, for example, although these were more common in India and Europe than in Japan or even the US. Firms that didn't use subcycles, by our definition, followed a more conventional waterfall process. Less than half the Japanese projects used subcycles, indicating that a waterfall process is still a popular choice in this region. Almost three-quarters of all projects used early beta releases, which have become a useful tool for testing and obtaining user feedback, especially since the arrival of the World Wide Web.[8,10,16] There was no clear difference between regions in the use of this practice.

At a micro level, there were also some interesting observations. Only a third of the sample used daily builds at some point in development—a surprising statistic given the publicity paid this technique over recent years. More interesting, however, is the pattern of usage noted over a project's life. US projects tended to decide at the start whether to adopt daily builds and then stick to this strategy throughout a project, whereas European projects (and Indian and Japanese projects to a lesser extent) tended to vary use of this practice over a project's life. Specifically, daily builds weren't used much at all early in a project but became much more common in the later stages. With regard to running regression tests on builds, the number of projects doing this was fairly high—over 80 percent—with the highest concentrations of this practice in Japan and India. Finally, more than 40 percent of the projects surveyed paired testers with developers, and nearly as many reported using paired-programmer techniques. Again, these practices appeared to be especially popular in India.

Performance

Project performance is difficult to measure and even more difficult to compare regionally, but we did collect traditional software development measures of performance to further characterize our sample. Table 3 presents these measures:

- Output per programmer-month of effort (in new LOC)
- Number of defects reported by customers per 1,000 lines of total source code in the 12 months after delivery to customers

We report median levels of performance to avoid outliers' impact on sample means. The performance differences observed between regions are likely due in part to the differing project types, underlying hardware platforms, coding styles, customer types, and reliability requirements. The numbers are therefore descriptive only of the data in this sample and not as the basis for projecting the performance of future projects in each region.

The Japanese projects in our sample achieved the lowest defect levels (median of 0.020). The Indian projects (0.263) and European projects (0.225) were quite similar to each other, but considerably higher than the Japanese. The US projects had the highest median defect rate (0.400), but this rate is somewhat similar to the European and Indian levels. In terms of LOC delivered per programmer per month—an admittedly limited measure of output in this context—the Japanese and European projects ranked at the top. (The SLOC/effort measure has limited utility when making comparisons across organizations. It has been used effectively, however, when comparing projects within a single organization, thus greatly reducing possibly unaccounted-for sources of variation.[17,18]) They had a median output level of about 450 LOC per programmer-month, unadjusted for programming language or project type. This was approximately twice the level of the Indian and US projects in our sample. The same caveats about differing project character-

istics cited earlier for defect data apply also to output data.

Various small-sample studies from the late 1980s and early 1990s also found higher levels of code output and fewer defects from Japanese software projects as compared to US projects, so the Japanese results might not be surprising.[3] US programmers often have different objectives and development styles. They tend to emphasize shorter or more innovative programs and spend more time optimizing code, which ultimately reduces the number of LOC and increases effort. The Indian organizations often have a significant fraction of US clients and might well have adopted a US-type programming style, although we expected their defect levels to be closer to Japan's, given their emphasis on more formal practices that historically have been associated with improving software reliability.

More interesting, however, are similarities in code productivity compared to projects from more than a decade ago in contrast to potentially large apparent improvements in quality. For example, in the survey published in 1990, Cusumano and Kemerer reported[3] median LOC productivity per year of 2,943 (245 per month) for a sample of US projects and 4,663 (389 per month) for a sample of Japanese projects—levels similar to the new survey. In the 1990 survey, however, median defect numbers reported by customers per 1,000 lines of source code in 12 months after shipping a product were 0.83 for the US sample and 0.20 for the Japanese sample, much worse than the current rates.

Links between practices and performance

The work that seeks to connect specific process choices with particular performance attributes in this broad sample is still ongoing. However, software engineering researchers and practitioners should be interested in the findings that emerged from the pilot data for this survey.[12]

First, we found that developers appeared to write more code per day when they had a more complete functional specification. Second, having more complete designs before coding appeared to correlate with fewer defects. These results make intuitive sense and have led many software managers to insist on having complete specifications before people start writing code—as is advocated in a waterfall process. In a technical sense, programmers can be more productive if they make fewer changes during a project and thus have less rework to do. They also have less chance of introducing errors when they make fewer design and code changes.

Yet, in a business sense, locking a project into a particular design early on might not produce the best product in a changing market. We also found that using early betas and prototypes—opportunities for customers to provide early feedback on a design—was associated with more output and fewer defects, probably because projects could make early adjustments based on the feedback (as opposed to finding out later that some things were wrong). Furthermore, running regression or integration tests with each build and conducting design reviews (both mechanisms for gaining early feedback on a design) were associated with fewer defects.

Importantly, when we captured the effect of all these practices in a model predicting performance, the presumed disadvantages that stem from not having a complete specification disappeared. That is, adopting practices associated with a more flexible process (that is, those geared to generating early feedback on product performance) appears to compensate for incomplete specifications. In a sense, these practices seem to provide an alternative mechanism for generating the type of information that a specification typically communicates. Our findings help explain why early software development research might have concluded that waterfall-style processes lead to improved performance; they might not have captured data on the use of other (more flexible) practices that were actually better performance predictors. And they also help us understand that in selecting a development model, we should be careful not to think that we can "cherry-pick" only those practices that look most appealing. On the contrary, development models are best regarded as coherent sets of practices, some of which are required to balance the potential performance trade-offs arising from the use (or absence) of others.

Adopting practices associated with a more flexible process appears to compensate for incomplete specifications.

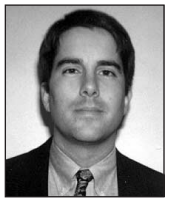
Our data suggests that anecdotal evidence emerging about the process and practice strengths developing in India are well founded. They also show some con-

About the Authors

Michael Cusumano is the Sloan Management Review Distinguished Professor at MIT's Sloan School of Management. He specializes in strategy, product development, and entrepreneurship in the software business. His most recent book, *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know, in Good Times and Bad*, is forthcoming in 2004. He received his PhD in Japanese studies from Harvard University and completed a postdoctoral fellowship in production and operations management at the Harvard Business School. Contact him at the MIT Sloan School of Management, 50 Memorial Dr., Rm. E52-538, Cambridge, MA 02142-1347; cusumano@mit.edu.

Alan MacCormack is an associate professor of business administration at the Harvard Business School. His research explores the management of product development in high-technology industries, particularly in the software industry. He teaches an MBA course titled "Managing Technology Ventures," which explores the factors that drive success in new technology-based initiatives. He received his PhD from Harvard Business School, where he received the George S. Dively Award for distinguished research. Contact him at Morgan Hall T39, Harvard Business School, Soldiers Field Park, Boston, MA 02163; amaccormack@hbs.edu.

Chris F. Kemerer is the David M. Roderick Professor of Information Systems at the Katz Graduate School of Business, University of Pittsburgh. His research interests include software evolution and management and measurement issues in information systems and software engineering. He is editor in chief of *Information Systems Research* and a member of the IEEE Computer Society. He received his PhD in systems sciences from Carnegie Mellon University. Contact him at 278A Mervis Hall, KGSB, Univ. of Pittsburgh, Pittsburgh, PA 15260; ckemerer@katz.pitt.edu.



Bill Crandall is director of product generation services at Hewlett-Packard. His team is responsible for delivering shared engineering services and for developing and disseminating new and better ways of developing products, services, and solutions across HP. He holds an MS in computer science and an MS in management from MIT, where he was a fellow in the Leaders for Manufacturing program. He is a member of the ACM. Contact him at Hewlett-Packard, 1501 Page Mill Rd., MS 1229, Palo Alto, CA 94304-1126; bill.crandall@hp.com.

tinued strengths in Japan. But, as is common in this type of research, due to the extreme variations in performance from project to project, drawing any definite conclusions is hard. It is important to remember as well that no Indian or Japanese company has yet to make any real global mark in widely recognized software innovation, long the province of US and a few European software firms. Code productivity taken in isolation might not be a good proxy for business performance, and it is probably less valuable than a defect measure for judging a development organization's performance. Unfortunately, comparable financial performance data is almost impossible to come by in these surveys, given the wide range of unique circumstances that each project addresses (for example, custom-developed software for internal use versus packaged software for retail sale). Nonetheless, above all, our data shows that Indian organizations are doing an admirable job of combining conventional best practices such as specification and review with more flexible techniques that should let them respond more effectively to customer demands. If the broader population can replicate this trend, the Indian software industry will likely experience continued growth and success.

References

1. S. McConnell, *Rapid Development*, Microsoft Press, 1996.
2. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
3. M. Cusumano and C.F. Kemerer, "A Quantitative Analysis of US and Japanese Practice and Performance in Software Development," *Management Science*, vol. 36, no. 11, Nov. 1990, pp. 1384–1406.
4. M. Cusumano, *Japan's Software Factories*, Oxford Univ. Press, 1991.
5. Y. Matsumoto, "A Software Factory: An Overall Approach to Software Production," *Tutorial: Software Reusability*, P. Freeman, ed., IEEE CS Press, 1987.
6. M. Zelkowitz et al., "Software Engineering Practices in the US and Japan," *Computer*, vol. 17, no. 4, June 1984, pp. 57–66.
7. W.W. Royce, "Managing the Development of Large Software Systems," *Proc. IEEE WESCON*, IEEE Press, Aug. 1970, pp. 1–9.
8. M. Iansiti and A. MacCormack, "Developing Products on Internet Time," *Harvard Business Rev.*, vol. 75, no. 5, Sept./Oct. 1997, pp. 108–117.
9. M. Cusumano and R. Selby, "How Microsoft Builds Software," *Comm. ACM*, vol. 40, no. 6, June 1997, pp. 53–62.
10. A. MacCormack, "Product-Development Processes that Work: How Internet Companies Build Software," *MIT Sloan Management Rev.*, vol. 42, no. 2, Jan. 2001, pp. 75–84.
11. S. Upadhyayula, *Rapid and Flexible Product Development: An Analysis of Software Projects at Hewlett-Packard and Agilent*, unpublished master's thesis, System Design and Management Program, MIT, June 2001; available in the MIT library or through interlibrary loan.
12. A. MacCormack et al., "Software Development Practices: Exploring the Trade-offs between Productivity and Quality," *IEEE Software*, vol. 20, no. 5, Sept./Oct. 2003, pp. 78–85.
13. P. Cheung, *Practices for Fast and Flexible Software Development*, unpublished master's thesis, Dept. Electrical Engineering and Computer Science, MIT, June 2002; available in the MIT library or through interlibrary loan.
14. M. Cusumano, *The Business of Software*, to be published by Free Press/Simon & Schuster, 2004.
15. M. Cusumano and R.W. Selby, *Microsoft Secrets*, Simon & Schuster, 1995 and 1998.
16. M. Cusumano and D. Yoffie, "Software Development on Internet Time," *Computer*, vol. 32, no. 10, Oct. 1999, pp. 2–11.
17. C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM*, vol. 30, no. 5, Sept. 1987, pp. 416–429.
18. R.D. Banker, S.M. Datar, and C.F. Kemerer, "A Model to Evaluate Variables Impacting Productivity on Software Maintenance Projects," *Management Science*, vol. 37, no. 1, Jan. 1991, pp. 1–18.