The Limits of the Division of Labor in Design and the Prospects for Off-Shore Software Development in Mexico

Michael J. Piore
David W. Skinner Professor of Political Economy
Department of Economics, MIT

Prepared for
The Software Industry in the Developing World Workshop
Yale University
February 20 to 22, 2004

This paper addresses the prospects for Mexico as a destination for off-shore development of software projects originating in the United States. Its starting point is a M.S. thesis written by Alexander Artola examining the staffing patterns on projects at a large off-shore software subcontractor in Mexico. Artola looked at three different sizes of projects, and for each size, identified the typical labor requirements, how these were distributed across various categories of labor, and how the size and distribution evolved over the life of the project. He also examined the geographic distribution of the assigned labor force between the sub-contractor's location in Mexico and the contractor's location in the United States and how that geographic distribution shifted over time. Finally, he traced the qualifications of the different types of labor with respect to formal education, company training and on the job experience. These findings are summarized in the accompanying charts (Appendix 1). They suggest two principle conclusions:

1. The relatively high qualifications of even lower level programmers and code writers.
2. The relatively large proportion of the subcontractors' staff located at any one time in the United States.

The management rule for these projects imposed by General Electric, which pioneered off-shore software development, and adopted by other U.S. companies, is that the number of foreign personnel in the United States at any one time should be limited to 30% of the project work force. However, interviews by Artola and others suggest that this rule is actually a minimum which is often exceeded in practice. What it reveals is a tendency for projects once sent off-shore to migrate back toward the United States.

How does one understand these findings? Here I draw heavily on a literature on the organization of software development but also on a study of the organization of product design and development that I have been conducting in collaboration with my colleague Richard Lester at the Industrial Performance Center at MIT.

The Division of Labor and Its Limits

The basic explanation suggested by our design studies is the limits of the division of labor. Since at least the writings of Adam Smith, the division of labor has been the basic principle of work organization. The central idea is captured by the example of the pin factory where one worker pulls the wire, a second worker cuts the wire, a third worker heads the pin, a fourth points the pin, etc. The pin factory is often contrasted with Robinson Crusoe, who performs all of the work on his island by himself. But the industrial stage of development represented by the pin factory is preceded by a possibly even more important stage in which the jack of all trades in a Robinson Crusoe economy is replaced by a social division of labor among a series of specialized crafts and professions, e.g., plumbers and carpenters, doctors and lawyers, who are masters of a domain of activity governed by a coherent body of knowledge which can be progressively perfected over time in a way that a generalist who works across many domains cannot possibly do. The division of labor is most effective when the skills involved in the distinct operations are different and, hence, breaking up the work in this way, at one and the same time, encourages specialization and benefits from the enhanced understanding that specialists bring to their work. Large projects or projects constrained by time are only feasible if the work can be divided and assigned to different workers, even when specialization is not possible. When work is organized in this way, however, the different pieces of the project must ultimately be brought back together and combined to form a “whole”. The process of doing this may be termed “integration”; the process of insuring that it can be done is generally referred to as “coordination”; integration and coordination are thus the other side of the division of labor.

Integration and Coordination

The problems of integration are minimized when the pieces into which the product is divided are independent of each other. And “independence” is thus the critical engineering principle in theories of project organization. Independence becomes a key principle in product design as well because only when the design can be broken into independent components can the project itself be so divided. The canonical example of a design with these properties is a church key which consists of a bottle opener at one end and can opener at the other, connected to each other as the two ends of a flat metal bar. But while engineers universally agree on the criticality of independence, there are no standard rules for achieving it. It is apparently a matter of intuition. As one engineer explained it to us: "Sure it is important. But once I see how to separate the components, the whole design is already done."

The problem of coordination in manufacturing is almost the mirror image of the problem in design projects. In manufacturing it is solved by freezing the design so that the interfaces between the different components of a product are standardized. The components are, in other words, not independent of each other at all; each part fits into the whole in a specific, completely deterministic way. Integration is then achieved through interchangeable parts. The key to modern manufacturing was thus the metal template that ensured that each and every copy of the same part would be produced to an exact specification.

When components are not independent, or the interfaces standard, integration becomes a much more complex and difficult problem. Airplane manufacturing has never been able to achieve the kind of standardization common in automobile production (or in pin factories). The production line in airplane assembly is characterized by workmen hammering and stretching the components to get them to fit together. All along the line are boxes of small metal plates known as "shimmies" which are inserted in an ad hoc way to fill spaces left at the joining of parts that are not fully compatible.

There is no equivalent in design to the "shimmy" in manufacturing. When the components of a design project are not independent, the individual designers on a project team must constantly consult with each other and cross-check their work as they go along to make sure that the pieces will be compatible in the end. Various other ways of achieving this have been proposed in the design literature. One of the most prominent is the "heavyweight" project manager who carries around the image of the whole in his head, imposes it on the team, and checks the work of the different team members as they go along to ensure conformity. The alternative is an integrated team that develops a common vision at the outset and then works together throughout the project to ensure by constant discussion and debate that the vision is maintained as the project evolves and the various pieces conform to it.

Integration in Software

A solution to the problem of integration and coordination perpetually eludes the software industry. It has proven impossible to invent a procedure for organizing software projects in a way that enables the components of the project to be easily reintegrated with each other. As a result, when a software program is broken into pieces that are assigned to different programmers and the pieces are then brought back together, they invariably interact in complex and unpredictable ways to produce "bugs". Integration then involves an elaborate and painful attempt to identify the sources of these bugs and to revise the software so as to eliminate them. A wide variety of organizational principles and programming tools have been developed to address this problem, ranging widely from the software factory and the nightly build to object-oriented programming. Each of these has in its turn been heralded as a solution, a panacea for the whole problem, but in the end each has also proved disappointing.

The underlying nature of the problem – and the debate it has engendered within the industry – is suggested by two books, written years apart, but with essentially the same theme. One of these is Frederick Brooks, The Mythical Man-Month, originally published in 1975; the second is Peter McBreen's Software Craftsmanship, the New Imperative published twenty-seven years later in 2002.

Brooks takes his title from the attempt to estimate the relationship between the time required to complete a software project and the staffing requirements; the paradox of this effort is that when the project falls behind schedule and you attempt to catch up by adding staff, you delay the project further. The problem, Brooks argues, is that the different programmers on a project need to understand the overall architecture of the program; failure to appreciate the interrelationship between your work and the rest of the program in a deep and profound sense is responsible for bugs and delays. Adding new members to the team who do not share the overall vision just increases the number of bugs. The optimal organization, Brooks argues, is in fact one in which a single master programmer designs and builds (and writes all the code) himself. Additional members of the team should work to support the master, freeing up his time to work on programming exclusively. They should be assigned to bring him food, fetch material from the library, etc., but not to work on the program itself. The solution is in essence the heavyweight project manager proposed in the automobile industry (although in some ways it seems to represent a return to Robinson Crusoe).

Brooks had an enormous influence upon the industry when his articles first appeared, but in conversation today there is a tendency to dismiss his work as characterizing an earlier era, when projects were of a different kind (though of exactly what kind is not always clear). It is therefore interesting to note the appearance of McBreen's book just two years ago with a very similar argument–although, unlike Brooks, tracing the problem to the difficulty of the division of labor and the craft-like (pre-pin factory) nature of programming. McBreen argues that these characteristics of software development are becoming more salient as time goes on, not less.

The Problem of Software Integration Inherent in Design Generally

While the books convincingly identify the nature of the problem, they do not explain from where the problem comes. Our work at the Industrial Performance Center suggests that the problem lies less in the nature of software itself than in the nature of design. The so-called production of software is trivial. Virtually anybody can copy an existing program on their PC. Hence the manufacturing model of the division of labor with standardized interfaces is largely irrelevant. What we are faced with is the problem of the division of labor in design. In order to be successful, then, the designer is required to identify independent components. This is the essential thrust of "object-oriented" programming – the latest "solution" to the software design problem. An object is a segment of the program which can be altered internally without affecting any other segment. But the key is to define objects which have this property. And that is evidently an art, similar to the art which leads to the church key design for kitchen openers.

The division of labor in software is, this suggests, equivalent to the division of labor in clothing design. Imagine breaking the design of a shirt into a series of components and assigning each component to a separated designer, the right sleeve to one designer, the left sleeve to another, the collar to a third, the front to a fourth and so on. The problem of putting these separate designs back together and integrating them to form a single, coherent garment is exactly analogous to debugging a software program. The pieces of the shirt are bound to clash with each other and to have to be revised when they are put together. But it would be difficult to say in the abstract what exactly is wrong with their initial fit. Indeed, individual garments are <u>never</u> designed in this way. Fashion collections (like that of the GAP or of Levi's) are more often split up among the members of a team and different whole garments assigned to different designers. But when the collection is finally put together to be presented to the public, it has to be "edited" and numerous items discarded. The editing typically occurs in a design meeting at which the various items are displayed and looked at in relation to each other; the issue of what fits is vigorously debated, and items are discarded by consensus. But again the criteria are never fully articulated, let alone applied scientifically. Again, the editing is equivalent to debugging, or it would be if instead of discarding the items which do not fit, those items were actually redesigned so that they did. But in the clothing industry, there is seldom time to go back to the drawing board.

In designing fashion collections, as in designing cars (and for small, embedded software programs) companies often use a single designer, the equivalent of the heavyweight project manager. But the alternative to designating a design czar is a team that develops a common vision of what the collection will look like. The nature of that vision and the process through which that team is created is helpful in understanding the staffing patterns that Artola observed in Mexico.

The vision is sometimes expressed as a theme or series of themes, but it is generally impossible to fully articulate; it is a much richer common understanding than any written statement can convey. Indeed, the collection when it finally emerges could be understood as an "objective" expression of what until then remains essentially tacit understanding. That understanding develops through interactions among the various team members which, in writing with my college Richard Lester, we have likened to a conversation at a cocktail party. In fashion, that conversation typically takes place around concrete examples of particular garments, picked up on communal shopping trips, or watching people thought to be representatives of the target customers shop with each other, listening to their comments and then coming home to discuss them with the rest of the team, interoperating what those comments imply for creating an appealing design. Thus, the meaning of a design theme when it is chosen tends to be associated with a particular comment which the team has discussed and interpreted and unless one took part in that discussion, it would be hard to understand what the theme the team had adopted actually meant. In this sense, what emerges out of the conversation amounts to a common language. The language is so closely linked to the vision the team developed that one could almost say that the vision is more like a language than a concrete plan or a set of architectural blueprints for the collection. That language, or vision, moreover, evolves as the team moves through the process of actually bringing a particular collection to market, and then (if the team stays together) as it moves from one collection to another. The expansion of the team, the drawing of new members into it, is correspondingly like the integration of foreigners into a language community.

The nature of clothing design – which incidentally is not very different from design in the other industries we studied (cellular telephones or medical devises) in the way it depends on conversation which generates something that more closely resembles a language than a concrete vision – and the analogy between clothing and software design suggests the nature of the relationship between the contractor and the subcontract in off-shore development. The architecture of the program is developed by the contractor or by the contractor in collaboration with the subcontractor. But the term "architecture" is misleading in the sense that it is nothing like a builder's blueprint. It begins with a list of the properties the program must have, the tasks which it is designed to perform, and a general characterization of how it will perform them. But the specification is very general and to the outsider vague. It often depends on the "reader" knowing about similar programs and their limits because it is these limits that the new project seeks to address. The precise understanding is thus linked to specific projects and approaches which the chief architect remembers from the past and the team understands because they too worked on these projects. Or understanding is linked to specific examples which may be used to illustrate the vision but whose complete meaning depends heavily on the context in which they arise and/or are presented. Thus, it is very hard to convey the vision except through face-to-face contact in which the expositor can see whether his or her interlocutors seem to understand and if they do not, seek on the spur of the moment other ways to explain what is meant and other examples which convey the meaning. And, of course, the architect's own vision changes as he or she seeks to explain it to others. Thus, the program architecture is more like a living organism: It is rooted in tacit knowledge built over time and embedded in the shared experience and the history which pins down and fills out knowledge that cannot be expressed in words. Artola, drawing his vocabulary from his interviews, in fact uses the term <u>ethos</u> as a synonym for vision.

All of this implies that a software development project cannot be "thrown over the wall". Rather, the subcontractor and its personnel must be drawn into the vision of the contractor and the community out of which that vision grows. Both the ethos of a program and the community in which it is embedded develop in a process of organic growth. That process is captured in Artola's interviews with the Mexican off-shore developers. On very large projects the project leader is permanently stationed in the U.S. and is in direct contact with the customer. Together, the project leader and the customer lay out the design of the project. The project leader then communicates with the Near Shore Coordinator in Mexico and he or she works with a team of analysts and designers at the Mexican site to flesh out the architecture, or what Artola calls the <u>ethos</u> of the project. In the construction stage, the analysts and designers actually become programmers, so they carry with them into the programming or construction phase of the project a direct knowledge of the overall design. The senior programmer then breaks up the project into pieces and assigns the pieces to junior programmers. They are the only members of the team who lack an understanding of the overall picture.

The model of organic growth is also consistent with the data collected on the evolution of off-shore contracting relations in India by Banerjee and Duflo. The authors themselves explain their observations in terms of the development of trust. But it would be very hard to distinguish the development of trust from the evolution of a language community through conversation. Indeed, Lester and I argue that in a certain sense trust is only really possible within a language community.

A third observation about off-shore contracting relationships is also supportive of the notion of organic growth: The way in which the U.S. contractors manage the evolution of these relationships. Our own interviews suggest that the preferred pattern is to develop a stable of subcontractors who are invited to bid for RFP's in competition with each other. Each subcontractor must first be qualified by the U.S. company on the basis of their capabilities to be admitted to the "stable". Once qualified, they are informed of the projects open for bid and invited to the conferences where the projects are discussed. But newly certified subcontractors typically start with small projects and work their way up to larger, more complicated ones, developing in this way a relationship with the contractor's personnel and an understanding of the contractor's language.

Some Implications

What does this pattern of organic growth suggest about the prospects for the software industry in Mexico?

1. The pattern of organic growth suggests the importance of personal contacts to start the process (and in fact all along the chain). Those countries which have developed extensive off-shore contracting, particularly India, but also Ireland, Armenia, and Israel, have a large Diaspora of educated manpower who have worked as programmers in the United States, and those people appear to be key bridges to the industry abroad. The stream of Mexican migration to the United States is extremely large, but its characteristics do not enable it to constitute bridges to Mexican industry. It is largely unskilled. The skilled manpower, especially those who come to the United States for higher education, have historically tended to return to Mexico as soon as their education is complete. That pattern is changing, and the pool of people with contacts in the U.S. out of which an off-shore industry might develop is growing, but at a moment Mexico must compete with other countries which already have established relationships with the U.S. Hence special efforts may be required to stimulate the industry, efforts to find Mexican nationals in the U.S. who are already working in software programming and would be interested in returning home to start their own business, to provide financing, etc.
2. Neither language nor the supply of skilled labor seem to be a problem. The language which is at issue is the insiders language that develops between the contractor and subcontractor over time and then diffuses to the subcontractor's personnel. But only people working directly with the contractor actually need to

speak English, and while Mexico would appear to be at a disadvantage in that respect relative to India, Ireland or even Israel (where English is widely spoken), it seems to have an adequate supply of English speakers in the programming community; and those people already working in the United States, who will constitute the initial contacts, are fluent and at home in the English language.

3. The importance of the chain of conversation in guiding the evolution of the program as it develops does place a premium on direct conversation and thus it would seem that the proximity of Mexico to the United States and the fact that it is in the same time zone should give the country a special advantage relative to its competitors. In the end, easy to pick up the telephone or even to fly to the contractor's home base when problems develop.

Bibliography

Artola, Alexandro. Implications of Nearshore Development in the Mexican Software Industry, Unpublished Masters Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2003.

Banerjee, Abhijit V., and Esther Duflo (2000) "Reputation effects and the limits of contracting: A study of the indian software industry," *Quarterly Journal of Economics*, 115(3), 989—1017.

Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. Boston: Addison-Wesley, 1995.

Cusumano, Michael A. Japan's Software Factories; A Challenge to U.S. Management. New York and Oxford: Oxford University Press, 1991.

Cusumano, Michael A. and Richard W. Selby. Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People. New York: Simon & Schuster, 1998 (1995).

Lester, Richard and Michael J. Piore. Innovation—The Missing Dimension, Harvard University Press, forthcoming, Fall 2004.

McBreen, Peter. Software Craftsmanship, the New Imperative. Boston: Addison-Wesley, 2002.

Appendix 1

Tables from: Artola, Alexandro. Implications of Nearshore Development in the Mexican Software Industry, Unpublished Masters Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2003.

Table 9:	Occupational distribution for small AD projects.
Table 10:	Occupational distribution for medium AD projects.
Table 11:	Occupational distribution for large AD projects according to location.
Table 12:	Time frame for AMS projects.
Table 14:	AMS project estimations according to size.

Table 9: **Occupational distribution for small AD projects.**

	Inception	Elaboration	Construction	Transition
Effort	30 hours	120 hours	390 hours	60 hours
Duration	4 days	12 days	3 weeks	4 days
Project Leader	x	x	x	x
Analyst	1			
Designer		1		
Senior Programmer/Tester			2	2
Junior Programmer/Tester			2	2
Total	1	1	4	4

Table 10: **Occupational distribution for medium AD projects.**

	Inception	Elaboration	Construction	Transition
Effort	50 hours	200 hours	650 hours	100 hours
Duration	6 days	16 days	3 weeks	5 days
Project Leader	1	1	1	1
Analyst	1			
Designer		1		
Senior Programmer/Tester			2	1
Junior Programmer/Tester			2	2
Total	2	2	5	4

Table 11: **Occupational distribution for large AD projects according to location.**

	Inception		Elaboration		Construction		Transition	
	US	Mexico	US	Mexico	US	Mexico	US	Mexico
Effort	300 hours		1,200 hours		3,900 hours		600 hours	
Duration	2 weeks		6 weeks		10 weeks		2 weeks	
Project Leader	1		1		1		1	
Nearshore Coordinator		1		1		1		1
Analyst	3	1						
Designer			2	2				
Senior Programmer					2	2	2	2
Junior Programmer						3		3
Tester						1		1
Total	4	2	3	3	3	7	3	7
	6		6		10		10	

Table 12: **Time frame for AMS projects.**

Definition		Maintenance & Support		Conclusion	
Formalize Project	Initialize Project	Implementation	Operation	Transition	Project End
1-2 weeks	1-2 months	1-4 months	At least 4 months	1-6 months	

Table 14: **AMS project estimations according to size.**

	Small	Medium	Large
Project Duration (months)	6	12	24
Average Number of Resources (persons)	5	10	20
Effort (hrs)	4,860	19,440	77,760
Cost Estimate for Customer (US$)	$100,000	$500,000	$2,000,000