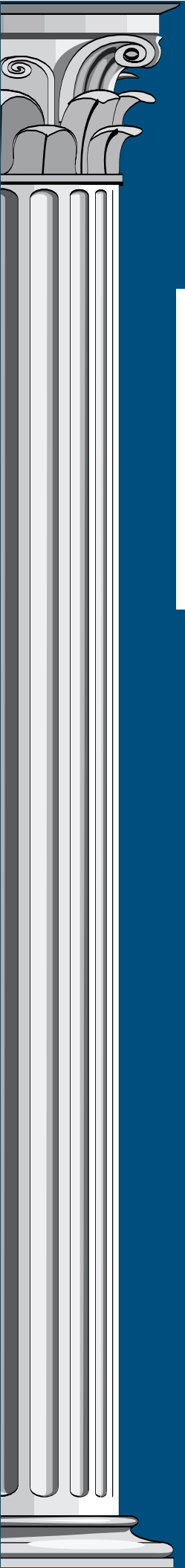

CUTTER CONSORTIUM

What Is Happening to the Global Software Village?

Is There Still a Case for Distributed Software Development?

AGILE PROJECT MANAGEMENT ADVISORY SERVICE

Executive Report, Vol. 3, No. 1

Fellows of the
Cutter Business Technology Council

Rob Austin

James Bach

Tom DeMarco

Jim Highsmith

Tim Lister

Dick Nolan

Ken Orr

Ed Yourdon

What Is Happening to the Global Software Village?

Is There Still a Case for Distributed Software Development?

AGILE PROJECT MANAGEMENT
ADVISORY SERVICE

Executive Report, Vol. 3, No. 1

by E.M. Bennatan

At the dawn of the 21st century, software development is an international activity. It is not uncommon to find parts of development organizations dispersed in distant locations. The world of software is indeed shrinking, and the global village is becoming more and more a reality. But after the events of September 11, many wonder whether this trend will continue. Will changes in global relationships and travel behavior affect the way software is being developed? Is a distributed software organization still a wise choice for today's corporations?

These questions are being deliberated in many boardrooms not just throughout corporate America but throughout the corporate world.

In order for companies to make the best decision, it is necessary to fully understand the advantages and the disadvantages of distributed software development (DSD), the problems and solutions that exist, and the tools and technology that have been developed to support it. This *Executive Report* provides the information necessary to help companies make these decisions. As we shall see, even with the concerns that have recently arisen, and possibly because of these concerns, there are many situations in which distributed software development is a sound corporate strategy.

AN EXAMPLE

An impressive cadre of infrastructure, tools, and techniques has been developed over the past two decades to support distributed organizations. A global communication infrastructure has indeed made the world seem much smaller, and many organizations have amassed a great deal of experience developing software in this shrinking world.

Let's consider a real-life example from my tenure at Motorola leading one of the company's international development centers, where we developed a large wireless telephony system (see Figure 1). The consumer units were designed and manufactured in Fort Worth, Texas, USA; the software for the infrastructure was designed in Tel Aviv, Israel; the infrastructure was manufactured in Chicago, Illinois, USA; and the

Figure 1 — Motorola's wireless telephony project teams.

testing and maintenance was provided from Bangalore, India. A project management nightmare? Quite the contrary. Actually, the project ran smoothly, was developed on time, and had software problems comparable to those of a project developed at a central location.

This should not be surprising. Developing high-tech projects, particularly software, at several locations is becoming increasingly common. What are the factors that are driving this trend? When is distributed development better than centralized development?

Both centralized development and DSD have their advantages and disadvantages. Centralized development is the classic way of developing software, and it may even appear to be the *natural* way of doing things. But there are many instances where this is not necessarily so. Consider the previous Motorola example of a DSD project. By producing the maintenance software in India, a local center of expertise was also being established in India that is capable of supporting the system in-country after its delivery. A centralized project would not have provided this advantage.

Even on a national level, centralized development may not always be the best solution. Within the US, there are regional proficiencies. Examples are: data processing expertise is more abundant in the New York area; aerospace in Florida; graphics and Internet-based technologies in California; and telephony in the Midwest. Companies have a tendency, therefore, to establish centers of expertise where the experts are.

On the international level, other factors play an important role in choosing locations for development centers, such as government incentives (e.g., Ireland, India, Israel), business needs (developing close to your market), and lower development costs. However, global development produces several challenges, primarily resulting from the inherent communication difficulties across distances, and also from some unexpected areas — such as cultural or local interpretations of documentation. Gary Anthes, writing in *Computerworld,* relates the story of a software developer having to travel from England to Germany to observe a problem being reported by a team of German software testers. Apparently the documentation said "to type a blank" and the testers were dutifully keying in "b-l-a-n-k" [1].

But there are even greater problems to overcome. Traveling from England to Germany is not as great a hassle as having to travel from Chicago to Bangalore (a trip undertaken by this author many times). The good news is that there

The Agile Project Management Advisory Service Executive Report is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com/consortium/. ISSN: 1536-2981.

are solutions provided by techniques, tools, and infrastructure.

The problems posed by DSD have generated a great deal of interest over the past few years. The Institute of Electrical and Electronics Engineers, Inc. (IEEE) devoted an entire issue of its software journal to the subject [14], and has organized several workshops [15] covering many new development tools and environments for both large-scale distributed projects and smaller, faster e-projects. In particular, the distributed development of Internet software has been a rapidly growing phenomenon, notably in Japan [17].

Lucent Technologies' Bell Labs has also devoted significant attention to the resolution of distributed development issues and has established a research team in that area. The Lucent research team summed up the questions it was studying as: "How can Lucent best use its people and capital resources to produce products with the desired cost, development interval, and quality when these resources are geographically distributed?"[1]

In the pages ahead we will review some of the best practices in DSD. We will look at a few real company experiences and see what they have learned. We will consider how a decision for or against DSD can be made; we will review some of the DSD methodologies and tools that exist; and we will discuss the structure of a distributed organization. We will then try to answer some of the questions about the impact of the current economy and the events of September 11, and we will attempt to look into the future.

WHAT CHARACTERIZES DISTRIBUTED DEVELOPMENT?

Let's start by clarifying what we mean by a distributed development project.

Distributed development refers to a project that is being developed at several remote sites. The definition is not entirely rigorous as there are no specific criteria that address when distance becomes significant, such as in the case of teams divided between different floors of a building, different adjacent buildings, or different buildings on a single campus. And what if the development at the different sites is not simultaneous? What if the different project components are really independent projects in their own right? The definition of a distributed development project is more one of convenience to enable us to address solutions for problems that arise when remote teams need to work together to develop a common project.

The distinguishing feature of a distributed system is therefore distance, and the distance is such that it produces specific challenges that can affect our ability to successfully complete the project. Different types of distance produce different types of problems, but the primary effect of distance is that it stretches the time needed for problem resolution [12]. This is what software organizations need to be aware of when they split up teams, whether they are divided between buildings, cities, states, or countries. Trivial as this seems, many project managers tend to overlook this.

Bridging the distance is the essence of any effort aimed at making a distributed project work well. This requires communications infrastructure, tools, methods, and techniques, and, above all, an understanding of the human aspect of distributed teams.

How exactly does distance affect team communication? Well, centralized team members have virtually unlimited opportunities to meet both formally and informally; even impromptu encounters at the water cooler can be an opportunity to exchange information, brainstorm, or just casually develop friendships between team members. These encounters are important not just because of the ease of information exchange, but also because of the team-building relationships that they can promote.

Let's see how this would work in a simple distributed team. We can imagine a typical distributed software project where three

[1]See www.bell-labs.com/org/11359/colab_prod/.

teams have divided the project into three parts (see Figure 2). The teams are distant from each other and cannot efficiently meet on a daily basis. On an overall project level, the three teams must function as a single team with a common goal: the successful completion of the full project. On a local level, each team functions with a significant degree of independence to achieve a local goal: the successful completion of the locally assigned part of the project. The degree of independence may vary, but it is significantly less than that of a centralized team.

Different companies have adopted this model in different ways. Their approach to DSD depends on many factors such as the location of the remote sites, the nature of the distributed project being developed, the internal culture within the company, and the company's previous experiences.

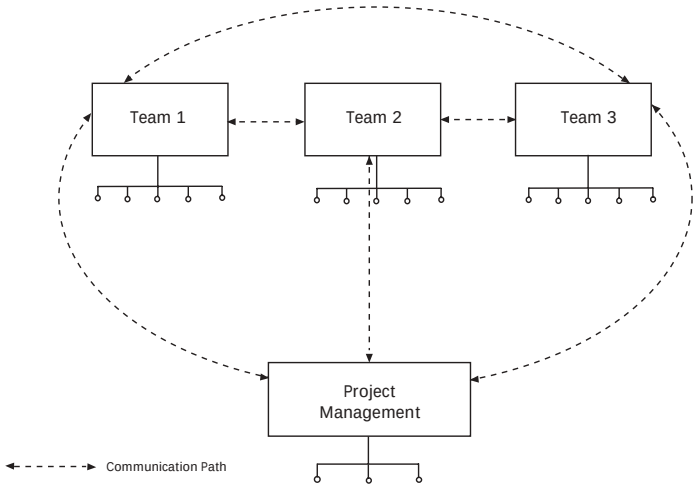

Figure 2 — Simplified example of a distributed team.

There is much experience on which to draw. Today, 203 of the US *Fortune* 500 companies already engage in offshore outsourcing [27]. They draw from more than 50 nations that are currently participating in collaborative software development internationally [4].

WHAT HAVE COMPANIES LEARNED FROM DSD?

There are common themes throughout the stories related by companies on their experiences with distributed development. Their success and failure have invariably been linked to their success or failure in bridging the human communications gap. This is always a major challenge in DSD because any form of remote team communication will always be inferior to direct human contact. Hence, the other advantages of a distributed project must always outweigh this disadvantage.

To put DSD more in perspective, let's discuss some of the lessons learned from the experiences of Alcatel and Motorola and one distributed experience in North America: the ESCOT project. We will then go on to discuss the characteristics and best practices of DSD in more detail.

The Alcatel Experience: The Importance of Collocation and Coherence

Christof Ebert from Alcatel in Belgium and Philip De Neve from Alcatel in India collaborated on an *IEEE Software* article on their company's experiences [8]. They report on the lessons they learned from some of the company's several thousand engineers who are dispersed in more than 15 development centers in Europe, the US, Asia, and Australia.

Ebert and De Neve identified the advantages of distributed development, such as time-zone effectiveness and reduced costs, as well as drawbacks, such as the additional overhead and expense for planning and managing people, language and cultural barriers, and jealousy (due to the different circumstances of some of the development sites). They list three of the primary drivers for adopting a global DSD strategy:

1. To be locally present for customization and after-sales service and to show the local customers how many new jobs were created, which in turn could justify more contracts

2. To support the growing number of acquisitions and mergers, which add new markets, products, engineers, and creativity to the existing team

3. The ability to hire young engineers with the necessary skills at a reasonable cost even in countries where neither the market nor the acquisitions would justify them

The Alcatel report relates to the company's experience in the Switching and Routing Business Division, which is highly software-intensive with 80%-90% of the business's research and development budget devoted to software. Most of the division's global development locations are at Capability Maturity Model (CMM) Level 2 with few at Level 3. Clearly, this is an excellent testbed for DSD, and, indeed several valuable lessons were learned from the experience.

Ebert and De Neve challenge a prevailing *virtual team* concept that would assume that anyone anywhere can be a member of any team. They recommend building what they refer to as *coherent* and *collocated* teams of *fully allocated* engineers.

Coherence means splitting the development work during development according to feature content and assembling each team so that it can implement a set of related functionality.

Collocation means that engineers working on such a set of coherent functionality should sit in the same building, perhaps within the same room. Locations providing experts with minor contributions should have them relocate to a more central project location for as long as they are needed on the project.

Full allocation implies that engineers working on a project should not be distracted by different tasks in other projects. Alcatel found that it would take two to three times the effort if people worked on several assignments in parallel.

Clearly, these requirements of a team are worthy for any type of project. They are particularly important in a distributed environment when communications and management are a challenge.

Alcatel found it useful to classify development staff into the following key roles:

- **Core competence —** highly experienced senior developers decide on architecture evolution, specify features, and review critical design decisions in the entire product line.
- **Engineering —** the majority of resources responsible for designing and integrating new functionality for all software.
- **Service —** specific functions for a group of projects with short or repetitive assignments, including industrialization, documentation, and maintenance activities.

These functions are allocated to various development teams, which together constitute the project team (there may be core competence teams, engineering teams, service teams, or even multifunctional teams). Alcatel used this classification of team functions as the basis for its method of allocating work to the various teams.

The Alcatel report also discusses the challenge of a changing and ever-improving organization that is heavily committed to DSD. Change, being often evolutionary, does not always occur at once across an organization, especially a large organization. It can produce incoherence among the various development sites, resulting in more complicated communication. This challenge was addressed by ensuring that all development locations working in one product line use the same process, methodology, and terminology, even when changes occur. This needed to be carried out without suppressing the ability of the company to change (which is firmly recognized by the authors as important for company survival).

What about language? Alcatel is based in France, so the issue of a standard language is both a technical obstacle and a political one. Interestingly, Alcatel chose English as the common language within the company.

Lastly, Ebert and De Neve conclude that their company benefits considerably from the very act of mixing teams from different global locations and cultures. Other companies have arrived at similar conclusions and have found that the interaction provides a diversity that promotes fresh thinking, new perspectives, and effective brainstorming.

The ESCOT Project Experience: Working with Autonomous Teams

Not all successful DSD endeavors are large projects. In fact, there are numerous successful examples of small DSD projects. Such is the case of a recent education software project supported by the US National Science Foundation. The project, reported recently by a team headed by Alexander Repenning of the University of Colorado [22] recounts the experience of a large testbed called Educational Software Components of Tomorrow (ESCOT) for the teaching of mathematics. ESCOT is a digital library being developed by a large pool of geographically distributed stakeholders in the US and Canada. The teams are relatively autonomous and are responsible for developing entire components, which are then added to the library.

A distinction of this type of project is that it may still succeed even if some components are not delivered or are delivered late. This is quite different from the Alcatel experience where different teams contribute critical elements to the project, without which the project may not be completed.

In addition to the development of the educational software testbed, the ESCOT project also had a second research goal: the exploration of the DSD process with the specific objective of building and deploying reliable software rapidly. The project employed a development process well suited for a distributed environment called Component-Oriented Rapid Development (CORD). CORD resembles Extreme Programming and is well suited for fairly autonomous teams. It includes the following elements:[2]

- The project start includes centralized analysis and design that attempt to anticipate connectivity issues (between the modules) that will arise later.
- Development is distributed to independent teams after initial centralized analysis and design is complete.
- Development is component-centered. This refers to the development of well-defined and distinct elements of the digital library system (the components). This approach is well suited for distributed teams using different tools and platforms because each component can be regarded almost as a mini-project on its own.
- Development and delivery are cross-platform. Individual developers (and users) can use the platform of their choice. Cross-platform development is possible with the appropriate tools, such as the CORBA[3] architecture and infrastructure or the Java programming language.

A conclusion from the ESCOT experience was that component-based approaches appear to be ideally suited for DSD. The use of the CORD method was found to be effective for projects with an aggressive project schedule, particularly in building educational applications. CORD's component-based nature enabled a high degree of parallelism involving distributed teams of domain experts, component framework coordinators, developers, and others involved in the software's development process.

The ESCOT production teams developed components on PC Windows, Macintosh, and Unix platforms using the process elements described above. Though several problems were reported (few would have surprised a DSD veteran), the ESCOT experience was successful. Among the lessons learned from the ESCOT project were:

- The importance of increasingly formal design representations to improve communications

[2]See [22] for a more detailed description of the CORD process steps.

[3]See www.omg.org/gettingstarted/corbafaq.htm for an overview of CORBA.

between the distributed team members and to avoid premature design commitments

- The need for a *component framework coordinator*, to ensure that components are properly integrated
- High test time, due to frequent platform- and virtual-machine-dependent implementation discrepancies that required a significant number of additional development cycles for debugging and workaround implementation

Figure 3 — Motorola's 3G Trial project development centers [2].

The Motorola Experience: Using Liaisons

While Alcatel concluded that a standard process across global teams was important, Motorola took a different approach. A recent article by Robert Battin et al. reports on the experience at Motorola developing a trial cellular system (called 3G Trial) with a global engineering team [2].

One of the incentives for using a distributed team was the unavailability of development staff; only about 20% of the required staff was available at Motorola's Arlington Heights-based facility near Chicago. The solution came from Motorola's software development centers worldwide. The project was staffed with engineers from six different centers (see Figure 3) across three continents.

One of the solutions employed by the project's Chicago-based management was the use of liaisons from the remote development sites. The liaisons were engineers from each location who moved to Chicago for up to three months. Their responsibility during this period was to meet the Chicago team, learn the system, help complete the system-level requirements and specifications, and communicate this information back to the development staff at their home location. The Motorola report describes the liaison model as a key factor in the successful completion of the project. Not only did the liaisons learn the system, but, more importantly, they developed relationships with the Chicago team and thus reinforced trust between the teams.

Once the liaisons returned home, the classic DSD problems of communications between teams appeared. They were addressed through intranet connectivity, conference calls, and travel. Travel was reserved for those instances where physical presence was absolutely required due to expense of travel in terms of both time and financial cost.

The 3G Trial project was a rich learning experience in areas of communication, development methodology, team structure, and local culture. Following are some of the lessons that were learned.

Loss of Communication Richness

- **Physical distances and time zones.** As related above, these issues were resolved through liaisons and later through intranet, conference calls, and travel.
- **Domain expertise.** This refers to the need for specific information and expertise to be transferred from the company's domain experts to the remote teams. This too was resolved by the liaisons.

Coordination Breakdowns

- **Architecture.** This refers to the innovative technology that was not familiar to all remote teams. This was resolved through low coupling between well-defined project components and their interfaces, a task assignment strategy based on local available expertise, and the abilities of the local facility.
- **Software integration.** Integration of components from remote sites is a classic DSD challenge. This was resolved through step-wise incremental integration, as opposed to "big bang" integration.
- **Software configuration management (SCM).** Configuration management too is a classic DSD challenge. This was resolved through a common SCM tool with multisite replication and a centralized problem report repository.

Problems of Geographic Dispersion

- **Vendor support.** This refers to the need for global support for development tools acquired from third-party vendors. This was resolved by ensuring international support contracts with the vendors and by using centralized reports for problems found in the tools.
- **Government issues.** This refers to immigration, visa, and travel issues; import and export rules; procedures for customs clearance; and US restrictions on export of some high-tech tools. This was resolved through experience and by learning the rules and regulations of each country.

Loss of "Teamness"

- **Differing development processes.** Different processes is also a common DSD problem. The Motorola team resisted the temptation of trying to impose a common process (in contrast to the Alcatel experience). This shortened the learning curve by letting each team begin producing immediately using the process that it knew. However, the definition of the work products and the working vocabulary was standardized. Also, it was ensured that each location's process encompassed a complete development lifecycle.

Cultural Differences

- **US impression of international development.** Motorola team members report that cultural differences were less of an issue for them. They attribute this to the vast experience that their company has had with global customers. There was, however, a concern among the US engineering team about the ability of the other teams to deliver functioning software. It was the act of working with the other global teams, and especially with the liaisons, that made the US team comfortable with its overseas colleagues.

  According to the Motorola report, the company's experiences with global development have been mostly positive. My own experiences at Motorola were similar. In fact, in one measure of quality — defects per lines of code — the global sites' code was marginally superior to the US team's code.

DECISION CRITERIA FOR A DISTRIBUTED ENVIRONMENT

A Major Driving Force: The Need for More Software Developers

With all these problems facing distributed development, why then have Alcatel, Motorola, and other development organizations adopted it? One of the reasons is the shortage of development engineers and the opening of a wider market to recruit developers. Though the demand for computer professionals ebbs and flows, as we have learned from the history of the past half-century, the trend has always been upward. Each new phase produces a stronger demand than its previous phase. Can we expect this trend to continue?

A decade ago, a prominent software author foresaw massive unemployment in the ranks of American programmers, systems analysts, and software engineers [26]. This prediction was based upon an expectation that international competition would put

American programmers out of work by the end of the 1990s just as Japanese competition put American automobile workers out of work in the 1970s.

It didn't happen. In fact, there was never such a high demand for software professionals in America as there was at the end of the 20th century. Was this prediction wrong, or was the timing off by a decade? Though the current economic slowdown has reduced the demand for software professionals (and indeed for most professionals), it has not been caused by international competition.

Even in the current sluggish economy, it is not bad to be a software engineer. This is true worldwide. Lawrence West provided some additional interesting data that seems to indicate that we are nowhere near seeing a lessening of the software engineer shortage in the US [25].

This is not an America-only quandary. The *Wall Street Journal* reported in 2000 that Western Europe had a shortage of 850,000 IT sector jobs with the shortage expected to grow to 1.7 million by 2003 [23]. Though this report predates the events of September 11, the data was so strong that many analysts predict that any effect of the events will be temporary. A good case can therefore be made for corporations worldwide to continue to strategize for a shortage of computer professionals even in the face of the current economy.

The most common response to this shortage over the past several decades has been increased training of computer professionals. But colleges and universities have never really been able to keep up with the growing demand. Other solutions are needed.

Distributed development has provided a partial solution, though not an easy one to successfully implement. We have seen that, in addition to the communications gap, there are problems to be overcome in determining development methods at multiple sites, project management, the division of work between the teams, and team structure. We will look at how these problems are being overcome, but first let us consider why all these problems do not scare off companies from using DSD.

When Is a Distributed Strategy Warranted?

James Herbsleb and Deependra Moitra [11] of Lucent Technologies, have listed five factors that, when present, favor distributed development:

1. The development resource problem: the need to capitalize on the global resource pool to successfully and cost-competitively use scarce resources, wherever located

2. The business advantage of proximity to the market, including knowledge of customers and local conditions, as well as the good will engendered by local investment

3. The quick formation of virtual corporations and virtual teams to exploit market opportunities

4. Severe pressure to improve time to market by using time zone differences in "round-the-clock" development

5. The need for flexibility to capitalize on merger and acquisition opportunities wherever they present themselves

In addition we may add the advantage discussed in the Alcatel story:

- The desire to benefit from cultural diversity

These factors will not be equally compelling for all companies. Due to the weakened global economy, the shortage of development resources may be less of a factor today for some companies than it was two or three years ago. However, for the very same reason, local investment may be more of a factor today due to the need of foreign governments to provide new jobs. The factor of proximity to the customer is generally evergreen in distant markets due to the invariable high cost of supporting remote systems.

At the beginning of this section, we may have wondered whether DSD was ever a good strategy.

We can now state that the answer is: yes, DSD is sometimes a good strategy. But does it necessarily have to be an arduous strategy? As we shall see, it need not be so. In the following sections, we will discuss some of the many methods, tools, and techniques that have evolved over the last two decades to ease the challenges of distributed development.

METHODOLOGIES AND TECHNIQUES IN DSD

What are the areas in DSD that require special methods and techniques? We have seen that communication is one such area, but the challenges of communications are so substantial that they are practically synonymous with DSD. Most of the DSD-specific methods and tools are geared toward bridging the communications gap caused by the distance between development locations. This affects the following four areas of the project:

1. **Work allocation.** In a distributed project, all teams need to communicate with each other; they need to coordinate their work so that all the project pieces fit together and form a complete working system. Though this is true in any project, it is a major consideration in DSD. As we have seen, when there is a distance between the teams, communication and coordination become more complicated. Work allocation for each team must be organized to minimize this difficulty.

2. **Project management.** It should not be surprising that the management of a distributed team requires special management processes and techniques. Distributed management differs from central management just as a federal government functions differently than a centralized government.

3. **Infrastructure.** The infrastructure for a distributed team is very different from that needed for a centralized team. Without Internets and intranets, e-mail, fax machines, and voice and video telecommunications, a distributed project would not be possible.

4. **Team structure.** Obviously, by its very nature, a distributed organization cannot be structured as a central organization. The distance between the teams and the differences in work practices and in culture require special attention to team organization. Here, too, it is interesting to note the similarities between the team structure of distributed project and the structure of a federal government. In many ways, the circumstances are not dissimilar.

These are not the only areas that require attention. After all, a distributed project is first and foremost a project and, as such, is subject to all the challenges of a regular project. Obviously, all the methods and tools used in regular projects are also applicable in distributed projects, but some of them are more complicated, such as configuration management, integration, and testing.

Let's start by looking at the method by which the development work is allocated to the remote teams.

Partitioning a Software Project

Can all projects be partitioned? Software is considered the more creative, or even artistic, of the engineering disciplines, and, as such, it is more difficult to partition. Imagine the result if Leonardo da Vinci had partitioned the *Mona Lisa* between his associates, or if Gershwin had subcontracted out parts of "Rhapsody in Blue." Is software development so creative that partitioning should be avoided?

Not surprisingly, the answer depends on the type of software project. Not all software projects can be, or should be, partitioned. It is a question of how closely together people need to work. For example, the software for a video game[4] is highly creative and thus requires a high level of interaction between its developers so it may best be developed at a single location. On the other hand, a large, commercial data-processing

[4]Today's video game software is quite complex with projects often taking around 40 person-years to develop over a period of two years.

project could require less interaction between its developers and therefore would more easily lend itself to partitioning between several development sites.

The level of interaction between the developers is a function of:

- The way in which the project is designed
- The characteristics of the project

In any branch of engineering, modular design practices are often a key factor for successful implementation. It is the basis of the divide-and-conquer approach to development of complex projects. It makes the project simpler — more manageable.

Modularity also enables a project to be subcontracted, or allocated to different development teams. For example, a suitably modular designed bridge can be partitioned into several components that can then be manufactured at different locations and shipped to a single site for assembly.

This is no different in software development. Modular software design is always good practice but especially so in a distributed project, as is evident from the analogy of the bridge.

Let's take a closer look at the characteristics of a project that make it partitionable.[5]

[5]This is also called chunking [19].

The concept of partitionability is closely linked to the concept of loose coupling, which has been around in computer engineering (both hardware and software) since the early days of structured design. In very broad terms, loosely coupled components are parts of a system that are relatively independent and have few points of interaction between them. In software design, loose modular coupling makes testing and integration easier [19].

We can therefore state that a software project is considered to be partitionable when it can be broken into loosely coupled components. In a distributed development environment, this reduces the need for communication between the development teams.

A distributed project must be designed from the ground up with partitioning in mind. There are many ways to do this:

- By functionality or features [20]
- By architectural subsystem
- By development phase
- By expertise
- By product version
- By a combination of the above

Partitioning by functionality means that each location is allocated one or more complete functions of the system. In a functional system design, the major project components can usually be designed for loose coupling; therefore, this is an ideal approach for a distributed project.

In large projects, the concept of a subsystem is critical. It is the highest level of the divide-and-conquer technique (also called stepwise refinement), without which large software projects would be unmanageable. Systems are always divided into subsystems by major functionality and are thus similar to partitioning by functionality. Subsystems are a natural choice for distributed development, and, in fact, very large software projects are commonly developed that way.

Partitioning by development phase is not particularly suitable for DSD except when there is some justification for specific phases being developed at different locations [19]. A common and rather obvious example would be the development of a project at one location while the maintenance team is located at the site where the system is installed. Also, occasionally, the requirements of a system may be developed at a location that is different from the development site. The US Department of Defense has been employing such tactics for many years, occasionally using different contractors for the development of the requirements and for the implementation of the other phases of the project.[6]

[6]This is rarely an option for iterative development paradigms such as spiral methods (e.g., Extreme Programming).

Partitioning by expertise is somewhat of a paradox. On the one hand, it is a poor way to partition a software project, but on the other hand, it is not uncommon. The reason is not surprising: development of complex software will occur where the experts are located (particularly when the experts will not or cannot come to where the project office is located). One way to resolve this paradox is to start out by noting the location of the expertise and to use that knowledge as a guide in functionally partitioning the project.

For many software products, partitioning the development by product version is suitable for DSD. An example may be the development of a video game on a Sony Playstation platform and assigning another facility to port the game to other platforms (e.g., Nintendo or Microsoft). This can happen parallel to the development of the main version or after the main version is complete.

Lastly, a combination of the above software partitioning methods can sometimes provide a good solution; for example, by using local expertise to guide the functional decomposition of a project. Another example would be incremental functionality where the initial project is developed at one location and additional functionality is added at a different location (e.g., foreign language localization).

Mockus and Weiss [19] report on their research at Bell Labs into defining an algorithm for DSD partitioning. The algorithm is based on an iterative process that allocates modification requests (defined as the elements of a new feature) to sets of assignments and continuously improves the allocation based on coupling between the sets. The result is a number of loosely coupled sets of assignments that corresponds to the number of distributed development locations.

At Alcatel, Ebert and De Neve [8] researched best DSD practices and reported on their allocation strategy, which employed an iterative/functional method of allocation. This meant that progressive versions of the system were released with new functionality added at each step. This approach is typical of the large legacy telephony systems developed at Alcatel, which were discussed earlier.[7]

Table 1 summarizes the different methods of partitioning a DSD project. Though the characteristics of each project may suggest a specific partitioning method, there are no strict rules on how to choose the winning strategy. As we can see from the Alcatel, Motorola, and ESCOT projects, experience is the ultimate guide for success.

The same approach to partitioning can be applied both to large projects with multiple remote teams and to smaller projects with remote individual developers (who can be considered a team of one). Clearly, the smaller the project, and the lesser the number of teams, the easier the partitioning. For larger projects, the divide-and-conquer approach

[7]The iterative/function approach has also been the method of choice for many Web and PC-based applications: Microsoft's DOS and Windows operating systems were developed that way.

Table 1 — Methods of Partitioning a Distributed Software Project

By Functionality	By Subsystem	By Development Phase	By Expertise	By Product Version
Partition by clusters of software features	Partition by high-level system component	Assign requirements, testing, or maintenance	Partition based on the expertise of the locations	Assign upgrade versions, language localization, etc.
Partition by combination of the above (e.g., functionality and expertise)				

There are the makings of a complexity function in this discussion where the basic arguments are the number of remote teams, the number of people in each team, a measure of the diversity between the teams, the diversity within the teams, and the distances between them. There is a challenge here for someone to describe a function that acts upon these arguments together with the partitioning of the project and produces a measure of the project complexity. This would produce an excellent early indication of the extent of the expected DSD challenge.

significantly reduces their complexity by turning them into a collection of smaller subprojects, which are then easier to partition.

DSD Project Management

The opening example in this report, quoted from my own tenure at Motorola, concludes with the rhetorical question: was the example a project management nightmare? It is indeed a fitting question to ask because, as all software project managers know, communications within the development team are a key factor in successfully managing the project. This is no different from any form of management — be it managing a *Fortune* 100 corporation or commanding an army. Sun Tzu recognized this 2,500 years ago, when he stated: "A well-developed communications network links your mind to your men so that instead of a collection of individuals more or less headed in the same direction, you now have a *unit*. Good communications is the key to controlling large groups of men." [18]

The Motorola wireless telephony project, described earlier, was developed in the mid-1990s. In many ways, its management was typical of large DSD projects. With teams in the US, Israel, and India, there was no problem confirming Sun Tzu's assertion: communications were indeed the primary management challenge.

The methods that resolve these management challenges can be divided into:

1. **Software project processes.** A DSD project is first and foremost a software project and, as such, requires all the standard processes necessary for the successful management of software development.

2. **Tools and technology.** There is a wide range of tools and technology available to help bridge the communications gap between facilities.

3. **DSD-specific management methods.** It is the distance between the teams that requires specific methods of management.

Process: One Standard Versus Many

One of the main challenges of software development process in a distributed project is the diversity of processes between the locations. From a management perspective, many of the elements that affect the direction, organization, and administration of the project may be different, such as: the development methodology, the development standards, the reporting format (e.g., status or progress reports), and particularly the terminology, which defines the technical language between the teams.

There are opposing perspectives on how this problem should be addressed. In the Motorola experience related earlier, Battin recommends not imposing a common process. The rationale is that learning a common process takes valuable time, and the learning curve would impact the project's delivery schedule.

Alcatel's Ebert takes the opposite position and recommends rigorously enforcing a CMM Level 3 interactive process model.

These opposing views may well be explained by the nature of the two organizations. Alcatel appears to promote a more process-based homogenous organization across all facilities, while Motorola tolerates more diversity. Clearly, there are advantages and disadvantages to both approaches. In a homogenous organization, the Alcatel practice of standardizing process makes sense, and in a diverse organization, Motorola's strategy makes sense. However, irrespective of the degree of diversity, the successful management of a DSD project requires modern

development processes just like any other type of software project.

The Evolution of Intelligent Tools

The use of tools and technology is what makes the management of DSD possible. In the early days of software development, a distributed team would be all but impossible to manage. Modern day technology, from fax machines to the Internet, have shaped the global village, where distances seem to have shrunk. The two principal mediums that have helped achieve this are:

1. Improved telecommunications, including reliable and inexpensive phone service, videoconferencing, fax machines, e-mail, and the Internet

2. Easy, convenient, and inexpensive travel

The telecommunications revolution has also spurred a host of intelligent tools and applications that are transported by the global communications infrastructure. Many of the applications are geared toward making the management of remote locations more practicable [22, 24]. These applications have become increasingly intelligent in their ability to help manage complex project tasks.

Intelligent software programs are not a new phenomenon. In software development, they can help with such tasks as decisionmaking (a natural candidate for artificial intelligence). Popular versions of decisionmaking applications are used for such tasks as the purchase choice between several automobiles based on the users' preferences and priorities and the grading of the candidate automobiles according to those preferences.

The concept of an intelligent assistant[8] for distributed projects was presented at an IEEE workshop in 1999 by Rory O'Connor and John Jenkins [21]. The O'Connor study based its research on the assertion that users of existing project management tools could benefit greatly from the inclusion of intelligent assistants' techniques when used together with the users' familiar tools. The study listed three applicable project activities where intelligent assistants can be valuable:

1. **Project planning** — particularly scheduling

2. **Process management** — in areas that support the framework and rules of project management.

3. **Risk analysis** — tools that help assess risk throughout the project lifecycle

[8]This was by no means a new concept, and, in fact, the authors traced it as far back as to the fourth century BC when Socrates claimed to have a nonhuman intelligent assistant, which he called Daemon. The development of computer technology raised interest in the concept at many stages, one of the early implementations being Eva in the 1980s — a computer software application that was purported to function as a personal psychoanalyst.

One of the characteristics of these activities is that they have the potential to accumulate vast amounts of information during the project lifecycle. This information is often too much for an individual to absorb, particularly when diverse and distant teams are producing the information. When captured into a database, intelligent assistants can process and analyze this information and provide valuable guidance to the developers and project management.

Examples of intelligent tools are the Distributed Software Project Management Tool (DSPMtool) [24], which gathers, analyzes, integrates, and disseminates the outputs of various software project management processes, and MS-Project, which has the intelligence to resolve planning conflicts between several different teams. More about these and other DSD tools in the "Overview of Tools and Technology" section starting on page 15.

DSD-Specific Management Methods

Erran Carmel, who has written extensively on DSD, states that an organization cannot function without coordination and control, and distance creates difficulties in both [4]. Carmel grouped the approaches to alleviating the problems of distance into six centripetal forces that exert inward pressure on the team for more effective performance:

1. Collaborative technology
2. Team building
3. Leadership
4. Product architecture and task allocation
5. Software development methodology
6. Telecommunications infrastructure

These forces cannot be expected to come to bear on their own. They need to be part of the overall project management DSD strategy for bridging the communication gap. Simply stated, project management must have a plan to address each of these six spheres of action. Just as we recommended earlier that a distributed project should be designed from the ground up with partitioning in mind, so we can state that the project management plan for a distributed project must be created from the beginning with DSD in mind.

Table 2 lists the basic steps to develop a DSD project management plan:

This is not a serial process. Steps 1, 2, and 3 can be implemented to a large degree in parallel. Step 4 can iterate back to step 3 and so forth.

Table 2 — Steps to Generate a DSD Project Management Plan

Step 1	Create a preliminary project management plan (e.g., use IEEE Std. 1058)
Step 2	Study the characteristics of each of the distributed teams
Step 3	Identify DSD challenges, (i.e., areas that require specific solutions such as: communication, culture, local methods and processes, government regulations, travel, available expertise, and available resources)
Step 4	Using the preliminary plan as a starting point, expand and enhance the plan to address the DSD challenges

OVERVIEW OF TOOLS AND TECHNOLOGY

As we have seen, a major role of technology in a distributed project is the bridging of the communications gap. James Herbsleb, who has been leading Bell Lab's research into distributed development, concludes that the problem with multisite development is largely one of the delay caused by intersite communications [10, 11]. We know that people communicate more when they are located at the same site, and the distance between the sites makes the resolution of problems more difficult.

Today's telecommunications technology has the ability to bring remote locations closer together. Most, if not all, tools in the DSD project manager's toolbox, are based, one way or another, on telecommunications. The Internet, videoconferencing, e-mail, fax transmissions, and improved phone service have brought remote sites closer together and have thus changed the way we work. Corporate communications networks now span the globe with broadband data capacities and high reliability and availability. This has provided the infrastructure on which more sophisticated tools can run.

The Bell Labs study identified the need for a more advanced toolset and concluded that the common communications tools were not enough to resolve the distance problem. Following are some of the areas in which new advanced tools and intelligent applications are being developed:

- Cross-site project management systems
- Cross-site configuration control
- Fast file transfer mediums
- Cross-site problem management systems [17]

The following paragraphs discuss some of the tools that are available today for distributed projects.[9]

- **Project planning** — this includes tools that help partition the overall project plan into individual team plans.
- **Process management** — these are the tools that support the framework and rules of software development, such as configuration management.
- **Project supervision** — this includes the facilities and tools that support the flow of information between the teams and the project manager, the quality assurance team, and the configuration management team. In a small project, the project manager may hold all, or some, of these responsibilities.
- **Interteam communications** — this includes the media that enable the teams to easily and effectively exchange information and hold discussions.
- **Identification of risks and the resolution of problems** — this includes tools that address specific risks and problems related to DSD, such as the loss of communication, incompatibility of implemented tasks, or the security of data.

[9]See also [21].

The project planning and process management tools are among the core functions required in the early stages of project launch. In particular, configuration management, which is important in any software development system, is especially critical in a distributed environment. This is the discipline that controls the individual parts of the overall system, administers changes to the items, records their implementation status, and verifies compliance with the project's specified requirements. Clearly, if not done well, this can be a frustrating challenge in any distributed environment. This is one of the chief areas where DSD either succeeds or fails.

The interteam communications tools become more important to the project as the teams grow. Toward the end of the project, the critical phases of testing and integration cannot occur without them.

As we have seen, some DSD tools function as intelligent assistants, analyzing data from several sources. In other cases, DSD tools are no more than an integrated set of individual tools that cover different project activities. Obviously, the ideal is for these tools to work together (though in DSD, that is not yet common).

Table 3 provides examples of both commercially available tools and internally developed proprietary tools.[10] This is not a list of recommendations but rather a collection of examples.

Table 3 — Examples of DSD Support Tools

DSD Activity	Examples of Tools
Communication between teams	TeamPortal [9] Microsoft's NetMeeting
DSD planning	Microsoft Project
Problem tracking	Rational's ClearQuest Multisite
Project management	DSPMtool [24]
Change control	Rational ClearQuest Multisite
Configuration management	Concurrent Versions System
Module/document check-in/check-out	Workgroup Cache [16] CHIME [7]
Problem tracker	Pragmatic Software's Software Planner Hitachi's Problem Management System [17]
Overall software engineering environment	PROSYT [5]

TEAM-RELATED ISSUES

Loss of teamness was a concern mentioned by Battin in his experiences with DSD at Motorola. There is strength in a good team that exceeds the sum of the strengths of the team members. In a distributed environment, the effect of this benefit can be lost.

[10]Of the many communications tools and software applications that support distributed development environments, many are commercially available but several are proprietary and were internally developed by large corporations.

A basic challenge in establishing a distributed team structure is the cultivation of as much team spirit as possible. This goes hand in hand with the establishment of the communications infrastructure and the project management structure, and with the partitioning of the project. How then is team spirit promoted within the overall organization of the distributed project?

First, the basics of team building are valid for a distributed team. Virtually all team-related issues that arise in software projects require no less attention in a distributed project. However, some of them are more greatly stressed in a distributed environment, and these include team structure, issues of trust, sensitiveness to differences in culture, and motivation.

Team Structure

"The structure of the system mirrors the structure of the organization that designed it." This sounds like a statement made recently after observing the impact of distributed teams on system architecture and design. It wasn't. It was, in fact, written back in 1968 by M.E. Conway and is referred to as Conway's law [6].

As we have seen, the structure of the system determines the ease in which it can be partitioned; consequently, the relationship between the structure of the team and the structure of the software seems to conform to Conway's law. This is especially true in a distributed software project.

Conway continued that this relationship is a necessary consequence of the communications needs of the people doing the work. This, almost 35 years ago. It is therefore important for the project management office to fully appreciate that, in a distributed development environment, it may be the team that determines the structure of the software and not the reverse. Because the final product doesn't care how it was built (i.e., the software's users are not sympathetic to development constraints), project management must be able to fully evaluate the impact of the distributed team on the appearance of the final product.

The following list summarizes some of the considerations in establishing a distributed team structure:

- The availability of a team is not in itself sufficient justification for it filling a set of openings in the staffing of a project. Though diversity can be a blessing in disguise [8], there must be some reasonable level of cohesion between the teams in regard to project goal, project leadership, quality of work, cooperation, and exchange of information. If any team falls short in one of these areas, its contribution to the project could turn out to be less valuable in comparison to the danger of it disrupting the project.
- The lessons of Conway's law should be taken into account fully to ensure that the intended team structure does not create a poor system structure. The team structure must promote some acceptable rationale in the way the software will be architected and partitioned. Ideally, the team structure should be shaped according to loosely coupled work allocations.
- One of the main consequences of a distributed team structure (in comparison to a centralized team) is the additional overhead for team leadership. Due to the relative independence of the local teams, local leadership and team structure must be clearly defined, with accountability and authority spelled out both on the team level and on the individual level.
- Similarly, central leadership, team structure, and individual roles must be clearly defined, with unambiguous designation of central authority and responsibility.
- Support functions such as quality assurance and configuration management must also be clearly defined and well coordinated both on the local and central level.
- The available infrastructure should be included in the determination of the team

structure. This includes the accessibility of the software project elements as well as the site itself (e.g., the location designated for the project management office should be easily accessible from all other sites).

Finally, it is noteworthy to recall one of the conclusions of the Alcatel report [8] discussed earlier. The report strongly advised that DSD teams be *coherent* (teams should be structured based on work allocation of features/functionality), *collocated* (within each team, members should work close to each other), and *fully allocated* (100% dedication of each team member's time to the project).

Issues of Trust Between Sites

Local patriotism, tribalism, or clanning is a human trait as old as humanity itself. In fact, it is one of the constituents of team spirit that, as we have seen, is quite a desirable quality. Competition between teams can be positive, but it can have negative effects, too. This is especially true between distributed teams.

Personal contact is an important catalyst for promoting trust between people.[11] The distance between distributed teams reduces the opportunities for personal contact and thus increases the risk of a growing distrust.

[11]It is true that personal contact also has the potential for promoting distrust between people. The outcome is a question of what happens during the personal contact.

This is one of the areas where a natural solution exists: travel.

A recent report from Lucent [12] recounts the experiences in this area of one of the company's departments. Lucent's real-time embedded systems department is engaged in a number of cross-site collaborations, within its own product group, with other corporate divisions, and with other companies. The challenges included working with teams with different languages, cultures, and within different time zones.

The report reveals the initial lack of trust between two sites due to concerns about job security. This led to a reluctance to share information, and the sites did not appear to regard themselves as partners cooperating toward the same end. This manifested itself in what the report called "uncharitable" interpretations of behavior. For example, if someone said, "We can't make that change," it was often interpreted as, "We don't find it convenient to make that change."

Lucent reports that improvement occurred when members of the two teams began to meet. One of the developers was quoted as saying: "Things eased a lot when we met these people face to face, instead of over the telephones and e-mail. We worked more closely and resolved things much quicker." Working face to face let the developers establish common goals and purposes. Also, the time spent at the other site familiarized each party with the terminology and style of the other.

Clearly, there must be some limitations placed on travel, as it is expensive in terms of both time and cost. However, good working relationships can also be fostered by encouraging frequent contact by other means — not as a substitute for travel, but in addition to it. E-mail is a useful medium, but voice contact (the telephone) is better, and videoconferencing is better still.

In fact, e-mail can be a rather poor medium for developing relationships, as it suffers from two serious shortcomings: it has no secondary language support (such as voice intonation or body language), and it does not provide immediate acknowledgement of understanding. This means that there is a delay in the correction of a misunderstood e-mail message, which provides time for misunderstandings to fester. Phone calls are almost always better relationship-builders.

Issues of Culture

Earlier in this report, we discussed a misunderstanding between British and German developers due to the way each interpreted the wording of an instruction. People from different cultures do not just interpret things differently; they also react to them differently (behavioral differences). This is less of an issue for nationally

distributed teams (e.g., within the US) but more of an issue for globally distributed teams.

Carmel [4] employs a term called cultural distance. It is a measure of the degree of difference between the project center (usually from where the project is managed) and the remote locations. This distance usually manifests itself in one of two forms: organizational culture or national culture.

Organizational culture refers to the culture of systems development, such as the use of methodologies and project management practices. Carmel relates the case of a Korean customer who recently accused an Indian outsourcing company of becoming "too American" in that they were devoting too much attention to documentation and were too stringent about deadlines.

National culture includes an ethnic group's norms, values, and spoken language, often delineated by national boundaries of countries. American companies generally prefer to work in foreign locations where cultural distance is small — for example, in Ireland — or where the language barriers are minimal, such as India or the Philippines.

Carmel discusses an alleviation of cultural distance through the role of expatriates or well-traveled individuals functioning as cultural liaisons, in a somewhat similar manner to the Motorola 3G project liaison discussed earlier. However, Carmel sees the role as ongoing (with frequent travel back and forth), in contrast to the Motorola liaison function that focused mainly on the initial stages of the project.

Herbsleb also focuses on the issues of culture in global development [11]. He brings the example of diverse teams not understanding each other. He observes from his experiences at Lucent that when people are puzzled about how to respond to what they see as odd-sounding messages, they often just ignore them or make uncharitable attributions about the sender's intentions or character. At the extreme, this has the potential of mimicking the tower of Babylon syndrome when communication totally breaks down.

Of the two forms of cultural distance, organizational culture is often easier to deal with. Project management has more control over organization culture (deciding on policies, standards, methods, and tools) than over national culture (which basically boils down to using or not using a specific team). As we have seen, in organizational culture there are two basic approaches: institutionalizing standard processes and methods across all teams (the path chosen in the Alcatel report), or accepting the divergence of work methods and making them a factor in the scheme of project development (the path chosen in the Motorola report). How do you choose between the two?

The answer is related to the degree to which the remote teams are integrated into the overall organization. In a tightly knit corporate structure, the Alcatel approach is possible and may be preferable. In a more loosely integrated organization, the Motorola approach may be preferable. The best path can therefore be determined by observing the structure of the organization.

These are not rigid rules. Most organizations are neither completely inflexible nor completely flexible, and just have tendencies one way or the other (this is true of both Alcatel and Motorola). For each individual distributed project, it is the degree of integration between the candidate development sites that should determine the decision on the appropriate organizational culture.[12]

The distance produced by national culture can be more of a challenge, primarily because most people are familiar with only one culture: their own. However, being aware of the problem is often a major part of its solution. Thus, a useful tactic in global development is to sensitize all members of all teams to the issue of cultural distance, early in the project. Ensuring awareness of

[12]Unfortunately, there is not always a decision to be made. Organizational culture is often predetermined by the corporate culture and is thus relatively inflexible.

the issue, together with occasional visits between teams and the use of liaisons, can significantly moderate the problem.

Team Motivation

Asked by an interviewer what he thought was the most efficient weapon in his arsenal, a renowned general once responded: "Without a doubt, high motivation."

Studies have shown a vast difference between the productivity of software developers [3]. There are many factors that determine productivity, but motivation is key. In a given situation, the function of project management and the environment in which the project is developed can have significant impact on the motivation and the productivity of the teams.

It is project management's responsibility to promote the motivation of teams. According to Fredrick Herzberg, the two strongest motivators are, by far, recognition and sense of achievement [13]. These are two factors that can easily get lost in the complexity of a large, distributed organization.

In a distributed environment, these two factors apply equally on the team level (recognition of the team's efforts and the team's sense of achievement) and on the individual level (recognition of each team member's efforts and their sense of achievement).

It is the local team leader's responsibility to celebrate local dedication and achievement and to provide the local vision for the team. On the overall project level, it is the project manager's responsibility, together with the management team, to recognize team dedication and achievement and to provide the overall project vision.

This is one of the reasons why travel is doubly important for the managers of a DSD project. It reinforces a critical link between the remote team and the project leadership, and it provides a sense of belonging. This is the best opportunity for project leadership to publicly recognize the achievements of the remote teams and thus to strengthen their motivation.

Similar to the issue of culture, just the awareness of the difficulty in motivating a distributed team is in itself part of the solution. This is an issue that must receive attention from the earliest phases of the project.

ON A PERSONAL NOTE

My experiences with distributed software development at Motorola included many of the distinctions that we have been discussing. At the early stages of every project, there was always an undertone of concern about whether we should have preferred centralized development.

It was only later in the project — when the organization was defined, the teams were in place, and development was under way — that the questions faded. Why was this so? Possibly because the preference for centralization is ingrained in our human nature — we don't like to relinquish control. And, also, possibly because there was a basic fear of the unfamiliar.

Interestingly, a second distributed project with the same teams was easier to initiate than the first. It had been demonstrated that the strategy worked, and the teams had become more familiar with each other and, more importantly, with Motorola's Chicago-based personnel.

How well did teams from the US, India, and Israel work together? Not too well, at first. The cultures were vastly different. At one stage, a team from Cork, Ireland, was also involved in the distributed experience. The US team, coming from the company's main corporate center, felt that they owned the project, and, to a large extent, they did. The Israeli team, with an entrepreneurial spirit, also felt that *they* owned the project, and to a large extent, they didn't. The Indian team, with their well-defined Software Engineering Institute Level 5 process and their polite culture, had a tough time dealing with their less-formal US and Israeli colleagues. The Irish team probably melded best with the others, as on the one hand, there was nothing in their culture that strongly differentiated them from the other teams, and, on the other hand, they were responsible

for a well-defined, very loosely coupled part of the project.

As the project progressed (over a period of almost two years), the teams learned to work better together. This was also achieved through tools and infrastructure that improved as the project proceeded. Travel was a major factor in the development of cordial relationships between the teams, and, as more people met face to face, the frictions and misunderstandings faded.

Though there was a basic set of project processes and methodologies in place, each location had its own flavor, which, in the case of the Indian team, was substantially different to the other teams. Most, but not all, tools were similar. And none of the projects could have succeeded without Motorola's sophisticated worldwide corporate communications network and its intranet.

CONCLUSION

On both the national and global levels, the case for distributed software development is being proven daily. Most major development corporations, and many smaller ones, have adopted DSD as part of their strategy. As technology continues to provide better solutions and the global village becomes more and more of a reality, a vast reservoir of software developers becomes increasingly accessible.

We asked at the beginning of this report: "Will changes in global relationships and travel behavior affect the way software is being developed? Is a distributed software organization still a wise choice for today's corporations?" We have reviewed many of the factors that can help you answer these questions.

First, the need to capitalize on the national or even global resource pool is still valid, though possibly in a temporarily diminished form. Also, the business advantages, such as proximity to the market, are still valid. The need to develop good will with foreign governments is more valid today than ever. Therefore, it appears that distributed development remains a sound strategy for companies with the right type of project. These are the projects that have a clear DSD business advantage and are suitable to be divided between times (they are *partitionable*).

As for the long-term future of DSD, technology will continue to develop, infrastructure will continue to improve, and this will make travel less critical. The phenomenon of the global village will continue regardless of whatever happens to airline travel. And if airline travel eventually returns to its previous level, and many expect that it will, then it will only further reinforce this trend.

As the old saying goes, the proof is in the pudding. For distributed development, the best proof of its merit is in the fact that so many companies have adopted it as part of their strategy, linking remote global and national development locations together. Apparently, the global village is alive and well.

REFERENCES

1. Anthes, Gary H. "Software Development Goes Global," *Computerworld*, June 28, 2000 (www.cnn.com/2000/TECH/computing/06/28/global.development.idg/).

2. Battin, Robert D., Ron Crocker, Joe Kreidler, and K. Subramanian. "Leveraging Resources in Global Software Development," *IEEE Software*, March/April 2001.

3. Bennatan, E.M. *On Time Within Budget: Software Project Management Practices and Techniques*, 3rd edition. John Wiley & Sons, 2000.

4. Carmel, Erran, and Ritu Agarwal. "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Software*, March/April 2001.

5. Cugola, Gianpaulo, and Carlo Ghezzi. "Design and Implementation of PROSYT: A Distributed Process Support System," *IEEE 2nd Workshop on Coordinating Distributed Software Development Projects*, Stanford University, California, USA, 16-18 June 1999.

6. Conway, M.E. "How Do Committees Invent?" *Datamation*, April 1968, Vol. 14, No. 4, pp. 28-31.

7. Dossick, Stephen E., and Gail E. Kaiser. "Distributed Software Development with CHIME," *ICSE-99 Second Workshop on Software Engineering over the Internet*, University of Calgary, May 1999.

8. Ebert, Christof, and Philip De Neve. "Surviving Global Software Development," *IEEE Software*, March/April, 2001.

9. Handel, Mark, and Graham Wills. "TeamPortal: Providing Team Awareness on the Web," Product Development Collaboratory, Lucent Technologies, Bell Labs, 2001.

10. Herbsleb, James D., Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. "Distance, Dependencies, and Delay in Global Collaboration," *Proceedings of CSCW 2000*, December 2000.

11. Herbsleb, James D., and Deependra Moitra. "Global Software Development," *IEEE Software*, March/April 2001.

12. Herbsleb, James D., and Rebecca E. Grinter. "Architectures, Coordination, and Distance: Conway's Law and Beyond," *IEEE Software*, September/October 1999.

13. Herzberg, Fredrick I. "One More Time: How Do You Motivate Employees?" *Harvard Business Review*, September/October 1987.

14. *IEEE Software*, "Global Software Development," March/April 2001.

15. *IEEE 2nd Workshop on Coordinating Distributed Software Development Projects*, Stanford University, California, 1999. http://wwwagr.informatik.uni-kl.de/~koetting/WETICE99/.

16. Kaiser, Gail, Christopher Vaill, and Stephen Dossick. "A Workgroup Model for Smart Pushing and Pulling," *IEEE Eighth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1999.

17. Kudo, Yutaka, Shinobu Koizumu, Osamu Ohno, Hiroshi Kawabe, and Yukari Furuhata. "Problem Management System for Distributed Software Development," *ICSE-98 Workshop on Software Engineering over the Internet*, University of Calgary, 1998.

18. Kuo, T.W. (translator/editor). *Sun Tzu: Manual for War*, ATLI Press, 1989.

19. Mockus, Audris, and David M. Weiss. "Globalization by Chunking: A Quantitative Approach," *IEEE Software*, March/April 2001.

20. Murer Tobia, Michael L. Van De Vanter. "Replacing Copies with Connections: Managing Software Across the Virtual Organization," *IEEE 2nd Workshop on Coordinating Distributed Software Development Projects*, Stanford University, California, 16-18 June 1999.

21. O'Connor, Rory, and John Jenkins. "Using Agents for Distributed Software Project Management," *IEEE 2nd Workshop on Coordinating Distributed Software Development Projects*, 16-18 June 1999, Stanford University, California.

22. Repenning, Alexander, Andri Ioannidou, Michele Payton, Wenming Ye, and Jeremy Roschelle. "Using Components for Rapid Distributed Software Development," *IEEE Software*, March/April 2001.

23. Rhoads, C. "Germany Faces Storm over Tech Staffing," *Wall Street Journal*, March 7, 2000.

24. Sujaputra, Raymond, and Piyush Maheshwari. "A Distributed Software Project Management Tool," *IEEE 2nd Workshop on Coordinating Distributed Software Development Projects*, Stanford University, California, 16-18 June 1999.

25. West, Lawrence A., and Walter A. Bogumil. "Immigration and the Global IT Work Force," *Communications of the ACM*, July 2001, Vol. 44, No. 7.

26. Yourdon, Edward. *The Decline and Fall of the American Programmer*. Prentice Hall, 1993.

27. "Offshore's New Horizons," *Global Technology Business*, March 2000, Vol. 3, No. 3, pp. 12-15.

FURTHER READING

Agarwal, Ritu, and Thomas W. Ferratt. "Crafting an HR Strategy to Meet the Need for IT Workers," *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Altmann, Josef, and Heinz Dobler. "Organizational Aspects of Distributed Software Development," *Proceedings of the 6th Interdisciplinary Information Management Talks*, Zadov, CZ, 1998, Trauner Universitätsverlag, pp. 111-120.

Altmann, Josef, and Gustav Pomberger. "Cooperative Software Development: Concepts, Model and Tools," *Proceedings of TOOLS-30 Conference*, IEEE Society Press, 1999.

Baentsch, Michael, Georg Molter, and Peter Sturm. "WebMake, Integrating Distributed Software Development in a Structure-Enhanced Web," *The Third International World-Wide Web Conference: Technology, Tools and Applications*, Darmstadt, Germany, 10-14 April 1995.

Benford, Steve, Chris Greenhalgh, Tom Rodden, and James Pycock. "Collaborative Virtual Environments, *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Boyer, David G., Mauricio Cortes, James Herbsleb, and Mark J. Handel. "Virtual Community Awareness," Product Development Collaboratory, Lucent Technologies, Bell Labs, 2001.

Boyer, David G., Mauricio Cortes, and Mark J. Handel. "Presence Awareness Tools for Virtual Enterprises," Product Development Collaboratory, Lucent Technologies, Bell Labs, 2001.

Dargan, P.A. "The Ideal Collaborative Environment," *Crosstalk: The Journal of Defense Software Engineering*, April 2001.

Embar, Chellam. "The State of Software Development in India," *Crosstalk: The Journal of Defense Software Engineering*, August 2001.

Favela, Jesús, and Feniosky Peña-Mora. "An Experience in Collaborative Software Engineering Education," *IEEE Software*, March/April 2001.

Finholt, Thomas A., Elena Rocco, Danial Bree, Nishant Janin, and James D. Herbsleb. "NotMeeting: A Field Trial of NetMeeting in a Geographical Distributed Organization," Product Development Collaboratory, Lucent Technologies, Bell Labs, 2001.

Govindarajan, Vijay, and Anil K. Gupta. "Building an Effective Global Business Team," *MIT Sloan Management Review*, Summer 2001.

Gupta, Phalguni. "Growth Scenario of IT Industries in India," *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Heeks, Richard, S. Krishna, Brian Nicholson, and Sundeep Sahay. "Synching or Sinking: Global Software Outsourcing Relationships," *IEEE Software*, March/April 2001.

Lai, Vincent S. "Interorganizational Communication with Intranets," *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Mander, Keith. "The Decline and Fall of the American Programmer?" *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Schuff David, and Robert St. Louis. "Centralization vs. Decentralization of Application Software," *Communications of the ACM*, June 2001, Vol. 44, No. 6.

Tobias, Murer, and Michael L. Van De Vanter. "Replacing Copies with Connections: Managing Software Across the Virtual Organization," *IEEE Eighth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Stanford University, California, 16-18 June 1999.

Trauth, Eileen M. "Mapping Information-Sector Work to the Work Force," *Communications of the ACM*, July 2001, Vol. 44, No. 7.

Whitehead, Jim. "The Future of Distributed Software Development on the Internet," *Web Techniques*, October 1999.

advisory service

at a glance

Agile Project Management Advisory Service

Today's projects require a new perspective. Instead of best practices, you've got to consider the next practices. The **Agile Project Management Advisory Service** is designed to help you implement a balance of practices that support innovation, discipline, and adaptability. You'll discover which of the major Agile Methodologies are right for your organization, and you'll get a platform on which you can begin to create project management methodologies that support your enterprise.

As a client, you will receive:

- Monthly *Executive Reports* and *Summaries* — providing strategic advice from today's project management experts
- Twice-monthly *Executive Updates* — with analysis of exclusive project data
- Weekly *E-Mail Advisors* by Practice Director Jim Highsmith

Topics Covered in the *Executive Reports* and *Updates* include:

- The effects of Agile Methodologies on system architecture, user interface design, database design, and software design
- Best strategies for executing second-generation e-projects
- Trends in length, team size, and project type for agile projects
- How to remain agile while using rigorous methods
- Proven ways to improve organizational decisionmaking
- Performing risk assessments using object-oriented metrics
- Successful strategies for managing distributed software teams
- What project managers need to know to leverage requirements
- Optimizing software inspection practices to maximize the return on investment

Senior Consultants

Cutter Consortium has assembled the world's preeminent IT consultants — a distinguished group of internationally recognized experts committed to delivering top-level, critical, objective advice.

Each Consortium practice area features a team of Senior Consultants whose credentials are unmatched by any other service provider.

The Senior Consultants who write for the **Agile Project Management Advisory Service** — and are available for inhouse workshops and custom consulting engagements — include:

- Jim Highsmith, Director
- Scott W. Ambler
- Sam Bayer
- E.M. Bennatan
- Tom Bragg
- Robert N. Charette
- Alistair Cockburn
- Doug DeCarlo
- Tom DeMarco
- Ian Hayes
- Ron Jeffries
- Brian Lawrence
- Tim Lister
- Michael C. Mah
- Lynne Nix
- Ken Orr
- Chris Pickering
- Roger Pressman
- Ram Reddy
- James Robertson
- Suzanne Robertson
- Alexandre Rodrigues
- Johanna Rothman
- Lou Russell
- Rob Thomsett
- Colin Tully
- Richard Zultner

For More Information:

To learn more about Cutter Consortium's **Agile Project Management Advisory Service**, contact David Gijsbers by phone at +1 781 641 5104, by fax at +1 781 648 1950, or send e-mail to dgijsbers@cutter.com.

About Cutter Consortium:

Cutter Consortium offers high-level advisory services, on-site assessments, consulting, and training to help organizations forge solutions to the IT challenges they face. The Consortium is dedicated to providing completely objective information and to customizing its services to meet each client's needs.

Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474, USA
Phone: +1 781 648 8700; Fax: +1 781 648 1950
Web site: www.cutter.com/consortium/

CUTTER CONSORTIUM

ABOUT THE CUTTER CONSORTIUM

Cutter Consortium's mission is to help senior executives leverage technology for competitive advantage and business success.

Cutter's offerings are entirely unique in the research/analyst industry sector because they are produced and provided by the top thinkers in IT today — a distinguished group of internationally recognized experts committed to providing high-level, critical advice and guidance. These experts provide all of Cutter's written deliverables and perform all of the consulting and training assignments.

Cutter Consortium's products and services include: high-level advisory/research services, online and print publications, benchmarking metrics, management and technical consulting, and advanced training. The content is aimed at both a technical and business audience with an emphasis on strategic processes and thinking.

An independent, privately held entity that has no allegiance or connections to any computer vendors, Cutter has a well-earned reputation for its objectivity. Cutter's more than 5,300 clients include CIOs, CEOs, CFOs, and senior IT managers in *Fortune* 500 companies and other businesses, national and state government, and universities around the world.

As a smaller information provider, the Consortium customizes its services to meet each client's individual needs and ensure them *access to the experts*.

FOR MORE INFORMATION

To learn more about the Cutter Consortium, call 800 964 5118 (toll-free in North America) or +1 781 648 8700, send e-mail to sales@cutter.com, or visit the Cutter Consortium Web site: www.cutter.com.

CUTTER CONSORTIUM
37 Broadway, Suite 1, Arlington, MA 02474-5552, USA
Tel: +1 781 648 8700; Fax: +1 781 648 8707
sales@cutter.com
www.cutter.com