Surviving Global Software Development

Christof Ebert and Philip De Neve, Alcatel

Software development involves teamwork and a lot of communication. It seems rational to put all your engineers in one place, encourage them to share objectives, and let the project run. Why use distributed sites when it’s easier to work in one location without the overhead of remote communication and planning? How is it possible to survive (and succeed with) globally dispersed projects?

Working in a global context has its advantages, but it also has drawbacks. On the plus side, you gain time-zone effectiveness and reduced cost in various countries. However, working on a globally distributed project means operating costs for planning and managing people, along with language and cultural barriers. It also creates jealousy as the more expensive engineers (who are afraid of losing their jobs) are forced to train their much cheaper counterparts.

In this case study, we try to summarize experiences and share best practices from projects of different types and sizes that involve several locations on different continents and in many cultures.

Case study setting

Alcatel is a global telecommunications supplier with a multitude of development projects that typically involve several countries. We will focus in this article on the company’s Switching and Routing business division. Software development in this business division is handled in a central R&D group of several thousand software engineers who are distributed throughout the world in more than 15 development centers in Europe, the US, Asia, and Australia. Strong functional and project organizations—interacting in a classic matrix—facilitate both project focus and long-term skill and technology evolution.

Telecommunication suppliers have worked in a global context for years. The primary drivers in the past were the need to be locally present for customization and after-sales service and to show local customers how many new jobs were created, which in turn could justify more contracts. A second source for internationally dispersed software development is the growing number of acquisitions and mergers, which add new markets, products, engineers, and creativity to the existing team. A third reason for starting new development activities in countries where neither the market nor the acquisitions would justify such
evolution is that it’s often impossible to hire young engineers with the necessary skills at a reasonable cost at the existing site. The answer in such cases is to start business in areas such as Eastern Europe or India, which Alcatel has done.

Although development centers have operated autonomously in the past, aligned R&D teams have replaced such local activities with globally managed product development lines. This effectively avoids overhead in terms of redundant skills and resource buffers in various locations. Having no option to act locally, product development lines organize projects to achieve the best possible efficiency—by optimizing the trade-offs between temporarily collocating teams, reducing overheads, and having the right skills in due time.

This, of course, directly relates to the organization’s overall profit and loss layout, which is adjusted to have R&D provide engineering results internally toward business units. Decisions about work allocation are R&D’s full responsibility. The parameters at which the department is measured are sales and R&D’s cost, which let it internally manage quality, productivity, and lead-time. If, for instance, a specific location is too expensive, it’s R&D’s decision to move engineers toward a higher productivity location.

This study describes projects within Alcatel’s Switching and Routing Business Division. The product spectrum ranges from the proprietary S12 switching system to Corba/Java middleware and front ends. Alcatel is registered for the ISO 9001 standard. The majority of development locations are ranked on CMM Level 2; few are on CMM Level 3. In terms of effort and cost, the software’s share is increasing continuously and currently occupies 80 percent to 90 percent of R&D’s budget. Our focus in this study is on switching projects because they are typically developed in at least two or three sites, sometimes on several continents. The projects vary in size between a few person-years and several hundred person-years, depending on how much new development is involved.

Lessons learned from global development

There are many ways to organize and manage global development. Let’s examine some of Alcatel’s experiences. We elaborate on several practices, but due to space restrictions, we can’t cover all of them. Nevertheless, we welcome email regarding specific questions.

Organization and allocation

Work organization highly impacts globally distributed software development. Although some research recommends building virtual teams, we strongly advise building coherent and collocated teams of fully allocated engineers. Coherence means splitting the work during development according to feature content and assembling a team that 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. Full allocation implies that engineers working on a project should not be distracted by different tasks in other projects.

At their kick-off, projects are already split into pieces of coherent functionality that will grow incrementally. Functional entities are allocated to development teams, which are often based in different locations. Architecture decisions, decision reviews at major milestones, and tests are done in one location. Experts from countries with minor contributions will relocate for however long the team needs. This allows effective project management, independent of how the project is globally allocated.

Team members must communicate whenever necessary to make the team efficient.
To clarify different organizational needs, we distinguish different roles. The key roles that facilitate allocation in global development include

- **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 roles are then allocated to various development teams, which constitute a project. Our objectives for improved allocation were as follows:

- Allocate the majority of people contributing to the project (the Engineering role) to almost full-time. This is measured with a scatter factor that relates the people contributing to a task to the total effort of the task. This scatter factor should be a maximum of 1.5 (1.5 engineers on average contribute to a one-engineer task), with a clear tendency to further reduce.
- Ensure reliable allocation, with agreements on start and end dates. Having time fixed means that with clear quality and cost targets, the only variable factor is content. Content thus serves as a buffer to mitigate unexpected overruns and is facilitated by incremental development and continuous build.\(^4\text{--}^6\)
- Ensure team collocation, even if the project is distributed across sites. Teams that are assigned across several locations face many challenges that could impact their ability to work as a team.
- Distinguish development (new functionality) from maintenance activities (such as defect correction). They should be organized as separate projects.

Such changes in allocation are a big cultural change—the clear target is to replace isolated expertise with skill-broadening tasks and effective teamwork. This implies a clear individual responsibility for overall project results. Such simple yet effective rules demand a sufficiently detailed project plan at a project’s start that breaks down resource needs into skills and duration and provides a feature development breakdown into teams and increments. Alcatel’s Switching and Routing business division institutionalized these changes starting in 1998.

Enriching jobs also requires more training and coaching. We saw, however, that coaching pays off the most. Looking only at the engineering cost of nonquality—the time needed to detect and correct defects—we found that projects with intensive coaching (1 to 2 percent of accumulated phase effort) reduced the cost of nonquality in the phase by over 20 percent.\(^3\) Break-even is typically reached at around 5 percent coaching effort. This means there are natural limits when involving too many inexperienced engineers.

**Concurrent engineering**

Previously, engineers lost sight of how their own contributions affected the overall project. Several project postmortems indicated that activities and work product quality were extremely isolated. The effect was that whenever we tried to build the complete product or iteration, we required huge overhead to bring the pieces together. This held true for individual work products as well.

Without a product perspective, engineers tended to handle work products inefficiently. Results were moved forward to the next link in the chain, and the cost of nonquality (and the number of delays) accumulated. For example, inspections typically didn’t follow the defined process, which involves checkers, inspection leaders, and a maximum reading speed. Instead of applying reasonable exit criteria, many inspections were considered finished when they reached their respective milestone dates—and before continuing the defect detection with the next and more expensive activity. Test was conducted with a rather static set of test cases that were not dynamically filtered and adjusted to reliability growth models.

The root causes were obvious but so embedded in the culture that the situation required a complete process reengineering to facilitate global development at a competitive cost.

The major changes that we implemented involved concurrent engineering and team-
work. For instance, we assembled cross-functional teams at the beginning of the project—before project kick-off, we called in an expert team to ensure a complete impact analysis, which is a prerequisite to defining increments. Concurrent engineering means that, for example, a tester is also part of such a team—experience shows that designers and testers look at the same problem very differently. Testability is only ensured with a focus on test strategy and the potential impacts of design decisions made during the project’s initial phases.

Teamwork was reinforced to the degree that a team had sole responsibility for realizing a set of functionally related customer requirements. A designer would no longer leave the team when her work product was coded—she would stay to test the product in the context of changes provided by other team members. Feature orientation clearly dominates artificial architectural splits. The team’s targets are based on project targets, and all the team members share them. The team follows up the targets on the basis of delivered value, such as feature content. Periodic reviews of team progress with the project lead are necessary to follow up and help when case risks arise that the team cannot mitigate.

We evaluated the effects of this reengineered process carefully over the past two years. As a result, we see two effects. We could reduce response time and overall cycle time with earlier defect detection (see Figure 1). The engineering cost of nonquality in the overall project and field defects could be reduced significantly due to earlier defect detection. We tested both hypotheses on a set of 68 projects over the past four years (before and after the change). As a result, we can accept with a significance level of more than 95 percent in a T-test that the change toward feature-oriented development impacts both cycle time and cost positively.

**Product line concept**

Global products with different local markets both stimulate and hinder true global software development. The broad flexibility of modern software can easily result in variants and local evolutions that make it impossible to manage synchronization of any kind of maintenance activity, be it corrective or additive. The absence of clear linkages to business value invites gold plating—that is, implementing functions that might be rarely used or adding excessive functions that are not necessary to attain the desired business results.

This trend had to stop, and the approach was a rigid introduction of a coherent worldwide product line concept. The product line concept is based on few core releases that are further customized according to specific market requirements around the world. The structuring of a system into product families allows the sharing of design effort within the family and, as such, counters the impact of ever-growing complexity.

Based on a mapping of customer requirements to architectural units (such as modules, databases, subsystems, and production tools), we achieved a clustering of activities that allowed for splitting activities into three parts:

1. Small independent architectural units that we could separate and leave out from any customization. Typically, they are subject to moving into separate servers, and development is collocated at one place.
2. Big chunks that any project would impact and thus need a global focus to facilitate simple customization (such as different signaling types captured with generic protocol descriptions and translation mechanisms). Development happens in multiskilled teams, and these skills are replicated in almost all locations.
3. Market- or customer-specific functional clusters based on the requirement analysis and ultimately forming the project team responsible for a customer project. This type of requirement must be the exception and asks for a dedicated pricing strategy because it creates the most overhead.
Such separation of architectural units is also the necessary precondition for splitting a global project into teams that we can individually collocate.

Change management

Improving the allocation of engineers to only one project naturally means that there is no global owner of a specific work product across projects. Instead, many developers in different places simultaneously share the responsibility of enhancing functionality within one product. Often a distinct work product (or a file with source code) is replicated as variants are concurrently updated and synchronized to allow the centralized and global evolution of distinct functionality.6

Effective tools and work environments are thus the glue to successful global software development. Most commercial tools cause problems when used in sites around the globe. Almost no tool seamlessly synchronizes or backs up database contents without disturbing engineers that are logged on 24 hours a day, seven days a week. Performance rapidly decreases when multisite use is involved, due to heterogeneous server and network infrastructures.

Managing corrections perfectly illustrates the observed challenges and solutions we implemented. Defects impact globally distributed product line architectures, and the risk is high that the same defects will occur again and again. The product line concept implies that engineering teams must align and synchronize feature roadmaps and deliveries of both new and changed (or corrected) functionality. However, synchronization of deliveries adds complexity to the development process. Engineers cannot easily copy corrections from one code branch to the other because they impact ongoing development. Therefore, effective synchronization of the individual corrections involves global visibility of all defects, impacts of defects, correction availability, and evaluation of correction impacts.

To facilitate easier communication of appropriate corrections, we introduced a new synchronization mechanism into our worldwide defect database. Based on the detected failure and the origination fault, a list of files in different projects is automatically prepopulated by telling which variants of a given file the engineer needs to correct. Although this is rather simple with a parent and variant tree on the macroscopic level, careful manual analysis is needed due to localized small changes on the code procedure and database content level. The change management system then automatically triggers those variants (within customization projects). Depending on the project manager’s trade-off analysis of failure risk and stability impacts, the developer responsible for the specific customization would correct these defects.

This approach immediately helped us focus on major field problems and ensure that they would be avoided in other markets. However, it also showed the applied product line roadmap’s cost. Too many variants create overhead. Obviously, variants must be aligned to allow for better synchronization of contents (both new functionality and corrections) while still preserving the desired functional flavors necessary in a specific market. Global development thus impacts product line architecture heavily toward fewer and simpler threads of design variants.

Incremental development

Although developers have known about and applied incremental development and related life-cycle models for many years,1,4 it’s not so obvious how to implement an entirely incremental approach to a legacy architecture that is primarily driven by heavily interacting subsystems instead of small add-on functionality or independent components. In such architectures, which are typical for legacy systems, development during top-level (or architectural) design not only agrees on interfaces and impacts on various subsystems, but also on a work split that aligns with subsystems. The clash comes when these subsystems should be integrated with all new functionality. Such processes are characterized with extremely long integration cycles that don’t show any measurable progress in terms of feature content.

The changes we introduced at Alcatel to achieve real incremental development were to

- Analyze requirements from the beginning in view of how they could be clustered to related functionality.
- Analyze the context (interfaces and data structures that are common for several
modules) impacts of all increments up front before starting development. The elaboration phase is critical to make real incremental development and a stable test line feasible. Obviously, not all context impacts can be addressed immediately without extending the elaboration phase toward an unacceptable range. It is thus necessary to measure context stability and follow up with root cause analysis why certain context impacts were overseen. As a target, the elaboration should not take longer than one third of total elapse time. The remainder of the project duration is related to development activities.

- Provide a project plan that is based on these sets of clustered customer requirements and allocates development teams to each set. Depending on the impact of the increments, they can be delivered to the test line more or less frequently. For instance, if a context impact is detected too late, a new production cycle is necessary that takes more effort and lead-time than regular asynchronous increments of additional code within the originally defined context.

- Develop each increment within one dedicated team, although a team might be assigned to several increments in a row. Increments must be completed until the end of the unit and feature integration test to avoid discovering that various components cannot be accepted to the test line. A key criterion for the quality of increments is that they don’t break the build.

- Base the progress tracking of development and test primarily on the integration and testing of single customer requirements. This gives visibility on real progress because a requirement can only be checked off if it is successfully integrated in the test line. Traceability is improved because each customer requirement links to the related work products.

- Extensively feature test increments by using the independent test line.

Increments toward a stable build proved one of the key success factors in global development. We realized that a project’s cycle time is heavily impacted by whether continuous build is globally applied. Figure 2 shows the results from an overview on projects conducted in the past three years. Especially with the described reengineering efforts, we could continuously reduce cycle time.

**Culture at work**

In a company like Alcatel, handling change effectively is mandatory to remain a key player in the industry. Without change, a company will stagnate and eventually disappear. What is crucial in a global development organization is that all development locations working in one product line use the same processes, methodology, and terminology even when changes occur. This looks obvious, but in an organization with several thousand engineers, separated thousands of kilometers from each other, having different languages and cultures, it is quite a challenge.

To convince top management of that need is not so difficult. The real challenge is to spread the awareness, communication, and knowledge to all levels in real time—from the different levels of management to the many engineers who want to have simple yet efficient processes. We thus focused on integrated workflow and online documentation of process and project contents. Today each team and project has globally accessible hompages that allow easy browsing of all internal work products. Work products and roles allow direct navi-
gating to process descriptions and related workflow tools (see Figure 3).

Key factors to continuous change are not new challenging tools or processes, but the ability of people (management and developers) to be open to change. This attitude is an integral part of the company’s culture. Large efforts have helped create the right attitude and spirit and a company culture based on common goals. One of the major milestones in creating such a common culture and thus facilitating our status as a global company was to choose English as the common language within Alcatel (not so obvious for a company based in France). English is mandatory, and language classes are provided in most non-native speaking countries to leverage skills.

A common syntactical language does not necessarily mean the same semantics and pragmatics, however. This is obvious with commitments and negotiations in different cultures. We still face some barriers, especially with countries that clearly follow their own culture. One way to improve is a heavy exchange of teams and management to face and live within the different cultures and thus gradually build mutual understanding.

Different development locations and cultures react differently to change. The longer a location works with similar tools and processes in a culture well protected from the outside, the more difficult the change is. This is not surprising, but we experienced direct proof of it. During the past three years, we opened two new and fast-growing development centers, one in Eastern Europe and one in India. By using the Greenfield approach—starting over—for these two centers, we successfully introduced our processes without friction. During the past three years, methodologies and processes changed several times without difficulty in these new centers. After three years in operation, one of these centers operates in a CMM Level 3 mode and plays a piloting role with Alcatel to become a CMM Level 5 location.

Process maturity positively impacts global development. CMM Level 2 is an absolute minimum to achieve sound project management (which is a prerequisite for distributed development). However, only a standardized organizational process framework allows a seamless integration of different development centers and ensures that interfaces for various work products—including workflow management—facilitate an exchange of results and shared contributions.

Such continuous learning impacts motivation and thus reduces turnover rates. Keeping turnover rates low is clearly a key objective in dealing with legacy software, otherwise nobody has the necessary long-term view.

Managing global software development is not easy and risks lowering overall productivity. Still, the positive impacts should not be forgotten. A major positive effect is innovation. Engineers with all types of cultural backgrounds actively participate to continuously improve the product, innovate new products, and make processes more effective. Achievements are substantial if engineers of entirely different educations and cultures try to solve problems. Best practices can be shared, and sometimes small changes within the global development community can have big positive effects. One example in Alcatel was the introduction of quiet rooms throughout the world to use for code reviews.

Here are some of those best practices, which we identified over the past few years as clearly supporting global software development:

- Agree and communicate at project start the respective project targets, such as quality, milestones, content, or resource allocation.
- Ensure that commitments exist in written and controlled form.
- Have one project leader who is fully responsible for achieving project targets, and assign her a project management team that represents the major cultures within the project.
- Within each project, follow up continuously on the top 10 risks, which in a global project are typically less technical than organizational.
- Define at a project’s beginning which teams are involved and what they will do in each location.
- Set up a project homepage that summarizes project content, progress metrics,
planning information, and team-specific information.

- Provide an interactive process model based on accepted best practices that allows tailoring processes for the specific needs of a project or even team.
- Rigorously enforce within a product line using the agreed standard process that relates to a CMM Level 3 organization pattern.
- Provide sufficient communication means, such as videoconferencing or shared workspaces and global software libraries.
- Rigorously enforce CM and build management rules (such as branching, merging, and synchronization frequency) and provide the necessary tool support.
- Rotate management across locations and cultures to create the necessary awareness for cultural diversity and how to cope with it.
- Set up mixed teams from different countries to integrate individual cultural background toward a corporate and project-oriented spirit.
- Make teams fully accountable and responsible for their results.

Global software development is not the target per se, but rather the result of a conscious business-oriented trade-off. The guiding principles are to optimize the cost of engineering, while still achieving the best feasible integration of all R&D centers worldwide.

These needs must be carefully balanced with additional costs that might occur only at a later point. This includes staff turnover rates, which in other countries might be higher than in Europe; cost overheads related to traveling, relocation, communication, or redundant development and test equipment; unavailability of dedicated tools that allow for globally distributed tools and work environments; impacts on the learning curve, which slows down with more locations involved; cultural differences that can impact work climate; insufficient language skills; different legal constraints related to work time, organization, or participation of unions; and building up redundant skills and resource buffers to be prepared for collocated teams and unforeseen maintenance activities. We faced all these obstacles, and even the best training cannot substitute for extremely cooperative engineers and highly effective management.

To be successful in a global market, a company should manage the risks of global software development, but can use the positive aspects as input to shape the development process in detail and the culture in general. History has shown us time and again that mixing blood is the best thing that can be done in the path of evolution. Globalization achieves exactly that.

**References**


**About the Authors**

**Christof Ebert** is Alcatel’s director of software coordination and process improvement. He is also IEEE Software’s associate editor in chief for requirements engineering. His research interests include requirements engineering, software metrics, software quality assurance, real-time software development, and CASE support for such activities. He studied electrical engineering and software engineering at the University of Stuttgart and Kansas State University. He received his PhD from the University of Stuttgart. He is a member of the IEEE, GI, and the Alpha Lambda Delta honor society. Contact him at Alcatel, Fr.-Wellesplein 1, B-2018 Antwerpen, Belgium; christof.ebert@alcatel.be.

**Philip De Neve** is the design center manager of Alcatel’s Chennai Development Center in India. His interests include human responses in multicultural software organizations, real-time software development, and artificial intelligence. He studied electrical engineering in the KUOGhent. Contact him at Alcatel SKO, 94/95 Thiruvika Industrial Estate, Guindy, 600 032 Chennai, India; philip.de_neve@alcatel.be.