Experience with Global Software Architecture Design & Development

Michael L. Greenberg
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
+1 609 734 3347
mgreenberg@scr.siemens.com

Daniel J. Paulish
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
+1 609 734 6579
dpaulish@scr.siemens.com

Peter Hess
Siemens Metering
Feldstrasse 1
CH-6301 Zug, Switzerland
+41 41 724 4374
Peter.Hess@zug1.siemens.ch

Abstract: In this paper, we describe our experience designing and developing a system---for acquiring and processing data from electric, gas, and water meters---among four development sites located in Switzerland, Germany, and the U.S. Some of the techniques we used for project planning and management are described. We observed that a number of multicultural variables affect the overall performance of the development team. Based on our experience, a set of recommendations is given for managing global software development teams. Although we collectively felt that a single-site project team is likely to be more efficient than a multi-site team, the diversity of ideas and skills offered by a multi-site team resulted in a product line architecture that is flexible, modifiable, and adaptable to different market requirements.

Keywords: Software architecture, project planning, high-level design, global analysis, global development, multicultural variables.

Background

In 1999, we initiated the design of a software architecture (called Athena) for meter data acquisition and processing central stations. A product line architecture was required since multiple application packages were envisioned in order to support the future business needs. In addition, existing product architectures were being evaluated for supporting the envisioned application packages.

Historically, these existing product platforms were often modified and tailored towards customer specific requirements within project engineering centers located throughout the world. As a result, proliferation of platforms and products was occurring so that it was difficult to bring new features back into the baseline product. Furthermore, there was a strong trend towards deregulation in the power distribution industry that was believed would cause changes in requirements as well as new business opportunities.

The headquarters for the Siemens Metering business is located in Zug, Switzerland. Siemens Metering was formed by the merger of the metering businesses from Siemens Power Distribution and Transmission located in Nuremberg, Germany and Landis and Gyr, located in Zug. In addition to these two sites, Athena team members were also located in the U.S. in Princeton, New Jersey and Charlotte, North Carolina. Thus, location, country, language, ethnicity, and corporate culture created a high degree of diversity within the development team.

A meter data processing central station collects data from electric, gas, and water meters connected via telephone lines and other media. The meter data is stored and processed. The type of processing depends on the type of consumer using the resource and its contractual agreement with the supplier of the resource. Thus, many different types of software applications must run on the Athena platform. For example, the processing software for commercial consumers using electricity would be quite different than for residential water consumers. Furthermore, control functions are provided. For example, commands are sent out to high power utilization equipment when load must be shed during high demand periods. Modern electric meters can provide measurement data as often as once a minute, which allows *tariff agreements* to be specified between energy consumers and providers such that the price of energy varies depending on the time of day, week, and year and also the amount of energy used. Athena performs calculations on the meter data, the results of which are typically sent to a utility's billing system.

Five initial application packages were planned for implementation on the Athena platform. These ranged from meter data acquisition through calculation and reporting to load management control and payment systems implementation. The application requirements were quite diverse and required a high degree of flexibility from the product line architecture design. The first two applications were directly supported by the architecture since their development was initiated shortly after the architecture was designed and reviewed. At the time of the architecture design, marketing requirements specifications existed for the first two applications but not for the other three applications.

Why Global Development?

We decided very early during the high-level architecture design that a global development was necessary. This decision was a result of an analysis technique called *global analysis* [1], in which we systematically looked at influencing factors and the resulting design strategies. The analysis made it visible that it would be difficult to implement Athena at any single location, since no site had a full set of necessary development skills. Unfortunately, global analysis and global development use the word *global* in different ways. Global analysis is a high-level design technique for determining design and project strategies.

Global Analysis

The global analysis considered factors that would influence the design of Athena, grouped into categories of organizational, technological, and product influences. Analysis of the influencing factors resulted in a set of design strategies that have been used to guide both the product line architecture design and implementation of the application packages. The organizational influencing factors very much pushed us towards a design strategy of using a global development team for the Athena design and implementation.

Organizational Influencing Factors

Organizational factors such as schedule and budget apply only to the product currently being designed. Other organizational factors such as organizational attitudes, culture, development site(s) location, and software development process can impact every product developed by an organization.

An example of an organizational influencing factor for Athena was that the technical skills necessary to implement the application packages were in short supply since prior products had been Unix-based with local user interfaces and marketing required new products to be Windows-based with web-based user interfaces. The resulting strategy to address this influence was to bootstrap and exploit expertise located at multiple development sites, to invest in training courses early in the development and to make use of consultants. Also, a second level of design specification documentation was developed at a lower level than the high-level design. This system design specification concentrated on describing the interfaces between major subsystems of the architecture, so that it was easier to parcel out a subsystem development to a remote software engineering site.

Another organizational factor was that company management wanted to get the product to the market as quickly as possible. Since the market was rapidly changing, it was viewed as critical to quickly get some limited features of the product to potential users so that their feedback could be solicited. Our strategy to address this factor was to develop the product incrementally such that scheduled release dates were met even if some features were missing from the release. Thus for Athena, project schedule took priority over functionality. A build plan was developed for each engineering release identifying the sequence for adding functionality. The project functionality and schedule were baselined after each engineering release. We found that a 6-8 week development cycle for each engineering release worked well for the development team to provide a reasonable set of features that could be tested and evaluated. With such a strong emphasis on speed to market, we needed to acquire development resources wherever we could find them, which resulted in a development team located in four locations.

Design Strategies

Design strategies determine the priorities and constraints of the architecture and help identify potential risks associated with the implementation of the software system. As a result of the Athena global analysis, twenty-four design strategies were identified that we believed could address the influencing factors. From these twenty-four design strategies, six major conclusions were derived and used as guiding principles for the Athena architecture design and resulting development. One of the major conclusions resulting from the global analysis was to set up a global development team. This put constraints on the design so that components could be more easily distributed for development at multiple locations, and the development environment and tooling were set up for multiple locations.

Architecture Design

A high-level design team was formed originally consisting of five engineers with a mixture of domain and architecture design expertise. A chief architect was appointed from Princeton, and the team began the architecture design in Zug by analyzing the marketing requirements, developing the conceptual architecture, investigating applicable development technologies, and doing some simple prototyping.

The Athena high-level software architecture was designed using our four views approach - conceptual, module, execution, and code [1]. This design approach decreases the complexity of the implementation and improves understanding, reuse, and reconfiguration (Figure 1). The architecture was reviewed using the Architecture Tradeoff Analysis Method (ATAM^{SM}) [2], which is a structured analysis technique that evaluates a software architecture with respect to multiple desired qualities (e.g., survivability, modifiability, performance, security), developed at the Software Engineering Institute (SEI).

We also analyzed the architecture to provide inputs to the project's development cost and schedule estimation [3]. With our architecture-centered software project planning approach (ACSPP) (Figure 2), the software architecture design document is a primary input to the top-down and bottom-up project schedule and effort planning processes. From this we derived an incremental development build plan such that the product functionality is built up feature by feature within engineering releases that are system tested until the functionality and quality are adequate for beta testing with prospective customers.

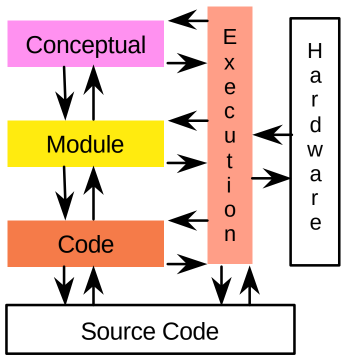

Figure 1. Four views of software architecture.

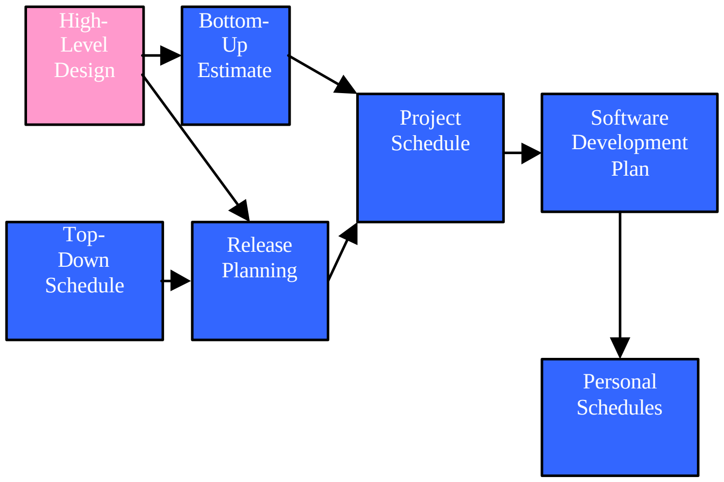

Figure 2. Architecture-centered software project planning.

The software architecture of Athena is based on a three-tiered model (user interface tier, business logic tier, and database tier), such that new metering applications can be easily added in the future at the middle or business logic tier (Figure 3). The user interface tier consists of a set of web pages, a web server, and a web browser for interaction with the user. The business logic tier is a group of subsystems that defines the business applications. The database tier contains the database interface, database tables, and database procedures. The business logic subsystems use the database interface or call database procedures to obtain and manipulate data in the database tables.

Customer accounts are managed through the consumer tree subsystem, where the relationships among master accounts, accounts, contracts, and consumers are described within a hierarchical structure (Figure 4) called the *consumer tree*. Each node in the consumer tree contains information about the account and a set of *active elements.* An active element is an object that encapsulates a part of the knowledge necessary to describe the data processing required for a node in the tree (e.g., a contract). Active elements include meters, tariff agreements, calculations, and reports. Active elements are interconnected to form *calculation chains*. Schedules of actions are maintained at each node such that a daily schedule is automatically generated and loaded to the meter data acquisition package from the data processing package. A design goal was to make the system easy to extend by the addition of new active elements.

Microsoft's Excel is used as a general-purpose computation and reporting engine. This provides both power and ease of use since most users are already experienced with using Excel spreadsheets for calculations.

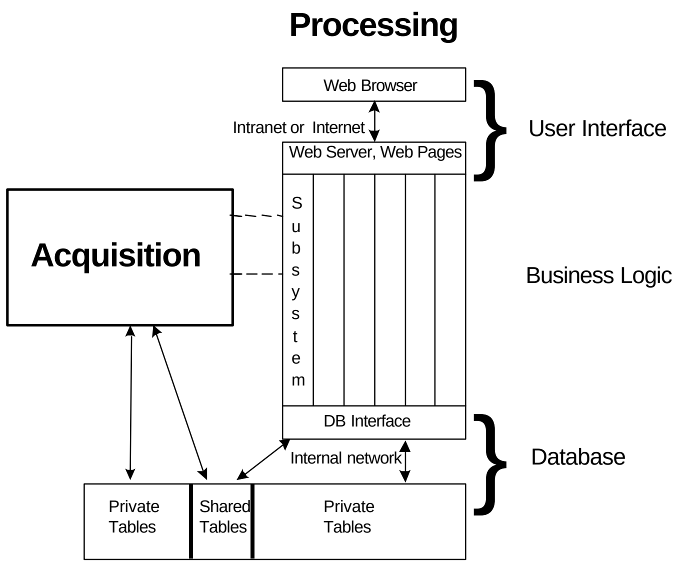

Figure 3. Three-tiered architecture.

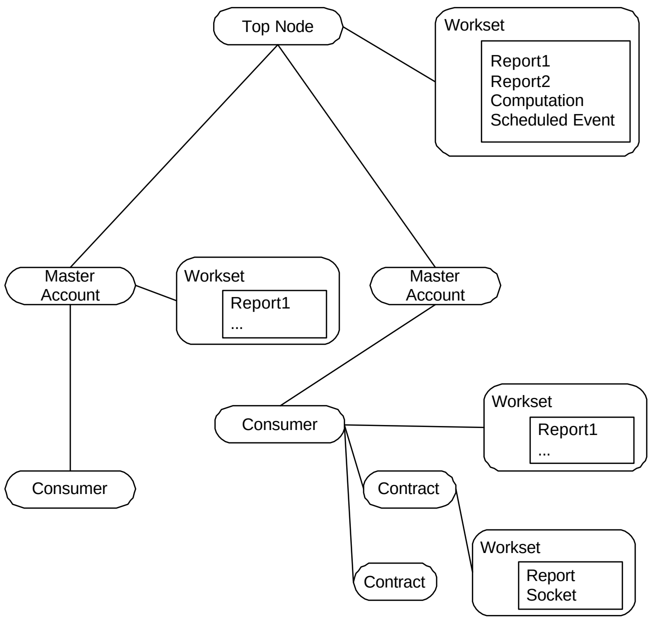

Figure 4. Consumer tree.

The web-based GUI (top tier) provides marketing, implementation, and cost advantages. User capability is simplified and empowered through the use of templates. Operators of Athena can manage the system (depending on their security profile) from a web browser on any client computer connected to the Internet or their intranet. The architecture is designed for scalability since multiple users can access the system simultaneously, with customer data segmentation and checkin/checkout capabilities for avoiding conflicts.

We believe that the resulting architecture is much more general and flexible than it would have been if it were designed within a single-site development effort. We observed that within a multi-site/multicultural project environment, communication is more difficult. By being more flexible, we are allowing more possibilities for increasing the functionality. As understanding is increased, we are able to increase functionality without changing the design. This generality does come at a cost. The architecture is more complicated than it would have been otherwise. One way in which the flexibility has paid off is with the ease of adding active elements. Initially, we expected to implement five types of active elements. Now there are eight, with several more expected. Another example is with the use of Excel. At first, there was significant resistance to using a general-purpose tool such as Excel, since prior projects had implemented specialized "lighter weight" calculation engines. Had we instead implemented our own calculation and reporting engine, we would have had significant problems as the requirements have evolved and become clearer.

Project Planning

We planned the software development as a sequence of incremental engineering releases with increasing functionality. The first release consisted of a 'vertical slice' through the architecture layer diagram, which functioned as a prototype of the architecture. The last release is the first set of functionality that can be sold as a package to a customer. We also planned alpha and beta releases that were tested by an in-house system testing function in Switzerland or by lead users. The project schedule was structured such that while in-house testers or users are testing a release, the development team is working on the next release. The cycle time for these incremental releases depended on the times needed for testing and feature development, as well as business constraints such as when the first set of useful functionality is needed by customers. For our global development, we found a 6-8 week cycle time between incremental releases to work best. The release plans were also driven by the dates of trade shows, at which time a new release with the latest functionality is required. We suggest fixing the release dates and release cycles, and then fitting in the functionality for each release depending on the bottom-up estimates and what makes sense for functionally testing the software as a set of user visible features.

The incremental release process was also a result of our global development process. We found that one of the best means of communication was by using the system itself. Only after parts of Athena were prototyped was it possible to fully understand and discuss the requirements. Perhaps this is because the operation of the system itself became a common language for us all. Incremental releases also encouraged the developers to quickly learn and start using new technologies, since they needed to learn enough to be able to implement even partial functionality.

	ER1	ER2	ER3	R1	R2+
Schedule Maker					
Search Consumer Tree for Scheduled Events	ō			ō	
Create a Schedule	ō			ō	
Handle Report Events	ō			ō	
Handle Acquisition Events		ō		ō	
Optimize Acquisitions					ō
Handle set Parameter Scheduled Events		ō		ō	
Display and manual Update of Schedules			ō	ō	

Figure 5. Build Plan.

With consultation from marketing and project management, a build plan (Figure 5) was completed detailing which product feature would be incorporated in which engineering release. From the build plan, a Software Development Plan (SDP) for each release was made. The SDP contains individual task assignments, sets of features, dependencies between features, risks, and test plans. Like the architecture, the SDP is a communications vehicle, and thus it must be simple enough to be understood by everyone on the team. Also, it is better for the engineers to plan the details of their work within their personal schedules, rather than for project management to schedule everything for them. This is especially true for multi-site development, where the developers must be flexible and responsive to the work going on at the other sites.

During the planning process for each incremental release, drafts of the proposed schedules and task assignments were circulated to the team members. The first version was distributed as a proposed development schedule and staffing plan to get feedback from all the team members. We would often receive feedback from the team members such as "this feature cannot be achieved in the time frame planned", "Joe is better qualified for this task than I am", or "I have vacation or training planned during these weeks". Based on this feedback, a second version of the plan would be developed and distributed which contained the "final" schedule committing the release dates and desired feature set. We later learned during our intercultural training that the Swiss are more comfortable with such a consensus building approach (called *Vernehmlassung* in Swiss German), while the Germans are more used to management specifying the development schedule.

We found that a clear division of development responsibilities was very important to the European team members. The SDP represents a commitment by the project team to develop the software product within the schedule and staffing requirements described.

We have found that a description of the overall project goals is useful and should be stated within the software development plan. An example statement of project goals is, quality will have higher priority than schedule, which will have higher priority than functionality. Such a statement of project goals helped the project manager make the tradeoffs that inevitably must be decided right before a release. It helped give guidance in answering the last minute questions such as, should I slip schedule to put in a few more features? The project goals are important to explicitly state for a global project team, since we observed cultural biases concerning qualities such as timeliness, perfectionism, quality, openness, and transparency.

Project Management

Each development site had a local manager to manage the team members at that site. There was also an overall project manager and a project manager for the software application package development. Thus, there was overlapping management responsibility for achieving the project goals. These managers had to negotiate individual work assignments; for example, if an individual was planned to work on multiple application packages in parallel. The overall project manager resolved conflicts that couldn't be handled at a lower level. In practice, many of these types of staffing conflicts surfaced when the proposed software development plan was distributed for feedback.

The chief architect was responsible for decision making and resolving technical conflicts for the application package. Analogous to the overall project manager, there was an overall technical manager. In practice, key technical decisions that affected the overall project goals were reviewed with the project and technical management in Switzerland, before they were implemented.

Each subsystem that was designed and implemented for Athena had a responsible engineer assigned to it. This approach was strongly supported by the European technical team members, who wanted clear division of responsibilities and ownership of subsystem code. In some cases, team members (especially in the U.S.) took on project nicknames consistent with their role within the project. For example, "GUI guy" was the person responsible for the user interface development.

A key role for the project was the "buildmeister" who was responsible for building and the integrating the system daily as new functionality was added. A centralized source code repository was used with access by all team members.

Project status tracking was done during weekly teleconferences. Each team member was encouraged to report on their development progress and raise information or issues to be shared with other team members. We noticed during these teleconferences culturally based communication differences where the Americans primarily wanted to discuss progress while the Europeans primarily discussed problems. To help balance the reporting of problems and progress we requested team members to experiment with role reversals such that the Americans initially described their problems during development before describing their progress and visa versa for the Europeans.

Multicultural Variables

Multicultural variables are a factor for projects developed in multiple countries [4]. These variables can be exploited as strengths for the development or they can get in the way, depending on how they are handled. Our project was not only multi-site, but also multinational. There were also strong multi-company cultural variables, since part of the development team had come from a company that had been recently acquired by Siemens.

Having gone through team training and multicultural training for Americans, Swiss, and Germans, we were very aware of the role of multicultural variables on our project, although we sometimes didn't know what to do about them. For example, we observed that there are culturally biased attitudes concerning basic qualities such as punctuality. These biases often indirectly affect our project schedule planning and execution. This can lead to frustrations for certain team members. Knowing that the frustration is culturally based sometimes helped us better accept our team members' perceived strengths and weaknesses, and may have reduced the frustration.

Some of the very basic project decisions that are taken for granted by single-site development teams could have significant overloaded meanings for multinational teams. For example, a decision such as our design documentation will be written in English is usually insignificant to a U.S.-based team. For a multinational team with team members who are not native English speakers, this decision can affect the task time estimates of team members with less experience writing specifications in English. This can frustrate American team members who are thinking, "why are those Europeans so slow"?

Language can also affect the way that design meetings and reviews are conducted. In an environment where design alternatives are expected to be "argued" until a consensus approach is determined, the non-native English speakers can feel disadvantaged. They may be less skilled in communicating their preferred design approach. Sometimes in such design meetings, the designer who speaks most and the loudest will bias the team to pursue a specific approach.

Software engineering is usually concerned with the management of tradeoffs. Project managers and development team members constantly wrestle with concerns such as, "if I had a few more weeks to perfect my design, the resulting code should be higher quality, but the product will take longer to get to the market". Cultural biases concerning qualities such as perfectionism, quality, work ethic, teamwork, etc. will often color the tradeoffs that are selected. Within a project with a tight schedule and limited staff, tradeoff decisions among quality, schedule, and functionality are made every day.

One of the more difficult cultural differences addressed within Athena was the frequency of interaction and collaboration among the team members. We observed that the European colleagues preferred to be assigned a task, and then they would pursue that task with minimal interaction with their colleagues. Americans tended to ask more questions of each other after they had initiated a task, sometimes changing the task characteristics, as new information became available. This sometimes made it difficult for project management to measure the progress of development tasks or the relative productivity of team members [5-8]. A project rule was proposed such that if an individual spent a predefined amount of time on a task, then they were encouraged to ask questions of a colleague to help move the task along. Setting up a chief architect was important to support this rule, since most of the time the team members would direct their questions to him.

Lessons Learned

At the current point of development, we are pleased about our experience with implementing the first two application packages using the Athena software architecture. The development of a system design specification was viewed by some team members as an unnecessary step that delayed the start of the application packages implementation. However, new development team members working on both the current and new application packages have successfully used this specification. It has been critical for partitioning work packages across the four development sites located within Europe and the U.S. We have observed that integration of the various subsystems has gone remarkably smoothly when the subsystem leaders are brought together in one location.

We've had good experiences with our approach to incremental development. By publishing the URL for the test system in Switzerland, all team members and their management can watch the progress of the development as new features are continually added. This was a big morale boost for the team, since everyone was aware of the rapid progress that was being made after the high-level design phase was completed and development began. The first engineering release was an implementation of a vertical slice through the architecture. This helped validate the architecture and gave the development team the confidence and understanding of the architecture to be able to implement the future engineering releases.

At the end of the planning phase for each incremental release, each team member has a personal schedule with weekly milestones for the components that he is responsible for developing. In addition, there is an overall project schedule, monitored by the project manager that identifies how the components are allocated to the incremental releases. The schedules have been developed to a large extent by the development team members, and thus their ownership of these schedules is relatively high. Business management, marketing, service, and sales can refer to the SDP and the Release Plan to see when functionality becomes available within the various incremental releases.

Since we put priority on meeting scheduled release dates and traded off functionality and quality as necessary, the development team successfully achieved every release date. This helped build up the credibility of the development team with management, since they knew that a new set of functionality would be ready for validation testing by the dates that were planned and committed to at the beginning of the baselined engineering release cycle. Fortunately, our quality remained relatively high throughout the development, so the tradeoff between meeting schedule and quality was never seriously challenged.

In the beginning, project meetings were held monthly, rotating among the development sites. Currently meetings are held mainly for major subsystem integrations or when training and/or detailed design work is necessary to get someone started on a new development task. We have weekly teleconference meetings to track schedule status and to bring up common problems that the developers should be aware of. We publish the goals of the teleconference in advance, using the techniques we learned to plan and conduct meetings during the team building training.

Despite our best efforts at communicating among the four development sites and our emphasis on design documentation with well-defined interfaces, global development is clearly more difficult than single-site development. This is a result of occasional miscommunications that are caused by different vacations and holidays in the three countries, time zone differences, and occasional network or computer outages. For example, if questions arise for colleagues in Europe during their evening hours while the U.S.-based teams are working, they likely will need to wait until the next day before they can be resolved. To compensate for the unexpected, team members often used the home telephone numbers of their colleagues in the other countries, and the system is rebuilt almost every day in multiple locations using the latest checked in source code. We have also invested in technical training, team building, and multicultural training for the development team members.

The Athena software product line architecture is designed to be very flexible and expandable to handle a wide variety of applications. This is a primary design requirement, since the power distribution industry is rapidly changing as a result of worldwide deregulation. The diversity of our development team members with differing skills and experience has helped us achieve a flexible design.

Recommendations for Global Development Teams

Based upon our experience on the Athena project, we provide the following recommendations for global software development teams split across the U.S., Germany, and Switzerland. The suggestions and lessons learned are probably most applicable to medium sized software development projects. We do not claim that very small or very large projects should be done this way.

- Develop a project culture: In order to reduce cultural variables, explicitly build a unique project culture. This project culture can select norms from a single culture or mix the "best" characteristics of multiple cultures. This project culture is established from the very beginning of the project. For example, from the beginning will team members address each other by their family names or given names? By explicitly developing this project culture, the comfort level of individual team members should increase. A skilled project manager can recognize the cultural differences while setting the norms for the project. For example, a project manager can state, "we realize that you normally don't spend so much time in design meetings, but for this project we will meet for another couple days before we break off to write specifications".
- Set project goals: Within a multicultural environment it is important to set clear project goals and define the criteria that must be satisfied for project success. This means putting relative priorities on project characteristics that must be traded off such as quality, schedule, and functionality. For Athena, we decided during global analysis that project schedule would take priority over functionality. Such goals definition at the beginning of the project helps the team recognize when it has achieved success.
- Overcommunicate: For global development, it is necessary to overcommunicate. This is necessary to overcome communications problems associated with language understanding. Abbreviations and slang should be avoided. Decisions made during verbal communications must be followed up in writing.
- Put as much as possible in writing: As related to overcommunicating, it is important to put as much project information in written form as possible. This includes both technical information such as specifications as well as project information such as the software development plan. For project team members who are working in non-native languages, put as much technical information as possible in diagrams that are easier to read than words. For software architecture descriptions, the UML notation has been helpful [1].
- Patience: Issues will not be resolved as quickly for a global development team as compared to a team located at a single site. In our case, with a six-hour time difference between the European and U.S. sites, the common workday was limited to a couple hours. Many issues were not resolved until the next day. Thus, all team members must practice patience.

- Identify team roles: Attempt to identify each development team member's role in the project. For Athena, we defined roles and named the roles with titles that we thought would provide some understanding of the role. In some cases, we needed to communicate and describe the roles a few times before team members understood them.
- Build consensus: Different cultures have different ways to indicate agreement and resolve conflicts. Within global development teams, consensus should be built at every opportunity. The European colleagues often were more comfortable with hierarchical decision making, but the power of a diverse multinational team with differing viewpoints can only be tapped when team members are actively solicited for their opinions.
- Invest in team building and multicultural training: This training helped us to formulate our project culture. Particularly important were the meeting management skills that we learned during the team building training. The training also gave us an opportunity to learn more about the various team members and some of their cultural biases and stereotypes. The group exercises done during the training can address real project issues such as defining the software development process.
- Get to know each other outside the work environment: Common meals and outings after working hours helped form personal as well as professional relationships among the team members. The personal relationships often helped bridge the gap when communications broke down during the workday. The personal relationships were also helpful for handling logistics and information issues that resulted from traveling: where to stay, shop, eat, relax, etc. when away from home.
- Travel: It is extremely important for key players to work together in person. We have consistently found that a day of face-to-face communication is more effective than a couple weeks of communication by telephone and email. This is especially true with team members from different countries, where non-verbal communication plays a much larger role in the exchange of information.
- Management support: Project management for global development must be both centralized and localized. Overall project leadership must be centralized, but every development team member needs a local manager for day-to-day support. Thus, multi-site collaboration and cooperation is necessary for the management team. Nevertheless, team members who are highly self-directed seem to perform better on global teams.

Conclusions

The high-level design and estimation process for Athena has been very useful for estimating the effort, dividing the work, and building buy-in and a common project culture for a team geographically distributed across three different countries. Six incremental releases were defined with an average interval time of 8 weeks. All planned release dates have been met so far. The biggest complication in developing the schedules has been to coordinate holiday and vacation times for development team members across the three countries. For example, development progress came to almost a complete stop at the end of 1999 when all countries were celebrating the new millennium. This pushed back an intermediate release milestone, which reduced the amount of time available for the development of the next release. We have learned that the greatest enemy of the global development manager is the varying holiday and vacation schedules of the team members within the different countries.

Team members of multi-site projects must, because of geography, be highly self-directed. Assigned tasks must be flexible enough to address the tradeoffs that come up that must be resolved without waiting for the next business day. Team members must collaborate with their technical interface counterparts frequently, especially when stuck on design issues. Face-to-face meetings still work best.

We have concluded from our experience on Athena that global development is not as fast or efficient as a team located in one place. Often, questions must wait a day to be answered since they usually do not arise during the limited common working times of team members in different time zones. However, the overall strength of global development is the flexibility and modifiability of the resulting design. Different views of the product colored by cultural biases, knowledge of local market conditions, experience with differing technologies, etc. result in a design that is more adaptable to changing markets than when the design is done by a single location team with less cultural and technical diversity.

Acknowledgements

We wish to acknowledge the contribution of the other members of the Athena high-level design team, namely Bill Sherman, Paul Bruschi, Henk LaRoi, and Sascha Lukic and the support of our management team, namely Uli Syre, Michael Sommer, Tom Murphy, and Ali Inan.

References

1. C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*, 2000, Addison-Wesley, Reading, MA.
2. R. Kazman, M. Barbacci, M. Klein, S. Carriere, and S. Woods, "Experience with Performing Architecture Tradeoff Analysis", *Proceedings of the 21st International Conference on Software Engineering*, New York, ACM Press, 1999, 54-63.
3. D. Paulish, R. Nord, and D. Soni, "Experience with Architecture-Centered Software Project Planning", *Proceedings of the Second International Software Architecture Workshop (ISAW-2)*, New York, ACM Press, 1996, 126-129.
4. E. Kopper, "Swiss and Germans: Similarities and Differences in Work-related Values, Attitudes, and Behavior", *International Journal of Intercultural Relations*, Pergamon Press, 1993, 167-184.
5. C. Jones, *Applied Software Measurement*, 1991, McGraw-Hill, New York.
6. L. Putnam and W. Myers, *Measures for Excellence*, 1992, Yourdan Press, New York.
7. R. Austin and D. Paulish, "A Survey of Commonly Applied Methods for Software Process Improvement," Tech. Report CMU/SEI-93-TR-27, ESC-TR-93-201, Software Engineering Institute, Carnegie Mellon Univ., Pittsburgh, PA, 1993.
8. K. Moeller and D. Paulish, *Software Metrics: A Practitioner's Guide to Improved Product Development*, 1993, IEEE Press, London.

Dan Paulish

Dr. Paulish is currently a software project manager at Siemens Corporate Research in Princeton, NJ, responsible for Siemens' software architecture research and development program. He has over twenty years' experience in software engineering management. He has been an international lecturer on software process improvement methods, project management, and measurement.

He is a co-author of *Software Metrics: A Practitioner's Guide to Improved Product Development*, published by IEEE Press and the author of a forthcoming book, *Architecture-Centric Software Project Management: A Practical Guide* to be published by Addison Wesley. He is formerly an industrial resident affiliate at the Software Engineering Institute (SEI), and he has done research on software measurement in Europe. He holds a Ph.D. in Electrical Engineering from the Polytechnic Institute of New York.

Siemens Corporate Research
755 College Road East
Princeton, NJ 08540 USA
+ 1 609 734-6579 (phone)
+1 609 734-6565 (fax)
dpaulish@scr.siemens.com