Issues in co-operative software engineering using globally distributed teams

Ian Gorton\textsuperscript{a}, Sanjeev Motwani\textsuperscript{b}

\textsuperscript{a}CaST Lab, School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia
\textsuperscript{b}Asia Pacific Systems Engineering Centre, BT Australasia Pty Ltd, 1 Market St, Sydney, NSW 2000, Australia

Received 16 December 1994; revised 19 December 1995; accepted 15 January 1996

Abstract

The use of geographically separated software development groups is proposed as a method for enabling 24-hour software development, or software shift work. The advantages of such an approach are explained, and potential organizational models for such virtual teams are described. Virtual team co-operation, information requirements and communication channels are explored. Activities in the software development life-cycle in which virtual teams can be advantageously utilized are explained, and examples of the successful use of virtual teams are cited. Finally a measure of the effectiveness of virtual teams known as the distributed working overhead is defined, which will enable project managers to clearly see the benefits and associated costs of employing virtual teams on a project.

Keywords: Software development; Shift work; Virtual teams

1. Introduction

Many large multinational organizations have numerous software development groups residing in different countries around the world. In most instances each group is primarily responsible for servicing its local and regional market. Occasionally development groups may be seconded to support a project for a remote customer. This is usually achieved by allowing the development team at the customer site to use the remote team as a resource to perform a particular aspect of the project. There are documented examples of complex products being developed by globally distributed teams [1], but these are few and far between and by no means the norm.

Utilizing multiple teams on a software development project enables concurrent engineering methods [2] to be applied to reduce product time-to-market [3]. Each team is allocated a portion of the overall project, allowing the development effort to proceed in parallel. If the tasks each team performs are relatively independent, inter-team communication overheads can be kept to a minimum, enabling each team to act autonomously and progress without substantial external hindrance.

For teams that are located in different places around the globe, a low level of interdependence would seem ideal as this reduces the problems and risks associated with long-distance communications and information sharing.

A further consideration when using globally dispersed teams is the time differences that exist between teams. For example, two co-operating teams located in the United Kingdom and Australia may (depending on the time of year and exact locations) have only a one-hour overlap of core working hours. This makes direct communications inconvenient at the least, and potentially introduces a major obstacle for effective team interaction.

Significant time differences between development teams do however introduce the possibility of software shift work. Many industries use shift work to provide continuous 24-hour construction of a product. Due to the nature of the products being manufactured (i.e. a new road or a car production line), staff are required to work unsociable hours in the same location to achieve greatly reduced production times. Software systems however differ in nature from most artefacts in that the output of the software production process is a collection of
binary files which comprise the documentation, source code and executables [4]. Using modern data communications networks, these files can be rapidly moved around and shared by teams anywhere in the world. Continuous 24-hour software development could therefore be achieved by exploiting time differences between teams, giving the effect of shift work but allowing teams to work the conventional hours for their location.

In this paper we speculate on how the software production process could be re-engineered to further exploit concurrent activities with globally distributed teams. Aspects of the software engineering process itself, infrastructure requirements and people-issues are considered. Where applicable, we mention our experiences gained from experiments in this area, and discuss our plans for a substantial development trial using groups from within BT subsidiaries worldwide.

It is important to stress that our approach differs from other work in this area (e.g. [5]). Our primary aim is to examine the project management, inter-team communications and software process aspects of global development teams. This will be achieved by designing and monitoring increasingly more ambitious live development trials, and utilizing the results to provide continued process improvement. We therefore look to utilize specialized co-ordination and communications support tools only when they are proven, commercially available technology and can be exploited or rapidly customized to support distributed working. It is not our intention to develop new tools such as shared editors or multiuser debuggers, although we have little doubt that such tools will increasingly move from the research laboratories into the commercial mainstream. We hope that the results of this project will define a fundamental framework for co-operative software engineering, into which advanced tools can be easily integrated as they become available.

2. Software development shift work

In general terms, a number of factors must be present in a development project to enable concurrent engineering techniques to reduce product time-to-market. These are [2]:

1. The division of the project into sizeable chunks of semi-independent activities.
2. Assignment of these activities to project teams who can proceed in parallel.
3. The performance of each assigned team activity as fast as possible.
4. The simultaneous starting of as many activities as possible.
5. The minimization of inter-dependencies between activities.
6. Transparent communications between and across parallel activities.

Software engineering projects typically exploit concurrency by dividing the development into loosely coupled sub-systems and assigning sub-systems to separate teams. Each team then proceeds relatively autonomously to design, code and test their allocated sub-system, allocating modules (i.e. programs) to individuals as appropriate. Wherever possible, design decisions which impact on other teams are made as early as possible in the module development cycle to reduce inter-team dependencies. When completed, individual modules are progressively integrated and tested to form the completed product.

In Fig. 1, the traditional decomposition of a software system is illustrated. Decomposing the overall project into sub-systems is depicted as a project team level task. Each sub-system is then allocated to a development group, which decomposes this into a number of modules which individuals can concurrently design, code and test. This manner of project decomposition is usually carried out using teams and groups which operate from the same geographical location. It is however amenable to using geographically separated development groups as reported in [1,6]. This requires development groups to interface at the sub-systems level, with the production of each sub-system carried out by individuals in a single co-located group.

In developments which utilize geographically separated teams with partially overlapping or disjoint
In order to exploit the overnight gain effect to a maximum, development groups need to cooperate on a mutual shared task. This implies close co-operation and co-ordination between members of geographically separated groups. When this is the case, we will use the term virtual team [7] to describe the relationship. Fig. 2 demonstrates a possible scenario using two developers from different sites and time-zones cooperating on the development of a module in a system. While one developer designs the module, the other produces a test specification. Each then reviews the other's work (denoted by the dotted lines to show the flow of documents), and after making any necessary amendments, the program coding and code review tasks are shared. The two developers then individually produce test data and test cases, which are then exchanged to enable their collaborator to run the tests and fix the problems wherever possible. With appropriate communications and co-ordination technology linking the two developers, the opportunities to exploit the overnight gain effect are considerable. Importantl, this manner of working allows small, fine-grain tasks of short duration to be decomposed and tackled concurrently, with productivity gains achieved through exploitation of the time difference between collaborators. This effectively introduces scope for concurrent, co-operative software engineering practices at the Individual Task level as depicted in Fig. 1.

It is envisaged that the effective use of virtual teams could have a number of beneficial effects for an organization involved in software development. These are:

- Overlap of development activity, giving potentially 24-hour continuous development effort.
- Reduce time-to-market for software products.
- Better utilization of available skilled employees.
- Add a global perspective to products during development [8].
- Improve software quality through improved development processes.
- Minimize duplication of expert skills in different locations.
- Exploit low operating overheads and skilled staff in developing nations [9].

Of course, there is a down-side to this scenario. Time differences between teams can also hinder the progress of a project if not managed correctly. A group may request a collaborator to carry out a task 'overnight', requiring it to be complete by the next morning so that further progress can be made. However, should this request get lost, delayed or be ignored, leading to the task not being completed, the requesting group may not be able to progress at all on the task the next morning. If the servicing group is no longer at work, or is reaching the end of their day, a delay of at least one whole day is incurred until the issue can be resolved. Such a problem we define as the overnight loss effect. This leads to the conclusion that productivity can be maximized by exploiting the overnight gain effect as much as possible, and minimizing the influence of the overnight loss effect.

How virtual teams should be formed, how they should cooperate and communicate, and what kind of software development tasks are suitable for virtual teams are all questions which will be considered in the next section.

3. Virtual teams—organization and infrastructure

3.1. Communications requirements

Virtual teams require a basic communications infrastructure in order to operate. All members of the virtual team should be connected via a high-speed computer network which provides electronic mail and file sharing.
services. It is likely that development machines at each site will be connected by a relatively fast local area network, with a slower wide-area network connecting the individual sites. Email delivery times between sites should be fast enough to allow email to be used as the primary asynchronous communications mechanism between team members. The wide-area network should also provide sufficiently low transfer times for project-related files. High reliability of the networks is essential to keep teams working co-operatively and concurrently: a sensible policy would be to provide a back-up communications network to cover for primary network and email failures.

In addition, telephone, teleconferencing and facsimile facilities should be provided for synchronous communications and transfer of paper-based information when necessary. Videoconferencing would further be a valuable asset to be used if available [8].

We expect that new high-speed networks, multimedia workstations and multimedia software development tools will profoundly influence virtual team communications in the years to come [5]. However, this technology is still relatively new and is unlikely to be in widespread use for a decade or perhaps longer. Moreover, we do not believe it to be fundamentally important in supporting virtual team activity. Fast, reliable data networks for asynchronous information sharing and email and voice links for synchronous communications comprise the required basic virtual team communications structure.

3.2. Virtual team organization

It is important to clearly establish the precise relationships and roles of each group which comprises a virtual team. We propose three major organizational models which can influence virtual team structure, namely co-operative, delegation and consultative.

Co-operative virtual teams comprise homogeneous distributed development groups who share the on-going process of developing a product. Groups co-ordinate closely on development tasks and each team relies on other teams to provide a range of services to help them exploit the overnight gain effect. In the co-operative model, although one group is likely to have overall project management responsibilities and customer accountability, groups co-operate to set consensus schedules and deadlines, and take responsibility for individual deliverables. Co-operative virtual teams could be involved in group decision-making or design meetings using tele- or videoconferencing, or share tasks such as program design and coding.

When virtual teams adopt a delegation structure, one distributed development group, the supervisor, assumes the responsibility for allocating tasks and setting schedules for another development group, the worker group. The supervisor and worker groups co-operate on a particular task, with the worker group providing a specific well-defined service to help the supervisor group exploit the overnight gain effect. Typically the supervisor group consists of more senior, experienced development staff, with the worker group members have less experience and a more narrow skill set. For example, the supervisor group may be a development group which devolves all its testing activities to a worker group, exploiting the overnight gain effect to obtain a fast turnaround of test results.

The consultative model relies upon the existence of distributed expert service groups who can be called upon to carry out a specific task within their area of expertise. Expert groups could provide services in particular areas of software development, such as user interface design, or provide specific knowledge in a particular product area. In such circumstances, a development group may decide to sub-contract out certain tasks that it feels it lacks expertise in, or enlist the help of an expert group to validate or assess its progress on some task. Deadlines are negotiated between the two groups, and the expert group assumes sole responsibility for completing the task by its deadline. Expert groups may service one or more virtual teams simultaneously, exploiting the overnight gain effect whenever sufficient time differences exist and the required consultations are of a sufficiently short duration. For example, an expert human-computer interaction group may advise on the suitability of user-interface designs, or the network software development group may support groups wishing to write high-performance distributed applications.

Although each model implies a different project management strategy and group interaction structures, the models should not be seen as mutually exclusive. All three may be adopted for some or all stages of a product development, as specific activities and group availability demand.

3.3. Virtual team information requirements

Regardless of the specific virtual team organization, groups need to exchange information which fits into the following three broad categories:

- **Project development documents**: including requirement specifications, design documents, program code, test specifications and data, and product documentation.
- **Ad hoc inter-group communications**: including general emails, discussions, queries, alerts to problems or unexpected changes in development documents.
- **Project management information**: including task specifications, status reports, current schedules, resource requests and project plans.

Each of these three categories can be further characterized by the volume of documents generated during a project, the longevity of documents, and the visibility of
documents. Visibility can be further subdivided into write access, read-only access and invisible. For example, project development documents will typically be very large in number (and size), be required to persist for the life of the project and beyond, and be widely visible in read-only mode. Write access to these documents should be strictly controlled.

It can be reasonably expected that a considerable number of ad hoc inter-group communications will be generated during a project. Some will need to be kept for the duration of the project, but many will have a limited lifetime of perhaps only a few days. Visibility will in general be restricted to the sender and receiver(s), although this may be expanded in appropriate, usually important, cases. Project management documents in comparison will be relatively few, with a small number of key documents being constantly updated throughout the project to reflect the current status. These key documents will exist for the lifetime of the project and beyond, and periodic status information that is gathered will in general need to be archived. Visibility of some of this information will be by necessity highly restricted.

These information requirements point to the need for a large, partitioned data repository or group memory [10] accessible to each distributed development group, as depicted in Fig. 3. Each group should 'see' a consistent view of all the data items in this global repository. Access controls should therefore ensure that documents cannot become inconsistent due to simultaneous change from two different locations. Although work is being carried out on concurrent document editing (e.g. [5]), we will assume only standard access controls such as one writer/many readers are available.

In a globally distributed virtual team environment, the data repository may permanently reside at one location, or be replicated amongst some or all locations. Should replication be used, mechanisms must be devised to ensure that developers are working with the latest version of a shared document. For example, a developer in London may be editing a program code file late in their working day. A developer in New York, late in their morning, wish to edit the latest replicated version of the same code file. If this is allowed to happen, chaos may occur. The replication mechanism must ensure that a replicated document remains locked until the developer in London finishes editing the file. When this happens, the latest version should be replicated to allow the file to be edited, and control of the file passed to the New York site. The London developer must then not be allowed to alter the same file while it is being edited elsewhere.

5.4. Virtual team communications

In a virtual team environment, information flows must exist between distributed development groups. Some of these will be formal and may exist for significant periods to facilitate co-operation and co-ordination between groups or individuals in groups. Others will be informal, coming into being as required and ceasing to exist when no longer needed.

In Fig. 3, some of the formal information channels are represented using an example of three distributed groups forming a virtual team. The example depicts two peer groups co-operating on some part of a project, with the Sydney group further acting in a supervisory capacity for a worker team located in London. The location of the project control group is not specified: for the purposes of this example it could co-reside with any of the three other groups, or be at a different location entirely. Bold lines represent a communications link for project control.
related information, such as task assignments, status reports and schedules. The direction of the arrow on bold lines indicates that a particular group has overall responsibility for the activities of the other. When teams co-operate on a peer basis (e.g. Sydney and New York), the arrow is bi-directional to indicate this relationship.

Narrow lines represent access paths to information in the data repositories. Note the partitioning of the repository to include an area only visible to the New York and Sydney groups, and an area only accessible to the project control group. If replication of the repository is required, advantage can be taken of this partitioning to minimize the overheads involved in the replication process. Only globally visible data need to be fully replicated.

4. Matching virtual team models to specific development tasks

A software engineering project involves a number of different activities which loosely follow some form of life-cycle model such as the traditional Waterfall Model. In general, the specific activities involved are requirements specification, architectural design, detailed design, module coding, module testing, integration testing and on-going product maintenance. These activities are inherently different in nature: they require different tools and resources, require different skills from team members, and have different inputs and deliverables, each with associated methods of validation.

Modern approaches to requirements specification and architectural design techniques (e.g. [8,10]) emphasize cross-functional teams, group collaboration and consensus decision-making techniques [3]. With video- or teleconferencing technology, widely dispersed groups can be brought together to carry out these activities. In [8] an example of this is reported in which teams from the USA and Europe (five hours time difference) collaborated using teleconferencing to define the requirements for a complex software product. When virtual team meetings were held in the afternoon for the European group, the results of the meeting could be written up by the USA group in their afternoon and be ready for the European group to review when they arrived at work the next morning. After review and any necessary amendments, both groups could meet the next afternoon to continue with the requirements definition process. Fig. 4 shows how this mode of working effectively exploits the overnight gain effect to reduce the cycle-time between virtual team meetings.

In this example two group meetings are required to perform a task. By using two time-displaced groups the task is completed in 51 hours instead of 75, yielding a 32% reduction in elapsed time. Note that in both cases the same amount of work is performed by each group, leaving them equal time to work on other activities. In [8], some overhead is introduced by the addition of a facilitator at each site: this role would normally be fulfilled by one person when groups meet together. Hence the reduced elapsed time for an activity is bought at the expense of some additional person-hours to compensate for and manage the distributed nature of the activities. We expect this to be a general rule for all distributed working projects. Additional resources (people, networks, computers, telecommunications, equipment) will be required to overcome the distances and time between groups. The expense of these resources will however be recouped by faster product development times and higher quality products.

Apart from exploiting the overnight gain effect, the use of virtual teams during requirements specification and architectural design can have several other advantages. First, areas of existing expertise, from across an organization, perhaps in a similar application area, can be drawn upon to take part in activities (consultative model) without incurring both the cost and the time overheads associated with long-distance travel. For products which

![Fig. 4. Exploiting the overnight gain effect.](image-url)
are intended to be used on a global scale, cross-functional groups from different parts of the world can be utilized to capture these international requirements at the very start of the project (co-operative model). Virtual team usage during analysis and design could also provide a solid basis for the use of larger virtual teams during the remainder of the product development cycle.

Group-based requirements analysis and architectural design can be characterized as highly creative iterative and subjective tasks — there is usually no ‘right’ answer, and the group seeks to find a ‘best-fit’ solution from the total solution space. Development tasks such as detailed design, coding and testing are in contrast more tightly defined activities with smaller solution spaces and more readily identifiable problems. Whereas large cross-functional groups tend to be most suited to analysis and architectural design, smaller groups and individuals are better suited to more methodical development tasks [10].

Despite this lesser emphasis on group interaction, virtual teams can be gainfully employed on development activities and exploit the overnight gain effect. Distributed development groups can co-operate on quality assurance tasks such as design, code and test specification reviews (co-operative model). As the amount of time typically spent by engineers in quality assurance roles is of the order of a few hours each review, this kind of activity is highly suited to time-displaced distributed groups. An engineer could distribute a document (design, code, test specification) for review to virtual team members at another site. When the originator returns to work the next day the comments of the reviewers would be already available, and the appropriate action could be taken, giving a net overnight productivity gain. Should problems occur, for example, one of the reviewers being completely unable to understand parts of the document, the potential productivity gain would be lost. However, if the problem lies with the document, a positive benefit in terms of product quality is still accrued, as a failing in the document has been captured by the review process.

Distributed development groups may also act co-operatively during software testing. Groups could individually develop modules and then pass them to a collaborating group to test. The testing group could run tests and fix problems wherever possible, only passing unresolved errors back to the development group in severe cases. When tests can be run and code repaired overnight, productivity gains are experienced. As well, a high level of scrutiny of code is achieved, as the test group must be able to understand the code in order to remove errors. This further has the effect of creating a larger group of developers who are familiar with each module, increasing group ownership of the code and potentially avoiding major problems should the original developer leave the company.

Module and system testing may also be approached by virtual teams using the delegation model. In contrast to the use of peer groups, the test group merely runs tests and reports back on their success or failure. It is the responsibility of the supervisory development group to fix all problems that are found and pass these back to the testers for regression testing.

5. Initial trial results

An example of usage of the delegation model of co-operation is given in [11], in which a small project was developed in Sydney and tested overnight in Bombay. The five and a half hours time difference was utilized so that test results were ready for the developer upon arrival at work in the morning. In this manner the overnight gain effect was successfully exploited. Test specifications were produced by the development group and reviewed before being passed to the tester. However, with an experienced testing group, there is no reason why test specifications and test cases should not be created by the testers and reviewed by the developers. This could potentially introduce more concurrent activities, and represents a compromise between the pure co-operative and delegation models.

This experiment in distributed working proposed and tested a number of low-tech project controls to facilitate effective task allocation, task synchronization, status reporting and information sharing between the two sites. These worked effectively, but the experience clearly identified the need for automation of many of these routine information flows. Miscellaneous problem tracking and resolution was also found to be problematic, as the potential for direct communications was limited by time differences. Tools to track problems and record actions, status and resolution were clearly necessary in a distributed group environment.

Based upon the experiences of this initial trial, a prototype groupware support tool was constructed in Lotus Notes [12]. This was then trialled on the development of a C++ database system by four developers working at three different sites in different timezones (the time differences were actually simulated by each person working disjoint hours during the two-week trial). The groupware system provided facilities for asynchronous communications amongst team members. Notes forms were created for each task in project. Each team member could use the system to view the status of dependent tasks, monitor Status Reports and Task Completions, and receive Review Requests. Deliverables such as design specifications, program code, executables and test results could also be easily transferred between team members. Notes and email provided a means to communicate ad hoc messages and files asynchronously.
The most important findings of the trial were:

(1) Team members spent on average 22% (17% asynchronous using Notes and email, 5% synchronous using phone) of the time devoted to the project communicating and co-ordinating activity with other team members, and performing project management activities.

(2) 82% of opportunities to exploit the overnight gain effect were successful. Although the trial did not attempt to quantify the productivity gains experienced, anecdotal evidence from the team members' experiences indicate that they were significant. Further research is needed to decide how such productivity gains should be measured.

(3) Blocking of activity due to the six overnight losses (failure to deliver results in time) caused estimated delays of 21 hours, an average of 3.5 hours each. This is a significant figure, and emphasizes the importance of minimizing overnight losses.

(4) Lotus Notes proved a suitable tool for implementing a groupware prototype for supporting distributed software development teams.

6. Project planning for distributed developments

As mentioned in a previous section, the reduced elapsed time for an activity performed by virtual teams is bought at the expense of some additional person/hours to compensate for and manage the distributed nature of the activities. The underlying philosophy therefore is that it is beneficial for a company to complete the product as quickly as possible (and incur the additional expenses of virtual teams) in order for the associated benefits (e.g. sales, productivity, reduce costs) to be experienced as soon as possible. An example of this is given in [1], in which it was estimated that for each hour that the time-to-profit for the Alpha microprocessor was reduced, Digital Corporation would achieve an additional $1 million dollars of revenue.

Consequently, once an overall project plan is completed, activities on the critical path should be carefully examined. Armed with the knowledge of the resources available, both human and technological, strategies can be formulated for tackling the critical activities using virtual teams. This will result in a new plan with a reduced critical path and revised resource estimates.

For example, an activity on the critical path of a project is estimated as 20 person-days effort. By employing two distributed groups and exploiting the overnight gain effect, the new elapsed time estimate for this activity is 14 person-days, thus reducing the critical path by six days. To achieve this reduction, it is estimated that 15 person-days effort are required by each of the two groups: this figure allows for the additional co-ordination and collaboration activities introduced by distributed working. In addition, a further 10 person-days are added to the plan for overall project management and monitoring of the activity.

Overall then, a planned 30% elapsed time reduction has been achieved at a 100% increase in total person-days for the activity. To capture this quantity formally, we define the distributed working overhead as the amount of additional effort required to achieve a single day reduction in the elapsed time for an activity, or:

Distributed working overhead

\[
\frac{\text{new person-days for activity}}{\text{old person days for activity}} - \frac{\text{old elapsed time for activity}}{\text{new elapsed time for activity}}
\]

In this example, the distributed working overhead is three and one-third. In planning distributed development activities, the aim should therefore be to keep the distributed working overhead as low as possible, with an ideal value of zero. By calculating this measure, project management can see the additional expense incurred in reducing the activity's elapsed time, and base decisions on this cost.

7. Future work and conclusions

As the technology to support global working becomes more and more ubiquitous, progressive software engineering organizations will be able to take advantage of the additional flexibility and productivity afforded by virtual development teams. Much work is however needed before such potential becomes reality. New work practices and project management techniques need devising to exploit time differences and conquer the problems of physical separation. New tools are also needed to automate routine and repetitive project coordination tasks, and facilitate seamless, controlled access to project data.

One of the most crucial aspects of this work will undoubtedly be the way in which individual engineers react to forming and working in virtual team environments. Research has shown that if co-operation is not part of an organizational culture, individuals will not fully participate in co-operative work [13]. For a software development project, such problems could spell disaster! Other issues of importance that require exploration include managing cultural and work ethic differences that will inevitably exist, and designing suitable internal group structures to work effectively in virtual teams. We expect that new roles may be needed to take responsibility for facilitating communication and resolving issues.
This project is currently focusing on investigating the work practices and project management tasks involved with virtual team operation. Two small-scale experiments have been carried out, and the experience gained from these (and from observing other distributed group developments in operation), is being utilized to plan a third larger-scale experiment on a live project. Concurrently we are investigating the potential of existing commercial groupware products to provide support for virtual teams in a software engineering project. To this end, a prototype support environment has been developed, and is currently being considerably enhanced for use and evaluation during the next trial in late 1995.

References