Software Development using the 24 – Hour Knowledge Factory Paradigm

Amar Gupta
Thomas R. Brown Chair in Management and Technology
University of Arizona
Eller College of Management
McClelland Hall, Tucson, AZ 85721-0108
Ph: (520) 307-0547
Email: gupta@eller.arizona.edu

Rajdeep Bondade
Doctoral Student, University of Arizona
Department of Electrical and Computer Engineering
1230 E. Speedway Blvd, Tucson, AZ 85721
Ph: (520) 360-1974
E-mail: rajdeepb@email.arizona.edu

Nathan Denny
Doctoral Student, University of Arizona
Department of Electrical and Computer Engineering
1230 E. Speedway Blvd, Tucson, AZ 85721
Ph: (520) 850-9501
Email: n.t.denny@gmail.com

Abstract— The 24- Hour Knowledge Factory is a software development paradigm that goes beyond the conventional global delivery model. It involves round-the-clock operations through the establishment of three or more development teams around the globe. Complex projects are iteratively broken down into simpler modules, with a “composite persona” (CP) being responsible for each of such modules. Developers around the globe work during day hours only; and the unfinished work is handed off to the next CP driver. The 24-Hour Knowledge Factory concept integrates advances in business processes and IT to provide a greater level of agility and efficiency.

Index Terms— global software development, offshoring, services computing, service oriented architecture
I. INTRODUCTION

The 24-hour Knowledge Factory paradigm is designed to enable a project to progress around the clock, while still permitting all employees to work during daytime in their respective countries.

In some respects, the above paradigm is analogous to the notion of geographically distributed call-centers. By judiciously placing a cluster of 3 to 4 call centers separated by times zones 6 to 8 hours apart, round-the-clock service can be provided, using employees that work during daytime in their respective countries. Based in part on advances in information technology, it is now possible for an organization to conduct a growing array of more sophisticated tasks around the clock by placing offices around the globe, enabling such endeavors to be completed in a much smaller timeframe. This is a special case of the global delivery model in which members of the global team work on the same phase of a project, and each developer works the regular day hours pertaining to his or her time zone. Note that the 24-hr knowledge factory paradigm is different from the concept of the conventional global development team. In the latter case, the overall project is iteratively segregated into modular units. Each offshore unit is responsible for a distinct subset of such modules, and collaboration between these sites is minimal. This represents horizontal division of the project. The 24-hr knowledge factory, in contrast, suggests a vertical breakdown of tasks, with several global developers working on the same task, sequentially at a temporal level. Thus, at the end of one work day, after a developer has worked on a portion of the project, this work is handed off to the succeeding offshore site. The developer at the latter site continues with the same task, with the goal of taking the work-in-progress incrementally closer to completion.
The 24-Hour Knowledge Factory model requires the collaborating sites to communicate often, with frequent transfer of data, information, and knowledge. This, in turn, places a need to create new types of IT concepts and services that are highlighted in this paper. In particular, new mechanisms are needed for individuals to handoff the work-in-progress to their contemporaries located in another continent. The efficiency of handoff is determined by the quality of the knowledge transfer and the duration it involves. This handoff process varies significantly by the nature of work involved, and the manner in which a task can be decomposed into a set of autonomous or semi-autonomous mini-tasks.

II. AGILE SOFTWARE DEVELOPMENT IN THE 24-HOUR KNOWLEDGE FACTORY

The 24-Hr Knowledge Factory leverages time differences between collaborating centers to foster agility and efficiency. If one used only 2 collaborating centers at opposite ends of the world, one center could do software development and the other could test the software developed by the first center. The first center can now receive test results on an overnight basis, thereby allowing the development and the test processes to progress on an interlaced basis. This provides new strategic advantage to the concerned organization.
The handoff process is one of the critical success factors (CSF) for the 24-hour knowledge factory concept. Handoff involves transfer of work along with the rationale for the decisions made during the particular shift. Close collaboration between geographically dispersed individuals can improve the quality and the productivity of the endeavor. This can enhance the utility of the service to the customer, and the revenues that accrue to the developer. Architecting the process of handoff requires awareness of other intangible critical factors, such as establishing trust, mitigating cultural differences, and overcoming security concerns between the various collaborating teams. While agile software processes have mostly been applied to collocated development teams, the underlying principles are relevant to the 24-hour Knowledge Factory paradigm. Agile software processes try to convey maximum information, mostly through the source code and project artifacts that deliver immediate information. Popular agile methods, such as extreme programming which emphasizes collective ownership of code [1], and Scrum stand-up meetings, can be utilized in the 24-hour knowledge factory.

III. ENABLING TECHNOLOGIES FOR THE 24-HOUR KNOWLEDGE FACTORY

A number of software vendors have incorporated collaborative software in their newer products. Software packages such as SharePoint and Quickr are embedded into applications such as Word, Excel, and PowerPoint; they allow developers to share information without having to locally save any information before it is transmitted via e-mail or a web-based upload.

The Wiki is an example of evolving technology--equipped with tools for version control, search engines and content indices-- that enables informal knowledge sharing [2], by allowing web surfers to add, modify, and remove existing content on their servers.

Distributed authoring and versioning systems, like WEB-DAV, facilitate collaboration between offshore teams. Any change that is made to an artifact, such as to program code or to a
diagram, should be reflected at all the other sites that are involved.

The above technologies facilitate the operation of the 24-Hour Knowledge Factory model. In the next section, we explore motivations for new concepts that are expressly related to the cyclic and sequential nature of work in such an environment. The discussion below incorporates results from the research performed at the University of Arizona and the Massachusetts Institute of Technology.

A. Knowfact

When collaborating on the same phase of a project, each site will make decisions during the period that it is active. In order to ease the transition of knowledge from one site to another, integrated decision support systems are needed to ensure that each worker is able to readily absorb and leverage the information from his or her colleague. A concept demonstration prototype, called KNOWFACT, was developed at MIT to demonstrate the capability to capture and to disseminate knowledge in an actual situation involving the design of satellites. Technical decisions relating to the designs of different sub-systems were constantly being made, which necessitated frequent re-evaluation by geographically dispersed stakeholders. This type of model holds direct relevance to the software engineering domain, where decisions are made on a continuing basis.

Figure 3 depicts the KNOWFACT decision support system. The Decision Rationale Module (DRM) is responsible for defining the important attributes that define a particular system, from which stakeholders can ascertain the activity and progress of the different facets of the project. The Decision History Module (DHM) accumulates the history of all decision parameters on an evolving basis. The importance of these parameters is then weighted to calculate the values for the attributes that were defined in the DRM. This utility function is utilized by the development
team to make decisions on the relative importance of attributes for the next stage of the endeavor. This concept of storing both the rationale and the history, on an evolving basis, is used in the 24-Hour Knowledge Factory paradigm.

Figure 2: KNOWFACT process flow

B. Composite Personae

The goal of software engineering is the development of software solutions that can be used to solve complex problems. This is achieved by decomposing the task into many different subtasks, each having its own solution. Each sub-problem is assigned to a single developer, who owns that artifact through the entire process of coding, testing, and maintenance, leading to the development of black-box solutions. Ideally, the task as a whole should be well defined, allowing for easy categorization of subtasks, and minimizing amount of lateral communication between the developers. However, in most cases, the problem is quite complex and has a high degree of correlation to many of the issues that need to be solved. Conditions external to the project may also change. Thus, changes in the requirements of one sub-system and its behavior will propagate to other systems, creating a ripple of changes that have to be incorporated through all the systems. The result of such an occurrence is a need for considerable cross-communication between all the developers, which can be an imposing challenge.
The development of large software programs and systems entails long hours being spent on collaborative tasks; and studies show that developers can expend 70% of their time on such activities [3]. This need for communication can lead to confusion between teams, which span a range of different languages, time zones, and cultures.

In order to overcome the communication skew and ambiguity between different development teams, caused by differences in culture and linguistics, the 24-Hr Knowledge Factory research group at the University of Arizona proposed the concept of the composite persona. A composite persona (CP) is a highly coordinated micro-team which may seem like a single unit, but consists of a collection of several individuals. In such a system, each offshore site mirrors its counterpart; i.e. each site has the same number of composite personae. However, this does not imply that an equal number of developers are present at each location, as each site can have individuals that belong to more than one composite persona. During the span of 24 hours, the CP is always active, but with a different driver for each shift of eight hours. Sub-tasks are now assigned to a particular composite persona, rather than to a single developer, and for the purpose of conflict resolution and discussion, each CP acts as a single entity.

In the composite persona model, tasks are divided vertically, with teams working in a time-sequenced manner. Communication plays a major role in determining the success of the composite persona model. There are three different types of communication with the first being handoff. As the succeeding site turns on and a new driver takes responsibility for the CP, the handoff procedure is necessary. For co-located developers, Scrum stand-up meetings can be conducted.

The second type of communication is lateral communication, which involves real-time synchronous communication between two active, co-located CPs. The third form of
communication consists of lateral communication with handoff. Such a form of communication may arise due to interaction between two CP drivers at one development site. When this site turns off and data are handed to another site, the same conversation can be resumed from the received data.

Overall, large-scale collaboration between different offshore sites around the world is attained with the company having at least one active CP driver working at all times of the day. This approach can provide high degree of agility to the business. This business model can lead to high operational throughput, while still retaining good control and coordination capabilities.

For any service to be effective for the customer and profitable for the organization, a very sound understanding of the capabilities and needs of both parties must be realized. However, much of the relevant pieces of tacit knowledge is not often communicated effectively as these pieces are inherently difficult to formalize. Thus, it is vital that effective strategies be developed and deployed for knowledge to be represented and transmitted effectively. The next sub-section of this paper focuses on the strategy employed to accomplish this objective.

C. CPro

CPro is an agile software process that improves the performance of the CP by incorporating automated algorithms for estimating delivery schedules, reducing software defects, and propagation of knowledge within the CP. It also assigns workloads to the different members of a CP, in a way that maximizes productivity.

CPro incorporates features that facilitate efficient planning during the course of a software project. During the planning phase of project it is necessary to determine the schedule to be followed. It is not possible for one person to estimate the entire agenda on behalf of the entire CP. This is because of the vertical decomposition of tasks, which can cause productivity to vary
based on the success of communication. Unforeseen external factors may also affect the project schedule. In order to address this challenge, CPro allows all developers to give their respective time estimates for all subtasks. A schedule caster then executes a Monte Carlo simulation in order to determine how the project should evolve. The emerging probabilistic schedule can be used by project managers to estimate the delivery dates of various project artifacts. The schedule caster in CPro simulates several possible work schedules based on the data obtained. This is shown in Figure 4.

![Simulated work schedules](image)

CPro employs a continuous distribution to determine productivity. This distribution is affected by data acquired from the developer’s estimates and from any previous estimates made, including the corresponding factual data. Tasks are treated as continuous entities, as they can be handed-off while still present in an incomplete state, to the next active site. The schedule caster records previous estimates and the corresponding actual performance from such instances. This data is utilized by CPro to generate high-probability distribution curves, especially ones that will lead to high productivity. For this iterative learning process, two mechanisms were analyzed in detail. The first mechanism used Bayesian probability with joint distributions; and the second one involved case-based reasoning, wherein cases were generated based on prior estimate and actual performance.

CPro helps to assimilate scheduling inputs from all the developers and to provide a cohesive and practical phased schedule for the distributed development team.
1) **CPro Development Process**

CPro disaggregates the project into different phases: planning, design, design review, coding, code review, and testing. Each of these phases is atomic; once a task has been accepted by a CP driver, the module must be accomplished in its entirety or dropped altogether. Handoff between two drivers can occur only at a phase boundary. Each succeeding phase relies heavily on understanding the previous phase, with such knowledge is not immediately distributed to developers not belonging to the particular CP.

When a module is assigned to a CP, each member approximates the difficulty of each phase of the task. Qualitative descriptors such as very simple, simple, moderately complex, complex and very complex can be used to describe each phase of the task. Every time a developer makes such an estimate and completes the task, the fact is recorded in the history and CPro adjusts the probability distribution for that developer. All the qualitative levels of complexity are associated with an offset exponential distribution. Based on the time interval that is available, each developer determines the difficulty of the task. Two methods were investigated; these are discussed in the following subsections.

a) **Random work assignment**

In this approach, multiple iterations are performed over the different tasks and each phase of the tasks, by randomly choosing a developer belonging to a CP. The assumption that all developers are equally committed to the project is considered, and the probability of choosing a developer is the same as the rest. However, in situations where different developers work for different lengths of time, the probability can be appropriately weighted. Once a particular developer is chosen and his or her estimates for the different phases of the task are obtained, a distribution is selected, from which a simulation of the effort required is generated. This effort is
recorded before the iteration continues to the next task.

The random work assignment model is designed to account for unusual cases, where the workload is not uniformly distributed. This scenario occurs, for example, when certain members of the CP are preoccupied with other tasks. This model does not take into consideration the personal preferences of individual developers.

\[ b) \quad \textit{Selected Work Assignments} \]

All classes that require coding begin with the planning stage. The driver of the CP starts out by determining the estimates for each phase of the different methods. After this, one of the methods is chosen and the planning phase is started. Once this is complete, the driver can move to the planning phase of another method, or can continue working on the design phase of the first method. Thus, at any given time, several open task-phases will exist that the developer can choose from. Since each phase cannot be segregated further, once a particular phase of a method has commenced, then it must be completed before the end of the developer’s shift. The choice of the task that will be taken on is influenced by shift timings and the complexity involved. This model considers these decision criteria and is more sophisticated to the random work assignment model.

The selected work assignment model uses a greedy policy of scheduling. When a developer has to make a choice of a task, all open tasks are arranged based on the estimates made by the developer, from very simple to very complex. Usually the easiest task is chosen first. Once a task has been selected, the simulation uses the estimates made by the developer to generate an associated distribution, and simulates the actual effort required for the completion of the task. If the task cannot be accomplished within the available time, and if the developer is unwilling to work beyond the allocated shift time, then that phase is abandoned and remains open for the next
developer. It is also possible for the developer to work overtime. The simulation algorithm is capable of locking a particular task, and preventing other developers in the next shift from working on this task. If the task is abandoned, then the lock is released, and the task can be selected by the incoming driver.

2) Defect Reduction Strategies

CPro incorporates multi-pronged strategy for defect reduction:

Artifact Review: These are design and code reviews, focused on quickly locating errors present in the code. While reading the code, the developer is provided with the logic behind the design, thereby facilitating the process of artifact review.

Test-driven Development: This method makes use of unit testing, and involves writing test cases that examine code artifacts in depth. Prior to coding a design, a minimum set of test cases are developed. As the design goes through the cycle of code-test-debug, any new defect that is identified is logged as a new test condition. Eventually, test cases are used to illustrate how the CP has evolved in its ability to understand the problem and the devised solution.

Heuristic Assignment: At any given point, there are multiple possibilities of tasks that can be performed by the driver of the CP, such as coding, adding a test case, and reviewing the code. CPro utilizes a set of heuristics for recommending a course of action based on the immediate productivity and the long term significance of knowledge sharing. For example, CPro might suggest a developer to review a code, which was not developed by him/her, with the intention of diffusion of knowledge throughout the CP. As a heuristic, CPro ties to match a particular task-phase to that developer who is likely to be the most productive for the particular task. Tools that incorporate the CPro business model will benefit from its look-ahead scheduling approach for obtaining local optimizations for assigning work to developers.
D. MultiMind

MultiMind is a concept development prototype collaboration tool that incorporates the concept of CPro. MultiMind aims to provide a rich operational environment that employs the 24-hr Knowledge Factory concept, specifically by allowing offshore developers to collaborate using the CP method. MultiMind is more enhanced in comparison to other collaborative engineering platforms such as DICE. A high-level architectural view of MultiMind is shown in Figure 5.

MultiMind is a tool that guides geographically distributed team members belonging to a CP along the CPro process. Business services concepts and IT technologies, such as Scrum meetings and XP respectively, are incorporated in MultiMind. The Scrum process is automated by an embedded project management system, which generates work summaries, thereby keeping developer intervention to a minimum. On sign-out, this system collects quick reports about the forthcoming progress of the task. During the next shift, upon signing-in, a scrum report is produced based on data obtained from the current state of the project and previously generated scrum reports.

MultiMind builds upon existing technologies as illustrated in the following subsections.
**Lifestream:** The 24-hour knowledge factory does not support real-time communication; as such, some form of proxy of the previous driver is needed. In order to address this requirement, MultiMind incorporates the concept of Lifestream. Lifestream is a monotonically increasing database that logs all objects that are accessed and all events that occurred, with the corresponding time of access or occurrence [4]. Lifestream enables developers to elicit the justification behind decisions that were made, when the original developer cannot be contacted in real-time. Thus, the developer can travel back in time across the Lifestream, by means of important document and artifacts accesses, web searches, posted messages and their corresponding replies, and relevant web pages visited, thereby enabling the current driver to replay the sequence of events that led to a particular decision.

**Speech Acts:** In MultiMind, an embedded discussion tool present in all project artifact interfaces utilizes speech act theory to tag all forms of communication according to their purpose. When a new message thread is created by a developer, it may be of the form ‘inquiry’ or ‘inform’. When a developer responds to a thread, he/she can choose from ‘response’, which answers a question, or ‘inquiry’, in order to pose a counter-question. MultiMind recognizes only three of the five high level speech acts: assertive, directive, and commissive [5].

**E. EchoEdit**

The simplest manner in which knowledge transfer can take place between two developers is through a one-on-one conversation. This is not possible for the 24-Hr Knowledge Factory scenario due to time differences and other reasons; hence, proper code documentation becomes very important. Approaches such as Literate Programming [6], as well as tools based on this concept (TeX, Cweb, METAFONT), are available to motivate developers to document their code in an efficient manner. The drawback with this approach is that it requires the developer to
possess a lucid writing ability, in order to create good documented code. More advanced solutions have been developed by the research group at the University of Arizona; one of them involves combining code documentation techniques with voice recognition capabilities [7].

The tool that integrates the code documentation and voice recognition is called Echo Edit; it is based on the composite personae model, and helps to minimize confusion among geographically dispersed developers. In its current implementation, EchoEdit is a plug-in for the Eclipse environment. This service was targeted towards the IBM Eclipse because the latter is an operational environment that many developers are already comfortable with. EchoEdit allows programmers to choose between using text comments and voice messages. For example, text comments could be used for syntactical or low-level details, while voice messages could be used for addressing high-level ideas. When another programmer enters this environment, voice annotations can be played back by selecting an icon displayed in the margin, similar to that in the PREP environment. As the voice message is played, the relevant block of code is highlighted. The purpose of the voice recognition software is to convert voice notes into text for easy indexing. EchoEdit supports three user actions: the recording of an audio comment, the playback of the audio file, and the conversion of the audio file into a text comment. When an audio comment is to be recorded, a recording event is generated and sent to the action handler, which initializes the audio recorder and starts listening for the record event. Two threads are created based upon this event: one to perform recording and the other to convert this voice to text. During recording, all data are stored into memory, and once conversion is completed, the text, along with the audio file is inserted back into the original file being worked on.

IV. CONCLUSION

The 24-Hour Knowledge Factory paradigm is a new business model that can enhance the
productivity of globally distributed software development teams. This paradigm incorporates emerging IT approaches as manifested in MultiMind and EchoEdit, along with state-of-the-art business methodologies as exemplified by CPro; these approaches, in turn, are founded on proven methodologies and technologies, such as Scrum, Extreme Programming, and Speech Acts. The successful integration of emerging technologies and business methodologies can greatly enhance the performance of globally distributed teams. By horizontally decomposing tasks between CPs and vertically breaking the phases within the CP, one can perform sophisticated tasks on a round-the-clock basis, with all individuals working exclusively in the day time of the respective countries. Just as the industrial revolution led to machines and ultimately to round-the-clock manufacturing operations, the 24-Hour Knowledge Factory concept could potentially lead to major innovations in the arena of software development.

REFERENCES


