Chapter 3.2 Hybrid Offshoring: Composite Personae and Evolving Collaboration Technologies

Nathan Denny
University of Arizona, USA

Shivram Mani
University of Arizona, USA

Ravi Sheshu Nadella
University of Arizona, USA

Manish Swaminathan
University of Arizona, USA

Jamie Samdal
University of Arizona, USA

ABSTRACT

Inspired by round-the-clock manufacturing, the 24-Hour Knowledge Factory endeavors to transform the production of software and other intangibles into a process of continuous development. While the concept of offshore software development is well established, few enterprises are currently able to develop the same code artifacts around the clock. We discuss the benefits of applying the 24-Hour Knowledge Factory to software development. We also present a representative scenario highlighting the problems of asynchronous communication in current offshore software development practices. Further, we introduce the notion of composite persona as a potential collaboration model within the 24-Hour Knowledge Factory and explain its ability to mitigate problems arising from communicating across cultures, languages, and time zones. Finally, we present a suite of new collaboration tools and techniques that are being developed specifically

for use by composite personae in the 24-Hour Knowledge Factory.

INTRODUCTION

Inspired by the paradigm of round-the-clock manufacturing, the concept of 24-Hour Knowledge Factory endeavors to transform the production of intellectual property and intangibles into a process of continuous development (Gupta & Seshasai, 2007). More specifically, we envision a 24-Hour Knowledge Factory as an enterprise composed of multiple sites that are evenly distributed around the globe. As the sun sets on one site, it rises on another; like an ongoing relay race chasing the sun, the day's work is handed off from the closing site to the opening site.

The benefits of implementing a software development enterprise as a 24-Hour Knowledge Factory are several. We expect to realize gains from significant compression in development schedules, faster turnaround time for localization and customization of existing products, and bug fixes and critical security patches released with greater celerity. However, we also admit that the challenges of establishing a 24-Hour Knowledge Factory are significant. More specifically, we anticipate technical challenges arising from asynchronous communication, which will likely be exacerbated by cultural and linguistic differences (Seshasai & Gupta, 2007). Non-technical challenges may grow from political and legal circumstances, and from difficulties in managing and operating in such a nontraditional business environment.

In this article, we discuss the current state of offshored and globalized software development and some of the underlying difficulties. Next, we introduce the concept of the composite persona. Finally, we discuss evolving collaboration technologies that support the concept of composite personae in the context of hybrid offshoring.

GLOBAL SOFTWARE DEVELOPMENT

Software development projects that involve multiple international sites have been a reality since the 1960s (Carmel, 1999, p. 17). However, those early efforts were relatively rare compared to the near ubiquity of contemporary global software development (GSD) (Gupta, 2007). By the year 2000, 200 of the Fortune 500 companies relied upon global software development teams or outsourced development to firms that use them (NASSCOM, 2000).

Our vision for the 24-Hour Knowledge Factory for software development is not synonymous with currently accepted GSD methods. Rather, the 24-Hour Knowledge Factory is a special case of GSD that has several unique properties while inheriting most of the problems of GSD. Subsequently, we present a brief survey of contemporary GSD methods to better define the problem of software development in the 24-Hour Knowledge Factory.

Convention and Practice

Engineering is the process of designing systems to solve problems. The activity of software engineering produces software systems to solve problems. Like nearly all engineering disciplines, software engineering advocates that large problems be recursively decomposed into smaller sub-problems and their corresponding sub-system solutions. Decomposition proceeds recursively until the problem-solution pair is tractable in both understanding the sub-problem and the resulting complexity of the sub-system. In modern parlance, the ultimate result of decomposition is a set of modules and classes (expressed in an object-oriented language such as Smalltalk, C++, or Java), where each class can be more-or-less completely understood by one person.

These modules and classes are then assigned to developers who then own that artifact throughout

Related Content