Mega Software Engineering

Katsuro Inoue[1], Pankaj K. Garg[2], Hajimu Iida[3],
Kenichi Matsumoto[3], and Koji Torii[3]

[1] Osaka University, Graduate School of Information Science and Technology,
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
inoue@ist.osaka-u.ac.jp
[2] Zee Source, 1684 Nightingale Avenue, Suite 201, Sunnyvale, CA 94087, USA
garg@zeesource.net
[3] Nara Institute of Science and Technology, Nara 630-0192, Japan
{iida, matumoto, torii}@is.naist.jp

Abstract. In various fields of computer science, rapidly growing hardware power, such as high-speed network, high-performance CPU, huge disk capacity, and large memory space, has been fruitfully harnessed. Examples of such usage are large scale data and web mining, grid computing, and multimedia environments. We propose that such rich hardware can also catapult software engineering to the next level. Huge amounts of software engineering data can be systematically collected and organized from tens of thousands of projects inside organizations, or from outside an organization through the Internet. The collected data can be analyzed extensively to extract and correlate multi-project knowledge for improving organization-wide productivity and quality. We call such an approach for software engineering **Mega Software Engineering**. In this paper, we propose the concept of Mega Software Engineering, and demonstrate some novel data analysis characteristic of Mega Software Engineering. We describe a framework for enabling Mega Software Engineering.

1 Introduction

Over the years, sometimes borrowing from traditional engineering disciplines, software engineering has adopted several methods and tools for developing software products, or more recently, software product families. For example, from hardware engineering the concept of specifying requirements before design and implementation have been useful for software engineering. A unique feature of software products, however, is that the end product has virtually no physical manifestation. Hence, composing or taking apart a software product has virtually no cost implications. As a result, software component reuse is a common practice for code sharing among multiple projects.

We posit that "sharing" among software projects can be extended beyond code or component sharing to more and varied kinds of "knowledge" sharing. Such sharing can be achieved using what we call *mega software engineering*. Instead of narrowly engineering a product, or a product family, an organization

F. Bomarius and S. Komi-Sirviö (Eds.): PROFES 2005, LNCS 3547, pp. 399–413, 2005.

can undertake the responsibility and benefits of engineering a large number of projects simultaneously. Examples of benefits that can accrue from such a perspective are: projects that share functionality can benefit from code sharing or reuse; experts in a particular implementation aspect can contribute their expertise to all projects that can potentially use that expertise (sort of like syndicated newspaper columnist or cartoonists); historical experiences of projects can be extrapolated to similar, newer projects to eliminate repeating process mistakes; and, 'outliers,' or projects with behavior deviant from the norm can be easily distinguished for rapid problem identification and resolution.

Many existing software engineering technologies remain focused on the individual project or programmer. For instance, code browsing tools typically allow a programmer to browse through single project code bases. Similarly, a navigation system might guide a developer utilizing data from her activities alone. While organizations can utilize global knowledge, for software reuse and other process improvements, an individual programmer or manager seldom enjoys the benefits of *mega or global knowledge*. Often, in large organizations its difficult for programmers to even discover projects related or similar to their own.

Prevailing organizational software engineering technologies for individuals are locally optimized to get local benefit for the individual developers or projects at most. They do not oversee global benefit and do not optimize the technologies using knowledge and software engineering data of other developers or other projects.

In modern times, the capacity, connectivity and performance of various networks ranging from local area network to the Internet are growing rapidly. Now, we are able to collect data from not only a single project, but *all* software development activities inside an organization (or company). If the organization has close relation to other software development organizations, as sub-contractor or co-developer, we can also collect software engineering data from the other organizations. A huge collection of Open Source software now exists on the Internet, which is sometimes a crucial resource for development projects. Such information is readily available via Internet tools.

Disk capacity and CPU power of recent computer systems are also rapidly increasing. Since vast disk space is available, we can archive project data at a detailed, fine granularity. Every change of a product can be recognized as a version and stored in a version control system. Every communication made among developers can be recorded. Not only single project data, but all project data spread over distributed organizations can be easily archived.

The collected mega software engineering data includes both process and product information. Various characteristics can be extracted by analyzing the collected data. Mining a single project data would be a relatively straightforward and light task. On the other hand, mining through mega data, say tens of thousands of projects, can be computationally expensive. Since now we have enormous computational power and memory space compared to, e.g., 10 years ago, however, such analysis becomes feasible. We may want to analyze, not only the organizational software engineering data, but also software engineering data