Heavyweight or Lightweight: A Process Selection Guide for Developing Grid Software

Cong Liu
Auburn University
2113 Shelby Center
Auburn, AL, 36849
(001)-334-844-6322
liucong@auburn.edu

David Umphress
Auburn University
3031 Shelby Center
Auburn, AL, 36849
(001)-334-844-8335
david.umphress@auburn.edu

ABSTRACT
Along with the advances in wide-area network technologies and computer hardware, the increasing availability of powerful computers and high speed networks are changing the high performance computing paradigm today. Recent related research has led to the emergence of a new paradigm known as grid computing. A major motivation of grid computing is to aggregate the power of widely distributed resources to provide services to users. Several unique characteristics of this paradigm make the development of grid software more challenging. In order to develop grid-aware applications, software engineering principles (e.g. software process, life-cycle model) for building conventional software must be adopted to cope with these challenges. Nowadays two kinds of software processes are frequently mentioned: heavyweight and lightweight. In this paper, we address the issues of comparing the two kinds of processes and selecting the better one for developing grid software. We choose to examine two well-known software processes: IEEE 1074 [6] and Extreme Programming [1], which respectively represents heavyweight process and lightweight process. They are first examined in detail and a comparison between them is conducted. Then we present a selection guide to choose the better methodology for a grid software project. Our analysis shows that lightweight software processes such as XP are more suitable for developing grid software.

Categories and Subject Descriptors
Software [**Software Engineering**]: Software Process

General Terms
Management, Human Factors, Languages

Keywords
software process, grid software, process selection

ACM-SE '08, March 28–29, 2008, Auburn, AL, USA.

1. INTRODUCTION
Metacomputing refers to computation on a cluster connecting different resources such as supercomputers, personal desktops, and storage systems [8]. It allows scientists and engineers to coordinate computational resources capable of sharing and analyzing complex simulations and data [9]. Recently, metacomputing evolved towards a more general paradigm, grid computing [4]. A major motivation of grid computing is to aggregate the power of widely distributed resources to provide services to users. This approach is innovative in the sense that it leverages existing IT infrastructure to optimize computing resources and manage data. Several distinguished characteristics make grid computing different from conventional parallel computing or cluster computing, which also make it challenging for software developers to design grid software.

1.1 Heterogeneity and Anonym
Unlike traditional parallel and distributed systems which usually run on homogeneous resources, in a grid resources are geographically distributed in multiple domains, not only the computational and storage resources but also the underlying networks connecting them are heterogeneous. The heterogeneity results in different capabilities for task processing and data access.

1.2 Scalability
Compared to traditional cluster systems, grid infrastructures have been deployed at larger and larger scales, with deployments incorporating tens of thousands of resources. This volume of resources raises scalability issues (e.g. resource discovery and monitoring).

1.3 Resource Dynamism
In a grid, resources capacity such as CPU speed, machine availability, network bandwidth and site loads vary significantly. There is no way to accurately estimate the performance of such resources. Such performance fluctuation might be the most important characteristic of Grid computing compared with traditional systems. Grid software should be able to be adaptive to such dynamic behaviors.

1.4 Computation-Data Separation
In traditional systems, codes to execute an application and required dataset, input/output data are always in the same local site. So the time for data transmission can be ignored. However, in a grid computing system, the data-intensive nature of

individual jobs means it can be important to take data location into account when determining job placement. Also given that the size of input and output data could be significant enough, their transmission time from executed site to local site should be considered. Grid software applications may also need to access, manage, store, and analyze data available in distributed data repositories.

The above characteristics make the development of grid software more challenging. In order to develop grid-aware software applications, software engineering principles (e.g. software process, life-cycle model) for building conventional software must be adopted to cope with these challenges. When an organization develops software, the criteria for choosing a process depend on many aspects, including skills, teaming, roles, techniques, standards, habits, size, and culture of the organization. In general, there are mainly two kinds of methodologies: heavyweight process and lightweight process. The traditional way to develop software is to use heavyweight processes, which develop software in a standard requirements-design-build procedure with well-defined processes. On the other hand, a current trend for developing software is to use lightweight methodologies, which apply short development iterations throughout the project life-cycle.

In this paper, we address the issues of comparing heavyweight process against lightweight process and selecting the better one for developing grid software. We choose two well-known software processes: IEEE 1074 [6] and Extreme Programming [1], which respectively represents heavyweight process and lightweight process. These two processes are first examined in detail. Then a comparison between them is conducted. Finally, we present a selection guide to choose the better methodology for a grid software project.

The rest of this paper is organized as follows. A review of grid software has been given in Section 2. Section 3 examines two chosen software processes in detail. Section 4 presents the proposed selection guide. Conclusions appear in Section 5.

2. GRID SOFTWARE-STATE OF THE ART

To date there are several commercial grid applications and middleware solutions that have been developed:

GridMATHEMATICA [5]: This application, which is developed by Wolfram Research, Inc, extends the famous Mathematica software by allowing the users to take advantage of remote Mathematica kernels installed on grid nodes to perform parallel computations. Moreover, it provides a quick way to set up and run large computations by offering a high-level programming language, a vast collection of mathematical algorithms, and parallel programming constructs.

Oracle Application Server 10g [10]: Oracle Application Server 10*g* offers a comprehensive solution for developing, integrating, and deploying your enterprise's applications, portals, and Web services. Based on a powerful and scalable J2EE server, Oracle Application Server 10*g* provides complete business integration and business intelligence suites, and best-of-breed portal software [10].

TurboWorx Enterprise [12]: TurboWorx Enterprise is a software product for companies in the life sciences, financial services, manufacturing, automotive, energy and aerospace industries. TurboWorx Enterprise provides the only integrated, end-to-end solution for creating, managing and accelerating computation-intensive and data-intensive applications and workflows in heterogeneous distributed computing environments and Grids [12].

Obviously, grid software development is still in its infancy, which is the major reason why the grid computing paradigm has not yet been widely adopted. It is believed that new trends and technologies such as grid computing will exacerbate the problem of dominating software complexity if developers still use traditional, heavy, and centralized software development processes, given that the current trend is toward evolvable, flexible, and decentralized software development processes. Emerging applications such as grid software will become driving factors for developing new software engineering approaches.

3. SOFTWARE PROCESS CANDIDATES

Since our goal is to compare heavy-weight against light-weight processes, we choose IEEE 1074 [6] and Extreme Programming [1] as the targeted processes. IEEE 1074 is a typical heavy-weight process while Extreme Programming is considered as a light-weight process.

3.1 IEEE 1074

IEEE 1074 is a standard for generating the process that governs software development and maintenance for a project. This standard requires the definition of a user's software life cycle and shows mapping into typical software life cycles [6]. It applies to the management and support activities that continue throughout the entire life cycle, as well as all aspects of the software life cycle from concept exploration through retirement [6]. The goal of IEEE standard 1074 is to establish a common framework for developing software life cycle models. The standard divides the set of activities into processes, and processes into process groups, as shown in Table 1 [7].

Table 1. Software Process Groups in IEEE 1074

Process Group	Processes
Life Cycle Modeling	Selection of a Life Cycle Model
Project Management	Project Initiation, Monitoring, and Control; Software Quality Management
Pre-development	Concept Exploration System Allocation
Development	Requirements; Design Implementation
Post-development	Installation Operation and Support; Maintenance Retirement
Integral Processes	Verification and Validation; Software Configuration Management; Documentation; Development Training

Below is a summary of the activities and work products of each process group.

Life Cycle Modeling: For a particular project, the manager customizes the selection and sequencing of the required activities [7]. Life cycle models describe the interrelationships between software development phases [11].

Project Management: The goal of project management is to initiate, control and manage the software development throughout

the life cycle. The plan of project management is documented during the project initiation process, and the plan is updated to reflect changes throughout the project [11].

Pre-Development: Before software development, the clients identify the ideas or needs, which help the manager establish the initial system architecture. The needs may be addressed through [11]: (*i*) greenfield engineering — a new development effort (*ii*) interface engineering — a change to the interface of an existing system (*iii*)reengineering — a software replacement of an existing business process. A Problem Statement or Statement of Need (IEEE 1074) describes the business requirements to be addressed by the project [11].

Development: The development process includes all the process directly involved in the construction of the system.

Post-Development: The post-development process is after the development phase. The aim of this process is to install, maintain, and give operational support to the developed software product.

Integral Processes: Integral processes mainly aim at verifying and validating the software, managing configuration, and training document development. Software verification and validation are defined as:

- verification — ensuring that the system models comply with the specification.
- validation — ensuring that the system addresses the client's needs.

3.2 Extreme Programming (XP)

Extreme Programming (XP) [1] is an agile and disciplined methodology to software development. XP is one of the most well-defined software processes since it generally consists of twelve interconnected practices [1]. XP has been used by many software companies around the world [1]. As stated in [1], "The basic advantage of XP is that the whole process is visible and accountable. The developers will make concrete commitments about what they will accomplish, show concrete progress in the form of deployable software, and when a milestone is reached they will describe exactly what they did and how and why that differed from the plan. This allows business-oriented people to make their own business commitments with confidence, to take advantage of opportunities as they arise, and to eliminate dead-ends quickly and cheaply.

XP becomes successful because it is designed to deliver the software according to customers' needs. One reason is because it is founded around small releases with iterations of a couple of weeks during which developers implement customers' requirements [1]. Moreover, software features are specified by both customer and developer. By doing so, the customer can be involved in planning and is able to answer developer questions in a timely manner, which helps produce a piece of software closely tailored to their needs [1]. Team work is also an emphasis to XP. Customers and developers are both part of a team dedicated to delivering high quality software. XP defines a simple, but effective way to enable groupware type of development. Essentially, it improves a software development in four ways: communication, simplicity, feedback, and courage [1]. XP developers always have frequent communications with their customers in order to keep their design correct, simple and clean. They get feedback by customer acceptance tests as early as possible. The initial software is also delivered to the customers as early as possible. Doing so enables developers to implement changes as suggested by customers.

3.3 Comparison

IEEE 1074 is a formal, disciplined software engineering methodology. It is considered as a heavyweight process since it requires a heavy degree of management effort, quality assurance reviews, and rigid procedures that developers should follow. It defines and documents a strict and consistent set of requirements. It is assumed that users know the requirements beforehand and they will not change over time. Moreover, heavyweight processes assume that it costs much to fix any error.

On the other hand, XP is considered as a lightweight software process, which is also known as agile process. XP is less structured. Developers are encouraged to skip comprehensive initial requirement analysis. In this approach, no design documentation or models need to be maintained in addition to source code. It produces frequent incremental versions based on the basic functioning software. The experience learned from each increment will be applied to later increments. Moreover, lightweight processes recognize the importance of people which are considered as the primary drivers of the project success. A summarized comparison between heavyweight and lightweight processes is shown in Table 2 [13].

Table 2. Heavyweight process vs. Lightweight process

Heavyweight	Lightweight
Predictive	Adaptive
Process Oriented	People Oriented
Design Oriented	Construction Oriented
Document Oriented	Artifact Oriented
Requirement Containment	Adaptation

4. SELECTION GUIDE

Since there are clear differences between light and heavy software processes, several factors involve in determining the relative suitability of lightweight or heavyweight methodologies in a grid software project situation. In the following sections, each factor is examined in detail to show how it affects the decision and which methodology is more suitable for developing grid software.

4.1 Scalability and Heterogeneity

For traditional computing systems such as single computing system and parallel computing systems, scalability is not a concern because the number of resources is usually small. More importantly, all the resources are homogeneous in nature, from physical devices to system software. The structure of the computation can be predicted beforehand, which makes the application design easy. For instance, traditional high performance software is developed for a specific supercomputer whose features are known beforehand, e.g., the computing capacity. However, scalability is a concern for grid computing environment. Grid infrastructures have been deployed at larger and larger scales, with deployments incorporating tens of thousands of resources. Such a large number of resources may demonstrate a heavy degree of heterogeneity, ranging from physical devices to system software and scheduling policies. Thus, grid software applications will run in a wide range of

environments, which results in the impossibility to predict the computation structure. Rather than using heavyweight processes which are predictive in nature, it is better to apply a lightweight process such as XP in developing grid software to address the scalability and heterogeneity issues. By using lightweight processes, developers do not need to have detailed design plans. It is also inapplicable to have detailed design plans for grid computing environment. Instead, developers may use iterative development to rework a part of a system once a new computation structure is added by customers. The advantage of using lightweight processes here is that new activities can be added if the targeting hardware configurations change, or more generally the goal cannot be achieved using the original planned activities

4.2 Dynamicity

Unlike traditional computing environment, dynamic behavior of the computation is a challenging characteristic of grid computing. It can not be assured that during the course of computation all of the system characteristics remain the same, e.g., the latency and network bandwidth can change widely. Moreover, since the resources in a grid are spanned at multiple administrative domains, there is no single authority in charge of the system, so that different authorization mechanisms and scheduling policies should be considered when developing grid software. It is difficult and maybe impossible to predict customers' requirements since in many cases even customers are unable to predict the dynamic behavior of the computation. Obviously it is more beneficial to use lightweight software processes to develop grid applications that run in a dynamic and loosely defined environment. By doing so, developers can give rapid response to changing software requirements and dynamic behaviors of the computation.

4.3 Complexity

New trends and technologies are exacerbating the problem of dominating software complexity. The grid will make this trend even worse with its need for increased dynamism and decentralization. Even the underlying network infrastructure is evolving from wired to wireless [4]. This level of dynamism brings additional complexity to software development. In order to reduce the impact of additional software complexity, flexible, evolvable, and decentralized software processes are definitely more preferable. Actually the current trend for software development is towards this direction [1].

4.4 People Factors

Experience and skill of a project team is a key factor when selecting the software development process [2]. Since grid computing is an emerging paradigm within the field of parallel and distributed computing, most grid software developers have expertise in designing parallel applications. Thus, we argue that lightweight methodologies can work in a grid software development environment where project team has skilled developers. Lightweight software processes focus on the talents and skills of individuals and molds process to specific people and teams, not the other way around [2]. The domain experts are able to realize the growth of software in the earliest stages [2]. Moreover, experts can give rapid feedback on the implications to the user of their design choices. By doing so, the cost of moving information and the elapsed time between making a decision to seeing the consequences of that decision can be reduced [2].

4.5 Summary

In conclusion, the following factors make lightweight software processes such as XP more favorable than heavyweight processes:

- Scalability and heterogeneity of the computing environment
- Requirements change very often
- Additional software complexity brought by grid applications
- Senior developers

5. CONCLUSION

In this paper, we deal with the problems of comparing heavyweight process against lightweight process. We propose a selection guide to choose the better methodology for grid software development. We examine two well-known software processes: IEEE 1074 and Extreme Programming in detail, which respectively represents heavyweight process and lightweight process. Our analysis shows that lightweight software process is preferred for developing grid software.

6. REFERENCES

[1] Beck, K.. 2000 Extreme Programming Explained: Embrace Change. Addison Wesley.

[2] Cockburn, A. and Highsmith, J. 2001 The People Factor. In Software Management, 131-133.

[3] Foreman, J., Gross, J., Rosenstein, R., Fisher, D., and Brune, D. 2007 C4 Software Technology Reference Guide-A Prototype. Handbook CMU/SEI-97-HB-001.

[4] Foster, I. and Kesselman C. 2003 The Grid2. Morgan Kauffmann Publishers.

[5] GridMATHEMATICA Introduction [Online]. Available: http://www.wolfram.com/products/gridmathematica [Accessed Nov. 5, 2007]

[6] IEEE 1074 Description [Online]. Available: http://standards.ieee.org/reading/ieee/std_public/description/se/1074-1997_desc.html [Accessed Nov. 10, 2007].

[7] IEEE 1074 Standards for Developing Life Cycle Processes [Online].Available: http://www.augustana.ca/~mohrj/courses/2005.fall/csc220/lecture_notes/lifecycle.html [Accessed Nov. 10, 2007].

[8] Metacomputing Introduction [Online]. Available: http://en.wikipedia.org/wiki/Metacomputing [Accessed Oct. 4, 2007].

[9] Metacomputing Capabilities [Online]. Available: http://www.hpti.com/tblTopicsHomeTemplate.asp?ID=114 [Accessed Nov. 8, 2007].

[10] Oracle Application Server 10g [Online]. Available: http://www.oracle.com/technology/products/ias/index.html [Accessed Nov. 8, 2007].

[11] Software Life Cycle Models [Online]. Available: http://www.levela.com/software_life_cycles_swdoc.htm [Accessed Nov. 8, 2007].

[12] TurboWorx Enterprise Introduction [Online]. Available: http://www.gridtoday.com/04/0419/103077.html [Accessed Nov. 8, 2007].

[13] Umphress D. 2007 Process Redux. COMP 6700 Software Process Course Material, Auburn University.