Global eXtreme Programming, a Software Engineering Framework for Distributed Agile Software Development

Ridi Ferdiana[1], Lukito Edi Nugroho[1], Paulus Insap Santoso[1], Ahmad Ashari[2]

[1]Department of Electrical Engineering and Information Technology, Gadjah Mada University
[2]Department of Computer Science and Electronics, Gadjah Mada University
Yogyakarta - Indonesia
ridi@te.gadjahmada.edu, lukito@mti.ugm.ac.id, insap@te.gadjahmada.edu, ashari@ugm.ac.id

***Abstract*:** Distributed software development or multi-site development may increase the development life cycle. This happens predominantly due to cross-sites communication and coordination difficulties, which have made software development an even more challenging task. The need of a simple and formal framework has been discussed through the concept of Global Software Development (GSD). However, the redundancy of development effort makes GSD need to be streamlined. This paper will make an effort to create a conceptual framework based on the well-known agile method called eXtreme Programming with the existing GSD process. The framework is called GXP, and it provides a formal framework that dedicated for distributed software development.

***Keywords*:** Distributed Software Development, Global Software Development, eXtreme Programming, Software Engineering Framework.

1. Introduction

Distributed software development comes with several types and level [7]. It can be global (different places, different organization), inter-organization (different places, same organization) or open source project development (different places, no organization). Holmstrom et al. [8] refines the types of distribution models, which are geographical, organizational, temporal, and stakeholder's boundaries. Those types are escalated through two main entities, which are people and its artifacts.

People challenges in distributed software development are caused by the dispersion of people among several locations. This is related to communication and cooperation between people. For example, the physical distance between people limited in formal communication. This might lead to a lack of information in a project. Extended communication effort, on the other hand, can lead to information overhead or too much discussion rather than develop the software. Thus, the software project should aim at a trade-off between lack information and information overhead.

Communication problems arise because several factors, which are social and cultural differences between distant sites [5], time zone separation [9], perceives distance within members of the given stakeholder group (Evaristo and Scudder, 2000), and different motivation background [7]. Those factors addressed by providing communication tools like Computer-supportive and Collaborative Work (CSCW) [6]. CSCW has been discussed long time to enhance communication in the distributed team. The issues within CSCW implementation are about the learning curve, the amount of such work is increasing, and the unusual way to communicate.

Artifact challenges are caused the need of task distribution, the level of synchronization, decision making, skills and knowledge of each member. Although it is not related directly with the people, this challenge should be answered through the organizational and software engineering process.

The rest of paper is organized as follows. First, we discuss the existing solution in the multi-site development. Secondly, we describe our research approach to synthesize the formal framework called GXP. The research then reports the result by a discussion of the implication of those results, limitation of the work and future research directions.

2. Current Research Solution

Distributed Software Development, Collaborative Software Development (CSD), and Global Software Development (GSD) process is related processes, which make an effort to manage artifacts, people, and product through software engineering disciplines for multi-site software development.

Distributed Software Development (DSD), Collaborative Software Development (CSD), and Global Software Development (GSD) are termed that interchangeably used to describe a software engineering process solution to overcome software engineering limitation in the distributed development model. Although those terms are used interchangeably, those terms have a different point of view to solve the problem.

DSD is the generic term which is used to describe management, development, and maintenance of software that being geographically distributed across the globe [13]. DSD research focuses in non-technical issues that related with distributed software development like coordination, awareness, and dependency management. DSD provides a problem-solution model that captured from field reports and adapted to the other's problem which has same context. For example, manufacturing organization is creating their production monitoring software through distributed software development model. The organization creates patterns and practices from their experience. Those patterns and practices afterward are adopted by different organization in the

different country to create the similar software.

CSD is another term that describes a set of tools that strengthen collaboration in software development [13]. CSD researches to provide several alternative tools that help collaboration in distributed software development. CSD is created based on CSCW concept, which is solving the distant and coordination problem through tools, the difference is that CSD provides a specific tool for distributed software development.

GSD is a contemporary form of software development undertaken in globally distributed locations and facilitated by advanced information and communication technology (ICT), with the predominant aim of rationalizing the development process [12]. GSD offers theoretical process to handle distributed software development. As a software engineering process, GSD offers planning strategy, organization structure, and progress control and monitoring. Rather than others approaches or terms, GSD provides more sufficient process and workflow in the software engineering framework.

Many of the GSD implementations are happened in an organization that has a software project in enterprise level. Company like Lucent, Microsoft, Philips, and Siemens is a small sample that done GSD for their software products. Nowadays, GSD is also happening in personal, small scale (1-6 people), and community software development. For example, a person can get a software development project from a freelance website, small group can get a client from different region or countries through an internet project bidding and community can build software like open source software through Sourceforge, or Codeplex system. Those opportunities give a clear view that GSD needs to be simplified.

This research has been motivating factors to deliver a simple approach in distributed software development. Simplification and effectiveness are the legacy problem for every software development. Therefore, many research focus in simplification and effectiveness. Agile process is one of the software engineering processes, which are dedicated to simplify the process and give a center of attention in delivering working software. Agile community through theirs manifesto promises a simple and standard way to build a software. However, agile process is fitting in collocated software development since the process extremely needs direct interaction without a distant.

Based on those hypotheses the research sees an opportunity to integrate the existing GSD process with the agile method. In a specific view, the research will choose one of the agile methods called eXtreme Programming (XP). XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements [3]. During its execution XP gets optimistic feedback to implement in personal software development [1], and also enterprise development [4]. Those researches provide factual information that XP has been sufficient in scalability aspect. Therefore, XP is chosen as an agile process in this research and will be integrated with the GSD in this research through a formal framework design process.

3. Framework Design

3.1 Software Engineering Framework

The high level of the framework describes the entities, building block and its relation. A framework is reusable design that requires components to functions. To create a framework, a researcher should provide the components required by the framework. In order to do this effectively, the framework-component interfaces must be specified so the researcher knows what expectations the framework makes about the component, and so the components can be verified against these assumptions. The framework itself can be designed to several points of view such as technical function, software engineering, and domain-specifics process.

A component is a software unit (for example, example module, set of function, or a class) or data unit that has a defined interface for which the component provides an instantiation. As framework entities, component should be easy to understand through its interface. To do so, component in a framework can be anything includes the data or non-software component.

To understand and use a framework, the framework must be specified, the engineer must understand what framework does, what components must be provided to instantiate the framework, and how to use the framework. In order to discover this information, the research defines the framework specification that includes three tasks.

1. Specify the syntax of the framework
2. Specify the semantics of the framework
3. Specify the framework component-interfaces.

The syntax of the framework specifies how the engineer uses the framework, that is, how the framework becomes a system of a part of the engineering system. For example, if a framework provides a test driven development function, the engineer knows how the function is executed. The semantics of the framework specifies what functionality it provides. The framework-component interfaces define the syntax of the components.

The result of the framework creation from the followed processes is called as Global eXtreme Programming framework (GXP). The research result is said as an unadjusted framework. Unadjusted term means that the framework needs several details, implementation, and assessments.

3.2 GXP Framework syntax

GXP framework syntax states how the distributed software development is executed. The framework has three main operations that are processed, method and tools operation. The process is the first operation that is executed to define an asset value and principles of the software development in GXP model. After the team understood the process, the team can learn the comprehensive daily execution of GSD through method operations. This operation makes the team understood the daily how-to and technical practices of the development execution. The last syntax of the framework is a tool. The tool gives productive understanding and real experience execution through the supporting tool. Figure 1 describes the three framework syntax in a sequential block

diagram.

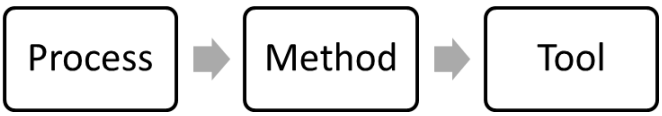

Figure 1. Framework syntax block diagram

GXP Process describes base framework activities that exist in global software development. Software development life cycles, software estimation, organization structure, and quality assurances are components that described in GSD process. GXP process works as umbrella activities for relating building block like GXP method or GXP tools.

GXP method describes the practical "how-to" in order to build software in the distributed model. GXP method includes software development practices, value, project artifacts, and project management practices. GXP method works as a guidance or step by step execution to solve distributed software development.

The last but not least is GXP tools. It is designed to give productivity tools for distributed software development. It consists of infrastructure requirements supports, communications tools, and collaboration software. The tools work as cross cutting building blocks that support GXP process and GXP method.

3.3 GXP Framework semantics

GXP semantics explicitly provides the component that contains within the operation. This step explicitly identifies components and its functionality. The aggregation of the result provides what the framework does and decision information for a software engineers to do properness and suitability of the framework based on their need.

Process semantics contains several components that related with the groundwork activities before the project is executed. Therefore, the process component displays several abstract components that related with the process. The research identifies the process semantics are preparation activities that need to be done before the project executed, the lists come up as follows.

1. Quality controls focus, since the software engineering purposes are to deliver good quality software. The quality controls focus dedicates several principles to cover good characteristic software in GXP framework. This component should be a first component to comprehend by the team.
2. Software development life cycle, this component acknowledges the SDLC of GXP. The SDLC describes the phases of the project that need to covers in GXP.
3. Software estimation, this component will allow the engineer to estimate the complexity the software by looking at the technical and experience factor. The early estimation will give the engineer better understanding about feasibility of the project based on the budget, time, and resources.
4. Team organization, this component is the last component that needs to learn in order create the jell team in GXP. Several roles are introduced and job descriptions are described.

Method semantics describes the technical "how-to" in terms of the project execution. This semantics only executed if the project is agreed in resources, budget, and time. Several components are identified as follows.

1. GXP values and principles, this component describes the GXP values and principles. This component covers what the team needs to understand the framework mindset.
2. GXP practices, this component describes the practical action which is believed to be more effective at delivering a particular outcome. GXP Practices also be defined as the most efficient (least amount of effort) and effective (best results) way of accomplishing a task.
3. GXP artifacts, this component describes any kind of tangible product that produced during the development of software. GXP artifacts consist of several document, template, and knowledge base to execute GXP software development.
4. GXP project management, this component describes the discipline of planning, organizing, and resource management to bring about the successful completion of specific project goals and objectives based on GXP point of view.

Tool semantics described the support of the software and infrastructure to improve the development productivity. The components are described as follows.

1. The communication Tools. This component discusses several communication tools and sample that appropriate to support GXP framework.
2. Infrastructure support. This component recommends some of the infrastructure specification to develop GXP environment.
3. Collaborative workspace. This component provides proof of concept recommendation by delivering suite tools for GXP framework projects.

Through the three framework syntax, the research proposes eleven components that dedicated to support the framework purposes. Those components will be rearranged in its execution through an interface identification step.

3.4 GXP Framework Interface

GXP framework component interfaces discuss the input and the output of the component that already defined in previous step. Table 1 describes the interface for GXP component.

Table 1. GXP Framework component interface

Component	Input interface	Out Interface
Quality control focus	Market intent and project vision	User stories baseline
SDLC	Project scope and time	Schedule plan
Software Estimation	User stories, technical factor, and experience factor	Software complexity, user story point, and effort rate
Team Organization	Effort rate, software complexity	Team profile
Values and principles	Team profile	Team mindset
Practices	Team mindset	Action plan
Artifacts	Schedule plan, action plan	Project artifact
Project Management	Schedule plan, action plan	Tracking execution
Communication Tool	Crosscutting interface	Crosscutting interface
Infrastructure Support	Crosscutting interface	Crosscutting interface

Collaborative workspace	Crosscutting interface	Crosscutting interface

Input interface is an input for the component, after the process of the component the output interface delivers the output channel for the process result. The process result can be processed further through another component. Several components also work as crosscutting interface, crosscutting interfaces describes a component that communicates intensively as a support component.

4. GXP framework prototype

4.1 Instantiation of the GXP Framework

Instantiation of the framework describes a step to pull the component into high level architecture. This step consists of two tasks, which are specifying the system properties and providing concrete components. The system properties define the system that implements when the framework is instantiated with all the components it needs. One of the complexities in framework use ensuring that the framework is applied to areas for which it is suitable. Stating system properties also allows for verification tasks to be undertaken ensuring that the framework with the instantiated component satisfies these properties. The second task is to provide concrete components that are instantiations of the components specified as part of framework specification. It can be concreted as file, document, or even the codes.

GXP framework properties cover several key points such as follows.

1. The input of the framework is an initiative to execute the software project distributed.
2. The output of the framework is an effective approach to manage and track the distributed project.
3. The process of the framework follows the XP phases which are exploration, planning, iteration, production, and maintenance.
4. The feedback mechanism of the framework which are executed through user acceptances test and production release feedback.
5. The environment of the framework is an environment where the ICT infrastructure like broadband the internet exists.

The concrete of the framework can be described through artifacts and the tools that support the framework. The artifact will exist in every phase of the project as main deliverables of the framework execution and tracking. The tool works as artifacts placeholder to manage, collaborate, and track the artifact.

4.2 GXP in the big picture

In a big picture, GXP provides overall building block of the framework architecture. The big picture covers the component, the dependency between components, and the semantics of the framework. As mentioned before, there are three building blocks, which are processing building block, method building block, and tool building block.

Process building block discusses about the starting point where the GXP should be started by the team to follow. It's provided several abstract values and principles that can be done by the team before the project starts and when the project is executed. The technical how to of the process is described in method building blocks, method building block discusses the implementation of the process through several values, principles, practices, and artifacts. Both process and method are supported by the existence of the tools as crosscutting layers that can communicate in the term process or method.

The proposed framework semantics and components provide probabilities to use GXP framework in several models of implementation. For the people who want to implement the GXP, it is recommended to start by seeing the project condition. GXP proposes three project conditions, which are remote model execution, virtual team model, and distributed team model.

Remote execution model is based on the situation, when the distributed context is only happened between team and the client. This execution model usually happens in a small scale project and less urgency. The team is on the same place while the client is separated by distance. The characteristic of this execution model is.

1. There is no management difference since the team is still one site. The working process can be like XP team with the small tweak from GXP framework in terms of tools and method.
2. Few number of the team members with the small complexity inside the project. The kind of projects that worked in this execution model most likely small in dependency and urgency for the client.
3. Project length between one to three months with small iteration for two weeks or less.

Virtual execution model is based on the situation where the distributed context is happen between the client and inside the team. The difference between the remote model executions is the location the team that also separated. The portion of the team is onsite with the client while the rest is separated. The others characteristics of virtual team model described as follows.

1. Management treats the team as a single virtual team. The team will have one single management but separated by distances.
2. Small to the medium numbers of the team with the medium complexity inside the project. The project typically is related with the core business for the client
3. The project length might be in three to twelve month with small medium iteration length between one into two months.

Global execution model is based on the situation where the distributed context is happening globally. The main characteristic of this execution model is huge numbers of team member's. The member might be different time zone and culture. The main characteristic of this execution model as follows.

1. Management divides clearly between central management and site management as separated instances.
2. Enterprise scale application with medium to large team numbers. The project typically is related with a core product that will be sold in the global market with different language, need, and culture.

3. The project length might be multiyear with medium to long iteration model between bimonthly into quarterly iteration.

Several execution models will make the team behave differently in terms of working model and initial configuration. Table 2 provides information about the recommended configuration for the team which wants to choose GXP as a distributed software development framework.

Table 2. GXP classification and initial recommendation

Recommendation	Remote execution model	Virtual execution model	Global execution model
Milestone length	1-3 months	3-12 months	12 months or more
Iteration length	1-2 weeks	4-8 weeks	8-16 weeks
Team model[12]	Hub-to-Spoke	Hub-to-Hub	Mesh (fully distributed)
Project management	Onsite	Offshore and centralized	Offshore and distributed

4.3 Using GXP Framework

Framework is a baseline for the team that wants to extend customized distributed development implementation without losing the essential of the framework component that need to be defined. Using GXP framework can be done through seven simple steps. These steps can be followed by the development team which wants to implement the framework for their distributed development need. The seven checklists are described as follows.

1. Verifying GXP Framework and project appropriates.
2. Choosing GXP classification.
3. Composing GXP team.
4. Learning GXP values, principles, practices, and artifacts.
5. Estimating the project.
6. Preparing the infrastructure and tools.
7. Executing and monitoring GXP through its SDLC.

GXP is not a silver bullet, there are several constraints where the framework becomes ineffective based on several conditions and constraints. The team should investigate the suitable project based on the discussion of the team. The first step is choosing the appropriateness between the project and the framework. Based on the GSD and XP concepts, GXP frameworks work appropriate when the several conditional.

1. The project at least partially distributed. Adopting GXP in the onsite project will make it work but not as efficient as just like adopting XP in the onsite project.
2. The project is not real time project or crucial project. Building software for the earth quake monitoring or other extensive resources project will not be appropriate with GXP.
3. The business culture of the client support working distributed. Some clients have a culture to appreciate the development team always onsite when this situation happens; the GXP framework is not effectively useful.
4. The business culture support continuous communication and improvement. Single meeting specification or the business of the offshore client make the GXP project won't work best when they do less communication.
5. The business culture support less document work but running software instead. GXP is dedicated to work in balance between document and software.

If the project is suitable based on those checklists, the team can start to choose the GXP as a framework and select the appropriate model of GXP such as remote, virtual, or global execution. Choosing the GXP model will give the team the initial configuration and recommendation of the project.

The initial configuration will help the team to choose the proper team scale and initial planning. GXP encourages the team follows XP roles such as the coach, developer, tracker, project manager, tester, and domain expert. These roles will be composed in a central team or a site team. Central team is a team that manages one or more site teams. In the small team, central team can be a team that has the management role for the overall team.

Just like in XP, team organization and structure creation is created several phases in the software development life cycle (SDLC). As the iteration phase begins, both site team and central team should be established and ready to build test and codes. Central team works as a monitoring team for the site team during the iteration, production, and maintenance phase. Ideally, both central team and site team has the same team composition. For example, central team can only have a management members like a project manager and domain expert, and site team has the rest technical team. There are no rules of thumb the team composition, since the situation is mostly driven by the resources.

After the team is composed, the team can start the learning curves by understanding the values, principles, practices, and the related GXP artifacts. Values, principles, and practices are lent from the XP methodology. However, several practices like pair programming and standup meeting is replaced with the equal interaction like online interaction through instant messaging or video conferences. In this step, it will be the good idea for the team to execute short term coaching workshops that learn principles, practices, and values.

The team can do estimating Estimation defines the approach to do better planning. It is executed in exploration phase. The result of the estimation technique is a quantitative result that works as a baseline for the software complexity measurement, user stories points, and effort rate. GXP estimation technique will answer how big or small is the project in a quantitative model.

The estimation result can make the team have clear visibility how complex the project, and how long it will take. After the estimation activities, team can implement the infrastructures that needed by the team. It means every communications and collaboration software should be installed, configured, and ready to use by the team. This step is including, training and testing the infrastructures between of sites.

After the infrastructures ready, the team can start the full of software development lifecycles. There are four phases in GXP SDLC. Figure 2 shows the GXP SDLC. It defines the overall lifecycles based on the well-known eXtreme Programming development life cycle. XP method adopts exploration, planning, iteration, production, and maintenance as a phase in a development cycle.

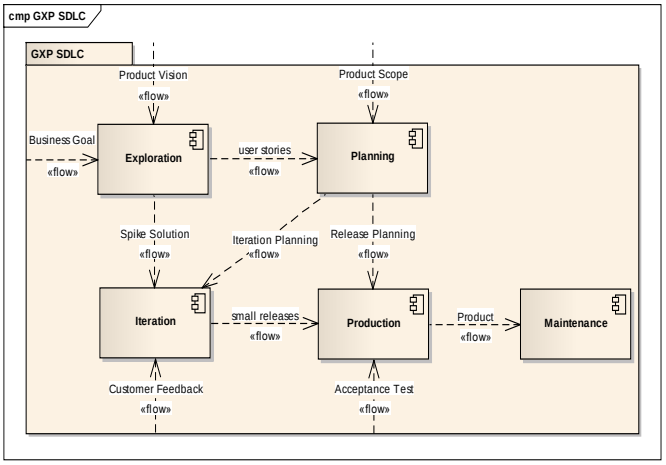

Figure 2. GXP Software Development lifecycles

Every phase has several inputs and the output results of the phase. The execution of each phase is not really sequential and one at a time. Except the exploration phase, the rest phases on the GXP SDLC are iterative and incremental. Table 3 describes the input and the output of every phase and also shows the workflow of the software development life cycle.

Table 3. GXP SDLC input and output process

Phase	Input	Output
Exploration	Product Vision	User Stories
	Business Goal	Spike Solution
Planning	Product Scope	Iteration Planning
	User Stories	Release Planning
Iteration	Spike Solution	Small Releases
	Iteration Planning	
	Customer Feedback	
Production	Small Releases	Product
	Release planning	
	Acceptances Test	
Maintenance	Product	Fixed Product

Based on the phases, the team should also prepare the artifacts for each phase. The artifacts are a useful documentation that provides a tracking history for the product evaluation that developed by the team. Unlike XP that mostly depends on the codes and comments, GXP should prepare the artifacts as a tool to exchange the knowledge between sites. Furthermore, the team should create just enough document and others resources so that the artifacts are not redundancy in terms of size and numbers.

5. Conclusion and Future Work

In the multi-site software development, software process like CSD, DSD, and GSD have begun to introduce.

The paper limits the discussion as a framework that can be fulfilled later in the specific aspect like values, principles, practices, and implementation. As a further work, the framework component should be detailed with the how an aspect that gives a framework user a detailed action that needs to be fulfilled. On the other's side, the framework also needs to be evaluated through several assessments and case studies implementation.

References

[1] Agarwal, R. and Umphress, D. 2008. Extreme programming for a single person team. In Proceedings of the 46th Annual Southeast Regional Conference on XX (Auburn, Alabama, March 28 - 29, 2008). ACM-SE 46. ACM, New York, NY, 82-87.

[2] Bass, M. 2006. Monitoring GSD projects via shared mental models: a suggested approach. In Proceedings of the 2006 international Workshop on Global Software Development for the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06. ACM, New York, NY, 34-37.

[3] Beck, K. 1999. Extreme Programming Explained. Addison-Wesley.

[4] Cao, L., Mohan, K., Xu, P., and Ramesh, B. 2004. How Extreme Does Extreme Programming Have to Be? Adapting XP Practices to Large-Scale Projects. In Proceedings of the Proceedings of the 37th Annual Hawaii international Conference on System Sciences (Hicss'04) - Track 3 - Volume 3 (January 05 - 08, 2004). HICSS. IEEE Computer Society, Washington, DC, 30083.3.

[5] Evaristo, J. R. and Scudder, R. 2000. Geographically Distributed Project Teams: A Dimensional Analysis. In Proceedings of the 33rd Hawaii international Conference on System Sciences-Volume 7 - Volume 7 (January 04 - 07, 2000). HICSS. IEEE Computer Society, Washington, DC, 7052.

[6] Grudin, J. 1994. Computer-Supported Cooperative Work: History and Focus. Computer 27, 5 (May. 1994), 19-26.

[7] Gumm, D. C. 2006. Distribution Dimensions in Software Development Projects: A Taxonomy. IEEE Software. 23, 5 (Sep. 2006), 45-51.

[8] Holmstrom, H., Conchuir, E. O., Agerfalk, P. J., and Fitzgerald, B. 2006. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In Proceedings of the IEEE international Conference on Global Software Engineering (October 16 - 19, 2006). ICGSE. IEEE Computer Society, Washington, DC, 3-11.

[9] Lanubile, F., Damian, D., and Oppenheimer, H. L. 2003. Global software development: technical, organizational, and social challenges. SIGSOFT Softw. Eng. Notes 28, 6 (Nov. 2003), 2-2.

[10] Mockus, A. and Herbsleb, J. 2001. Challenges of Global Software Development. In Proceedings of the 7th international Symposium on Software Metrics (April 04 - 06, 2001). METRICS. IEEE Computer Society, Washington, DC, 182.

[11] Pilatti, L., Audy, J. L., and Prikladnicki, R. 2006. Software configuration management over a global software development environment: lessons learned from a case study. In Proceedings of the 2006 international Workshop on Global Software Development For the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06. ACM, New York, NY, 45-50.

[12] Sangwan, R., Bass, M., Mullick, N., Paulish, D. J., and Kazmeier, J. 2007. Global Software Development Handbook (Auerbach Series on Applied Software Engineering Series). Auerbach Publications.

[13] Sengupta, B., Chandra, S., and Sinha, V. 2006. A research agenda for distributed software development. In Proceedings of the 28th international Conference on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM, New York, NY, 731-740.

Author Biographies

Ridi Ferdiana. Mr. Ridi Ferdiana was born in 1983. He is a doctoral student at Gadjah Mada University, Yogyakarta since 2008. He earned his master degree from the same university in 2006. In his professional area, he holds several professional certifications such as MCP, MCTS, MCPD, MCITP, MVP and MCT. In his daily research activities he really enjoys to learn about software engineering, business platform collaboration, and programming optimization.

Lukito Edi Nugroho. Born in 1966, Dr. Lukito Edi Nugroho is an Associate Professor in the Department of Electrical Engineering and Information Technology, Gadjah Mada University. He obtained his M.Sc and PhD degrees from James Cook University in 1995 and Monash University in 2002, respectively. His areas of interest include software engineering, distributed and mobile computing, and application of ICT in education.

Paulus Insap Santosa. Insap was born in Klaten, 8 January 1961. He obtained his undergraduate degree from Universitas Gadjah Mada in 1984, master degree from University of Colorado at Boulder in 1991, and doctorate degree from National University of Singapore in 2006. His research interest including Human Computer Interaction and Technology in Education.

Ahmad Ashari Place and date of birth: Surabaya, May 2nd 1963. Get Bachelor's degree 1988 in Electronics and Instrumentation, Physics department Gadjah Mada University, Yogyakarta. Master degree 1992 in Computer Science, University of Indonesia, Jakarta Doctor Degrees 2001 in Informatics, Vienna University of Technology. Major Field of study is distributed system, Internet, Web Services, and Semantic Web.