Global Extreme Prorgamming Framework Untuk Pengembangan Sistem Software Enginering Dengan Pendekatan Metode Extreme Programming

Novita Kurnia Ningrum
Fakultas Ilmu Komputer Universitas AKI Semarang

Abstract

The sophistication of information and communication technology today makes the software as the device is much needed by the community. Along with this, the client is more observant and knows how far and what kind of software product quality that are needed. Thus, there is a high demand from the client that must be followed up quickly by the software developer. Global eXtreme Programming (GXP) as a framework using agile approach by adopting eXtream Programming (XP) development system method provides solutions in the field of software engineering to provide the convenience to overcome the dynamic requirements of the client and can complete projects rapidly or quickly. GXP is available to be applied to teamwork which has geographically dispersed conditions but is still facilitated in terms of communication and information.

***Key words** : framework, GXP, XP, agile, dynamic require*

1. Pendahuluan

I.1. Latar Belakang Masalah

Kemajuan teknologi komunikasi dan informasi dari waktu ke waktu memberikan banyak manfaat bagi kemajuan masyarakat. Komukasi menjadi lebih murah, informasi menjadi lebih mudah dan cepat diperoleh dan bisa dilakukan oleh masyarakat dari belahan bumi manapun. Kondisi ini menjadikan dunia seolah-olah datar, sebagaimana dikemukakan oleh Friedman [HYPERLINK \l "Fre1" 1], komunikasi menjadi murah dan dapat dilakukan dimanapun. Bahkan dengan adanya internet dapat diperoleh informasi dan data yang cepat, murah, dan beragam.

Salah produk kemajuan teknologi yang sudah lekat dengan masyarakat adalah *software engineering product* baik yang berupa *software* maupun *software application.* Tanpa disadari keberadaan produk *software* sudah begitu lekat dengan

kebutuhan sehari-hari. Tidak hanya *personal computer (PC)* dan *laptop* yang menggunakan *software* namun *phonecell, iphone, gadgets,* dan banyak juga produk layanan jasa seperti *e-banking, telemarketing,* dan banyak lainnya yang menggunakan produk *software.*

Di era internet apa saja menjadi mudah, salah satunya memberikan kemudahan bagi *users* untuk mendapatkan informasi mengenai *software* sesuai dengan yang dibutuhkan. Beragam produk *software* ditawarkan melalui internet, baik itu berbayar ataupun *free* baik melalui *legal sites* maupun *illegal sites.* Fenomena tersebut menjadi tantangan bagi pelaku *software engineering* baik itu *software developer, programmer, engineer* untuk lebih kreatif dengan menciptakan produk teknologi yang tepat guna bagi *clients* dan *users*.

Tingginya kebutuhan akan *software* tersebut menjadi kendala bagi *software developer* karena jumlah antara pengguna dalam hal ini *clients* dan *users* tidak diimbangi dengan jumlah *developer* yang memadai. Sehingga disini *developer* dituntut untuk bekerja dengan *team* yang cepat dan tanggap terhadap kebutuhan *clients* dan *users* agar projek

Selain hal tersebut *developer* juga dituntut untuk dalam mengakomodasi *requests and requirements* (permintaan dan persyaratan) *clients* yang cenderung berubah-ubah. Dengan kemudahan mendapatkan referensi dan informasi mengenai kemajuan teknologi apa saja yang sedang berkembang, *clients* dapat mengajukan perubahan *requests and requirements* sebelum projek yang sedang dikerjakan selesai. Hal tersebut tentu menjadi masalah yang tidak sederhana baik dalam *developing process* maupun terhadap kinerja *team* yang juga dipaksa harus berubah mengikuti perubahan yang diajukan *clients.*

Untuk mengatasi permasalahan iklim *requests and requirements* dari *clients* yang cenderung berubah (sesuai dengan kebutuhan dan perkembangan teknologi informasi yang diakses) kapan saja sebelum projek selesai dikerjakan dibutuhkan satu metode pengembangan sistem yang dinamis, peka terhadap perubahan untuk mengatasi masalah tersebut.

Saat ini sudah dikenal metode pengembanagan sistem yang digunakan dalam *software development* diantaranya dengan metode konvensional *SDLC* seperti *waterfall* dan *RAD*, *eXtreme*

*Programming*2] oleh Beck dan kawan-kawan, dan juga kosep *Global eXtreme Programming* [HYPERLINK \l "Fer10" 3] yang pada tahun 2010 dikenalkan oleh Ferdiana dan kawan-kawan dari Indonesia.

Pada dasarnya metode *SDLC* masih relevan dengan kondisi pengembangan sistem saat ini. Namun ada kalanya metode *SDLC* mengalami kendala seperti *paralyze analisis*, *throw away design*, dan *unupdated design*4]. Pertama *paralyze analisis* terjadi karena *team* terlena dengan pendokumentasian sehingga tahap yang lainnya terabaikan. Kedua kondisi *throw away design* hal ini biasa terjadi karena adanya perubahan *request and requirement* dari *clients* sehingga *team* terpaksa menghilangkan atau membuang sistem yang sudah dibuat. Dan ketiga *unupdated design*, sebagaimana perubahan *request and requirement* dari *clients* terkadang terjadi dengan cepat sehingga antara dokumen yang ada dengan analisa yang dibuat tidak relevan.

Dengan adanya kendala dari metode pengembangan sistem tradisonal dengan *SDLC* diatas, sehingga saat ini pendekatan *agile* yang diterapkan di beberapa metode seperti *eXtreme Programming* dan *SCRUM* merupakan metode baru yang menjadi pilihan dalam *software development* untuk mengembangkan sistem pada projek yang dikerjakan.

Dalam artikel ini akan dianalisa bagaiman penerapan metode *software development* dengan *framework Global eXtreme Programming* dengan pendekatan metode *eXtreme Programming* dapat mengatasi permasalahan *software development* menghadapi *dynamic reguirements* dari *clients*.

I.2. Rumusan Masalah

Rumusan masalah dari artikel ini adalah menganalisa hasil penerapan *software development* dengan *framework Global eXtreme Programming* dapat mengatasi permasalahan *software development* menghadapi perubahan *dynamic reguirements* dari *clients*.

I.3. Pembatasan Masalah

Masalah pada artikel ini dibatasi pada bagaimana pengembangan sistem dengan *framework Global eXtreme Programming* dapat mengatasi permasalahan *software development*

menghadapi perubahan *requests and reguirements* secara *rapid* atau cepat dari *clients*.

II. Landasan Teori

II.1. *Software Enginering* dengan Pendekatan Konvensional

Secara umum terdapat dua macam pendekatan yang dalam *software development*:

1. Pendekatan konvensional

   Pendekatan konvensional ditandai dengan adanya proses *up-front-analisys.* Yaitu proses yang identik dengan *SDLC Models* yang diawali dengan identifikasi kebutuhan, *analyze, design, coding, testing, evaluating, implementation* serta pendistribusian *software* atau perangkat lunak.
2. Pendekatan *agile* (*agile concept)*

   Pendekatan *agile* dilakukan dengan menerapkan *minimalist design,* pengujian bertahap, dan pendokumentasian sesuai kebutuhan dan tidak berlebihan.

Systems Developments Life Cycles (SDLC) adalah proses menciptakan atau mengubah sistem informasi, dan model dan metodologi yang digunakan orang untuk mengembangkan sistem ini. Dalam *software enginering* konsep *SDLC* mendasari berbagai jenis metodologi pengembangan perangkat lunak.

Terdapat beberapa pemodelan yang dianut *software development* yang dikembangkan berdasarkan prinsip *SDLC,* diantaranya model *waterfall, prototype, incremental, Rapid Aplication Development (RAD),* dan model spiral.

Secara umum dari beberapa model di atas hampir memiliki kemiripan dalam setiap tahapan pengerjaannya, yaitu dimulai dengan *requirement analisis, design ,implementation, testing,* dan *evaluation* sebagaimana diilustrasikan pada Gambar 1 berikut ini,

Gambar 1. *Systems Developments Life Cycles (SDLC)* [5]

Tahapan-tahapan tersebut ditentukan agar dalam mengembangkan projek *engineers* tidak asal mengerjakan tanpa ada konsep dan langkah kerja yang jelas. Apabila kondisi asal mengerjakan ini dibiarkan yang ada pengerjaan projek menjadi kemana-mana (bisa jadi keluar dari konteks kebutuhan *users*) dan tidak terstruktur. Yang pada akhirnya apa yang dibutuhkan oleh *users* tidak tercover dan terakomodasi oleh produk *software* yang dihasilkan.

Pada saat projek dikembangkan harus ada komunikasi yang berkesinambungan dan konsisten dalam antara *client* dengan *engintieers* untuk menentukan spesifikasi kebutuhan *software* seperti apa yang dibutuhkan oleh *client* dalam hal ini *users* sehingga produk *software* yang dihasilkan benar-benar tepat guna secara effektif dan effisien mengakomodasi kebutuhan *client.*

Setelah analisa kebutuhan dirincikan, e*ngineers* yang dalam hal ini adalah pelaksana projek juga harus dapat melakukan komunikasi internal *team* secara konsisten dan berkesinambungan juga. Sebagian besar masalah dalam pengerjaan suatu projek adalah *team work* lebih terpaku bagaimana menyelesaikan *coding* tepat pada *deadline.* Padahal menentukan arsitektur sitem yang akan digunakan agar sesuai dengan kebutuhan *client* akan lebih bisa banyak membantu *engineers* untuk membagi tugas pada masing-masing anggota *team*nya.

Sebagai bahan acuan untuk perbandingan, Table 1 berikut ini merupakan paparan perbandingan pendekatan metodologi oleh Post and Anderson [6],

Tabel 1. Comparation of Methodology Approaches [6]

	SDLC	**RAD**	**Open Source**	**Objects**	**JAD**	**Prototyping**	**End User**
Control	Formal	MIS	Weak	Standards	Joint	User	User
Time frame	Long	Short	Medium	Any	Medium	Short	Short
MIS Staff	Many	Few	Few	Varies	Few	One or two	One
Transaction/DSS	Transaction	Both	Both	Both	DSS	DSS	DSS
Interface	Minimal	Minimal	Weak	Windows	Crucial	Crucial	Crucial
Documentation and training	Vital	Limited	Internal	In Objects	Limited	Weak	None
Integrity and Security	Vital	Vital	Unknown	In Objects	Limited	Weak	Weak
Reusability	Limited	Some	Maybe	Vital	Limited	Weak	None

Praktek *SDLC* memiliki keunggulan dalam model pengembangan perangkat lunak tradisional, yang cocok untuk lingkungan yang lebih terstruktur. Kelemahan menggunakan metode *SDLC* adalah ketika ada kebutuhan untuk pengembangan iteratif seperti pengembangan *website* atau *e-commerce* dimana *stakeholders* harus meninjau secara rutin perangkat lunak yang dirancang.

Selain itu ada pula beberapa tantangan berkaitan dengan *software engineering* dengan pendekatan konsep konvensional4] diantaranya adalah:

1. *Paralyze analisis*

Paralyze atau lumpuh, disini diartikan sebagai suatu kondisi dimana *team* terlalu *detile* melakukan analisa pada data-data projek sehingga terlena bahwa tujuan utama pengembangan projek adalah untuk membangun sistem bukan membuat dokumen-dokumen sistem yang terlalu berlebihan yang tidak memungkinkan untuk dibaca semua dokumen tersebut.

2. Throw away design

Dalam kondisi ini *team* tepaksa membuang *design system* yang sudah direncanakan pada awal pengembangan projek. Hal ini biasanya disebabkan *team* merasa *design system* tersebut sudah tidak sesuai atau tidak diperlukan lagi. *Team* akan langsung masuk pada tahap pengembangan atau *developing and coding* dan tahapan-tahapan selanjutnya.

3. Unupdated design

Dalam proses pengembangan projek suatu hal yang biasa terjadi perubahan dalam

tahap analisa. Namun perubahan-perubahan ini tidak seiring dengan perubahan pada dokumentasi *design*. Ketika projek sudah dikerjakan *engineers* akan lebih fokus pada tahap pengembangan sehingga dokumentasi *design* terbaikan. Setelah projek selesai baru akan terlihat ketidak cocokan antara analisa dengan dokumentasi *design.*

Pemaparan di atas menggambarkan bahwa saat ini pendekatan secara konvensional tidak selalu sesuai untuk diterapkan pada setiap projek. Setiap *team* work memiliki permasalahan masing-masing. Seharusnya *team work* itu sendiri secara independent menentukan tahapanp-tahapan yang harus dikerjan. Proses yang kaku pada akhirnya menjadikan sistem kerja tidak kondusif dan hasil dari projek tidak sesuai yang diharapkan.

Mengikuti perkembangan teknologi informasi modern beberapa berpendapat bahwa penerapan *SDLC* dengan sistem yang kaku perlu dilengkapi model komputasi *Agile* [HYPERLINK \l "And06" 6] agar lebih relevan dengan kebutuhan dunia teknologi itu sendiri. Meski demikian *SDLC* juga memiliki keunggulan yang masih dapat diterapkan dalam istilah luas dalam perkembangan teknologi.

II.2. *Software Enginering* dengan Pendekatan *Agile*

Konsep *agile* dipopulerkan oleh Jamie Linn Cooke dapat diartikan sebagai sesuatu hal yang dinamis, fleksibel, dan mudah menyesuaikan diri terhadap perubahan7]. Dewasa ini prinsip *agile* banyak diadopsi dalam praktek bisnis dunia, dengan *track record* yang terbukti untuk membantu organisasi mencapai efisiensi yang lebih besar, berkualitas tinggi output dan meningkatkan kepuasan pelanggan dengan *cost budged* yang dapat diminimalis [HYPERLINK \l "Coo10" 7].

Sebagaimana dipaparkan pada bab sebelumnya, pendekatan *agile* diperlukan untuk menghadapi tuntutan dari *clients* yang semakin tinggi. Diantaranya dengan semakin banyak dan informasi mengenai perkembangan teknologi yang diperoleh *clients* maka berdampak pula dengan *request and requirement* yang memiliki kecenderungan berubah-ubah dalam waktu yang cepat. Sehingga *developer* juga harus dapat mengikuti perubahan yang terjadi dalam sistem yang sedang dikembangkannya.

Beberapa metode pengembangan sistem *software engineering* yang sudah mengadopsi pendekatan *agile* diantaranya

adalah *SCRUM, eXtreme Programming (XP).*

Selengkapnya mengenai *eXtreme Programming* dijelaskan pada bab berikutnya.

II.3. *eXtreme Programming (XP)*

eXtreme Programming (XP) mengadopsi pendekatan *agile* untuk pengembangan perangkat lunak yang diasumsikan dapat membantu meningkatkan efisiensi dan fleksibilitas dari sebuah proyek pengembangan perangkat lunak dengan mengkombinasikan berbagai ide sederhana. *XP* dipopulerkan oleh Kent Beck2] pada tahun 2010 sebagai metode atau pendekatan untuk mengembangkan sistem *software engineering.*

XP tidak selalu cocok untuk setiap proyek pengembangan perangkat lunak. Kelebihan XP adalah sesuai untuk digunakan pada proyek yang memiliki *dynamic requirements* seperti permintaan dari *clients* yang sewaktu –waktu mengalami perubahan selama pengerjaan projek dilakukan. Proyek semacam ini memerlukan adaptasi cepat dalam mengatasi perubahan-perubahan yang terjadi selama proses *software development*. *XP* juga cocok untuk proyek dengan jumlah anggota tim tidak terlalu banyak (sekitar 10-20 orang) dan berada pada lokasi yang sama.

Nilai-nilai yang mendasari *XP* pada setiap tahapan proses pengembangan perangkat lunak diuraikan sebagai berikut:

1. *Communication*

Hubungan komunikasi yang baik antar anggota tim adalah hal yang utama dalam *software development*. Satu *team work* harus terbangun pengertian dan *sahring knowladge and skills* pada saat pengerjaan projek.

2. *Courage*

Team work dan *software developer* harus memiliki keyakinan dan integritas terutama pada saat terjadi tekanan dari *client.* Rasa saling percaya merupakan hal yang coba dibangun dan ditanamkan dalam *XP* pada berbagai aspeknya.

3. *Simplicity*

Mengerjakan dengan cara sederhana seperti menggunakan *method* yang pendek dan simpel, desain yang tidak terlalu rumit, *unused fitures* dihilangkan, dan berbagai proses penyederhanaan pada aspek lainnya.

4. *Feedback*

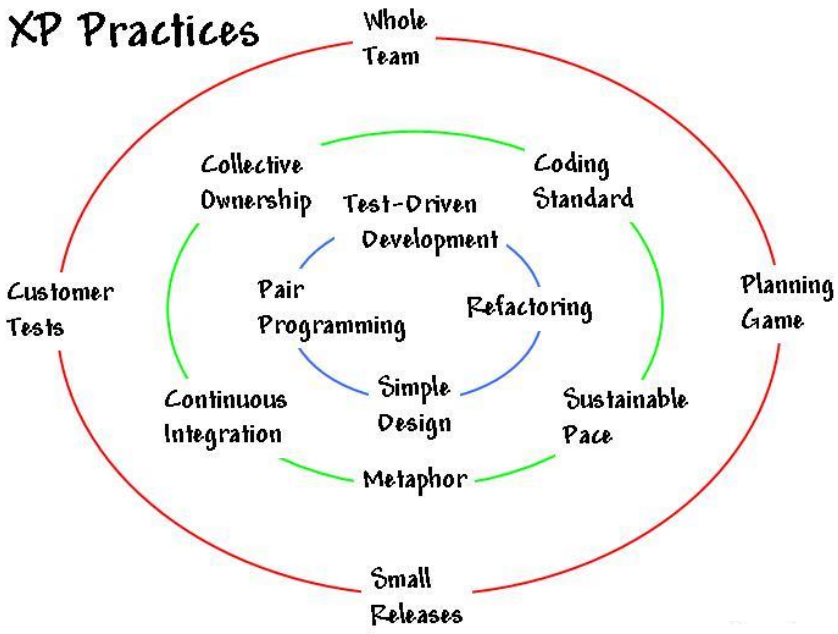
Membangun *feedback* yang komunikatif dalam *team work.* Setiap permasalahan dan perubahan yang terjadi diungkapkan dan anggota *team* diberi kesempatan untuk mengutarakan pendapat masing-masing.

5. *Quality Work*

Pada prinsipnya segala sesuatu yang dikerjakan diharapkan dapat meghasilkan produk dan hasil dengan kualitas baik. Oleh karena itu pula diberlakukan juga kualitas kerja yang baik dan optimal.

Aspek dasar *XP* terdiri dari berbagai teknik atau yang dapat diamati pada gambar berikut ini:

Gambar 2. Gambar Aspek Dasar XP **[** HYPERLINK \l "www08" **5]**



1. *The Planning Game*

Pendekatan XP dalam perencanaan sangat mirip dengan metode yang diterapkan pada *RAD (Rapid Application Development).* Proses pendek dan cepat, mengutamakan aspek teknik, memisahkan unsur bisnis dengan unsur teknis dan pertemuan

intensif antara klien dengan *developer*. Pada XP proses ini menggunakan terminologi "*game*" karena Beck menyarankan untuk menggunakan teknik *score card* dalam menentukan *requirements*. Semakin sulit aspek teknis yang dibutuhkan semakin tinggi pula skor pada kartu rencana tersebut.

2. *Small Releases*

Setiap *release* dilakukan dalam lingkup sekecil mungkin pada XP. Setiap *developer* menyelesaikan sebuah unit atau bagian dari perangkat lunak maka hasil tersebut harus segera dipresentasikan dan didiskusikan dengan klien. Jika memungkinkan untuk menerapkan unit tersebut pada perusahaan, hal itu juga dapat dilakukan sekaligus sebagai tes awal dari penerapan keseluruhan sistem. Kendati demikian hal ini tidak selalu perlu dilakukan karena harus dihitung terlebih dahulu sumberdaya yang dibutuhkan. Apakah lebih menguntungkan langsung melakukan tes terhadap unit tersebut atau melakukan tes setelah unit tersebut terintegrasi secara sempurna pada sistem.

3. *Metaphor*

Metaphor pada dasarnya sama dengan arsitektur perangkat lunak. Keduanya menggambarkan visi yang luas terhadap tujuan dari pengembangan perangkat lunak. Beck sendiri seperti para penandatangan Agile Manifesto lainnya bercita-cita menyederhanakan proses pengembangan perangkat lunak yang saat ini sudah dianggap terlalu rumit. Arsitektur yang saat ini banyak berisi diagram dan kode semacam UML dianggap terlalu rumit untuk dimengerti, terutama oleh klien. *Metaphor*, walaupun mirip dengan arsitektur lebih bersifat naratif dan deskriptif. Dengan demikian diharapkan komunikasi antara klien dengan *developer* akan berlangsung lebih baik dan lancar dengan penggunaan *metaphor*.

4. *Simple Design*

Sebagai salah seorang penandatangan Agile Manifesto, Beck adalah seorang yang tidak menyukai desain yang rumit dalam sebuah pengembangan perangkat lunak. Tidak heran jika dia

memasukkan *Simple Design* sebagai salah satu unsur XP. Pada XP desain dibuat dalam lingkup kecil dan sederhana. Tidak perlu melakukan antisipasi terhadap berbagai perubahan di kemudian hari. Dengan desain yang simpel apabila terjadi perubahan maka membuat desain baru untuk mengatasi perubahan tersebut dapat dengan mudah dilakukan dan resiko kegagalan desain dapat diperkecil.

5. *Refactoring*

   Refactoring adalah salah satu aspek paling khas dari XP. *Refactoring* seperti didefinisikan oleh Martin Fowler adalah ”Melakukan perubahan pada kode program dari perangkat lunak dengan tujuan meningkatkan kualitas dari struktur program tersebut tanpa mengubah cara program tersebut bekerja”. *Refactoring* sendiri sangat sesuai untuk menjadi bagian XP karena *Refactoring* mengusung konsep penyederhanaan dari proses desain maupun struktur baris kode program. Dengan *Refactoring* tim pengembang dapat melakukan berbagai usaha untuk meningkatkan kualitas program tanpa kembali mengulang-ulang proses desain. Fowler adalah salah satu kolega dekat dari Kent Beck karena itu tidak mengherankan bahwa cara berpikir mereka terhadap proses pengembangan perangkat lunak sangat mirip satu dengan lainnya.

6. *Testing*

   XP menganut paradigma berbeda dalam hal tes dengan model pengembangan perangkat lunak lainnya. Jika pada pengembangan perangkat lunak lainnya tes baru dikembangkan setelah perangkat lunak selesai menjalani proses *coding* maka pada XP tim pengembang harus membuat terlebih dahulu tes yang hendak dijalani oleh perangkat lunak. Berbagai model tes yang mengantisipasi penerapan perangkat lunak pada sistem dikembangkan terlebih dahulu. Saat proses *coding* selesai dilakukan maka perangkat lunak diuji dengan model tes yang telah dibuat tersebut. Pengetesan akan jauh lebih baik apabila dilakukan pada setiap unit perangkat lunak dalam lingkup sekecil mungkin daripada menunggu

sampai seluruh perangkat lunak selesai dibuat. Dengan memahami tahap ini kita dapat melihat bahwa siklus pada XP adalah *requirement analysis → test → code → design*. Sekilas terlihat hal ini tidak mungkin dilakukan tetapi pada kenyataannya memang gambaran inilah yang paling dapat menjelaskan tentang XP.

7. *Pair Programming*

   Pair programming adalah melakukan proses menulis program dengan berpasangan. Dua orang programer saling bekerjasama di komputer yang sama untuk menyelesaikan sebuah unit. Dengan melakukan ini maka keduanya selalu dapat berdiskusi dan saling melakukan koreksi apabila ada kesalahan dalam penulisan program. Aspek ini mungkin akan sulit dijalankan oleh para programer yang memiliki ego tinggi dan sering tidak nyaman untuk berbagi komputer bersama rekannnya.

8. *Collective Ownership*

   Tidak ada satupun baris kode program yang hanya dipahami oleh satu orang programer. XP menuntut para programer untuk berbagi pengetahuan untuk tiap baris program bahkan beserta hak untuk mengubahnya. Dengan pemahaman yang sama terhadap keseluruhan program, ketergantungan pada programer tertentu ataupun berbagai hambatan akibat perbedaan gaya menulis program dapat diperkecil. Pada level yang lebih tinggi bahkan dimungkinkan para programer dapat bertukar unit yang dibangunnya.

9. *Coding Standards*

   Pair programming dan *collective ownership* hanya akan dapat berjalan dengan baik apabila para programer memiliki pemahaman yang sama terhadap penulisan kode program. Dengan adanya *coding standards* yang telah disepakati terlebih dahulu maka pemahaman terhadap program akan menjadi mudah untuk semua programer dalam tim. Hal ini dapat diterapkan sebagai contoh pada penamaan variabel dan penggunaan tipe data yang sama untuk tiap elemen semua *record* atau *array* pada program.

10. *Continous Integration*

    Melakukan *build* setiap hari kerja menjadi sebuah model yang disukai

oleh berbagai tim pengembang perangkat lunak. Hal ini terutama didorong oleh keberhasilan penerapan sistem ini oleh Microsoft dan telah sering dipublikasikan. Dengan melakukan *build* sesering mungkin berbagai kesalahan pada program dapat dideteksi dan diperbaiki secepat mungkin. Apabila banyak tim pengembang perangkat lunak meyakini bahwa *build* sekali sehari adalah minimum maka pada XP hal tersebut adalah maksimum. Pada XP tim disarankan untuk melakukan *build* sesering mungkin misalnya setiap 4 jam atau bahkan lebih cepat lagi.

11. *40-hours Week*

    Beck berpendapat bekerja 8 jam sehari dan 5 hari seminggu adalah maksimal untuk tiap programer. Lebih dari itu programer akan cenderung membuat berbagai *error* pada baris-baris kode programnya karena kelelahan.

12. *On-Site Customer*

    Sebuah pendekatan klasik, di mana XP menganjurkan bahwa ada anggota dari klien yang terlibat pada proses pengembangan perangkat lunak. Yang lebih penting lagi ia harus ada di tempat pemrogaman dan turut serta dalam proses build dan test yang dilakukan. Apabila ada kesalahan dalam pengembangan diharapkan klien dapat segera memberikan masukan untuk koreksinya.

I. METODOLOGY

III.1. *eXtreme Programming (GXP)*

Sebagaimana dipaparkan pada latar belakang masalah di atas, saat ini telah dikenalkan sebuah *framework* baru yang menerapkan pendekatan *agile* yang dinamis menyesuaikan kondisi projek dan *team work. Global eXtream Programming* (*GXP*)4] sebagai *framework* yang membantu *team work* yang terpisah secara geografis namun masih terhubung dari segi komunikasi dan informasi. *GXP* ini diharapkan dapat mengatasi permasalah non teknis yang dihadapi pengembang sehingga dapat secara efisien dan efektif menjawab tantangan pengembang yang dituntut untuk menyelesaikan projek *rapidly* atau dengan cepat dan adanya *dynamic requirements* dari *client*. Dengan cepatnya pengerjaan projek maka berbanding lurus dengan *cost* yang

dikeluarkan oleh pengembangpun dapat ditekan seminimal mungkin.

GXP bukanlah satu produk *framework* baru dalam *software engineering*, sebelumnya ada banyak model *framework* yang sudah dikenalkan dan sudah digunakan. Keberadaan *GXP* saat ini untuk melengkapi dan menyempurnakan hasil penelitian yang sudah ada sebelumnya.

Terdapat tiga komponen utama yang mendasari *framework GXP*, yaitu proses *Global Software Development (GSD)* yang dipaparkan oleh Sangwan [HYPERLINK \l "San10" 8], *eXtream Programming*2], dan yang terakhir adalah alat bantu yang diasumsikan untuk meningkatkan produktivitas *team work* dalam *software development.*

GSD sebagai siklus hidup *software engineering* yang memfokuskan pada pengembangan perangkat lunak yang mendukung pola pengembangan terdistribusi. Pola pengembangan terdistribusi menekankan pada dukungan pengembangan jarak jauh seperti teknik pengembangan *shift* (*shift based model*) atau *round clock follow the sun techniques. GSD* sesuai diadopsi untuk *software development* saat ini dimana dunia yang makin global, memungkinkan *team work* berada di tempat yang terpisah secara geografis.

Sebagaimana dipaparkan pada bab sebelumnya *XP* dengan prinsip dan teknik praktis yang diterapkan pada metodenya sesuai untuk *software development* yang efektif dan efisien.

Alat bantu yang dikembangkan dalam *GXP* ditujukan untuk membantu kinerja *team work*. Dengan adanya alat bantu ini, diharapkan produktivitas *team work* dapat meningkat.

III.2. Eksekusi *Global eXtreme Programming (GXP)*

Banyak *framework* yang bisa digunakan untuk membantu pengembangan sisstem, diantaranya *Microsoft Solution Framework* (*MSF*), *Rational Unified Process* (*RUP*), dapat juga mengadopsi model *Scrum*. Demikian pula dengan *GXP*,

tidak semua projek *software development* sesuai dikerjakan dengan *framework GXP*. Oleh karena itu ada beberapa poin yang perlu dipenuhi sebelum projek dikerjakan dengan *framework GXP* antara lain:

- *GXP* sesuai untuk *team* work yang terpisah secara geografis

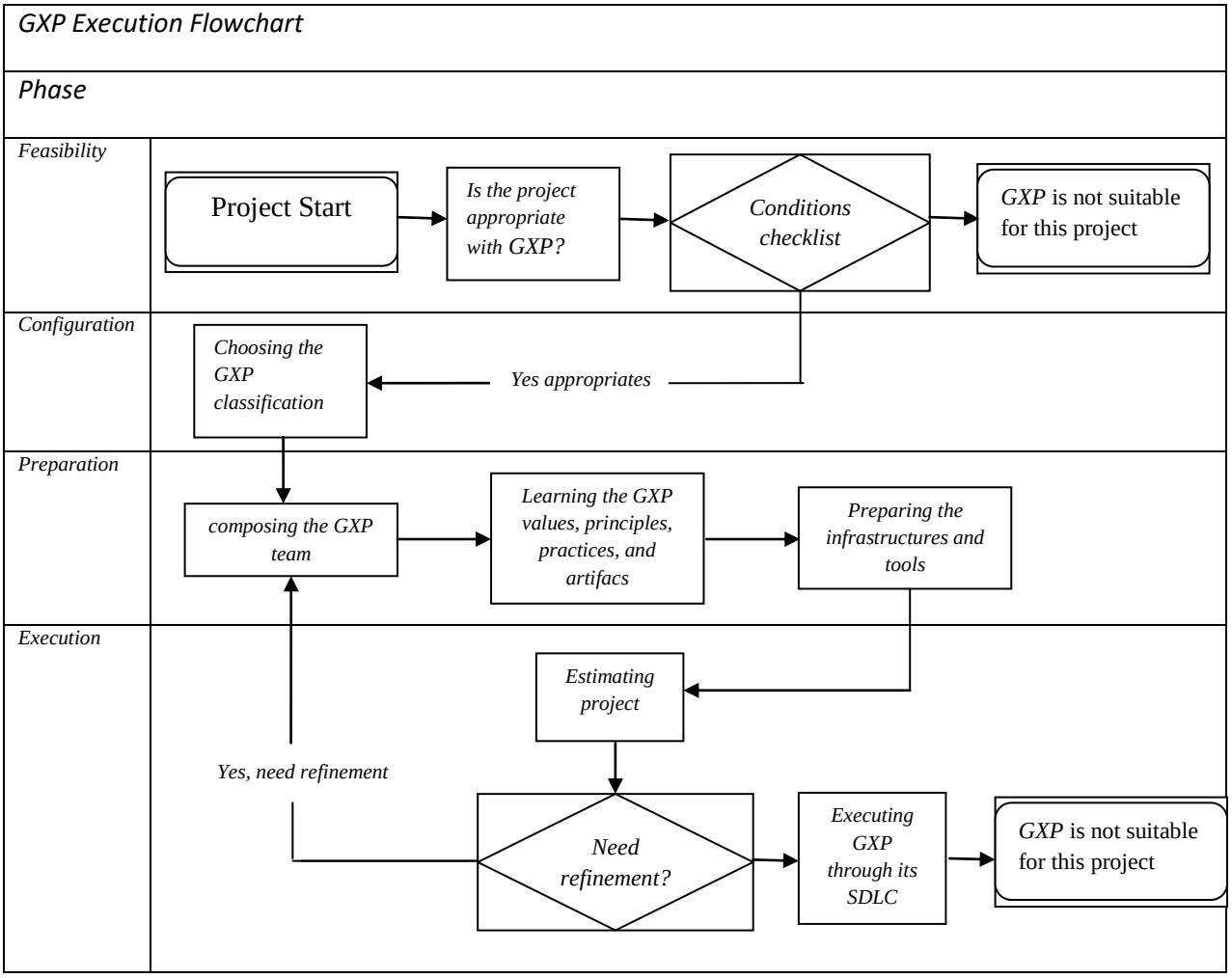
dan terfasilitasi secra komunikasi dan informasi. Namun, apabila terjadi kendala perbedaan bahasa dan kultur kerja yang tidak dapat diseragamkan, *GXP* menjadi tidak efisien diterapkan pada kondisi *team work* seperti ini.

- *GXP* sesuai untuk projek dengan skala kecil dimana *team work* yang terlibat antara 10-12 orang anggota. Untuk projek dengan skala besar dan membutuhkan banyak anggota *teami* serta projek yang bersifat *real time* tidak sesuai dikerjakan dengan *GXP*.
- Karakter *client* yang fleksibel dengan kondisi *team work* yang bekerja secara fleksibel melalui jarak jauh pentng untuk diperhatikan. Sehingga ada kesinambungan dan pengertian antara *software developer* dengan *client.*
- Komunikasi yang *intens* antara *software developer* dengan *client* harus bisa terjalin dengan baik. Jika hal ini tidak terpenuhi akan memungkinkan terjadinya *misunderstanding* yang berujung pada kegagalan pengerjaan projek.
- Harus dapat dikomunikasikan dengan baik bahwa *team work* menitikberatkan pada hasil kualitas *software* dibandingakan dengan pendokumentasian yang bewrlebihan dalam pengerjaan projek.

Dari penjelasan beberapa poin di atas, pada dasarnya pengerjaan dengan *framework* GXP banyak menitik beratkan pada prinsip komunikasi yang komunikatif dua arah antara *software developer* dengan *client*. Semakin baik dan lancar komunikasi yang dilakukan, akan mempengaruhi tingkat keberhasilan *team work* dalam pengerjaan projek.

Apabila sudah diketahui apakah projek sesuai untuk dikerjakan dengan *framework GXP* atau tidak sesuai, tahap berikutnya dapat dilanjutkan dengan eksekusi *GXP*. Langkah eksekusi *GXP* digambarkan pada gambar 3 berikut ini,

Gambar 3. Langah-langkah Eksekusi *GXP* [HYPERLINK \l "Fer12" 4]



Gambar 3 menunjukkan langkah-langkah eksekusi *GXP* terbagi menjadi empat fase utama, yaitu *feasibility*, *configuration*, *preparation*, dan *execution.*

Pada fase *feasibility* atau kelayakan *team work* mempertimbangkan apakah sesuai dan layak jika projek dikerjakan dengan *framework GXP.* Jika dinyatakan layak maka masuk ke fase berikutnya, *configuration.*

Setelah dinyatakan layak, selanjutnya *team work* mengkonfigurasi projek akan menggunakan model eksekusi *GXP* apa yang sesuai dengan projek tersebut. Ada tiga model eksekusi *GXP* yang bisa dipilih, antara lain:

1. *Remote Execution Model* (*REM*)

Model ini sesuai untuk kondisi dimana *team work* dengan *client* berada ditempat yang terpisah secara geografis, sebagaimana kondisi geografis Indonesia yang tepisah antar pulau. Karakteristik *REM* antara lain:

- Angggota *team* tidak banyak (kurang dari enam)
- Urgensi pengerjaan projek tinggi
- Managemen *team* tetap terintegrasi, meskipun berada di tempat yang berbeda
- Jangka waktu pengerjaan projek antara satu hingga tiga bulan

Dengan karakteristik seperti di atas *team* diharuskan bekerja dengan iterasi pendek (antara 1-2 minggu), dengan demikian setiap 1-2 minggu sekali *team* menyampaikan perkembangan projek kepada *client.* Sehingga dibutuhkan satu orang sebagai penghubung yang selau mengkomunikasikan kondisi projek. Model ini juga dikenal dengan istilah *hub to spoke.*

2. *Virtual Execution Model* (*VEM)*

Kondisi *team* dengan *client* tetap pada kondisi yang terpisah secara geografis, namun ada perwakilan dari *team* yang *onsite* berada di satu lokasi dengan *client.*

Karakteristik dari model ini antara lain:

- Managemen *team* tetap berada pada satu integritas, meskipun sebagian *team* berada di lokasi *client.*
- Pada umumnya sakala projek menengah dan cukup memiliki nilai bisnis.
- Jumlah *team* di atas enam orang
- Projek yang dikerjakan mencakup satu atau lebih sistem pada *client*
- Jangka waktu pengerjaan projek antara tiga hingga dua belas bulan.

Pada model ini *team* harus lebih detil dalam menentukan desain, analisa, dan arsitektur sistem yang akan dibangun. Iterasi pada model ini sekitar 4-8 minggu sekali. Model seperti ini biasa disebut juga *hub to hub model*

3. *Global Execution Model* (*GEM*)

GEM memungkinkan untuk kondisi *team work* yang terpisah secara geografis, berbeda negara sehingga berbeda zona waktu, pengaturan managemen, kultur, dan bahasa. Contoh pengembangan dengan *GEM*

seperti pengembangan *Windows 7, Linus Ubuntu, FreeBSD.* Pengembangan sistem ditekankan pada eksekusi yan terdistribusi dari masing-masing *team work.*

Karakteristik *GEM* antara lain:

- Managemen terpisah
- Komunikasi antar managemen *team work* meskipun terpisah terjalin dengan dengan baik
- Sistem yang dikembangakan umumnya berbasis produk
- Memungkinkan menggunakan bahasa pemrograman yang berbeda
- Jangka waktu pengerjaan projek lebih dari satu tahun.

Karena model ini terdiri atas managemen *team* lebih dari satu dan berbeda-beda, sehingga aspek non teknis seperti persamaan visi dan *center management* menjadi hal yang perlu diperhatikan.

Setelah *team* mengkonfigurasi model eksekusi yang sesuai fase selanjutnya persiapan. Diantaranya mempersiapkan komposisi *team*¸ mempelajari prinsip, nilai, teknis praktis, dan mempersiapkan infrastruktur yang dibutuhkan.

Persiapan pembentukan struktur *team* berdasarkan dilakukan berdasarkan tiga hal:

- Struktur *team* terdiri atas *manager, developer, tester, teracker, subject matter expert, coach.* Masing-masing melaksanakan sesuai dengan *job description* yang telah ditentukan.
- Model *team* dibentuk berdasarkan hasil fase konfigurasi sebelumnya. Ada dua macam model *team* yaitu *central team* dan *site team. Central team* bertindak sebagai pusat pengendalian visi misi produk, menentukan arsitektur *team,*menentukan kebijakan dan managemen projek. *Site team* lebih focus pada desain sistem dan pengembangan aplikasi. Dalam satu *central team* bisa terdapat beberapa *site team* sesuai kebutuhan projek.
- Pembentukan *team* dilakukan secara *incremwental* beraturan. Dimana *team* terbentuk secara bertahap sesuai dengan kebutuhan. Jika cakupan projek luas, maka *team* yang terbentuk akan menyesuaikan dengan kebutuhan projek tersebut
- Proses pembelajaran *team* adalah hal yang mutlak dalam penerapan *GXP*.

Dimana *team work* tidak dapat berdiri sendiri dengan tugas yang sudah ditentukan akan tetapi antar *team work* wajib untuk melakukan komunikasi secara *intens*, melakukan *sharing knolage*, dan selalu mendiskusikan permasalahn-permasalahan yang terjadi pada proses pengerjaan projek. Hal tersebut terangkum dalam prinsip kerja dan teknik praktis yagn dijabarkan pada Tabel 2 dan Tabel 3 berikut ini.

Prinsip kerja *GXP* yang mengadopsi pendekatan *agile* yang dinamis digabungkan dengan metode *XP* menghasilkan beberapa poin prinsip kerja4] yang diuraikan pada Tabel 2 sebagai berikut:

Tabel 2. Tabel Prinsip Kerja *GXP*

No	Prinsip	Penjelasan
1.	Umpan balik secara teratur	Umpan balik dari *client* serta respon dari *team* akan meyakinkan bahwa proyek berjalan dengan baik
2.	Penyederhanaan keberntungan	Melakukan
3.	Menyusun *roadmap* untuk *versioning*	Menyiapkan model pengembangan berkala untuk menjamin sistem agar selalu *up*
4.	Memformalkan manajemen perubahan	Pendokumentasian setiap perubahan pada kode, dokumen, maupun permintaan *client* secara baik dan rapi
5.	Kualitas yang seimbang	Menyesuaikan kemampuan *team* dengan kualitas yang akan dihasilkan
6.	Pembelajaran mandiri	*Team work* membiasakan diri dengan pembelajaran yang beragam dan memiliki suatu kajian mandiri
7.	Incestasi perangkat bantu komunikasi	*Team work* harus dapat memikirkan bagaimana agar komunikasi dapat tetap berjalan konsisten dan homogeny untuk mencapai komunikasi yang lebih baik
8.	Penyamaan visi	Dengan adanya pandangan visi yang sama akan memudahkan *team work* tetap dapat melakukan

		kolaborasi maksimal meskipun dengan keterbatasan sarana yang ada
9.`	Komunikasi multimodel	*Team work* diharapkan memiliki model komunikasi alternative selain tatap muka secara konvensional, misalnya *by email, IM,* maupun komponen komunikasi lainnya
10.	*Online progress*	*Progress* kondisi projek harus selalu diakses oleh *team work* sehingga tetap terpantau kondisi projek tersebut

Prinsip kerja *GXP* tersebutdiharapkan dapat diimplementasikan secara praktis *oleh team work* dalam menjalankan eksekusi projek. Secara praktis aspek yang diimplentasikan dalam Tabel 3 [HYPERLINK \l "Fer12" 4].

Tabel 3. Tabel Teknis Praktis *GXP*

No	Teknis Praktis	Penjelasan
1.	Sinkronisasi secara rutin	*Team work* membutuhkan jadwal yang rutin untuk berkomunikasi dengan *client* maupun dengan sesame anggota *t*
2.	*Online programming*	Pemrograman yang menekankan kolaborasi *online, offline* melalui media internet
3.	*Cloud codes*	Selalu mengunggah kode-kode terkini melalui sistem *CVS* atau *workspace* yang bisa diakses oleh seluruh *team work*
4.	*Standarts coding*	*Team work* harus menentukan standart *coding* yang digunakan agar seragam
5.	*Refactoring and review*	*team work* melakukan *review* pada setiap pekerjaan yang sudah berjalan sesuai dengan fungsi masing-masing dan optimalisasi *coding* melalui teknik

		refactoring
6.	*Wiki journaling*	Membuat halaman wiki yang dijadikan tempat untuk mencantumkan temuan dan pengetahuan yang ditulis oleh *team*
7.	*Multysite testing*	Pengujian dari berbagai *sites* untuk menjamin bahwa solusi yang dikembangkan dalam projek teruji dari berbagai sudut pandang
8.	*Centralized integration*	Memasang *source control* untuk kebutuhan manajemen yang lebih baik
9.`	*Online planning game*	Mendesain model sistem kemudian mebngestimasi baik secara *online* maupun *face to face*
10.	*Sprint prototyping*	*Team work* mengembangkan *prototype* yang memungkinkan *team* dan *client* saling memberikan *input* dan *feedback* dengan proporsional
11.	*Loosely coupled component*	Memisahkan sekumpulan komponen perangkat lunak yang bisa dikembangkan secara terpisah sehingga memudahkan *team work* membagi pekerjaan
12.	*Around the clock development*	Apabila pengembangan projek melibatkan rekan atau *team* yang berada di zona waktu yang berbeda maka dapat digunakan mekanisme *around the clock development* sehingga ada ketepatan waktu yang sama untuk menyelesaikan projek sesuai dengan *deadline* yang direncanakan
13.	*Knowlade based learning*	Mengaplikasikan pengetahuan bersama dengan memasangkan *repository knowlage system* untuk memudahkan *teamwork* melakukan *sharing* pengetahuan

Tiga belas teknik praktis *GXP* yang dijelaskan pada Tabel 2 diatas disusun berdasarkan projek yang mengadopsi model *GXP*. *Team* tidak harus menjalankan semua *point* tersebut, namun *team* dapat memilih *point* mana yang perlu dan tidak perlu menyesuaikan dengan kondisi *team* itu sendiri baik dari segi kebutuhan maupun dari segi kompleksitas projek yang dikerjakan.

Tahap persiapan yang terakhir adalah persiapan infrastuktur. Infrastruktur meliputi perangkat keras maupun perangkat lunak dari projek. Poin penting yang perlu diperhatikan dalam penerapan *GXP* antara lain:

- Perangkat komunikasi dan kolaborasi, sebagai contoh Yahoo Massanger, Shared View, Live Messenger, Groove.
- Perangkat bantu yang dapat memvisualkan model, sebagai contoh Open SVN, Team City, Visual Studio Team System, Rational Rose Requisite Pro, Sparx System, Enterprise Achitect.
- Perangkat bantu untuk menyimpan dan mengelola perkembangan projek atau *repository* yang dapat digunakan untuk acuan analisa projek selanjutnya. Contoh perangkat bantu ini seperti Open Wiki, Blogengine, SWiki

Setelah semua fase terpenuhi maka fase yang terakir adalah eksekusi. Pada fase ini lebih banyak pada kegiatan teknis dalam *software development.* Fase eksekusi ini terdiri atas lima tahapan, yaitu tahap eksplorasi, tahap perencanaan, tahap iterasi, tahap produksi, tahap pemeliharaan.

Kesimpulan

Kesimpulann yang dapat diambil dari penulisan artikel ini adalah dalam *software engineering* dikenal dua metode pendekatan pengembangan sistem perangkat lunak, yaitu pendekatan konvensional dan pendekatan *agile* (dinamis). Pendekatan konvensional diterapkan pada *SDLC* dimana tahapan pengembangan sistem secara umum meliputi *requirement analisis*, *design ,implementation, testing,* dan *evaluation.* Saat ini dengan makin tingginya kebutuhan *software* dan makin tinggi pula pengetahuan *client* mengenai produk *software* yang sesuai dengan yang dibutuhkan maka *software developer* harus dapat bekerja secara cepat serta mampu beradaptasi dengan *dynamic requirement* yang dapat sewaktu-waktu diajukan oleh

client. Ada pula beberapa projek *software development* yang bersifat sederhana dan tidak memerlukan rangkaian tahapan pengembangan sistem yang panjang. Oleh karena itu dibutuhkan suatu metode pemnegmbangan sistem yang dinamis yang mampu menyesuaikan antara kebutuhan projek dengan kebutuhan *team* dalam mengerjakan projek seperti yang ditawarkan oleh metode pendekatan *agile*. Pendekatan ini diterapkan pada metode pengembangan sistem *eXtreme Programming (XP)* yang kemudian diadopsi dalam *GXP* menjadi sebuah *framework* yang terdiri atas *tools* atau alat bantu untuk memudahkan *team* dalam menyelesaikan projek. Dimana *team* dibentuk disesuaikan dengan kondisi dan kebutuhan *team* berdasarkan konsep *GXP* yang ada.

Meskipun demikian tidak semua projek *software engineering* sesuai dikerjakan menkggunakan *GXP*. *SDLC* memiliki kelemahan dan kelebihan dalam *software engineering* dan masih tetap dapat digunakan untuk mengerjakan projek yang bersifat *run time* yang dikerjakan dalam jangka waktu yang lama serta membutuhkan *team work* dalam jumlah banyak.

Daftar Pustaka

Anderson, D, dan G Post. *Managemen Information Systems: Solving business problems with Information Technology*. New York: Mc Graw-Hill Irwin, 2006.

Beck, Kent. *Global Software Development and Collaboration: barries and solution.* Inroads, ACM, 2010.

Cooke, Jamie Lynn. *Agile Principles Unleashed.* It Governence Ltd., 2010.

Ferdiana, Ridi. *Rekayasa Perangkat Lunak yang Dinamis dengan Global Extreme Programming.* Yogyakarta: Penerbit ANDI, 2012.

Ferdiana, Ridi, Lukito Edi Nugroho, Paulus Insap Santoso, dan Ahmad Ashari. "Process Framework in Global eXtream Programming." *Computer Science and Information Security*, 2010.

Friedman, Thomas L. *The World is Flat.* Farrar, Straus & Giroux, 2005.

Sangwan, Raghvinder, Matthew Bass, Neel Mullick, Daniel J. Paulish, dan Juergen Kazmeier. *Global Software Development Handbook (Auerbach Series on Applied Software Engineering Series).* Boston: Auerbach Publication, 2010.