PROSES PEMODELAN SOFTWARE DENGAN METODE WATERFALL DAN EXTREME PROGRAMMING: STUDI PERBANDINGAN

Imam Fahrurrozi[1], Azhari SN[2]
[1,2]Program Studi Ilmu Komputer, Universitas Gadjah Mada
e-mail: [1]imam.fahrurrozi@ugm.ac.id, [2]arisn@ugm.ac.id

Abstract

The purpose of writing this paper is to compare Extreme Programming and Waterfall Modelling software processes. The Software Development Life Cycle (SDLC) describes the processes by which a piece of computer software goes from conception to delivery and beyond. There are many process models detailing the steps in an SDLC, each with different strengths and weaknesses. Models are useful for explaining, evaluating and exploring the different processes which teams use to move through the SDLC.

Extreme Programming (XP) is very different from Waterfall. XP itself is a collection of practices for Software Development, some of which create a development process comparable to Waterfall. It also contains values and best-practices to aid in the SDLC. At its core it is a multi-iterative approach with a focus on what the customer wants [1].
Keywords: *compare, sdlc, xp, processes.*

PENDAHULUAN

Model waterfall dicetuskan pada tahun 1970 sebagai contoh metodologi pengembangan perangkat lunak yang tidak bekerja secara baik [2]. Dokumen model waterfall, yang dibuat oleh Winston W. Royce, yang terdiri dari dua kategori: satu dari model itu sendiri dan yang kedua menggambarkan masalah utama yang melekat dalam model, atau alasan untuk tidak menggunakan model waterfall. Mungkin tampak aneh, kemudian, bahwa model waterfall menjadi salah satu metodologi pemrograman yang paling populer setelah publikasi dan tetap seperti itu selama bertahun-tahun. Bahkan saat ini, menurut beberapa survei, model waterfall digunakan oleh sebagian besar dunia rekayasa perangkat lunak [3].

Mengapa begitu banyak perusahaan perangkat lunak masih menggunakan model waterfall untuk mengembangkan perangkat lunak, terutama ketika berkembangnya *agile* sistem, seperti XP dan scrum, telah menjadi "topik panas" dari komunitas rekayasa perangkat lunak untuk beberapa tahun ini? Ada banyak alasan mengapa, tapi yang paling penting adalah daya tahan alami untuk mengubah tradisi dan komplikasi dari isu tersebut.

Dalam rekayasa perangkat lunak, metodologi baru umumnya ditawarkan sebagai pembawa perbaikan yang murah pada waktu untuk rilis, efisiensi, dan jumlah kesalahan per rilis. Tapi perbaikan setiap kata sangat berarti bahwa harus ada sistem mapan atau badan hasil terhadap yang untuk membandingkan klaim metodologi baru. Banyak kali, sistem dibandingkan pengembangan perangkat lunak adalah model waterfall, dan perbaikan diukur terhadap keuntungan yang ada dan kerugian dari model tersebut. Jadi, ketika mengusulkan sebuah sistem yang, misalnya, "lebih efisien" dan menghasilkan lebih sedikit kesalahan per baris kode, seseorang harus menambahkan "dari model waterfall" atau sesuatu yang lebih

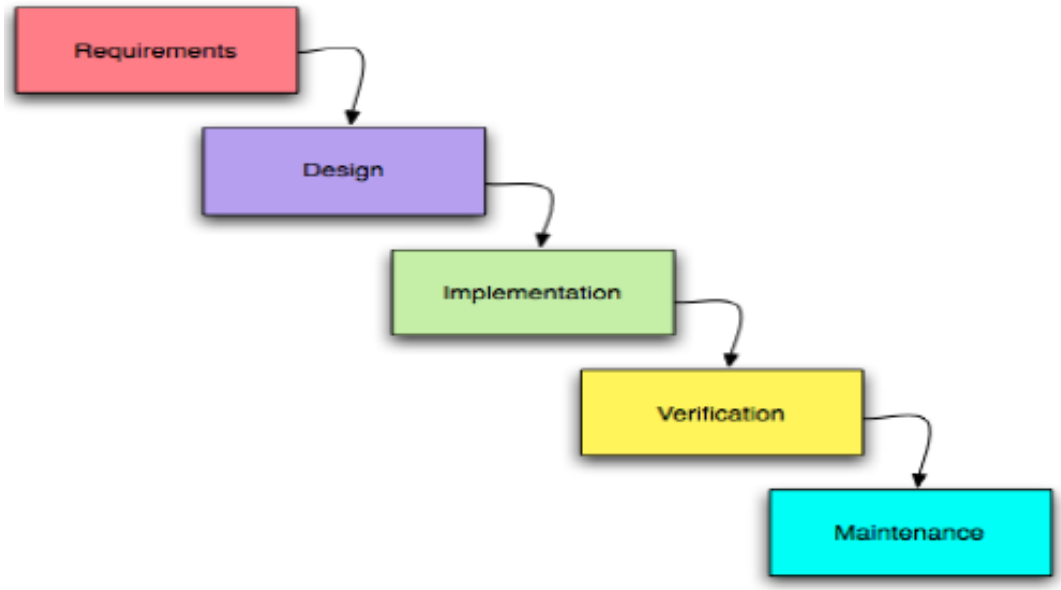
generik, seperti "daripada metode sebelumnya digunakan perangkat lunak pembangunan "untuk memungkinkan klaim kepercayaan.

Untuk membuat motif untuk kepentingan kita dalam berapa banyak pengembang menggunakan model waterfall dibandingkan pemrograman ekstrim, kita harus menetapkan bahwa ada beberapa alasan untuk mengharapkan pengembang untuk memilih satu atau yang lain. Manfaat lebih gesit model waterfall di berbagai daerah dari proses pengembangan perangkat lunak dasar yang cukup untuk menjamin kepentingan.

PEMBAHASAN

Sekuensial Software Pengembangan: Model Waterfall

Model waterfall adalah proses pengembangan perangkat lunak tradisional yang umum digunakan dalam proyek-proyek perangkat lunak yang paling pembangunan. Ini adalah model sekuensial, sehingga penyelesaian satu set kegiatan menyebabkan dimulainya aktivitas berikutnya. Hal ini disebut waterfall karena proses mengalir "secara sistematis dari satu tahap ke tahap lainnya dalam mode ke bawah [4]". Membentuk kerangka kerja untuk pengembangan perangkat lunak. Beberapa varian dari model ada, setiap label yang berbeda menggunakan untuk setiap tahap. Secara umum, bagaimanapun, model ini dianggap memiliki enam tahap yang berbeda seperti yang ditunjukkan pada Gambar 2 yaitu: analisis Kebutuhan, desain, implementasi, verifikasi, instalasi dan pemeliharaan [5].



Gambar 1. Waterfall Model

1. Kebutuhan berbasis Pengujian

Ini adalah langkah pertama dan paling penting dari model waterfall. Ini melibatkan pengumpulan informasi mengenai solusi akhir dari kebutuhan pelanggan pelanggan dan pemahaman. Ini melibatkan definisi yang jelas tentang tujuan pelanggan, harapan terhadap proyek dan masalah produk akhir diharapkan untuk memecahkan. "Analisis meliputi pemahaman konteks bisnis pelanggan dan kendala, fungsi produk harus melakukan, tingkat kinerja itu harus mematuhi dan sistem eksternal itu harus sesuai dengan" [5].

Elisitasi persyaratan adalah proses mengumpulkan informasi dari para pemangku kepentingan dari sistem. Beberapa teknik digunakan untuk elisitasi adalah pelanggan wawancara, prototyping cepat, kasus penggunaan dan brainstorming. Kasus penggunaan umumnya persyaratan fungsional dari sistem .

2. Desain

Langkah ini dimulai dengan menggunakan informasi yang ditangkap di SRS. Ini dapat dianggap sebagai memberikan solusi untuk masalah di lingkup menggunakan sumber daya yang tersedia. Tahap ini terdiri dari bagaimana perangkat lunak akan dibangun, dengan kata lain perencanaan solusi perangkat lunak. Para pemangku kepentingan yang terlibat dalam modul ini adalah para perancang sistem. Desain perangkat lunak mungkin mencakup desain sistem dan desain komponen [6].

"Tahap desain melibatkan mendefinisikan perangkat keras dan perangkat lunak arsitektur, menentukan kinerja dan parameter keamanan, merancang kontainer penyimpanan data dan kendala, memilih IDE dan bahasa pemrograman, dan menunjukkan strategi untuk menghadapi masalah-masalah seperti penanganan eksepsi, pengelolaan sumber daya dan konektivitas antarmuka [5] ".

Di sinilah desain pengguna sistem juga diakui tergantung pada kemudahan akses dan navigasi. Output dari tahap ini adalah satu atau lebih software desain deskripsi (SDD) yang berfungsi sebagai masukan untuk tahap berikutnya [5].

Pada 9 Maret 2009, Google merilis Android versi 1.1. Android versi ini dilengkapi dengan pembaruan estetis pada aplikasi, jam alarm, *voice search* (pencarian suara), pengiriman pesan dengan Gmail, dan pemberitahuan email.

3. Implementasi

Masukan dari fase ini adalah SDD sistem. "Di sinilah perkembangan aktual sistem terjadi sesuai dengan spesifikasi desain. Langkah ini dilakukan oleh pengembang, desainer interface dan stakeholder lainnya dengan menggunakan alat seperti compiler, debugger, penerjemah dan editor media.
Output dari langkah ini adalah komponen produk satu atau lebih yang dibangun berdasarkan standar yang telah ditetapkan coding dan perbaikan, pengujian dan terintegrasi untuk memenuhi kebutuhan arsitektur sistem "[5].

4. Pengujian: Verifikasi dan Validasi

Pada fase ini kedua komponen individu dan solusi terintegrasi yang diverifikasi untuk melihat itu adalah bug gratis dan memenuhi spesifikasi kebutuhan perangkat lunak. Tester adalah stakeholder yang terlibat dalam fase model. Uji kasus ditulis untuk mengevaluasi apakah sistem sepenuhnya atau sebagian memenuhi persyaratan sistem. Pengujian dapat dikategorikan ke dalam unit testing (dilakukan pada modul tertentu kode), sistem pengujian (untuk melihat bagaimana sistem bereaksi ketika semua modul yang terintegrasi) dan penerimaan pengujian (dilakukan dengan atau nama pelanggan untuk melihat apakah semua kebutuhan pelanggan puas). Cacat yang ditemukan pada tahap ini diberikan sebagai umpan balik kepada para pengembang yang pada gilirannya memperbaiki masalah. Ini adalah tahap di mana produk dikembangkan didokumentasikan. Buku pedoman ini diterbitkan pada fase ini [5].

5. Maintenance

Instalasi

Fase ini terjadi setelah akhir-produk telah diuji dan disetujui oleh pelanggan. "Langkah ini melibatkan penyusunan sistem atau produk untuk instalasi dan penggunaan di lokasi pelanggan". Pengiriman produk dilakukan melalui internet atau melalui metode fisik. Sejumlah revisi biasanya ditandai samping diserahkan untuk memfasilitasi update atau perubahan pada tahap berikutnya [5].

Pemeliharaan

Ini adalah tahap akhir dari model waterfall dan terjadi setelah instalasi sistem produk / di lokasi pelanggan. "'Tahap ini melibatkan membuat modifikasi pada sistem atau komponen individu untuk mengubah atribut atau meningkatkan kinerja sistem". Modifikasi yang muncul karena perubahan permintaan dipicu oleh pelanggan atau cacat yang ditemukan saat menggunakan sistem secara real time. Nomor revisi diperbarui dalam setiap rilis pemeliharaan [5].

Persamaan Model Proses

1.Kebutuhan berbasis Pengujian Waterfall, dan XP mengakui bahwa sebelum pembangunan dapat terjadi saling pengertian antara pelanggan dan pengembang harus ada. [7].

2.Berurusan dengan Risiko

Waterfall meringankan risiko persyaratan rumit yang harus dipahami dengan seksama sebelum proses pembangunan dapat dimulai. Dengan perencanaan dan teliti menganalisa kebutuhan pelanggan ia menawarkan pengembang gambaran yang jelas tentang apa yang perlu disampaikan untuk memenuhi kebutuhan tersebut. XP berkaitan dengan risiko perubahan, yaitu ia tidak pernah berencana depan [8]. Perencanaan ini dilakukan hanya untuk saat ini dan iterasi / atau rilis. Hal ini juga berkaitan dengan risiko moneter; selalu ada penyampaian berfungsi penuh bagi pelanggan jika anggaran proyek dikurangi atau dipotong sepenuhnya.

Kontras antara Model Proses

2.1 Dokumentasi dan Publikasi

Salah satu yang terjun berfokus pada setiap tahap memiliki dokumentasi dan pengiriman untuk lolos ke tahap berikutnya. Tujuannya adalah bahwa setiap tahap tahu persis apa yang dibutuhkan dan kapan persyaratan yang harus dipenuhi. Seluruh proyek merupakan salah satu alur kerja besar dengan jelas, tonggak tidak berulang. Sifat linear Waterfall memberikan pelanggan garis yang jelas kapan proyek harus selesai dan apa yang akan disampaikan, sejak awal dari SDLC.

Seperti setiap langkah hanya dapat dilakukan sekali, dan dalam rangka itu, cocok untuk masalah di mana persyaratan sangat dipahami dengan baik dari awal dan ada risiko rendah perubahan selama proyek. XP mencoba untuk mengurangi jumlah dokumentasi sebanyak mungkin dengan merencanakan hanya sebelum setiap iterasi, dan setiap iterasi memberikan produk kerja.

2.2. Kualitas dan Kebenaran

Waterfall juga memiliki fokus yang kuat pada kualitas dan kebenaran perangkat lunak, tapi gagal untuk menawarkan proses di mana perangkat lunak tersebut dapat diperbaiki jika

masalah ditemukan pada akhir proses pembangunan. XP bekerja untuk memecahkan ini.

2.3. Ringan kode dan Keterlibatan Pelanggan

Fokus XP adalah pada kepuasan pelanggan dengan efektif menanggapi berubah. XP memaksa beberapa iterasi pengumpulan kebutuhan, pengembangan, dan penilaian. Tes, dengan kode minimal yang diperlukan untuk membuat mereka lulus, ditulis untuk memastikan validitas terus-menerus dan mengurangi kode yang mungkin tidak diperlukan sama sekali [1].

XP membutuhkan Pelanggan di tempat untuk membuat persyaratan, merespon dengan cepat atas pertanyaan pengembang, dan melakukan pengujian penerimaan. Pelanggan selalu terlibat dalam memutuskan apa yang dibutuhkan dan apa yang dapat diterima. Pada akhir iterasi atau melepaskan pelanggan selalu akan divalidasi dan diverifikasi perangkat lunak, dan itu akan menjadi perangkat lunak yang paling berharga bagi mereka. Waterfall merencanakan semua fungsi dan kode hanya pada awal SDLC, dan memberikan setelah proses selesai. Persyaratan pelanggan mungkin telah berubah pada akhir proses ini.

2.4. Stabilitas Persyaratan

Model seperti Waterfall dan V-Model tergantung pada stabilitas persyaratan, yang sering kurang. Joe Marasco (2006) menyatakan dalam 50% dari proyek yang gagal alasannya terletak pada analisis kebutuhan. [9].

Manifesto untuk Agile Software (2001), yang XP mencoba untuk mematuhi, mengekspresikan nilai "menanggapi berubah menyusul rencana". Waterfall kedua tidak memungkinkan untuk perubahan persyaratan pelanggan. Setelah tahap persyaratan dilewatkan di Waterfall, proses tersebut harus selesai. XP mencoba untuk mengatasi masalah ini, dan ini dijelaskan dalam bagian berikut.

Kelebihan dan Kekurangan Waterfall

Pada model waterfall, beberapa prinsip utama yaitu project dibagi-bagi dalam beberapa fase yang saling berurutan, penekanan pada perencanaan, jadwal (schedule), deadline, budget, dan implementasi keseluruhan sistem sekaligus, sekaligus kontrol yang ketat dalam siklus hidup project dengan menggunakan bantuan dokumentasi tertulis.

Sedangkan kelebihan dari metode waterfall meliputi model ini mempunyai kemudahan untuk dimengerti, mudah digunakan, requirement dari sistem bersifat stabil, baik dalam manajemen kontrol, serta bekerja dengan baik ketika kualitas lebih diutamakan dibandingkan dengan biaya dan jadwal (deadline).

Selain kelebihan-kelebihan di atas, model Waterfall ini memiliki kekurangan yang sangat banyak. Dikarenakan sangat banyak tim yang telah mengimplementasikan projectnya dengan menggunakan model ini. Beberapa diantara kekurangannya itu yakni semua kebutuhan sistem harus diketahui terlebih dahulu, dapat memberikan kesan palsu dari progresnya, tidak menunjukkan menunjukkan prinsif ”ProblemSolving” dalam Pengembangan Perangkat Lunak dikarenakan fase yang harus berurut, integrasi sekaligus di akhir sistem, customer hanya memiliki sedikit kesempatan untuk melihat dan mereview sistem (yakni di akhir project), resiko sangat tinggi, karena testing hanya dilakukan pada setiap akhir fase, bahkan tidak jarang testing hanya dilakukan di akhir-akhir project, membutuhkan waktu yang cukup lama (walaupun projectnya tidak terlalu besar), perubahan requirement dapat merubah keseluruhan proses yang telah dilaksanakan.

Kelebihan dan Kekurangan Extreme Programming dibandingkan Waterfall

Sebuah survei yang dilakukan tahun 2008 menunjukkan bahwa 67% dari praktisi agile mengalami perbaikan proses pengembangan perangkat lunak mereka [10]. Perbaikan yang dibawa oleh pemrograman agile datang dalam banyak tahap perkembangan yang berbeda,

dan sering tidak butuh waktu lama untuk menunjukkan diri mereka dalam proses secara keseluruhan. Memang, beberapa arsitek perangkat lunak mengalami manfaat sejak awal [11]. Daripada menghabiskan jumlah besar waktu peramalan durasi proyek, memprediksi semua fitur mungkin untuk faktor mereka ke dalam prediksi, dan penulisan spesifikasi dan dokumentasi, pengembang dapat memulai

segera setelah mendapat informasi dari set pertama fungsi. Pendekatan ini mengurangi waktu keseluruhan yang dihabiskan pada desain, karena pengembang tidak harus dokumen fitur yang tidak dapat dikembangkan pula karena batas waktu larangan. Dalam arti, metodologi agile pendukung "mendapatkan penggelindingan bola," menyebabkan produktivitas cepat sementara juga memberikan manajemen waktu untuk lebih menentukan arah untuk keseluruhan proyek.

Iterasi pendek agile dalam pemrograman juga membantu memastikan bahwa tim pengembangan adalah tugas yang erat memaksa menangani tujuan utama setiap iterasi, daripada hanyut terpisah dalam tujuan seperti yang umum di waterfall model.
Fenomena hanyut terpisah dalam tujuan satu masalah ketika berbicara tentang programer, dan sama sekali berbeda ketika berbicara tentang sebuah memutuskan antara klien dan pengembang. Salah satu nyata kekuatan dari metodologi pemrograman agile adalah fokus pada komunikasi meningkat antara pengembangan tim dan klien [10]. Peningkatan hasil komunikasi dalam beberapa manfaat. Tidak seperti model waterfall, gesit memperhitungkan fakta bahwa kebanyakan klien tidak tahu fitur apa yang mereka ingin produk mereka untuk memiliki, atau bahkan persis produk apa yang seharusnya, sejak awal.

Sebuah tim menggunakan model waterfall mungkin mengumpulkan ide dari harapan klien, hanya untuk menemukan bahwa mereka harapan disalahartikan, setelah berbulan-bulan produksi [12].
Dalam sistem agile, seperti salah tafsir akan ditemukan langsung setelah iterasi yang melibatkan fitur yang diharapkan (jika tidak selama iterasi itu sendiri), yang oleh sifat lincah, tidak pernah merupakan jangka waktu yang panjang. Aliran konstan komunikasi antara klien dan pengembang membantu menyempurnakan sedikit demi sedikit produk dari waktu ke waktu, daripada membuat pengembang menebak apa yang diinginkan klien untuk mencapai.

Karena klien itu sendiri sering tidak tahu apa yang mereka inginkan, prototipe dibuat sebagai sesering setelah setiap iterasi dalam metodologi agile membantu mereka mempersempit rencana mereka untuk proyek tersebut, sehingga membantu pengembang fokus pada bagian yang paling penting dari produk.
Ini tak menyimpang, fokus pada bagian yang paling penting dari perangkat lunak mengarah ke kemungkinan penurunan tanggal rilis tertunda, karena fitur yang lebih sepele dapat dibatalkan atau dirilis sebagai pemutahiran jika tanggal rilis datang lebih cepat dari diantisipasi [13]. Hal ini berbeda dengan model waterfall, di mana infrastruktur sistem yang penting dapat tidak sepenuhnya dibangun sampai akhir proyek. Pada saat itu mungkin terlalu terlambat untuk menerapkan utama

Fitur di atas infrastruktur yang jika ada masalah ditemukan di sepanjang jalan, dan model waterfall telah kemudian menimbulkan rilis lain terlambat. Menyampaikan produk pada waktu berdiri tepat di samping merilis sebuah produk kesalahan-bebas dalam disiplin rekayasa perangkat lunak [14]. Kesalahan catching dan koreksi daerah lain di mana waterfall

Model gagal untuk bersaing dengan pemrograman agile. Dengan agile, fungsi masing-masing diuji secara ketat selama dan pada akhir setiap iterasi, sehingga menangkap kesalahan ketika mereka sangat lokal dan murah untuk memperbaikinya. Di model waterfall, kesalahan kaskade menuruni waterfall proses, hanya untuk mendarat di tumpukan di akhir proyek.

Dengan demikian, pada saat para pengembang perangkat lunak harus gembira tentang pelepasan mereka produk, mereka malah berebut di sekitar di dasar waterfall, bertanya-tanya bagaimana mereka akan menggali semua kesalahan sebelum klien tiba pada hari tenggat waktu untuk melihat apa yang mereka harapkan akan menjadi, berkilau elegan waterfall. Semua citra samping, matematika juga mendukung keunggulan agile dalam koreksi kesalahan: dalam survei tersebut, 45% peserta mencatat penurunan cacat dalam produk mereka [15].

Setiap metodologi rekayasa perangkat lunak memiliki pendukung dan lawan, dan di atas penalaran tidak bermaksud untuk menunjukkan bahwa metodologi agile bebas dari kritik. Salah satu yang terbesar masalah adalah kurangnya fokus pada dokumentasi dan spesifikasi yang beberapa metode agile meresepkan [10]. Proses dokumentasi dapat menjadi kacau, misalnya, ketika setelah melihat prototipe, klien ingin membuat banyak perubahan ke produk. Versi persyaratan ini dan menjaga dokumentasi up to date adalah masalah nyata, dan beberapa harus memakai metode yang kurang dari ideal, seperti menjaga catatan di atas kertas, untuk bersaing ketika proyek akan kecepatan penuh.

Masalah lain dengan metodologi agile yang lawan mengutip adalah kesulitan yang melekat dalam mencoba menjalankan sebuah proyek gesit ketika ts anggota tim tidak semua bekerja di lokasi fisik yang sama - bukan kritik kecil, karena 48% responden survei mengatakan tim mereka dibagikan [15]. Tim tersebar di zona waktu memiliki terutama sulit waktu dengan komunikasi yang efektif antara anggota, dan merasa lebih sulit untuk menerima klien umpan balik, sehingga menipiskan beberapa kekuatan dari agile. Lain kritik dari gesit metodologi pusat pada sifatnya yang tampaknya kacau, dengan kurangnya tonggak berbeda dan jelas mengabaikan untuk disiplin dalam perencanaan desain dan dokumentasi [12].

Dari sudut pandang beralasan, mengambil pro dan kontra dari kedua gesit dan metodologi waterfall ke rekening, manfaat agile pemrograman jelas lebih besar daripada kritiknya. Mereka kritik Meskipun demikian, memberikan setidaknya awal penjelasan mengapa pengembang perangkat lunak lebih tidak menggunakan agile. Seseorang tidak mungkin mengumpulkan pentingnya penuh dari kesenjangan antara lincah dan waterfall metodologi tanpa memahami bagaimana luas setiap hari. Sebuah studi yang dilakukan tahun 2002 menunjukkanbahwa meskipun metode agile telah ada sejak 1980-an, pengembangan perangkat lunak dominan proses adalah model waterfall, digunakan oleh lebih dari sepertiga dari 200 pengembang yang menanggapi, akuntansi untuk ribuan proyek perangkat lunak [3].

Sisa dari peserta digunakan baik pemrograman agile, sebuah custom built metodologi, atau digunakan (seperti yang dominan di kategori ini) tidak ada metode tertentu pengembangan perangkat lunak sama sekali. Update untuk survei yang dilakukan pada tahun 2008 dengan hampir pergeseran dalam hasil [16]. Angka-angka menggunakan model waterfall tetap sekitar sepertiga, namun yang digunakan metode agile tidak melebihi jumlah mereka yang menggunakan model waterfall. Jelas ada beberapa kekuatan yang tangguh yang mencegah perusahaan pengembangan perangkat lunak lebih dari konversi untuk pengembangan agile.

Sering kali manajemen senior akan menjadi halangan untuk perubahan yang diperlukan [17].

Sebuah software tim pengembangan dapat meyakinkan manajemen mereka untuk memberi mereka pergi hijau untuk menggunakan metode agile pada mereka berikutnya proyek. Masalah dimulai ketika manajemen tidak memahami implikasi penuh bergerak ke sistem gesit, dan tidak memungkinkan para pengembang kebebasan yang cukup untuk mengimplementasikan sistem, dengan semua nya perbedaan dengan metodologi warisan, apakah itu waterfall atau metodologi sama sekali. Ketika sebuah pengertian yang tepat dari kemajuan tidak dapat ditentukan, manajemen dapat tarik steker, bersikeras restart menggunakan metode tradisional, atau memang tidak memungkinkan proyek-proyek masa depan untuk dikembangkan dengan agile. Sejak saat itu manajemen senior telah memiliki pengalaman buruk dengan agile bahwa mereka akan mengingat kapan saja seseorang mengusulkan perubahan untuk mencoba metodologi tersebut lagi, mungkin memimpin mereka untuk menolak sambil mengutip logika "Setidaknya model waterfall dapat diandalkan."

Ini keengganan untuk mengubah sebenarnya adalah masalah besar memperlambat perluasan metode agile. Metode baru yang sangat berbeda dari pendekatan sequencing ketat model waterfall, dan dengan demikian banyak persiapan - mental dan prosedural - diperlukan untuk memberikan sistem agile kesempatan [13]. Berulang pendekatan yang metode agile mengusulkan kebutuhan komitmen untuk tingkat sangat meningkat komunikasi yang tidak diperlukan oleh model waterfall, dan tanpa pergeseran paradigma, manfaat jatuh agile untuk pinggir jalan [11]. Jika klien tidak memiliki pengalaman dengan sistem pembangunan agile, mereka mungkin waspada terhadap jumlah input mereka harus memberikan, atau tentang bagaimana karya desain sedikit mereka akan melihat sebelum proyek lepas landas. Karena adanya ketidakpastian ini, mengkonversi ke pengembangan agile dapat lebih rumit dari satu mungkin berpikir. Artinya, beberapa masalah dalam migrasi dari model waterfall untuk agile tidak hanya terletak pada mendapatkan perangkat lunak perusahaan pengembangan untuk berubah, tetapi juga meyakinkan klien yang bahwa perubahan itu menjadi lebih baik. Tanpa basis klien yang luas mendukung gesit, sulit untuk membuat

manfaat dari program gesit menarik cukup bahwa pengembang akan menganggapnya sebagai pilihan yang valid. Ini putuskan dalam metodologi antara pengembang dan klien tetap menjadi salah satu penyebab stagnasi industri, dan alasan mengapa lebih tidak beralih ke pemrograman agile [13].

Selain kelebihan-kelebihan di atas, Agile Model Terutama **Extreme Programming** juga mempunyai beberapa kekurangan, yaitu developer harus selalu siap dengan perubahan karena perubahan akan selalu diterima,tidak bisa membuat kode yang detail di awal (prinsip simplicity dan juga anjuran untuk melakukan apa yang diperlukan hari itu juga).

KESIMPULAN

Perbandingan dan kontras antara model proses perangkat lunak telah diuraikan kekuatan dan kelemahan masing-masing, yang memberikan nilai dalam dan dari dirinya sendiri. Dengan memahami model-model terbaru, tema dan tren kita dapat lebih akurat menerapkannya dalam berbagai situasi yang unik. Saya percaya tren masa depan diperkirakan akan menjadi bagian dari proses perangkat lunak yang memberikan nilai lebih

besar dari orang-orang yang mendahului mereka. Dengan memahami masa lalu kita dapat lebih memenuhi kebutuhan masa depan.

Tidak ada model proses yang akan sesuai dengan semua situasi yang ditemukan dalam situasi pengembangan perangkat lunak, sebagai studi yang disebutkan dalam tulisan ini telah menunjukkan. Pertanyaan diajukan di awal, jangan model ini menyediakan perangkat lunak nilai, adalah salah satu yang tidak mudah dijawab. Pada tingkat individu, dalam beberapa kasus ya, dalam beberapa kasus tidak. Saya percaya bahwa, pada umumnya, setiap proyek mengikuti model proses memberikan kontribusi nilai. Hal ini memungkinkan kita untuk menilai dan berkembang proses yang digunakan untuk mengembangkan model proses yang lebih baik untuk masa depan proyek.

DAFTAR PUSTAKA

[1] Beck, K. (1999b) 'Embracing change with extreme programming', *Computer*, 32(10): 70-77

[2] Toikkanen, Tarmo. "Don't draw diagrams of wrong practices - or: Why people still believe in the Waterfall model." 9 Sept 2005. 19 Oct 2008, http://tarmo.fi/blog/2005/09/09/dont-drawdiagrams- of-wrong-practices-or-why-people-still-believe-in-the waterfall-model/>.

[3] Davidson, Michelle. "Survey: Agile interest high, but waterfall still used by many." *Search Software Quality*. 27 June 2008. 20 Oct 2008 <http://searchsoftwarequality.techtarget.com/news/article/0,,sid92_gci1318992,00.html>.

[4] [Online]: http://www.waterfall-model.com/

[5] TechRepublic [Online]: http://www.techrepublic.com/article/understanding the-pros-and-cons-of-the-waterfall-model-of software-development/6118423

[6] [Online]: http://www.coleyconsulting.co.uk/waterfall-model.htm

[7] Gao, Y. (2010) 'Research on the rule of evolution of software development process model ' *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*. pp. 466 – 470.

[8] Beck, K. (1999a) *Extreme programming explained: embrace change*. Addison-Wesley Professional.

[9] Marasco, J. (2006) 'Software development productivity and project success rates: Are weattacking the right problem?', *IBM developerWorks,* Available from http://www.ibm.com/developerworks/rational/library/feb06/marasco/index.html

10] Laplante, Phillip, & Colin Neill. ""The Demise of the Waterfall Model Is Imminent' and Other Urban Myths." *ACM Queue*. Feb 2004. 19 Oct 2008

http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=110

[11] Kidwell, Jim. "The agile software development model." *The Extensis Blog*. 16 Nov 2007. 19 Oct 2008 <http://blog.extensis.com/?p=987>.

[12] "Understanding the pros and cons of the Waterfall Model of software." *Builder AU*. 19 Feb 2007. 19 Oct 2008 http://www.builderau.com.au/strategy/designprinciples/soa/Understanding-the-prosand-cons-of-the-Waterfall-Model-of-softwaredevelopment/0,339028846,339273696,00.htm

[13] Ritchie, Rose and Bernie Michalik. "Converting a project from a waterfall to an iterative approach." *developerWorks*. 15 May 2006. 19 Oct 2008 <http://www-128.ibm.com/developerworks/rational/library/may06/ritchie/index.html>.

[14] Schach, Stephen. *Object-Oriented & Classical Software Engineering*. 2007.

[15] Frye, Colleen. "Agile practitioners face challenges, but see process improvements." *Search Software Quality*. 27 June 2008. 20 Oct 2008<http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318820,00.html>.

[16] Davidson, Michelle. "Survey: Agile interest high, but waterfall still used by many." *Search Software Quality*. 27 June 2008. 20 Oct 2008 <http://searchsoftwarequality.techtarget.com/news/article/0,,sid92_gci1318992,00.html>.

[17] Ambler, Scott. "Converting to Agile against resistance." *Dr. Dobb's Portal*. 1 July 2003. 19 Oct 2008 <http://www.ddj.com/showArticle.jhtml;?articleID=184415008>.