REQUIREMENTS MANAGEMENT PADA EXTREME PROGRAMMING

Widodo, Massus Subekti
Jurusan Teknik Elektro, Fakultas Teknik, Universitas Negeri Jakarta
Jl. Rawamangun Muka, Jakarta. 13220
E-mail: widodo03@yahoo.com

ABSTRAKSI

Metodologi pengembangan perangkat lunak yang berkembang saat ini telah beralih ke metodologi yang lebih sederhana. Metodologi yang dikenal sebagai agile methods ini mengutamakan fleksibilitas terhadap perubahan-perubahan yang terjadi selama pengembangan. Bahkan perubahan ataupun penambahan pada saat fase terakhir pun teratasi apabila menggunakan metodologi ini. Salah satu yang paling popuper yaitu eXtreme Programming. Metodologi ini memiliki keunikan yang sebenarnya juga bisa menjadi kelemahannya, yaitu tanpa dokumentasi formal. Makalah ini akan mencoba memasukkan dokumentasi sederhana pada extreme programming sebagai requirements management. Selanjutnya hasil penambahan sebagai fase requirements management tersebut diuji dengan distribusi normal untuk menunjukkan bahwa penambahan tersebut mempertahankan eXtreme Programming dalam batas-batas agile method.

***Kata kunci**: Agile methods, eXtreme Programming, requirements management.*

1. PENDAHULUAN

Bidang pengembangan perangkat lunak berkembang sangat pesat dewasa ini. Dalam perkembangannya ini tidak terlepas dari peran metodologi yang digunakan untuk membangun. Jika kita lihat kembali ke belakang, berbagai metodologi untuk mengembangkan perangkat lunak telah cukup banyak diperkenalkan. Mulai dari model *waterfall* sampai dengan model-model *incremental*. Semua metodologi yang berkembang sebelumnya tidak mampu menangani kemungkinan perubahan atau penambahan *requirements* yang terjadi pada saat pengembangan dilaksanakan atau bahkan pada waktu fase terakhir pengembangan. Hal inilah yang mendorong munculnya metodologi atau model pengembangan perangkat lunak yang baru yang mampu mengatasi kekurangan tersebut.

Pada dekade 90-an diperkenalkan metodologi baru yang dikenal dengan nama *agile methods.* Metodologi ini sangat revolusioner perubahannya jika dibandingkan dengan berbagai metode sebelumnya. *Agile Methods* dikembangkan karena pada metodologi tradisional terdapat banyak hal yang membuat proses pengembangan tidak dapat berhasil dengan baik sesuai tuntutan *user*. Saat ini metodologi ini sudah cukup banyak berkembang, di antaranya adalah:

- *eXtreme Programming (XP)*
- *Scrum Methodology*
- *Crystal Family*
- *Dynamic Systems Development Method (DSDM)*
- *Adaptive Software Development (ASD)*
- *Feature Driven Development (FDD)*

Jika kita lihat, *agile* bisa berarti tangkas, cepat, atau ringan. *Agility* merupakan metode yang ringan dan cepat dalam pengembangan perangkat lunak. *Agile Alliance* mendefinisikan 12 prinsip untuk mencapai proses yang termasuk dalam *agility*:

1. Prioritas tertinggi adalah memuaskan pelanggan melalui penyerahan awal dan berkelanjutan perangkat lunak yang bernilai.
2. Menerima perubahan *requirements* meskipun perubahan tersebut diminta pada akhir pengembangan.
3. Memberikan perangkat lunak yang sedang dikerjakan dengan sering, beberapa minggu atau beberapa bulan, dengan pilihan waktu yang paling singkat.
4. Pihak bisnis dan pengembang harus bekerja sama setiap hari selama pengembangan berjalan.
5. Bangun proyek dengan individu-individu yang bermotivasi tinggi dengan memberikan lingkungan dan dukungan yang diperlukan, dan mempercayai mereka sepenuhnya untuk menyelesaikan pekerjaannya.
6. Metode yang paling efektif dan efisien dalam menyampaikan informasi kepada tim pengembangan adalah dengan komunikasi langsung *face-to-face*.
7. Perangkat lunak yang dikerjakan merupakan pengukur utama kemajuan.
8. Proses *agile* memberikan proses pengembangan yang bisa ditopang. Sponsor, pengembang, dan *user* harus bisa menjaga ke-konstanan langkah yang tidak pasti.
9. Perhatian yang terus menerus terhadap rancangan dan teknik yang baik meningkatkan *agility*.
10. Kesederhanaan –seni untuk meminimalkan jumlah pekerjaan– adalah penting.
11. Arsitektur, *requirements*, dan rancangan terbaik muncul dari tim yang mengatur sendiri.

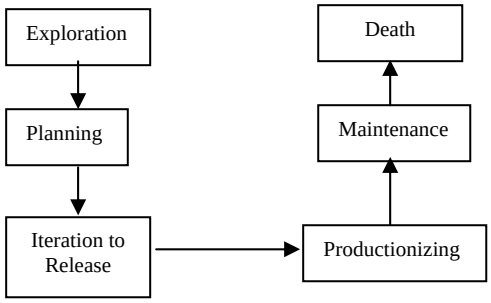
12. Pada interval reguler tertentu, tim merefleksikan bagaimana menjadi lebih efektif, kemudian menyesuaikannya.

2. EXTREME PROGRAMMING

Extreme Programming (XP) adalah metode pengembangan perangkat lunak yang ringan dan termasuk salah satu *agile methods* yang dipelopori oleh Kent Beck, Ron Jeffries, dan Ward Cunningham. *XP* merupakan *agile methods* yang paling banyak digunakan dan menjadi sebuah pendekatan yang sangat terkenal. Sasaran *XP* adalah tim yang dibentuk berukuran antara kecil sampai medium saja, tidak perlu menggunakan sebuah tim yang besar. Hal ini dimaksudkan untuk menghadapi *requirements* yang tidak jelas maupun terjadinya perubahan-perubahan *requirements* yang sangat cepat.

XP sebagai sebuah metode yang dinamis diperlihatkan dalam empat *values* yang dimilikinya dan keempatnya merupakan dasar-dasar yang diperlukan dalam *XP*. Kent Beck menyatakan bahwa tujuan jangka pendek individu sering berbenturan dengan tujuan sosial jangka panjang. Karena itu dibuatlah *values* yang menjadi aturan, hukuman, dan juga penghargaan. Keempat *values* tersebut adalah:
1. Komunikasi *(Communication*)
2. Kesederhanaan (*Simplicity*)
3. Umpan Balik (*Feedback*)
4. Keberanian (*Courage*)

Sebagai sebuah metodologi untuk mengembangkan peragkat lunak XP tentu memiliki siklus hidup. Siklus hidup pada XP ini terdapat lima fase yaitu :
1. *Exploration Phase*
2. *Planning Phase*
3. *Iteration to Release Phase*
4. *Productionizing Phase*
5. *Maintenance Phase*
6. *Death Phase*



Gambar 1. Fase pada *eXtreme Programming* (sebelum modifikasi)

Meskipun para *developer* mungkin memiliki *practices* yang berbeda namun secara mendasar *XP* memiliki 12 *practices* utama yaitu:

1. *Planning Game*
2. *Small Releases*
3. *Metaphor*
4. *Simple Design*
5. *Testing*
6. *Refactoring*
7. *Pair Programming*
8. *Collective Ownership*
9. *Continuous Integration*
10. *40-hour week*
11. *On-site Customer*
12. *Coding Standard*

Pada *practice Planning Game* sendiri memiliki tiga fase yaitu *exploration phase, commitment phase,* dan *steering phase*. *Planning game* ini adalah pertemuan antara dua pihak, yaitu dari pihak *developer* dan pihak bisnis, dibahas mulai membuat *story* oleh *on-site customer* sampai *re-estimate* oleh pihak *development.*

Tabel 1. Fase pada *planning game* (sebelum modifikasi)

NO	*PHASE*	*BUSINESS*	*DEVELOPMENT*
I	**Exploration phase**	*Write story*	-
		-	*Estimate story*
		Split story	-
II	**Commitment phase**	*Sort by value*	-
		-	*Sort by risk*
		-	*Sort by velocity*
		Choose scope	-
III	**Steering phase**	*Iteration*	-
		-	*Recovery*
		New story	*Re-estimate*

Pada fase-fase XP tidak terdapat sebuah fase ataupun bagian dari satu fase yang melakukan dokumentasi formal selama proses pengembangan. Satu-satunya dokumentasi adalah penulisan *requirements* pada *index card* yang disebut dengan *stories. Stories* ini ditulis pada fase *exploration,* di mana satu *function* yang akan diimplementasikan akan ditulis dalam satu *index card.* Selanjutnya pada proses pengembangan apabila satu *function* pada *stories* telah berhasil diimplementasikan, *index card*-nya akan dibuang. Ini bisa menjadi kelemahan XP karena tanpa dokumentasi formal maka proses pengembangan ini akan kembali seperti proses yang tidak terpola dan primitif. Apabila ditarik kepada model CMM (*Capability Maturity Model*) maka proses yang tanpa dokumentasi formal ini akan masuk ke level paling bawah yaitu lebel *initial*. Namun jika terdapat dokumentasi formal yang cukup berat, diperkirakan metodologi ini tidak menjadi ringan lagi sehingga tidak masuk dalam kategori *agile*.

3. ANALISIS PENAMBAHAN FASE REQUIREMENTS MANAGEMENT

Pada bagian ini akan dilakukan penambahan fase pada XP yaitu dengan menyisipkan sebuah fase yang disebut *requirements management phase*. Fase ini disisipkan tidak sekuensial tapi paralel dengan fase *planning*. Tujuan penyisipan secara paralel ini adalah mengurangi kemungkinan XP keluar dari lingkup *agile*. Tiga hal yang dilakukan yaitu:

 ISSN: 1907-5022

1. Penambahan *Requirements Management*

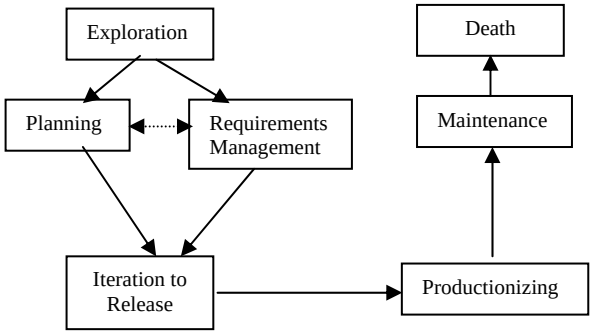
Requirements management akan dijadikan sebuah fase tersendiri namun kronologis pelaksanaannya adalah paralel dengan fase *planning.* Hal ini dimaksudkan agar penambahan fase ini tidak akan berpengaruh secara signifikan terhadap waktu pengembangan secara keseluruhan. Penambahan anggota tim juga tidak diperlukan untuk melakukan proses *requirements management* ini.

2. Pendokumentasian sederhana (*simple documenting*)

Ini merupakan implementasi dari fase *requirements management.* Pihak *development* melakukan peringkasan *stories* (*summarize stories*) ke dalam dokumen sederhana (*simple document*) dari pihak bisnis. Teknis pelaksanaannya dilakukan tidak menggunakan komputer tapi ditulis tangan. Ini dimaksudkan agar metodologi ini tetap ringan. Penulisan dengan menggunakan komputer bisa dilakukan setelah proses pengembangan berakhir. *Requirements* yang ditulis hendaknya mencantumkan identitas dari *stories* yang didokumentasikan yaitu nomor *story* pada *index card.* Ini diperlukan sebagai *trace* karena *requirements* yang ditulis pada *simple document* tidak serinci pada *story*, namun sudah dirangkum menjadi sebuah *feature.* Jika *requirements* yang dirangkum tersebut mengalami perubahan maka pada perubahan itu juga harus ada *trace*-nya untuk mengetahui asal *requirement*-nya.

3. Mempertahankan *index card* yang telah diimplementasi dengan baik.

Pada metode XP, *index card* akan dibuang apabila *story* yang terdapat di dalamnya telah berhasil diimplementasikan dengan baik. Namun pada modifikasi ini *index card* tidak perlu dibuang, karena selain sebagai bahan dokumentasi, *index card* adalah bukti dari *requirements* pada *simple document.* Pada *requirements* tersebut terdapat satu atribut yang mengarah ke *index card*, yaitu *story* asal dari *requirements* tersebut yang tertulis lebih rinci.



Gambar 2. Fase pada *eXtreme Programming* setelah modifikasi

Setelah dilakukan modifikasi terdapat beberapa pengaruh pada *practice*-nya. Dari dua belas *practice* yang terdapat pada XP, ada empat buah *practice* yang mempunyai singgungan kuat dengan *requirements management* yaitu *planning game, metaphor, 40-hour week, On-site customer.* Dari keempatnya *planning game*-lah yang paling besar keterkaitannya dengan *requirements.* Pada bagian ini akan dipaparkan yang terjadi pada keempat *practice* tersebut setelah dilakukan modifikasi pada XP tersebut.

1. *Planning Game*

Berikut ini adalah tabel setelah dilakukan perubahan pada siklus:

Tabel 2. Fase pada *planning game* (setelah modifikasi)

NO	*PHASE*	*BUSINESS*	*DEVELOPMENT*
I	**Exploration phase**	*Write story*	-
		-	*Estimate story*
		Split story	-
		-	***Summarize stories (simple documenting)***
II	**Commitment phase**	*Sort by value*	-
		-	*Sort by risk*
		-	*Sort by velocity*
		Choose scope	-
III	**Steering phase**	*Iteration*	-
		-	*Recovery*
		New story	*Re-estimate*
		-	***Summarize stories (update simple document)***

Sebelum modifikasi, pada *exploration phase* terdapat tiga kegiatan yaitu *write story, estimate story, dan split story.* Setelah dilakukan modifikasi satu kegiatan lagi yang dikerjakan oleh pihak development adalah *summarize stories into simple document.* Pihak bisnis melakukan *split story* menjadi dua atau lebih *story* apabila pihak *development* tidak dapat mengestimasi *story*, atau *story* yang ditulis terlalu kompleks. Sebaliknya pihak *development* melakukan *summarize story* yaitu *story* dirangkum menjadi sebuah *requirements* yang menjadi sebuah *feature* pada sistem yang akan diimplementasikan. Proses ini dikerjakan langsung pada waktu *customer* selesai menulis *story*, tidak perlu menunggu sampai semua *stories* selesai ditulis.

Demikian juga pada *steering phase* yang pada awalnya hanya terdiri atas kegiatan-kegiatan *iteration, recovery, new story*, dan *re-estimate*, setelah modifikasi ditambah satu lagi yaitu *update simple document.* Proses ini sebenarnya sama dengan *summarize stories* pada *exploration phase*, perbedaannya pada *steering phase* adalah untuk mengantisipasi adanya *new story* yang dikerjakan oleh pihak bisnis.

2. *Metaphor*

Practice berikutnya yang bersinggungan dengan *requirements* adalah *metaphor*. *Metaphor* merupakan panduan (*guidance*) dalam melakukan pengembangan secara keseluruhan. *Metaphor* di sini memerlukan penjelasan rinci. *Requirements* yang diperoleh melalui proses *summarize stories* tersebut berperan sebagai *metaphor*, sedangkan penjelasan rincinya ada pada *story* awal. Karena alasan tersebut *index card* yang *story*-nya telah berhasil diimplementasikan tidak perlu dibuang. Jika terjadi perubahan pada *requirements* tersebut maka untuk melacak *story* awalnya harus melihat ke *index card*, sehingga *requirements* pada *simple document* tersebut menjadi *metaphor* yang memandu proses pengembangan agar berjalan sesuai keinginan.

3. *40-hour week*

Story yang telah selesai ditulis oleh *customer* langsung dirangkum ke dalam *simple document* pada fase *requirements management.* Sementara fase tersebut berlangsung secara paralel dengan fase *planning.* Perangkumannya adalah dengan ditulis tangan, tidak melalui komputerisasi. Model dokumennya juga dibuat sesederhana mungkin agar semua pihak dapat mengerti dengan mudah. Dengan proses tersebut di atas maka tidak akan terjadi penambahan waktu secara signifikan.

4. *On-site Customer*

Komunikasi dengan *on-site customer* tetap dilakukan terus, termasuk dalam melakukan *summarize stories.* Dokumentasi yang sederhana juga akan dapat dimengerti dengan mudah oleh *customer*, karena sifatnya hanya merangkum *story* ke dalam *requirements* yang akan menjadi *feature* pada sistem yang dibuat. Untuk menjaga berbagai kemungkinan agar keinginan *user* cukup representative maka pada saat *requirements gathering* sebaiknya minimal ada dua *customer*. Satu orang menulis *stories* yang bersifat fungsional, yang lainnya menulis *stories* yang non-fungsional. *Stories* yang bersifat fungsional adalah *stories* yang berhubungan dengan *core business*-nya, sedangkan yang non-fungsional adalah yang berhubungan dengan *support* dari bisnisnya.

4. PERHITUNGAN KATEGORI *AGILE*

Dalam perhitungan batas-batas *agile* ada 14 unsur yang akan dinilai yaitu 4 *practice* yang paling bersinggungan yaitu *planning game, metaphor, 40 hour week*, dan *onsite customer*. Untuk *planning game* akan dibagi 3 langsung yaitu *exploration phase, commitment phase,* dan *steering phase*. Selain itu terdapat 10 unsur untuk menilai kategori *agile* yaitu *communication, simplicity, feedback, courage, embrace change, number of person, iterative development and design, emergence, adaptation, software first.* Penilaian untuk setiap unsur adalah 0-3, di mana semakin kecil berarti pengaruh penambahan tersebut tidak terlalu signifikan dalam mengganggu ke-*agil*-annya.

Unsur-unsur tersebut pembobotannya akan dibagi lagi dari 1-5. Pembobotan yang tertinggi ada pada fase-fase *planning game.*

Tabel 3. Pembobotan untuk penilaian *agile*.

BOBOT	***INDIKATOR***
5	*Planning game: exploration, commitment, and steering phase*
4	*Metaphor, 40 hour week, On-site customer*
3	*Communication, simplicity, feedback, embrace change, number of person*
2	*Adaptation*
1	*Courage, iterative development and design, emergence, software first*

Penilaian yang dilakukan adalah dengan mengalikan nilai pada setiap unsur dengan masing-masing bobot yang dimilikinya. Kemudian hasil perkalian unsur–unsur tersebut dijumlahkan sehingga diperoleh hasilnya.

Template untuk perhitungan *agile* adalah sebagai berikut:

Tabel 4. *Template* indikator *agile*

NO	***INDIKATOR***	***ANGKA MAX***	***BOBOT***	***NILAI TERTIMBANG***
1	*Planning Game:*			
	- exploration	3	5	15
	- commitment	3	5	15
	- Steering	3	5	15
2	*Metaphor*	3	4	12
3	*40 hour week*	3	4	12
4	*Onsite customer*	3	4	12
5	*Communication*	3	3	9
6	*Simplicity*	3	3	9
7	*Feedback*	3	3	9
8	*Courage*	3	1	3
9	*Embrace change*	3	3	9
10	*Number of person*	3	3	9
11	*Iterative development and design*	3	1	3
12	*Emergence*	3	1	3
13	*Adaptation*	3	2	6
14	*Software First*	3	1	3
TOTAL				144

Langkah berikutnya adalah menghitung batas-batas *agile*. Pengelompokkan *template* pada tabel 4 berdasarkan kelas adalah:

Tabel 5. Pengelompokkan data

Kelas	***Frekwensi/Jumlah***
1 – 3	4
4 – 6	1
7 – 9	5
10 – 12	3
13 – 15	3
Jumlah	**16**

Data pada tabel 5 dikelompokkan lagi berdasar nilai tengahnya menjadi:

Tabel 6. Pengelompokkan data dengan nilai tengah

Kelas (k)	***Nilai Tengah (x_i)***	***Frekwensi/Jumlah (f_i)***	***N.T X Fr (x_i. f_i)***
1 – 3	2	4	8
4 – 6	5	1	5
7 – 9	8	5	40
10 – 12	11	3	33
13 – 15	14	3	42
Jumlah		**16**	**128**

Dari tabel tersebut diperoleh rata-rata:
$\mathbf{x_r = \sum(x_i . f_i) / \sum(f_i)}$
$\mathbf{x_r =} 128 / 16$
$= 8$

Distribusi frekwensi dari hasil perhitungan adalah:

Tabel 7. Data distribusi frekwensi

K	***$(x_i - x_r)^2$***	***$(x_i - x_r)^2 . f_i$***
1 – 3	36	144
4 – 6	9	9
7 – 9	0	0
10 – 12	9	27
13 – 15	36	108
Jumlah		**288**

Dengan menggunakan rumus deviasi standar (SD) diperoleh:
$SD = \sqrt{288/15}$
$= \sqrt{19,2}$
$SD = \pm 4,38$

Dengan data-data yang diperoleh di atas, kita sudah dapat menghitung klasifikasi *agile*-nya dengan menggunakan distribusi normal. Data yang sudah didapatkan adalah deviasi standar (SD), rata-rata (x_r), dan angka-angka pada indikatornya. *SD = 4,38, X_r = 8.*

Nilai X sendiri ada 16. Data yang tertinggi yaitu 15 akan dijadikan sebagai batas *agile* dengan menghitung luas kurva normalnya. Hal yang sama akan dihitung dengan dua buah data yang nilainya berada di bawah 15, yaitu 12 dan 9, sebagai klasifikasi cukup *agile*, *agile*, dan sangat *agile*.

Berikut perhitungannya:

1) Untuk nilai X tertinggi yaitu x = 15
$\check{Z} = (x - X_r) / SD = (15 – 8) / 4,38$
$= 1,6$

Luas kurva normal untuk variabel random ($\check{Z}$) = 1,6 sesuai tabel luas kurva normal adalah 0,4452.
Angka ini artinya batasan modifikasi ini tetap berada pada kondisi **cukup *agile*** adalah **44,52.** Lewat dari ini sudah tida *agile* lagi.

2) Untuk nilai X berikutnya yaitu x = 12
$\check{Z} = (x - X_r) / SD = (12 – 8) / 4,38$
$= 0,91$

Luas kurva normal untuk variabel random 0,91 adalah 0,3186.
Angka ini menunjukkan batas maksimal ***agile*** adalah **31,86.** Lewat angka ini asal belum melewati 44,52 adalah masih cukup *agile.*

3) Untuk nilai X berikutnya yaitu x = 9
$\check{Z} = (x - X_r) / SD = (9 – 8) / 4,38$
$= 0,23$

Luas kurva normal untuk variabel random 0,23 adalah 0,0910.
Angka ini menunjukkan batas maksimal **sangat *agile*** adalah **9,10.** Lewat angka ini sudah tidak masuk klasifikasi sangat *agile*, namun asal masih belum melewati angka 44,52 berarti masih di lingkungan *agile*, atau cukup *agile.*

Dengan melihat hasil perhitungan di atas, maka dapat diklasifikasikan dalam skala 0–100, posisi yang menempatkan pada keadaan *agile*:
1. Sangat *Agile* (**A**): **0 – 9,10**
2. *Agile* (**B**): **> 9,10 – 31,86**
3. Cukup *Agile* (**C**): **> 31,86 – 44,52**
4. Tidak *Agile* (**D**): **> 44,52.**

Hasil perhitungan tersebut telah mebuat klasifikasi *Agile* dan Tidak *Agile* menjadi empat kelompok. Klasifikasi A apabila pengukuran menghasilkan angka sampai dengan 9,10. Klasifikasi B apabila pengukuran menghasilkan nilai >9,10 sampai 31,86. Klasifikasi C apabila pengukuran menghasilkan nilai >31,86 sampai 44,52. Apabila melewati angka 44,52 maka modifikasi yang dibuat tidak lagi berada dalm posisi *agile.*

Setelah diukur penambahan fase pada XP dengan menggunakan *template* yang disediakan dihasilkan tabel perhitungan sebagai berikut:

Tabel 8. Hasil perhitungan *agility* untuk penambahan fase *requirements management*

NO	***INDIKATOR***	***ANGKA***	***BOBOT***	***NILAI TERTIMBANG***
1	*Planning Game:*			
	- exploration	1	5	5
	- commitment	0	5	0
	- Steering	1	5	5
2	*Metaphor*	1	4	4
3	*40 hour week*	1	4	4
4	*Onsite customer*	2	4	8
5	*Communication*	1	3	3
6	*Simplicity*	1	3	3
7	*Feedback*	0	3	0
8	*Courage*	0	1	0
9	*Embrace change*	0	3	0
10	*Number of person*	1	3	3
11	*Iterative development and design*	0	1	0
12	*Emergence*	0	1	0
13	*Adaptation*	0	2	0
14	*Software First*	0	1	0
TOTAL				35

Total yang didapatkan adalah 35. Untuk mendapatkan skala 0–100, agar seperti luas kurva normal angka tersebut dibagi 1,44 karena total maksimumnya adalah 144. T = 35/1,44 maka T = 24,306. Hasil akhir pengukuran adalah didapatkan angka 24,306. Sesuai dengan klasifikasi yang telah dibuat, angka ini masuk ke dalam klasifikasi **B** yaitu posisi ***Agile.*** Posisi **B** ini berada pada kisaran angka >9,10 – 31,86. Dengan melihat hasil pengukuran ini berarti penambahan fase *requirements management* yang dibuat paralel dengan fase *planning*, metodologi *eXtreme Programming* tetap pada posisi *agile.* Berbagai penambahan dan perubahan yang telah dilakukan tidak menambah terlalu berat pada hasil akhirnya.

5. KESIMPULAN

Secara garis besar penambahan fase *requirements management* pada *eXtreme Programming* sangat membantu untuk lebih menstrukturkan metode ini. *Requirements Management* yang disisipkan terutama difokuskan dalam hal dokumentasi. Pendokumentasian pada *eXtreme Programming* ini tidak akan mengganggu proses secara keseluruhan karena dilaksanakan secara paralel dengan *planning phase.*

Hasil pengukuran yang dilaksanakan terhadap penambahan fase *requirements management* yang paralel dengan *planning phase* pada siklus hidup *eXtreme Programming* menunjukkan bahwa metodologi ini tetap pada posisi *agile.* Pengukuran yang dilakukan dengan menggunakan statistik yaitu dengan menilai indikator-indikator *agile*-nya yang kemudian diklasifikasikan dengan menggunakan distribusi normal memperoleh empat klasifikasi. Klasifikasi itu adalah A untuk posisi sangat *agile*, B untuk posisi *agile*, C untuk posisi cukup *agile*, dan D untuk posisi tidak *agile.* Setelah pengukuran, penambahan fase ini menunjukkan berada pada posisi B. Dengan hasil tersebut berarti penambahan fase *requirements management* tidak menjadikan *eXtreme Programming* keluar dari metodologi *Agile*.

DAFTAR PUSTAKA

[1] Abrahamsson, Peka., Salo, Outi., Ronkainen, Jussi and Juhani Warsta. *Agile Software Development Methods: Review and Analysis.* VTT Publication 478. Finland

[2] Beck, Kent., Jeffries, Ron., and Ward Cunningham. *Extreme Programming: Embrace Change.*Addison-Wesley.2000.

[3] Fagan, Mary Helen. Compiring Traditional and Agile Development Approach: The Case of Extreme Programming. *Issues in Information Systems*. Volume VI No.2, 2005

[4] Hayes, Steve. *An Introduction to Extreme Programming.* http://www.khatovartech.com**.** Khatovar Technology. 2001

[5] Hayes, Steve and Martin Andrews. *An Introduction to Agile Methods.* http://www.khatovartech.com**.** Khatovar Technology. 2001.

[6] Leffingwell, Dean., and Don Widrig. *Managing Software Requiremnents: A Unified Approach.* Addison-Wesley. Boston. 2000

[7] Paulk, Mark C. *Extreme Programming from a CMM Perspective*. http://www.sei.cmu.edu/cmm/papers/xp-cmm-paper.pdf. 2001

[8] Pressman, Roger. *Software Engineering: A Practitioner's Approach.* 6th Edition. McGraw-Hill. 2005.