BAB II
LANDASAN TEORI

2.1 Landasan Teori

Landasan teori yang digunakan untuk memperkuat penelitian ini meliputi Skripsi/Tugas Akhir, Rekayasa Perangkat Lunak (RPL), *Software Development Process* meliputi *The Waterfall Model, Incremental Development, Reuse-oriented Software Model, Agile Methodology,* Scrum dan *Burndown Chart*. Dibawah ini akan dibahas mengenai landasan teori yang digunakan dalam penelitian.

2.1.1 Skripsi

Skripsi/Tugas Akhir adalah merupakan karya ilmiah yang disusun oleh mahasiswa berdasarkan hasil penelitian laboratorium atau penelitian lapangan dengan bimbingan pembimbing dosen, untuk dipertahankan dihadapan penguji skripsi sebagai syarat untuk memperoleh gelar sarjana (Peraturan Rektor No. 53 Universitas Sebelas Maret, 2005). Prosedur pelaksanaan Skripsi/Tugas Akhir disesuaikan dengan kondisi dan situasi fakultas masing-masing dengan mengakomodasi kekhasan Jurusan/Program Studi. Penelitian Skripsi/Tugas Akhir dapat bersifat penelitian laboratorium maupun studi lapangan. Pada penelitian Skripsi/Tugas Akhir, mahasiswa akan dibimbing oleh dosen pembimbing. dosen pembimbing bertanggung jawab atas kelancaran pelaksanaan bimbingan Skripsi/Tugas Akhir. Hasil dari Skripsi/Tugas Akhir ditentukan dengan beberapa ujian yaitu ujian seminar proposal, ujian seminar hasil dan sidang atau pendadaran. Mahasiswa yang dinyatakan telah selesai menyusun naskah Skripsi/Tugas Akhir dapat diusulkan kepada panitia Skripsi/Tugas Akhir untuk mempertahankan Skripsi/Tugas Akhir di hadapan majelis penguji Skripsi/Tugas Akhir, yang terdiri dari minimal 3 (tiga) orang (Peraturan Rektor No. 53 Universitas Sebelas Maret, 2005).

2.1.2 Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak (RPL) berasal dari dua kata yaitu *Software* (Perangkat Lunak) dan *Engineering* (Rekayasa). Perangkat Lunak (*Software*) adalah *source code* pada suatu program atau sistem. Sedangkan Rekayasa (*Engineering*) adalah aplikasi terhadap pendekatan sistematis yang berdasar atas ilmu pengetahuan dan matematis serta aplikasi tentang produksi terhadap struktur, mesin, produk, proses atau sistem (Nugroho, Ratnasari, Ramadhani, & Putro, 2009). Rekayasa Perangkat Lunak merupakan perihal kegiatan yang kreatif dan sistematis berdasar suatu disiplin ilmu yang membangun suatu perangkat lunak berdasar suatu aspek masalah tertentu. Dalam rekayasa perangkat lunak, metodologi pengembangan perangkat lunak (juga dikenal sebagai *system development methodology, software development life cycle, software development process, software process*) adalah sebuah divisi dari pekerjaan pengembangan perangkat lunak dalam tahap yang berbeda atau tahap yang berisi kegiatan dengan maksud baik perencanaan dan manajemen

2.1.3 Software Development Process

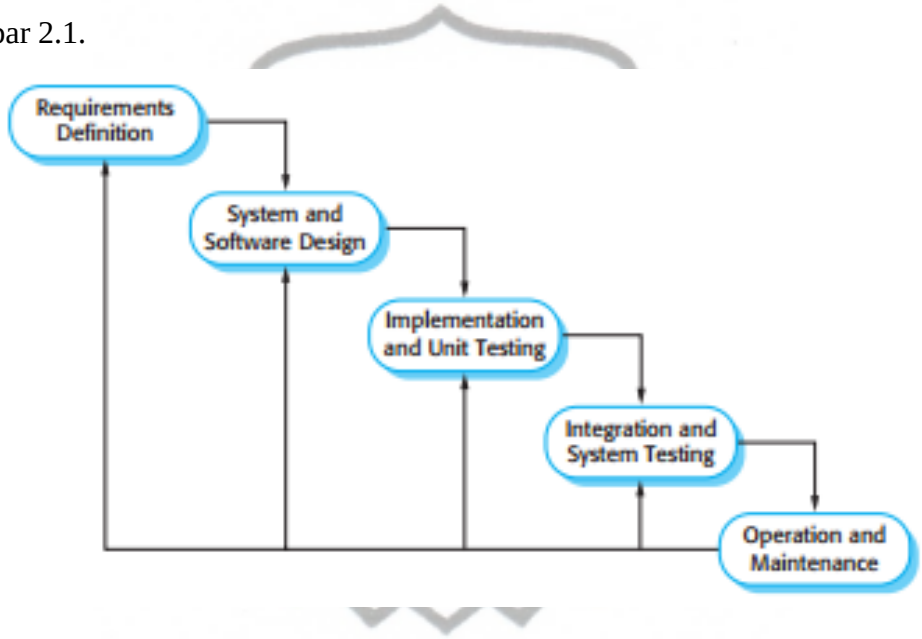
Software Development Process adalah suatu struktur yang diterapkan pada pengembangan suatu produk perangkat lunak yang bertujuan untuk mengembangkan sistem dan memberikan panduan yang bertujuan untuk menyukseskan proyek pengembangan sistem melalui tahap demi tahap (Britton, 2001). Proses ini mencakup aktivitas penerjemahan kebutuhan pemakai menjadi kebutuhan perangkat lunak, transformasi kebutuhan perangkat lunak menjadi desain, penerapan desain menjadi kode program, uji coba kode program, dan instalasi serta pemeriksaan kebenaran perangkat lunak untuk operasional (Oestereich, 1999).

Rangkaian proses aktivitas pengembangan perangkat lunak tersebut disederhanakan menjadi beberapa model proses perangkat lunak yaitu *The Waterfall*

Model, Incremental Development, Reuse-oriented Software Engineering, dan *Agile Methodology*. (Sommerville, 2011)

2.1.3.1 *The Waterfall Model*

Model air terjun (*Waterfall*) adalah proses pengembangan perangkat lunak yang berurutan (sekuensial) mengalir ke bawah seperti air terjun melalui tahap *requirements analysis* and *definition, System* and *software design, implementation* dan *unit testing, integration* dan *system testing*, serta *operation* and *maintenance* (Sommerville, 2011). Berikut adalah alur proses *waterfall model* yang ditunjukkan pada Gambar 2.1.



Gambar 2.1 *The Waterfall Model* (Sommerville, 2011)

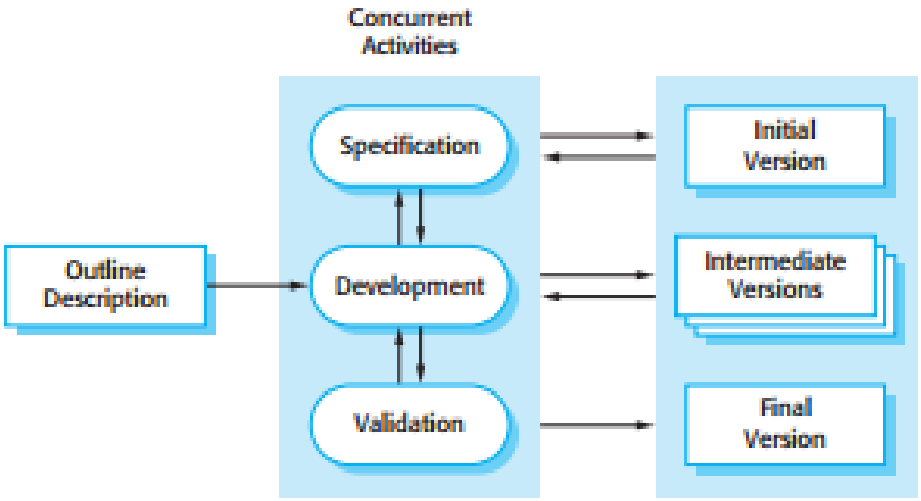
Gambar 2.1 menjelaskan bahwa pada tahap *requirements definition*, semua kemampuan sistem ditentukan dengan konsultasi kepada pengguna atau klien. Selanjutnya sistem didesain dengan mengalokasikan persyaratan baik perangkat keras atau perangkat lunak sistem dengan membentuk arsitektur secara keseluruhan. Kemudian perangkat lunak dibuat berdasarkan sistem yang telah dirancang sebelumnya dan di uji setiap kemampuan sistem apakah sesuai dengan spesifikasi. Setelah semua sistem lolos uji, kemudian perangkat lunak diberikan kepada klien

ataupun pelanggan. Pada tahap terakhir, sistem akan dioperasikan dan diperbaiki bila ada kesalahan (Sommerville, 2011).

Beberapa kelemahan dari model *waterfall* ini adalah proyek yang sesungguhnya jarang mengikuti alur sekuensial, sehingga perubahan yang terjadi menyebabkan hasil produk yang sudah dibuat harus diubah kembali. Masalah lain yaitu, ketika semua kebutuhan pemakai sudah dinyatakan secara eksplisit di awal proses, kadang tidak semua kebutuhan tersampaikan sehingga tidak memenuhi semua keinginan pengguna (Sommerville, 2011).

2.1.3.2 *Incremental Development*

Incremental Development merupakan pengembangan dari model air terjun (*waterfall*) yang sebelumnya dibahas. Model ini memiliki tahapan yang dilakukan berulang-ulang. Setiap tahapan model air terjun menghasilkan *deliverable increment* bagi perangkat lunak, dimana *increment* pertamanya merupakan sebuah produk inti yang mewakili kebutuhan dasar sistem dan disebut sebagai *initial version*. Produk inti ini nantinya akan dikembangkan menjadi *increment-increment* selanjutnya melalui *proses specification*, *development*, dan *validation* menghasilkan *intermediate version*. Setelah digunakan dan dievaluasi sampai akhirnya diperbaharui kembali dan didapat produk yang lengkap dan memenuhi kebutuhan pengguna (Sommerville, 2011). Metode *Incremental Development* dijelaskan dengan alur proses seperti Gambar 2.2 di bawah ini.



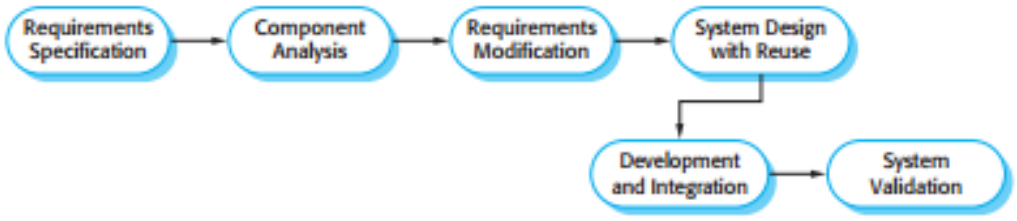
Gambar 2.2 *Incremental Model* (Sommerville, 2011)

Metode *incremental development* memiliki tiga keuntungan penting bila dibandingkan dengan metode air terjun. Keuntungan pertama yaitu biaya yang dibutuhkan untuk mengakomodasi permintaan pengguna apabila ada kebutuhan sistem yang diganti akan berkurang. Sejumlah analisis dan dokumentasi yang harus dibuat ulang lebih sedikit dibanding menggunakan metode air terjun. Keuntungan kedua yaitu membuat pengguna dengan mudah memberikan umpan balik terhadap perangkat lunak apabila ada yang tidak sesuai dengan keinginannya. Keuntungan ketiga yaitu pengguna lebih cepat menggunakan dan memberikan nilai kepada produk meskipun belum mencapai seluruh fungsi yang diinginkan (Sommerville, 2011). Namun, beberapa kekurangan juga dimiliki oleh metode ini, yaitu butuh waktu yang relatif lama untuk menghasilkan produk yang terlengkap, tidak cocok untuk diaplikasikan pada proyek berskala besar, dan juga biaya dapat meningkat jika desain fisikal dan fungsi tidak terstruktur penuh.

2.1.3.3 *Reuse-oriented Software Engineering*

Reuse-oriented Software Engineering merupakan metode yang sering terjadi ketika proyek yang akan dikerjakan mirip dengan desain atau algoritma yang pernah

dibuat sebelumnya. Desain dan algoritma ini dicari, kemudian dimodifikasi sesuai kebutuhan dan kemudian digabungkan ke dalam sistem yang akan dibuat (Sommerville, 2011). Berikut adalah model *Reuse-oriented* yang ditunjukkan pada Gambar 2.3.



Gambar 2.3 *Reuse-oriented Model* (Sommerville, 2011)

Gambar 2.3 menunjukkan bahwa pada tahap *Component Analysis* dicari semua komponen yang berhubungan dengan spesifikasi sistem. Kemudian komponen yang sudah ditemukan dimodifikasi sesuai dengan kebutuhan. Setelah itu *framework* dari sistem digunakan kembali untuk dipakai dan kemudian diintegrasikan ke dalam sistem baru.

2.1.3.4 *Agile Methodology*

Metode *agile* merupakan metode pengembangan *incremental* yang fokus pada perkembangan yang cepat, perangkat lunak yang dirilis bertahap, mengurangi *overhead* proses, dan menghasilkan kode berkualitas tinggi dan pada proses pengembangannya melibatkan pelanggan secara langsung (Sommerville, 2011).

Metode agile ditemukan oleh tujuh belas pengembang yang sedang berdiskusi untuk menciptakan *lightweight development methods.* Tujuh belas pengembang mendiskusikan dan merumuskan beberapa tujuan metode *agile* yaitu (Collier, 2011):

1. *High-value & working App system*, diharapkan dengan memakai *agile development methods* dapat dihasilkan perangkat lunak yang mempunyai nilai jual yang tinggi, biaya pembuatan bisa di tekan dan perangkat lunak bisa berjalan dengan baik.

2. *Iterative, incremental, evolutionary*, agile adalah metode pengembangan perangkat lunak yang iteratif, selalu mengalami perubahan, dan evolusioner. Tim harus bekerja dalam waktu yang singkat (biasanya 1-3 minggu) dan juga selalu menambah fungsionalitas dari perangkat lunak sesuai dengan kebutuhan klien. Agile dapat dianalogikan ketika seseorang ingin pergi ke suatu kota dan dia tidak tahu jalannya. Lalu agar dapat sampai pada tujuan yaitu dengan sering bertanya kepada orang yang dia temui dijalan hingga dia sampai di tempat tujuan.
3. *Cost control & value-driven development*, salah satu tujuan dari *agile* yaitu pengembangan perangkat lunak disesuaikan dengan kebutuhan pengguna, tim bisa dengan cepat merespon kebutuhan yang diinginkan pengguna sehingga waktu dan biaya pembuatan perangkat lunak bisa dikontrol.
4. *High-quality production*, walaupun biaya pembuatan perangkat lunak bisa ditekan dan proses pembuatan bisa dipercepat tetapi kualitas dari perangkat lunak yang dibuat harus tetap dijaga. Dengan melakukan tes setiap fungsionalitas perangkat lunak setelah selesai dibuat berarti agile juga mengakomodir kebutuhan ini.
5. *Flexible & risk management*, jika menggunakan metode pembuatan yang biasanya dipakai, jika ingin mengubah fungsionalitas dari *wireframe* yang telah dibuat di butuhkan proses yang rumit. Mulai dari pertemuan dengan sistem analis untuk mengubah sistem perangkat lunak, perubahan rencana rilis produk hingga perubahan biaya produksi. Pertemuan dengan klien untuk melakukan tes perangkat lunak juga sering dilakukan sehingga fungsionalitas perangkat lunak mudah diubah dan akhirnya kegagalan perangkat lunak pun bisa diminimalisir.
6. *Collaboration*, dengan menggunakan *agile*, tim pengembang diharuskan sering bertemu untuk membahas perkembangan proyek dan *feedback* dari klien yang nantinya akan ditambahkan dalam perangkat lunak, sehingga tim bisa berkolaborasi dengan maksimal.
7. *Self-organizing, self-managing teams*, rekrut orang terbaik, beri dan dukung kebutuhan mereka lalu biarkan mereka bekerja. Itulah perbedaan *agile* dan

metode lainnya. Dengan *agile*, *developer* dapat memanajemen dirinya sendiri, sedangkan manajer tim hanya bertugas mengkolaborasikan *developer* perangkat lunak dengan klien. Sehingga terciptalah tim yang solid.

Studi literatur tentang Metode *Agile* menjelaskan beberapa alasan yang cocok mengapa metode *agile* digunakan sebagai metode dalam pengembangan perangkat lunak. Menurut hasil analisis (Strode, 2005) dalam thesisnya yang berjudul "*The Agile Methods: An Analytical Comparison of Five Agile Methods and an Investigation of Their Target Environment", Agile Methods* adalah metodologi pengembangan perangkat lunak yang dirancang untuk mendukung pengulangan dan tahapan pengembangan sistem bisnis yang berubah secara konstan. Dalam thesisnya, (Strode, 2005) membandingkan lima metode *agile* untuk mengetahui kelebihan, kekurangan, dan lingkungan yang cocok dalam mengembangkan perangkat lunak. Berikut adalah kelima metode *agile* tersebut:

a. DSDM (*Dynamic Systems Development Method)*
b. XP (*Extreme Programming)*
c. Scrum
d. ASD (*Adaptive Software Development)*
e. *Crystal*

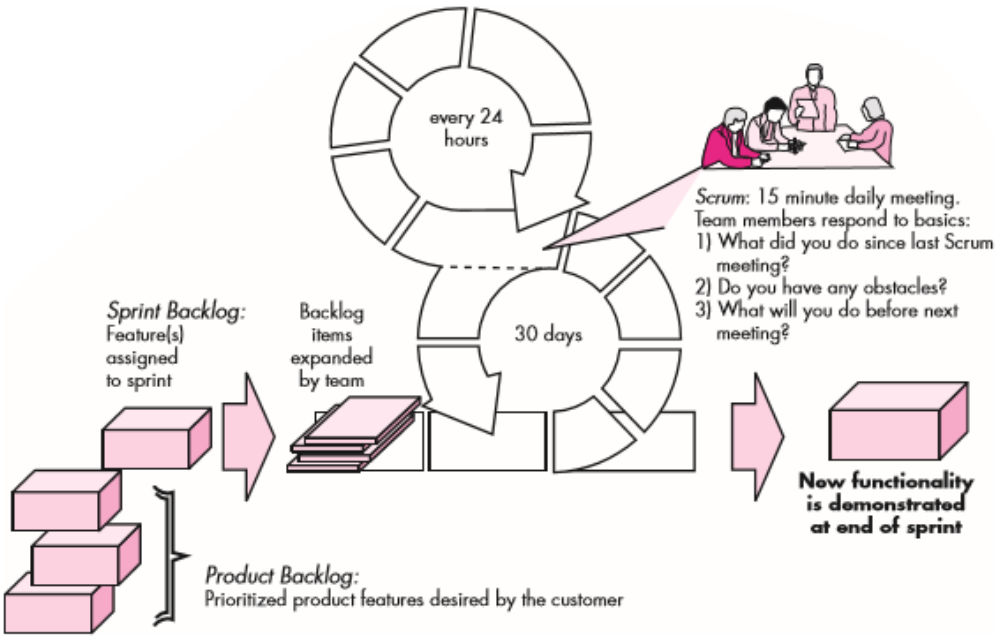
Semua metode yang dibandingkan oleh (Strode, 2005) objektif dalam memberikan solusi terhadap bisnis. Karena kelima metode tersebut menyediakan iterasi, inkremental, umpan balik, pembelajaran, keterlibatan pengguna, kerjasama tim, dan pemberdayaan tim dalam membuat suatu keputusan. Metode *Agile* dikerjakan oleh tim yang relatif kecil dan menekankan adanya komunikasi informal antar tim. Metode *agile* meminimalkan dokumentasi dan produksi sistematis. Karena metode *agile* lebih menekankan *coding* dan *testing* (Strode, 2005).

Karakteristik dari metode *agile* berbeda-beda. DSDM dan ASD adalah kerangka kerja yang mengadopsi teknik dari metodologi lainnya, XP mendefinisikan satu set eksplisit untuk beradaptasi dengan cepat terhadap perubahan lingkungan dan

berfokus pada komunikasi dengan pelanggan. *Crystal* adalah metodologi untuk memilih teknik yang efektif dalam setiap proyek pengembangan perangkat lunak, sedangkan scrum berfokus pada manajemen proyek. DSDM dan ASD menawarkan detail pada banyak teknik yang digunakan, XP mengedepankan teknik dalam mengembangkan perangkat lunak, sedangkan Scrum mengedepankan kejelasan dalam mengelola proyek. Hasil tersebut dikemukakan oleh (Strode, 2005) berdasarkan kerangka kerja analisis yang ia gunakan dalam membandingkan kelima metode *agile*.

2.1.3.5 Scrum

Scrum adalah salah satu *framework* dari metode *agile development process* yang telah dikembangkan oleh Jeff Sutherland dan tim pada tahun 1990. Kemudian pengembangan lebih lanjut dilakukan oleh Schwaber dan Beedle. Prinsip yang digunakan scrum dalam mengembangkan perangkat lunak menggabungkan kerangka kegiatan mulai dari *requirements, analysis, design, evolution, dan delivery* (Pressman, 2005). Gambar 2.4 Menunjukkan aliran keseluruhan dari proses scrum.



Gambar 2.4. Scrum Process (Pressman, 2005)

a. Product Backlog

Product backlog adalah sebuah daftar fitur atau *requirements* suatu proyek yang sudah diprioritaskan untuk memaksimalkan nilai bisnis terhadap pelanggan. Isi dari *backlog* dapat ditambahkan setiap waktu untuk kemudian dinilai oleh *product manager* dan memperbaharui prioritas sesuai kebutuhan (Pressman, 2005).

b. Sprint Backlog

Sprint backlog adalah daftar sederhana dari tugas-tugas yang harus dilaksanakan oleh tim dalam rangka untuk menyelesaikan fungsi suatu perangkat lunak secara inkremental. Pembuatan *sprint backlog* terjadi pada saat rapat perencanaan *sprint* dengan partisipasi dari setiap anggota tim (Pressman, 2005).

c. Sprint

Sprint adalah unit pekerjaan yang dibutuhkan untuk menyelesaikan *requirements* yang terdapat pada *sprint backlog* dan harus diselesaikan pada waktu yang ditentukan (biasanya 1-4 minggu atau 30 hari) (Pressman, 2005). Seberapa

banyaknya *sprint* tergantung dari seberapa besar dan kompleks produk perangkat lunak yang akan dibuat. Selama *sprint* berlangsung, *item* yang terdapat pada *sprint backlog* dibekukan atau tidak dapat diubah sampai *sprint* selesai. *Sprint* memungkinkan anggota tim untuk bekerja dalam jangka pendek, tetapi pada lingkungan yang stabil. *Scrum meetings (Daily Stand-ups)* merupakan pertemuan pendek (biasanya 15 menit) yang dilakukan oleh tim scrum dalam fase *sprint* (Kung, 2014). Scrum master atau ketua tim memimpin jalannya pertemuan tersebut dan menilai respon dari setiap orang. Scrum *meetings* menolong tim tersebut untuk menemukan masalah yang berpotensi terjadi sedini mungkin. Selain itu, pertemuan ini dapat menjadi ajang untuk berbagi pengetahuan dengan demikian sesuai dengan struktur *self-organizing team* (Pressman, 2005). Tiga pertanyaan kunci ditanyakan oleh Scrum master kepada para anggota tim. Tiga pertanyaan kunci tersebut adalah (Pressman, 2005):

1. Apa yang telah kamu lakukan kemarin?
2. Apa saja rintangan yang kamu hadapi saat ini?
3. Apa rencana yang akan kamu lakukan besok?

Selama masa *sprint* terdapat waktu kerja atau *work days* atau *man days*, dimana waktu kerja didapat dari hari kerja tim pengembang (senin sampai jumat). Waktu kerja tidak selalu seratus persen waktu untuk bekerja, karena pasti ada faktor luar yang menginterupsi komitmen dalam bekerja seperti mengerjakan pekerjaan lain diluar pekerjaan mengenai produk, sakit, komputer yang rusak, dan lain-lain (Kniberg, 2007). Waktu kerja ideal *(ideal man-days)* adalah waktu kerja ideal yang seharusnya tidak ada interupsi dan merupakan waktu yang efektif dalam melakukan pekerjaan. Untuk menghitung *ideal man days* dapat menggunakan rumus (Kniberg, 2007):

$$Ideal\ man\ days = Avaiable\ man\ days\ \times Focus\ factor \quad \text{.....(2.1)}$$

Menghitung waktu kerja ideal yaitu menggunakan kebijakan *focus factor*, dimana *focus factor* adalah estimasi dari seberapa fokus sebuah tim dalam mengerjakan pekerjaannya. *Focus factor* pada dasarnya adalah berapa banyak waktu yang dihabiskan tim pengembang untuk fokus kepada *task* atau pekerjaan yang sudah mereka komitmenkan untuk dikerjakan (Kniberg, 2007). *Focus factor* yang rendah dapat berarti tim tersebut mengharapkan untuk mengalami banyak gangguan atau mengharapkan perkiraan waktu mereka sendiri untuk menjadi optimis (Kniberg, 2007). Cara terbaik untuk menentukan *focus factor* yang wajar adalah dengan melihat kinerja dari *sprint* terakhir. Untuk menghitung *focus factor* dari *sprint* terakhir dapat digunakan rumus (Kniberg, 2007):

$$Focus\ Factor = \frac{Actual\ velocity}{Available\ man\ days} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

Actual velocity merupakan jumlah estimasi awal dari semua *backlog item* atau *task* (pekerjaan) yang sudah diselesaikan saat *sprint*. Apabila tim pengembang benar-benar baru dalam mengerjakan *sprint* sehingga tidak dapat dilihat kinerja sebelumnya, maka dapat dilihat *focus factor* tim pengembang lain dengan kondisi yang sama. Apabila tetap tidak menemukan tim pengembang lain tersebut, maka *focus factor* dapat diisi dengan *default value* yaitu 70% (0.7) ataupun dengan menebak karena tebakan hanya berlaku untuk *sprint* pertama sehingga akan memiliki statistik di *sprint* selanjutnya dan dapat terus menerus mengukur *focus factor* selanjutnya (Kniberg, 2007).

d. Demos

Produk didemonstrasikan kepada pelanggan secara incremental untuk kemudian dievaluasi setiap fungsi-fungsinya oleh pelanggan tersebut dan mengulangi proses dan membuat sprint baru jika pelanggan meminta perubahan fungsi atau menambah fungsi baru (Pressman, 2005). Terakhir, produk yang sudah sesuai diberikan kepada pelanggan dan proses pengembangan selesai.

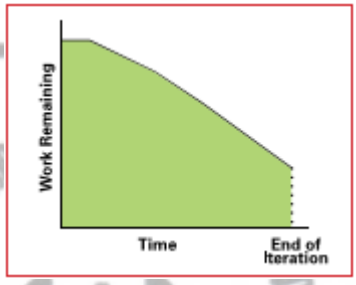
2.1.4 Burndown Chart

Burndown chart adalah metode yang sederhana untuk memantau proses pengerjaan suatu produk yang dikerjakan menggunakan metode *agile*. *Burndown chart* adalah cara terbaik untuk memvisualisasikan hubungan antara jumlah pekerjaan yang tersisa pada setiap titik waktu dan kemajuan Tim pengembang (Schwaber, 2004). *Burndown chart* dapat digunakan untuk membuat prediksi tentang tanggal dirilis nya suatu produk dan juga untuk manajemen proses dalam suatu proyek agar proyek dapat berjalan lebih baik (Dinwiddie, 2009). *Burndown chart* dapat diaplikasikan dalam berbagai jenis proyek yang dikembangkan secara iteratif.

Parameter yang diukur dalam *burndown chart* dapat berupa *work (task) remaining* atau *story point*. Pengukuran yang paling sering dilakukan yaitu dengan menggunakan parameter *work remaining* dimana unit yang digunakan untuk mengukurnya berdasarkan unit satuan waktu (jam, hari, atau minggu) (Dinwiddie, 2009).

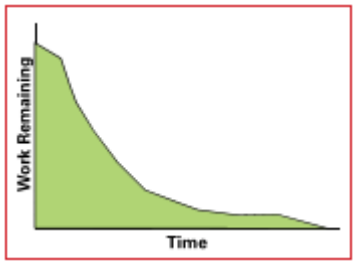
Burndown chart memiliki *ideal task remaining line* dan *actual task remaining line. Ideal task remaining line* adalah garis lurus yang menghubungkan titik awal ke titik akhir. Pada titik awal, garis yang ideal menunjukkan jumlah dari perkiraan untuk semua tugas (pekerjaan) yang perlu diselesaikan. Pada titik akhir, garis yang ideal penyadapan sumbu x menunjukkan bahwa tidak ada pekerjaan yang tersisa untuk diselesaikan. Beberapa orang tidak setuju dengan menyebut ini sebuah "ideal" *line*, karena umumnya tidak benar bahwa tujuannya adalah untuk mengikuti garis ini. Garis ini adalah perhitungan matematis berdasarkan perkiraan, dan perkiraan lebih cenderung berada dalam kesalahan dari pekerjaan (Dinwiddie, 2009). *Actual task remaining line* adalah pekerjaan yang sebenarnya tersisa. Pada titik awal, *actual task remaining line* adalah hamper sama dengan *ideal task remaining line* tapi seiring berjalannya waktu, *actual task remaining line* berfluktuasi di atas dan di bawah garis yang ideal tergantung pada seberapa efektif tim pengembang bekerja. Secara umum, titik baru ditambahkan ke baris ini setiap hari selama proyek berjalan. Setiap hari, jumlah dari estimasi perkiraan waktu untuk pekerjaan yang baru saja diselesaikan

dikurangi dari titik terakhir di baris untuk menentukan titik berikutnya (Dinwiddie, 2009). Jika garis aktual terus berada di atas garis ideal, maka terdapat lebih banyak pekerjaan yang tersisa dari yang diperkirakan dan proyek akan mengalami keterlambatan. Jika garis aktual terus berada di bawah garis ideal, maka terdapat sedikit pekerjaan yang tersisa dan proyek akan diperkirakan selesai lebih cepat dari jadwal (Dinwiddie, 2009). Gambar 2.5 sampai Gambar 2.9 menunjukkan contoh dari *burndown chart tracking progress* pada pengembangan perangkat lunak secara iteratif. Berikut adalah contoh *burndown chart 1* yang ditunjukkan pada Gambar 2.5.



Gambar 2.5 Contoh *Burndown Chart* 1 (Dinwiddie, 2009)

Gambar 2.5 menunjukan bahwa *burndown chart* tidak mencapai garis tujuan. Pada akhir dari iterasi terdapat pekerjaan yang tersisa tidak terselesaikan. Kondisi seperti ini mungkin jarang terjadi, namun jika sering terjadi maka dapat disimpulkan ini adalah tanda bahwa tim pengembang terlalu membuat janji yang berlebihan akan pekerjaan yang dapat mereka selesaikan. Hal ini karena tim pengembang yang terlalu optimis tentang kemampuan mereka. Mereka tahu bagaimana melakukan hal-hal yang mereka lihat perlu dilakukan. Seringkali mereka tidak ingat atau mempertimbangkan waktu mereka terjebak, baik pada masalah yang sulit atau menunggu ketergantungan eksternal. Mereka mungkin mengabaikan waktu yang dihabiskan melakukan hal-hal lain selain menciptakan kode program (Dinwiddie, 2009). Selanjutnya adalah contoh *burndown chart 2* yang ditunjukkan pada Gambar 2.6.



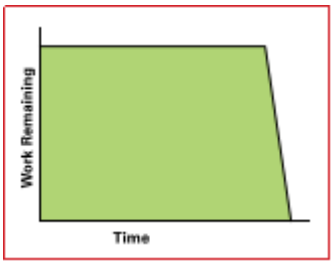
Gambar 2.6 Contoh *Burndown Chart 2* (Dinwiddie, 2009)

Gambar 2.6 menunjukkan kemajuan yang mantap dan kemudian mengulur-ulur waktu ketika mencapai tujuan yang tampaknya meyakinkan. Pada grafik ini, ada beberapa kemungkinan, yaitu kemungkinan pertama tim pengembang tidak ingin melewatkan komitmen mereka dan mengecewakan pelanggan, mereka mempertahankan komitmen mereka. Kemungkinan kedua yaitu *task* selanjutnya yang diestimasikan rendah, relatif terhadap *task* yang sebenarnya terjadi yaitu membutuhkan waktu yang lama untuk mengerjakannya. Kemungkinan ketiga yaitu hutang pekerjaan sebelumnya menghambat tim pengembang dalam mengerjakan pekerjaan selanjutnya (Dinwiddie, 2009). Gambar selanjutnya adalah contoh *burndown chart 3* yang ditunjukkan pada Gambar 2.7.

Gambar 2.7 Contoh *Burndown Chart 3* (Dinwiddie, 2009)

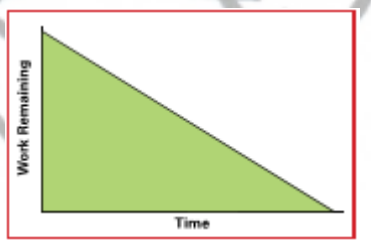
Gambar 2.7 menunjukkan bahwa tidak terdapat indikasi perubahan pada proses dalam waktu yang cukup lama, ini dikarenakan adanya *user story* yang besar atau membutuhkan waktu yang lama untuk mengerjakannya. Melakukan pekerjaan pada *user story* yang besar membuat perkembangan proses suatu proyek sulit untuk

dilihat (Dinwiddie, 2009). Gambar selanjutnya adalah contoh *burndown chart 4* yang ditunjukkan pada Gambar 2.8.



Gambar 2.8 Contoh *Burndown Chart* 4 (Dinwiddie, 2009)

Gambar 2.8 menunjukkan terdapatnya *user story* yang sangat besar sehingga memakan waktu yang sangat lama atau kemungkinan lainnya yaitu pihak *developer* mengerjakan pekerjaan lainnya daripada mengerjakan pekerjaan yang harusnya dikerjakan. Problem yang didapati jika grafik seperti ini muncul yaitu tidak adanya keyakinan oleh tim pengembang pada masa *sprint* tersebut untuk menyelesaikan *user story* karena waktu yang sudah terlampau mendekati akhir dari iterasi (Dinwiddie, 2009). Gambar terakhir adalah contoh *burndown chart 5* yang ditunjukkan pada Gambar 2.9.



Gambar 2.9 Contoh *Burndown Chart* 5 (Dinwiddie, 2009)

Gambar 2.9 merupakan gambar *burndown chart* yang sempurna dimana proyek berjalan sesuai dengan waktu yang diestimasikan. Hal ini sangat jarang terjadi dan seharusnya tidak mungkin terjadi karena waktu yang diperkirakan atau diestimasikan tidak selalu tepat pada aktualnya. Hal seperti ini dapat terjadi dengan berbagai kemungkinan, pertama yaitu karena tim pengembang mengerjakan dengan

komitmen tinggi sesuai perancangan estimasi waktu dan tidak terhambat oleh halangan apapun, kedua jika terdapat seseorang yang memanipulasi sendiri data yang digunakan agar proses pengembangan terlihat baik dan tidak bermasalah (Dinwiddie, 2009).

2.2 Penelitian Terkait

Berikut adalah beberapa penelitian yang pernah dilakukan mengenai keterkaitannya terhadap Sistem Informasi Manajemen Skripsi/Tugas Akhir.

1. ***Waterfall Vs V-Model Vs Agile: A Comparative Study On SDLC.*** **Ditulis tahun 2012 oleh Balaji & Murugaiyan**

   Penelitian yang dibuat oleh (Balaji & Murugaiyan, 2012) memberikan solusi kepada organisasi yang mengalami kesulitan dalam memilih metode pengembangan perangkat lunak yang tepat. Penelitian ini melakukan komparasi terhadap tiga model pengembangan perangkat lunak, yaitu *waterfall model, v-model, agile model* berdasarkan *client requirement* untuk mengembangkan aplikasi berbasis *website*. *Website* yang akan dibuat adalah sistem pemesanan tiket kereta api, bis, dan pesawat dalam satu sistem. Pengguna dapat memesan untuk enam bulan kedepan. Selanjutnya pengguna dapat melakukan pembayaran dan tiket secara otomatis telah terpesan. Hasil yang didapat dari penelitian ini mengungkapkan bahwa setiap metode mempunyai kelebihan dan kekurangan. Bagaimana menentukan yang terbaik tergantung dari beberapa aspek yaitu jika perubahan *requirement* begitu sering dan proyek lebih kecil serta memiliki sumber daya yang mumpuni atau waktu yang singkat maka dipilihlah metode *agile.* Namun, jika *requirement* yang diberikan di awal jelas dan dalam skala proyek yang besar, maka dipilihlah metode *waterfall.* Selain itu, dipilih metode *v-model* jika *requirements* jarang berubah, skala proyek besar, validasi dalam setiap tahap, dan tester terlibat dalam tahap awal pengembangan.

2. ***Accelerating Software Development through Agile Practices - A Case Study of a Small-Scale, Time-intensive Web Development Project at a College-level IT Competition.*** **Ditulis tahun 2012 oleh Zhang & Dorn.**

Agile Software Development banyak diterapkan dalam industry maupun akademi dikarenakan keuntungan dalam mengembangkan perangkat lunak lebih cepat, lebih murah, sesuai dengan kebutuhan pelanggan, dan terus dapat berjalan walaupun *requirements* sering berubah-ubah. Penelitian yang dilakukan oleh (Zhang & Dorn, 2012) menguji nilai *agile practices* dalam sebuah grup skala kecil yang membuat suatu proyek berbasis *web* serta manfaat dan tantangannya. Hasil yang didapat setelah diuji coba berdasarkan observasi pada proses pengembangan, *interview* anggota tim, dan pembelajaran dari beberapa dokumen terkait yaitu bagaimana *agile practices* seperti *daily scrums, backlogs,* dan *sprints* telah berhasil diaplikasikan pada pengembangan proyek tersebut. Selain itu, tantangan dan manfaat yang didapat dalam pengembangan *web* dengan menggunakan metode ini yaitu metode *agile* sesuai untuk tim yang dibentuk dengan sukarela, *self-organized,* dan memiliki fungsi yang berbeda-beda. Metode *agile* juga sesuai bila diterapkan pada proyek skala kecil dan memilii waktu yang intensif.

3. **Pengembangan Sistem Informasi Monitoring Tugas Akhir Ber basis *Short Message Service* (SMS) *Gateway* di Fasilkom Unsri. Ditulis tahun 2011 oleh Ibrahim.**

Kegiatan administrasi di Fasilkom Unsri memiliki beberapa kendala serta kurang efisien dan tidak efektif. Hal tersebut terlihat seperti Sistem Informasi masih belum dapat melacak posisi proposal saat proses pengajuan hingga selesai, sehingga menyebabkan mahasiswa harus pergi ke kampus untuk menanyakan status proposalnya. Selain itu, administrasi juga harus berkoordinasi terlebih dahulu dengan Ketua Jurusan jika ada mahasiswa yang menanyakan status proposalnya. Akibatnya kinerja administrasi terlihat kurang efektif. Pemanfaatan teknologi informasi dan komunikasi berupa telepon seluler dalam mengolah, memproduksi, serta

mengirimkan ataupun menerima segala bentuk pesan komunikasi untuk memberikan informasi mengenai Tugas Akhir mahasiswa dilakukan oleh (Ibrahim, 2011)**.** Pengembangan perangkat lunak yang dilakukan menggunakan metode *incremental model.* Metode ini dapat melakukan pengerjaan tahapan proses secara parallel. Perangkat lunak yang dihasilkannya mengimplementasikan teknologi SMS *gateway* untuk memonitoring tugas akhir mahasiswa Fasilkom Unsri. Pemanfaatan telepon seluler dalam mengelola informasi membuat proses transfer informasi menjadi lebih cepat, akurat, efisien dan efektif. Produk yang dibuatnya mampu melakukan proses transfer informasi dalam bentuk SMS yang cepat, efisien, efektif, interaktif dan akurat tentang status proposal, jadwal seminar, dan jadwal sidang tugas akhir. Selain itu, dengan menggunakan metode incremental, pengerjaan perangkat lunak menjadi lebih efisien karena tahap demi tahap dilakukan secara parallel.

4. ***Process Development and Integration through Incremental Process for Web Applications.*** **Ditulis tahun 2013 oleh Rameshwariah.**

Teknologi web sangat berpengaruh terhadap proses informasi yang terjadi di jaringan internet. Web dalam pengembangannya sangat kompleks selama proses evolusi dan harus memenuhi semua *requirements* yang dibutuhkan seperti yang dikemukakan oleh (Rameshwariah, 2013). Kompleksitas yang terdapat pada proses pengembangan dapat dapat diatasi dengan mempertimbangkan beberapa langkah-langkah selama fase perkembangannya. Pengembangan yang dilakukan oleh Rameshwariah menggunakan metode *incremental model* dalam mengembangkan aplikasi berbasis web. Hasil yang didapat dengan menggunakan metode *incremental model* pada saat pengembangan aplikasi web adalah kemudahan dalam uji coba dan perbaikan aplikasi apabila dibandingkan dengan metode lainnya. Hal ini dikarenakan perubahan aplikasi yang sedikit selama setiap iterasi yang ada. Sehingga dengan digunakannya *incremental model*, pengembangan aplikasi *web* menjadi mudah dan efisien.

5. ***Implementation of Scrum Framework of Agile Methodology for an Online Project.*** **Ditulis tahun 2014 oleh Mohamed.**

Penelitian yang dilakukan oleh (Mohamed, 2014) mengungkapkan tentang bagaimana mengimplementasikan metode scrum terhadap proyek *website* melalui skenario yang dibuatnya. Kemudian juga dapat diketahui kenapa menggunakan scrum dan kapan untuk tidak menggunakan scrum. Melalui skenario yang diberikan terdapat masalah dimana *website* suatu perusahaan yang rusak dan semua detail dari *website* tersebut terhapus dari server. Kemudian pihak manajer menghimbau kepada tim pengembang yang disebut "*scrum team"* untuk membuat *website* baru. *Scrum team* membangun sistem dengan menggunakan scrum *framework.* Hasil yang didapat setelah proyek selesai adalah *framework* scrum menawarkan tingkat fleksibilitas yang tinggi untuk keberhasilan proyek. Komunikasi sangat penting dalam anggota tim untuk mengambangkan *website.* Scrum menyediakan kualitas yang baik dalam suatu proyek, dapat diaplikasikan dalam *requirements* yang sering berubah, memberikan kemudahan pada scrum master untuk mengestimasi pekerjaan dari tim proyek, dan mendukung proyek untuk selesai pada jadwal yang ditetapkan.

6. ***An Analysis on Scrum Methodology Used For the IT Project for Effective Software Deliverable.*** **Ditulis tahun 2014 oleh Mahalakshmi & Sundararajan.**

Penelitian yang dilakukan oleh (Mahalakshmi & Sundararajan, 2014) melakukan analisis pada metode scrum yang digunakan pada proyek IT untuk efektifitas pembuatan perangkat lunak juga komparasi terhadap metode *agile* lainnya. Hasil yang didapat menjelaskan bahwa scrum jika dibandingkan dengan metode *agile* lainnya menyediakan kualitas yang terbaik dan juga dari segi produktifitas. Scrum dapat diterapkan dalam segala tipe ukuran proyek dan mudah dalam mengestimasi pekerjaan dan waktu. Di dunia yang bergerak cepat dan tidak stabil, perubahan perangkat lunak yang terjadi seiring proses pengembangan harus dapat diterima. Banyak perusahaan perangkat lunak mulai menerapkan metode Scrum dalam proyek mereka ketika dibandingkan dengan metodologi lainnya untuk mengatasi perubahan

tersebut. Scrum dapat digunakan dalam proyek IT maupun proyek non-IT dengan cara yang paling efektif.

Tabel 2.1 Keterkaitan Penelitian Sebelumnya dengan Penelitian yang Dilakukan

No	Penelitian Terkait	Judul	Tujuan	Metode
1	Balaji & Murugaiyan (2012)	*Waterfall Vs V-Model Vs Agile: A Comparative Study On SDLC*	Memberikan studi banding terhadap model waterfall, v-model, dan agile model dalam membangun aplikasi berbasis website.	Waterfall, V-model, Agile
2	Zhang & Dorn (2012)	*Accelerating Software Development through Agile Practices - A Case Study of a Small-Scale, Time-intensive Web Development Project at a College-level IT Competition*	Menguji nilai *agile practices* dalam sebuah grup skala kecil yang membuat suatu proyek berbasis *web* serta manfaat dan tantangannya	Scrum
3	Ibrahim (2011)	Pengembangan Sistem Informasi Monitoring Tugas Akhir Ber basis *Short Message Service* (SMS) *Gateway* di Fasilkom Unsri	Mengimplementasikan teknologi SMS gateway untuk manajemen Tugas Akhir di Fasilkom Unsri	Incremental

Tabel 2.1 Lanjutan

No	Penelitian Terkait	Judul	Tujuan	Metode
4	Rameshwariah (2013)	*Process Development and Integration through Incremental Process for Web Applications*	Mengimplementasikan metode incremental dalam pengembangan aplikasi web	Incremental
5	Mohamed (2014)	*Implementation of Scrum Framework of Agile Methodology for an Online Project*	Mengimplementasikan metode scrum dalam pengembangan aplikasi proyek online berbasis web	Scrum
6	Mahalakshmi & Sundararajan (2014)	*An Analysis on Scrum Methodology Used For the IT Project for Effective Software Deliverable*	Melakukan analisis pada metode scrum pada proyek IT dan komparasi terhadap metode agile lainnya	Scrum, Agile method
7	Primadharma (2015)	Pengembangan Aplikasi Skripsi (Tugas Akhir) Berbasis Web Menggunakan Metode Scrum	Membangun Aplikasi Skripsi/Tugas Akhir di Jurusan Informatika UNS Berbasis Web dengan menggunakan metode pengembangan perangkat lunak Scrum	Scrum