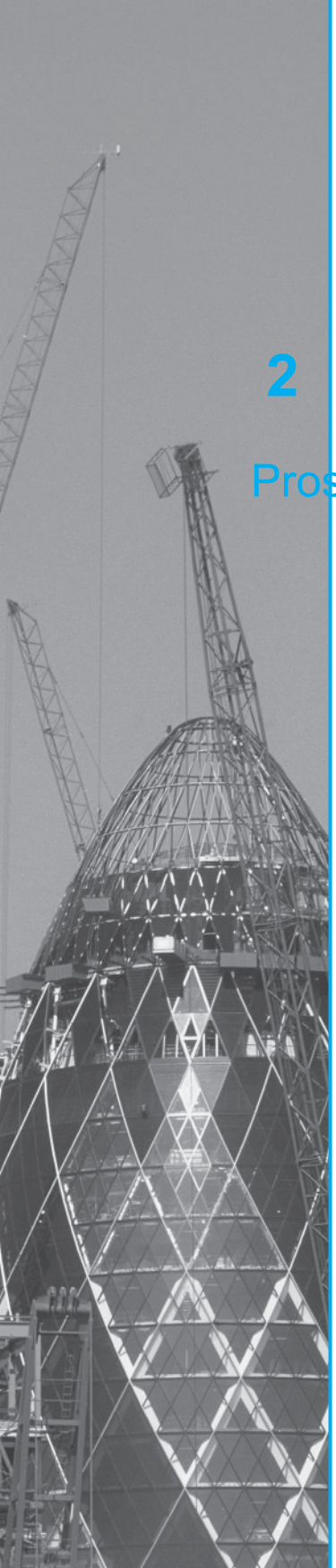


2

Proses perangkat lunak

Tujuan

Tujuan dari bab ini adalah untuk memperkenalkan Anda pada gagasan tentang proses perangkat lunak — serangkaian aktivitas yang koheren untuk produksi perangkat lunak. Setelah Anda membaca bab ini, Anda akan:

- memahami konsep proses perangkat lunak dan model proses perangkat lunak;
- telah diperkenalkan pada tiga model proses perangkat lunak umum dan kapan model tersebut dapat digunakan;
- mengetahui tentang aktivitas proses dasar rekayasa persyaratan perangkat lunak, pengembangan perangkat lunak, pengujian, dan evolusi;
- memahami mengapa proses harus diatur untuk mengatasi perubahan dalam persyaratan dan desain perangkat lunak;
- memahami pengertian perbaikan proses perangkat lunak dan faktor-faktor yang mempengaruhi kualitas proses perangkat lunak.

Isi

Proses perangkat lunak adalah sekumpulan aktivitas terkait yang mengarah pada produksi sistem perangkat lunak. Seperti yang telah saya bahas di Bab 1, ada banyak jenis sistem perangkat lunak, dan tidak ada metode rekayasa perangkat lunak universal yang dapat diterapkan pada semuanya. Akibatnya, tidak ada proses perangkat lunak yang dapat diterapkan secara universal. Proses yang digunakan di berbagai perusahaan bergantung pada jenis perangkat lunak yang dikembangkan, persyaratan pelanggan perangkat lunak, dan keterampilan orang yang menulis perangkat lunak tersebut.

Namun, meskipun ada banyak proses perangkat lunak yang berbeda, semuanya harus menyertakan, dalam beberapa bentuk, empat aktivitas rekayasa perangkat lunak dasar yang saya perkenalkan di Bab 1:

1. *Spesifikasi perangkat lunak* Fungsionalitas perangkat lunak dan batasan pada operasinya harus ditentukan.
2. *Pengembangan perangkat lunak* Perangkat lunak untuk memenuhi spesifikasi harus diproduksi.
3. *Validasi perangkat lunak* Perangkat lunak harus divalidasi untuk memastikan bahwa ia melakukan apa yang diinginkan pelanggan.
4. *Evolusi perangkat lunak* Perangkat lunak harus berkembang untuk memenuhi kebutuhan pelanggan yang terus berubah.

Aktivitas ini sendiri merupakan aktivitas kompleks, dan mencakup subaktivitas seperti validasi persyaratan, desain arsitektur, dan pengujian unit. Proses juga mencakup aktivitas lain, seperti manajemen konfigurasi perangkat lunak dan perencanaan proyek yang mendukung aktivitas produksi.

Ketika kita mendeskripsikan dan mendiskusikan proses, kita biasanya berbicara tentang aktivitas dalam proses ini, seperti menentukan model data dan merancang antarmuka pengguna, dan urutan aktivitas ini. Kita semua dapat memahami apa yang dilakukan orang untuk mengembangkan perangkat lunak. Namun, saat mendeskripsikan proses, penting juga untuk mendeskripsikan siapa yang terlibat, apa yang diproduksi, dan kondisi yang memengaruhi urutan aktivitas:

1. Produk atau kiriman adalah hasil dari suatu aktivitas proses. Misalnya, hasil dari aktivitas desain arsitektur dapat berupa model arsitektur perangkat lunak.
2. Peran mencerminkan tanggung jawab orang-orang yang terlibat dalam proses tersebut. Contoh peran adalah manajer proyek, manajer konfigurasi, dan programmer.
3. Kondisi sebelum dan sesudah adalah kondisi yang harus ada sebelum dan sesudah aktivitas proses diberlakukan atau produk diproduksi. Misalnya, sebelum desain arsitektural dimulai, prasyaratnya adalah konsumen telah menyetujui semua persyaratan; setelah aktivitas ini selesai, kondisi akhir mungkin adalah model UML yang mendeskripsikan arsitektur telah ditinjau.

Proses perangkat lunak itu kompleks dan, seperti semua proses intelektual dan kreatif, bergantung pada orang yang membuat keputusan dan penilaian. Karena tidak ada proses universal yang tepat untuk semua jenis perangkat lunak, sebagian

proses pengembangan. Proses telah berkembang untuk memanfaatkan kemampuan para pengembang perangkat lunak dalam suatu organisasi dan karakteristik sistem yang sedang dikembangkan. Untuk sistem yang kritis terhadap keselamatan, proses pengembangan yang sangat terstruktur diperlukan di mana catatan terperinci disimpan. Untuk sistem bisnis, dengan persyaratan yang berubah dengan cepat, proses yang lebih fleksibel dan gesit kemungkinan besar akan lebih baik.

Seperti yang saya bahas di Bab 1, pengembangan perangkat lunak profesional Profesional adalah aktivitas yang dikelola, jadi perencanaan adalah bagian yang melekat dari semua proses. Proses yang digerakkan oleh rencana adalah proses di mana semua aktivitas proses direncanakan sebelumnya dan kemajuan diukur terhadap rencana ini. Dalam proses tangkas, yang saya bahas di Bab 3, perencanaan bersifat inkremental dan berkelanjutan saat perangkat lunak dikembangkan. Oleh karena itu, lebih mudah untuk mengubah proses untuk mencerminkan persyaratan pelanggan atau produk yang berubah. Seperti yang dijelaskan oleh Boehm dan Turner (Boehm dan Turner 2004), setiap pendekatan cocok untuk jenis perangkat lunak yang berbeda. Umumnya, untuk sistem besar, Anda perlu menemukan keseimbangan antara proses yang digerakkan oleh rencana dan proses yang gesit.

Meskipun tidak ada proses perangkat lunak universal, ada ruang untuk perbaikan proses di banyak organisasi. Proses mungkin termasuk teknik yang sudah ketinggalan zaman atau mungkin tidak memanfaatkan praktik terbaik dalam rekayasa perangkat lunak industri. Memang, banyak organisasi masih belum memanfaatkan metode rekayasa perangkat lunak dalam pengembangan perangkat lunak mereka. Mereka dapat meningkatkan proses mereka dengan memperkenalkan teknik seperti pemodelan UML dan pengembangan berbasis pengujian. Saya membahas perbaikan proses perangkat lunak secara singkat nanti di teks bab ini dan lebih detail di web Bab 26.

Model proses perangkat lunak

Seperti yang saya jelaskan di Bab 1, model proses perangkat lunak (kadang-kadang disebut Siklus Hidup Pengembangan Perangkat Lunak atau model SDLC) adalah representasi yang disederhanakan dari proses perangkat lunak. Setiap model proses merepresentasikan proses dari perspektif tertentu dan dengan demikian hanya memberikan sebagian informasi tentang proses itu. Misalnya, model aktivitas proses menunjukkan aktivitas dan urutannya tetapi mungkin tidak menunjukkan peran orang-orang yang terlibat dalam aktivitas ini. Pada bagian ini, saya memperkenalkan sejumlah model proses yang sangat umum (terkadang disebut *proses paradigma)* dan menyajikannya dari perspektif arsitektur. Artinya, kami melihat kerangka proses tetapi bukan detail aktivitas proses.

Model generik ini adalah deskripsi abstrak tingkat tinggi dari proses perangkat lunak yang dapat digunakan untuk menjelaskan pendekatan berbeda untuk pengembangan perangkat lunak. Anda dapat menganggapnya sebagai kerangka proses yang dapat diperluas dan disesuaikan untuk membuat proses rekayasa perangkat lunak yang lebih spesifik.

Model proses umum yang saya bahas di sini adalah:

1. *Model air terjun* Ini membutuhkan proses dasar kegiatan khusustion, pengembangan, validasi, dan evolusi dan merepresentasikannya sebagai fase proses terpisah seperti spesifikasi persyaratan, desain perangkat lunak, implementasi, dan pengujian.

Proses Terpadu Rasional

Rational Unified Process (RUP) menyatukan elemen dari semua model proses umum yang dibahas di sini dan mendukung pembuatan prototipe dan pengiriman perangkat lunak secara bertahap (Krutchen 2003). RUP biasanya digambarkan dari tiga perspektif: perspektif dinamis yang menunjukkan fase model dalam waktu, perspektif statis yang menunjukkan aktivitas proses, dan perspektif praktik yang menyarankan praktik yang baik untuk digunakan dalam proses. Tahapan RUP dimulai, di mana kasus bisnis untuk sistem ditetapkan; elaborasi, di mana persyaratan dan arsitektur dikembangkan; konstruksi tempat perangkat lunak diimplementasikan; dan transisi, tempat sistem diterapkan.

http://software-engineering-book.com/web/rup/

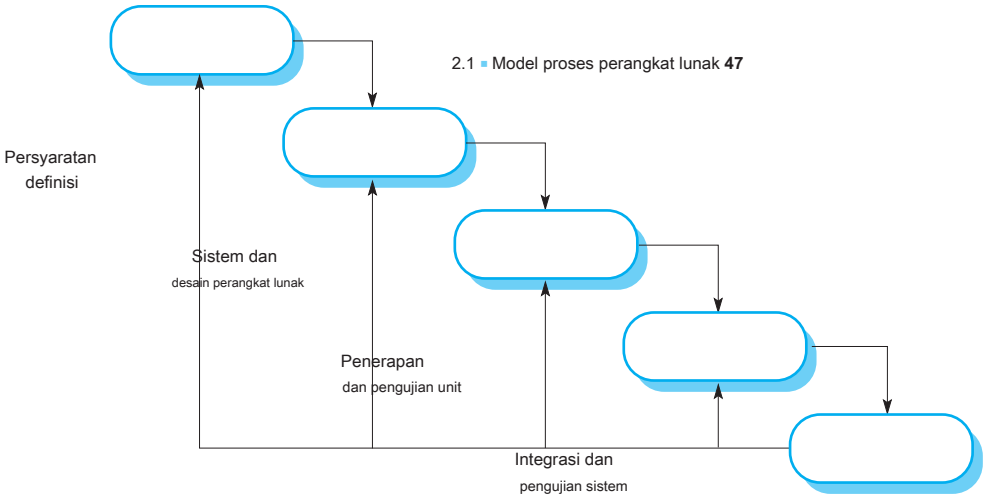
2. *Perkembangan bertahap* Pendekatan ini menyisipkan aktivitas spesifikasi, pengembangan, dan validasi. Sistem ini dikembangkan sebagai serangkaian versi (penambahan), dengan setiap versi menambahkan fungsionalitas ke versi sebelumnya.

3. *Integrasi dan konfigurasi* Pendekatan ini bergantung pada ketersediaan komponen atau sistem yang dapat digunakan kembali. Proses pengembangan sistem berfokus pada konfigurasi komponen ini untuk digunakan dalam pengaturan baru dan mengintegrasikannya ke dalam sistem.

Seperti yang telah saya katakan, tidak ada model proses universal yang tepat untuk semua jenis pengembangan perangkat lunak. Proses yang benar bergantung pada pelanggan dan persyaratan peraturan, lingkungan tempat perangkat lunak akan digunakan, dan jenis perangkat lunak yang dikembangkan. Misalnya, perangkat lunak keamanan kritis biasanya dikembangkan dengan menggunakan proses waterfall karena banyak analisis dan dokumentasi diperlukan sebelum implementasi dimulai. Produk perangkat lunak sekarang selalu dikembangkan menggunakan model proses inkremental. Sistem bisnis semakin dikembangkan dengan mengkonfigurasi sistem yang ada dan mengintegrasikannya untuk membuat sistem baru dengan fungsionalitas yang diperlukan.

Mayoritas proses perangkat lunak praktis didasarkan pada model umum tetapi sering kali menggabungkan fitur model lain. Hal ini terutama berlaku untuk rekayasa sistem besar. Untuk sistem yang besar, masuk akal untuk menggabungkan beberapa fitur terbaik dari semua proses umum. Anda perlu memiliki informasi tentang persyaratan sistem yang penting untuk merancang arsitektur perangkat lunak untuk mendukung persyaratan ini. Anda tidak dapat mengembangkan ini secara bertahap. Subsistem dalam sistem yang lebih besar dapat dikembangkan dengan menggunakan pendekatan yang berbeda. Bagian dari sistem yang dipahami dengan baik dapat ditentukan dan dikembangkan menggunakan proses berbasis air terjun atau dapat dibeli sebagai sistem siap pakai untuk konfigurasi. Bagian lain dari sistem, yang sulit ditentukan sebelumnya, harus selalu dikembangkan dengan menggunakan pendekatan inkremental. Dalam kedua kasus tersebut,

Berbagai upaya telah dilakukan untuk mengembangkan model proses "universal" yang mengacu pada semua model umum ini. Salah satu model universal yang paling terkenal adalah Rational Unified Process (RUP) (Krutchen 2003), yang dikembangkan oleh Rational, sebuah perusahaan rekayasa perangkat lunak AS.



Gambar 2.1 Itu model air terjun

dapat dipakai dalam berbagai cara untuk membuat proses yang menyerupai salah satu model proses umum yang dibahas di sini. RUP telah diadopsi oleh beberapa perusahaan perangkat lunak besar (terutama IBM), tetapi belum diterima secara luas.

2.1.1 Model air terjun

Model pertama yang diterbitkan dari proses pengembangan perangkat lunak berasal dari model proses rekayasa yang digunakan dalam rekayasa sistem militer besar (Royce 1970). Ini menyajikan proses pengembangan perangkat lunak sebagai sejumlah tahapan, seperti yang ditunjukkan pada Gambar 2.1. Karena kaskade dari satu fase ke fase lainnya, model ini dikenal sebagai model air terjun atau siklus hidup perangkat lunak. Model air terjun adalah contoh proses berbasis rencana. Pada prinsipnya setidaknya, Anda merencanakan dan menjadwalkan semua aktivitas proses sebelum memulai pengembangan perangkat lunak.

Tahapan model air terjun secara langsung mencerminkan kegiatan pengembangan perangkat lunak dasar:

1. *Analisis dan definisi kebutuhan* Layanan sistem, batasan, dan tujuan ditetapkan melalui konsultasi dengan pengguna sistem. Mereka kemudian didefinisikan secara rinci dan berfungsi sebagai spesifikasi sistem.
2. *Desain sistem dan perangkat lunak* Proses desain sistem mengalokasikan persyaratan ke sistem perangkat keras atau perangkat lunak. Ini menetapkan arsitektur sistem secara keseluruhan. Desain perangkat lunak melibatkan mengidentifikasi dan menjelaskan abstraksi sistem perangkat lunak dasar dan hubungannya.
3. *Implementasi dan pengujian unit* Selama tahap ini, desain perangkat lunak diwujudkan sebagai sekumpulan program atau unit program. Pengujian unit melibatkan verifikasi bahwa setiap unit memenuhi spesifikasinya.

Model proses spiral Boehm

Barry Boehm, salah satu pelopor dalam rekayasa perangkat lunak, mengusulkan model proses tambahan yang didorong oleh risiko. Proses ini direpresentasikan sebagai spiral daripada urutan aktivitas (Boehm 1988).

Setiap loop dalam spiral mewakili fase proses perangkat lunak. Jadi, loop paling dalam mungkin berkaitan dengan kelayakan sistem, loop berikutnya dengan definisi persyaratan, loop berikutnya dengan desain sistem, dan seterusnya. Model spiral menggabungkan penghindaran perubahan dengan toleransi perubahan. Ini mengasumsikan bahwa perubahan adalah hasil dari risiko proyek dan mencakup aktivitas manajemen risiko eksplisit untuk mengurangi risiko ini.

http://software-engineering-book.com/web/spiral-model/

4. *Integrasi dan pengujian sistem* Unit program atau program individu diintegrasikan dan diuji sebagai sistem yang lengkap untuk memastikan bahwa persyaratan perangkat lunak telah terpenuhi. Setelah pengujian, sistem perangkat lunak dikirimkan ke pelanggan.

5. *Operasi dan pemeliharaan* Biasanya, ini adalah fase siklus hidup terpanjang. Sistem dipasang dan mulai digunakan secara praktis. Pemeliharaan melibatkan koreksi kesalahan yang tidak ditemukan pada tahap awal siklus hidup, meningkatkan implementasi unit sistem, dan meningkatkan layanan sistem saat persyaratan baru ditemukan.

Pada prinsipnya, hasil dari setiap tahapan dalam model waterfall adalah satu atau lebih dokumen yang disetujui ("ditandatangani"). Fase berikut tidak boleh dimulai hingga fase sebelumnya selesai. Untuk pengembangan perangkat keras, yang melibatkan biaya produksi yang tinggi, ini masuk akal. Namun, untuk pengembangan perangkat lunak, tahapan ini tumpang tindih dan memberikan informasi satu sama lain. Selama desain, masalah dengan persyaratan diidentifikasi; selama masalah desain pengkodean ditemukan, dan sebagainya. Proses perangkat lunak, dalam praktiknya, tidak pernah menjadi model linier sederhana tetapi melibatkan umpan balik dari satu fase ke fase lainnya.

Saat informasi baru muncul dalam tahap proses, dokumen yang dihasilkan pada tahap sebelumnya harus dimodifikasi untuk mencerminkan perubahan sistem yang diperlukan. Misalnya, jika ditemukan bahwa persyaratan terlalu mahal untuk diterapkan, dokumen persyaratan harus diubah untuk menghapus persyaratan tersebut. Namun, ini membutuhkan persetujuan pelanggan dan menunda proses pengembangan secara keseluruhan.

Akibatnya, baik pelanggan dan pengembang mungkin membekukan spesifikasi perangkat lunak sebelum waktunya sehingga tidak ada perubahan lebih lanjut yang dilakukan terhadapnya. Sayangnya, ini berarti bahwa masalah dibiarkan untuk diselesaikan nanti, diabaikan, atau diprogram. Pembekuan persyaratan yang terlalu dini dapat berarti bahwa sistem tidak akan melakukan apa yang diinginkan pengguna. Ini juga dapat menyebabkan sistem terstruktur dengan buruk karena masalah desain dielakkan oleh trik implementasi.

Selama fase siklus hidup akhir (operasi dan pemeliharaan) perangkat lunak mulai digunakan. Kesalahan

Kesalahan program dan desain muncul, dan kebutuhan akan fungsionalitas baru diidentifikasi. Oleh karena itu, sistem harus berkembang agar tetap berguna. Membuat perubahan ini (pemeliharaan perangkat lunak) mungkin melibatkan pengulangan tahapan proses sebelumnya.

Pada kenyataannya, perangkat lunak harus fleksibel dan mengakomodasi perubahan saat dikembangkan. Perlunya komitmen awal dan pengerjaan ulang sistem saat perubahan dilakukan berarti model air terjun hanya sesuai untuk beberapa jenis sistem:

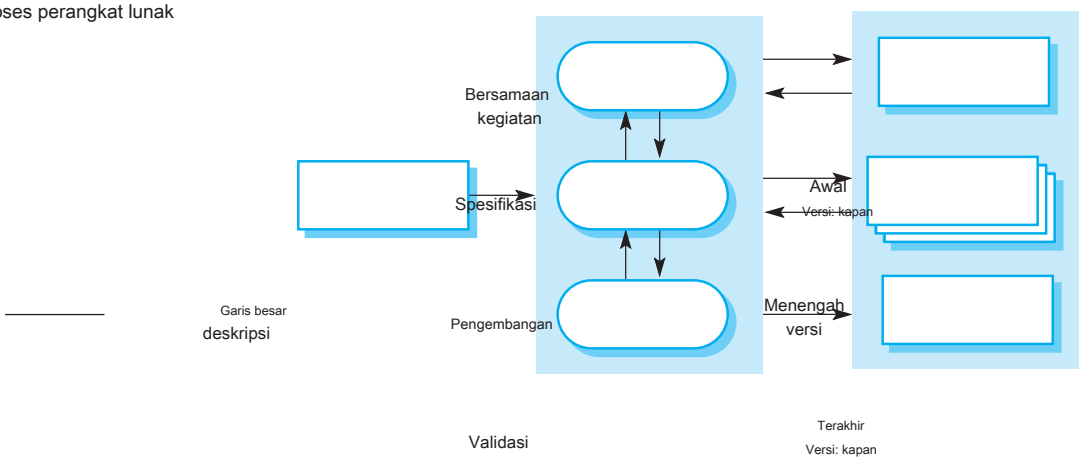
1. Sistem tertanam di mana perangkat lunak harus berinteraksi dengan sistem perangkat keras. Karena perangkat keras tidak fleksibel, biasanya tidak mungkin untuk menunda keputusan tentang fungsionalitas perangkat lunak sampai perangkat lunak itu diimplementasikan.
2. Sistem kritis yang memerlukan analisis keselamatan dan keamanan ekstensif dari spesifikasi dan desain perangkat lunak. Dalam sistem ini, spesifikasi dan dokumen desain harus lengkap agar dapat dilakukan analisis. Masalah terkait keselamatan dalam spesifikasi dan desain biasanya sangat mahal untuk diperbaiki pada tahap implementasi.
3. Sistem perangkat lunak besar yang merupakan bagian dari sistem rekayasa yang lebih luas yang dikembangkan oleh beberapa perusahaan mitra. Perangkat keras dalam sistem dapat dikembangkan dengan menggunakan model yang serupa, dan perusahaan merasa lebih mudah menggunakan model umum untuk perangkat keras dan perangkat lunak. Lebih lanjut, jika beberapa perusahaan terlibat, spesifikasi lengkap mungkin diperlukan untuk memungkinkan pengembangan subsistem yang berbeda secara independen.

Model air terjun bukanlah model proses yang tepat dalam situasi di mana komunikasi tim informal dimungkinkan dan persyaratan perangkat lunak berubah dengan cepat. Pengembangan berulang dan metode tangkas lebih baik untuk sistem ini.

Varian penting dari model waterfall adalah pengembangan sistem formal, di mana model matematis dari spesifikasi sistem dibuat. Model ini kemudian disempurnakan, menggunakan transformasi matematis yang menjaga konsistensinya, menjadi kode yang dapat dieksekusi. Proses pengembangan formal, seperti yang didasarkan pada metode B (Abrial
2005, 2010), sebagian besar digunakan dalam pengembangan sistem perangkat lunak yang memiliki persyaratan keselamatan, keandalan, atau keamanan yang ketat. Pendekatan formal menyederhanakan produksi kasus keselamatan atau keamanan. Hal ini menunjukkan kepada pelanggan atau pembuat peraturan bahwa sistem tersebut benar-benar memenuhi persyaratan keselamatan atau keamanannya. Namun, karena tingginya biaya pengembangan spesifikasi formal, model pengembangan ini jarang digunakan kecuali untuk rekayasa sistem kritis.

2.1.2 Pengembangan bertahap

Pengembangan bertahap didasarkan pada gagasan mengembangkan implementasi awal, mendapatkan umpan balik dari pengguna dan lainnya, dan mengembangkan perangkat lunak melalui beberapa versi hingga sistem yang diperlukan telah dikembangkan (Gambar 2.2). Spesifikasi, pengembangan, dan aktivitas validasi disisipkan alih-alih terpisah, dengan umpan balik cepat di seluruh aktivitas.



Gambar 2.2 Tambahan pengembangan

Pengembangan bertahap dalam beberapa bentuk sekarang menjadi pendekatan yang paling umum untuk pengembangan sistem aplikasi dan produk perangkat lunak. Pendekatan ini bisa digerakkan oleh rencana, gesit atau, biasanya, campuran dari pendekatan ini. Dalam pendekatan berdasarkan rencana, peningkatan sistem diidentifikasi sebelumnya; jika pendekatan tangkas diadopsi, kenaikan awal diidentifikasi, tetapi pengembangan kenaikan selanjutnya bergantung pada kemajuan dan prioritas pelanggan.

Pengembangan perangkat lunak tambahan, yang merupakan bagian fundamental dari metode pengembangan tangkas, lebih baik daripada pendekatan air terjun untuk sistem yang persyaratannya cenderung berubah selama proses pengembangan. Ini adalah kasus untuk kebanyakan sistem bisnis dan produk perangkat lunak. Perkembangan bertahap mencerminkan cara kita memecahkan masalah. Kami jarang mencari solusi masalah yang lengkap sebelumnya, tetapi bergerak menuju solusi dalam serangkaian langkah, mundur ketika kami menyadari bahwa kami telah membuat kesalahan. Dengan mengembangkan perangkat lunak secara bertahap, akan lebih murah dan lebih mudah untuk membuat perubahan pada perangkat lunak saat dikembangkan.

Setiap kenaikan atau versi sistem menggabungkan beberapa fungsionalitas yang dibutuhkan oleh pelanggan. Umumnya, peningkatan awal sistem mencakup fungsionalitas yang paling penting atau paling mendesak. Ini berarti bahwa pelanggan atau pengguna dapat mengevaluasi sistem pada tahap yang relatif awal dalam pengembangan untuk melihat apakah sistem memberikan apa yang dibutuhkan. Jika tidak, maka hanya kenaikan saat ini yang harus diubah dan, mungkin, fungsionalitas baru yang ditentukan untuk kenaikan selanjutnya.

Pengembangan bertahap memiliki tiga keunggulan utama dibandingkan model air terjun:

1. Biaya penerapan perubahan persyaratan berkurang. Jumlah analisis dan dokumentasi yang harus dilakukan ulang jauh lebih sedikit daripada yang dibutuhkan dengan model air terjun.

2. Lebih mudah mendapatkan umpan balik pelanggan atas pekerjaan pengembangan yang telah dilakukan. Pelanggan

Masalah dengan perkembangan tambahan

Meskipun pengembangan bertahap memiliki banyak keuntungan, itu tidak bebas masalah. Penyebab utama dari kesulitan tersebut adalah kenyataan bahwa organisasi besar memiliki prosedur birokrasi yang telah berkembang dari waktu ke waktu dan mungkin terdapat ketidaksesuaian antara prosedur ini dan proses berulang atau gesit yang lebih informal.

Terkadang prosedur ini ada untuk alasan yang bagus. Misalnya, mungkin ada prosedur untuk memastikan bahwa perangkat lunak tersebut memenuhi peraturan eksternal yang diterapkan dengan benar (misalnya, di Amerika Serikat, peraturan akuntansi Sarbanes Oxley). Mengubah prosedur ini mungkin tidak dapat dilakukan, jadi konflik proses mungkin tidak dapat dihindari.

http://software-engineering-book.com/web/incremental-development/

banyak yang telah diterapkan. Pelanggan merasa sulit untuk menilai kemajuan dari dokumen desain perangkat lunak.

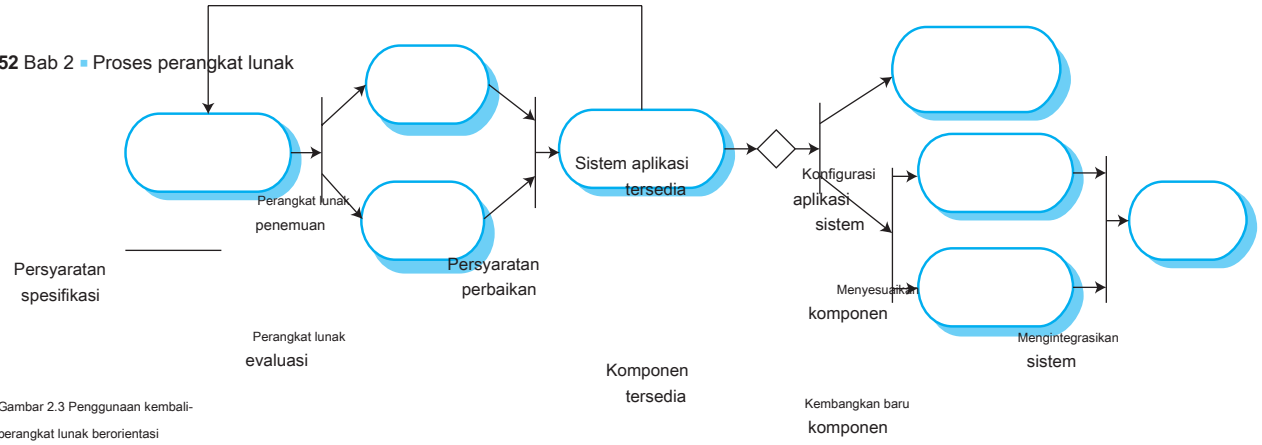
3. Pengiriman lebih awal dan penyebaran perangkat lunak yang berguna kepada pelanggan dimungkinkan, meskipun semua fungsionalitas belum disertakan. Pelanggan dapat menggunakan dan mendapatkan nilai dari perangkat lunak lebih awal daripada yang dimungkinkan dengan proses air terjun.

Dari perspektif manajemen, pendekatan inkremental memiliki dua masalah:

1. Prosesnya tidak terlihat. Manajer membutuhkan kiriman rutin untuk mengukur kemajuan. Jika sistem dikembangkan dengan cepat, tidak efektif biaya untuk menghasilkan dokumen yang mencerminkan setiap versi sistem.
2. Struktur sistem cenderung menurun seiring penambahan baru. Perubahan reguler menyebabkan kode berantakan karena fungsionalitas baru ditambahkan dengan cara apa pun yang memungkinkan. Menambahkan fitur baru ke sistem menjadi semakin sulit dan mahal. Untuk mengurangi degradasi struktural dan kekacauan kode umum, metode tangkas menyarankan agar Anda secara teratur melakukan refactor (memperbaiki dan merestrukturisasi) perangkat lunak.

Masalah pengembangan bertahap menjadi sangat akut untuk sistem yang besar, kompleks, dan tahan lama, di mana tim yang berbeda mengembangkan bagian sistem yang berbeda. Sistem yang besar membutuhkan kerangka kerja atau arsitektur yang stabil, dan tanggung jawab tim yang berbeda yang bekerja pada bagian-bagian sistem perlu didefinisikan dengan jelas sehubungan dengan arsitektur tersebut. Ini harus direncanakan terlebih dahulu daripada dikembangkan secara bertahap.

Pengembangan inkremental tidak berarti Anda harus mengirimkan setiap kenaikan ke pelanggan sistem. Anda dapat mengembangkan sistem secara bertahap dan memaparkannya kepada pelanggan dan pemangku kepentingan lainnya untuk dikomentari, tanpa harus mengirimkannya dan menerapkannya di lingkungan pelanggan. Pengiriman inkremental (dibahas dalam Bagian 2.3.2) berarti bahwa perangkat lunak digunakan dalam proses operasional yang nyata, sehingga umpan balik pengguna cenderung realistis. Namun, memberikan umpan balik tidak selalu memungkinkan karena bereksperimen dengan perangkat lunak baru dapat mengganggu proses bisnis normal.



Gambar 2.3 Penggunaan kembali-perangkat lunak berorientasi teknik

2.1.3 Integrasi dan konfigurasi

Di sebagian besar proyek perangkat lunak, ada beberapa penggunaan ulang perangkat lunak. Hal ini sering terjadi secara informal ketika orang yang mengerjakan proyek mengetahui atau mencari kode yang mirip dengan yang diperlukan. Mereka mencari ini, memodifikasinya sesuai kebutuhan, dan mengintegrasikannya dengan kode baru yang telah mereka kembangkan.

Penggunaan kembali informal ini terjadi terlepas dari proses pengembangan yang digunakan. Namun, sejak tahun 2000, proses pengembangan perangkat lunak yang berfokus pada penggunaan kembali perangkat lunak yang ada telah digunakan secara luas. Pendekatan berorientasi penggunaan kembali bergantung pada basis komponen perangkat lunak yang dapat digunakan kembali dan kerangka kerja yang terintegrasi untuk komposisi komponen ini.

Tiga jenis komponen perangkat lunak sering digunakan kembali:

1. Sistem aplikasi yang berdiri sendiri yang dikonfigurasi untuk digunakan di lingkungan tertentu. Sistem ini adalah sistem tujuan umum yang memiliki banyak fitur, tetapi harus disesuaikan untuk digunakan dalam aplikasi tertentu.
2. Koleksi objek yang dikembangkan sebagai komponen atau sebagai paket untuk diintegrasikan dengan kerangka komponen seperti kerangka Java Spring (Wheeler dan White 2013).
3. Layanan web yang dikembangkan sesuai dengan standar layanan dan yang tersedia untuk pemanggilan jarak jauh melalui Internet.

Gambar 2.3 menunjukkan model proses umum untuk pengembangan berbasis penggunaan kembali, berdasarkan integrasi dan konfigurasi. Tahapan dalam proses ini adalah:

1. *Spesifikasi kebutuhan* Persyaratan awal untuk sistem diusulkan. Ini tidak harus diuraikan secara rinci tetapi harus mencakup deskripsi singkat tentang persyaratan penting dan fitur sistem yang diinginkan.
2. *Penemuan dan evaluasi perangkat lunak* Mengingat garis besar persyaratan perangkat lunak, pencarian dilakukan untuk komponen dan sistem yang menyediakan fungsionalitas yang diperlukan. Komponen dan

Alat pengembangan perangkat lunak

Alat pengembangan perangkat lunak adalah program yang digunakan untuk mendukung kegiatan proses rekayasa perangkat lunak. Alat ini termasuk alat manajemen persyaratan, editor desain, alat pendukung refactoring, kompiler, debugger, pelacak bug, dan alat pembangunan sistem.

Alat perangkat lunak menyediakan dukungan proses dengan mengotomatiskan beberapa aktivitas proses dan dengan menyediakan informasi tentang perangkat lunak yang sedang dikembangkan. Sebagai contoh:

- Pengembangan model sistem grafis sebagai bagian dari spesifikasi kebutuhan atau perancangan perangkat lunak
- Pembuatan kode dari model grafis ini
- Pembuatan antarmuka pengguna dari deskripsi antarmuka grafis yang dibuat secara interaktif oleh pengguna. Program debugging melalui
- penyediaan informasi tentang program yang sedang dijalankan
- Terjemahan otomatis dari program yang ditulis menggunakan versi lama bahasa pemrograman ke versi yang lebih baru

Alat dapat digabungkan dalam kerangka kerja yang disebut Lingkungan Pengembangan Interaktif atau IDE. Ini menyediakan seperangkat fasilitas umum yang dapat digunakan oleh alat sehingga lebih mudah bagi alat untuk berkomunikasi dan beroperasi secara terintegrasi.

http://software-engineering-book.com/web/software-tools/

mereka memenuhi persyaratan penting dan jika secara umum cocok untuk digunakan dalam sistem.

3. *Penyempurnaan persyaratan* Selama tahap ini, persyaratan disempurnakan menggunakan informasi tentang komponen yang dapat digunakan kembali dan aplikasi yang telah ditemukan. Persyaratan diubah untuk mencerminkan komponen yang tersedia, dan spesifikasi sistem ditentukan ulang. Jika modifikasi tidak memungkinkan, aktivitas analisis komponen dapat dimasukkan kembali untuk mencari solusi alternatif.

4. *Konfigurasi sistem aplikasi* Jika sistem aplikasi off-the-shelf yang memenuhi persyaratan tersedia, itu mungkin kemudian dikonfigurasi untuk digunakan untuk membuat sistem baru.

5. *Adaptasi dan integrasi komponen* Jika tidak ada sistem off-the-shelf, komponen individu yang dapat digunakan kembali dapat dimodifikasi dan komponen baru dikembangkan. Ini kemudian diintegrasikan untuk membuat sistem.

Rekayasa perangkat lunak yang berorientasi pada penggunaan ulang, berdasarkan konfigurasi dan integrasi, memiliki keuntungan yang jelas dalam mengurangi jumlah perangkat lunak yang akan dikembangkan dan dengan demikian mengurangi biaya dan risiko. Biasanya juga mengarah pada pengiriman perangkat lunak yang lebih cepat. Namun, kompromi persyaratan

yang tidak memenuhi kebutuhan pengguna yang sebenarnya. Selain itu, beberapa kendali atas evolusi sistem hilang karena versi baru dari komponen yang dapat digunakan kembali tidak berada di bawah kendali organisasi yang menggunakannya.

Penggunaan kembali perangkat lunak sangat penting, dan beberapa bab di bagian ketiga saya telah mendedikasikan beberapa bab di bagian ke-3 buku ini untuk topik ini. Masalah umum penggunaan ulang perangkat lunak dibahas dalam Bab 15, rekayasa perangkat lunak berbasis komponen di Bab 16 dan 17, dan sistem berorientasi layanan di Bab 18.

Proses kegiatan

Proses perangkat lunak nyata adalah rangkaian kegiatan teknis, kolaboratif, dan manajerial yang diselingi dengan tujuan keseluruhan untuk menentukan, merancang, menerapkan, dan menguji sistem perangkat lunak. Umumnya, proses sekarang didukung oleh alat. Ini berarti bahwa pengembang perangkat lunak dapat menggunakan berbagai alat perangkat lunak untuk membantu mereka, seperti sistem manajemen persyaratan, editor model desain, editor program, alat pengujian otomatis, dan debugger.

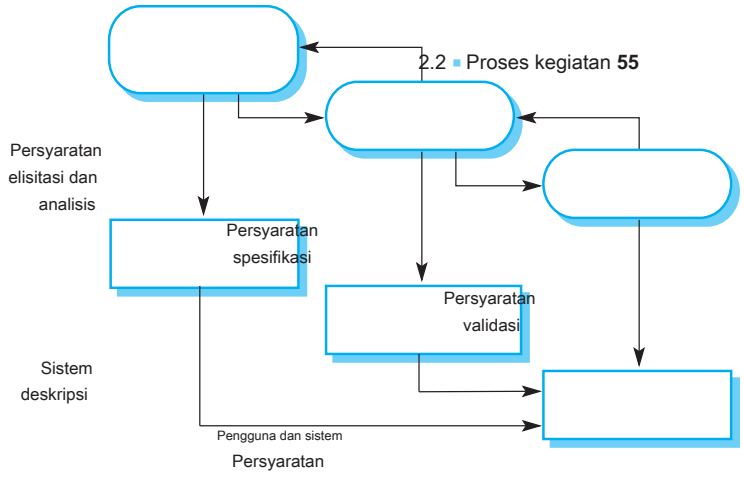
Empat aktivitas proses dasar yaitu spesifikasi, pengembangan, validasi, dan evolusi diatur secara berbeda dalam proses pengembangan yang berbeda. Dalam model air terjun, mereka disusun secara berurutan, sedangkan dalam perkembangan bertahap mereka disisipkan. Bagaimana kegiatan ini dilakukan tergantung pada jenis perangkat lunak yang dikembangkan, pengalaman dan kompetensi pengembang, dan jenis organisasi yang mengembangkan perangkat lunak tersebut.

2.2.1 Spesifikasi perangkat lunak

Spesifikasi perangkat lunak atau rekayasa persyaratan adalah proses memahami dan mendefinisikan layanan apa yang dibutuhkan dari sistem dan mengidentifikasi kendala pada operasi dan pengembangan sistem. Rekayasa persyaratan adalah tahap yang sangat penting dari proses perangkat lunak, karena kesalahan yang dibuat pada tahap ini pasti akan menyebabkan masalah di kemudian hari dalam desain dan implementasi sistem.

Sebelum proses rekayasa persyaratan dimulai, perusahaan dapat melakukan studi kelayakan atau pemasaran untuk menilai apakah ada kebutuhan atau pasar untuk perangkat lunak tersebut dan apakah realistis secara teknis dan finansial untuk mengembangkan perangkat lunak yang diperlukan. Studi kelayakan adalah studi jangka pendek dan relatif murah yang menginformasikan keputusan apakah akan melanjutkan analisis yang lebih rinci atau tidak.

Proses rekayasa persyaratan (Gambar 2.4) bertujuan untuk menghasilkan dokumen persyaratan yang disepakati yang menetapkan sistem yang memenuhi persyaratan pemangku kepentingan. Persyaratan biasanya disajikan pada dua tingkat detail. Pengguna akhir dan pelanggan membutuhkan pernyataan tingkat tinggi tentang persyaratan; pengembang sistem membutuhkan spesifikasi sistem yang lebih rinci.



Gambar 2.4 Itu Persyaratan proses rekayasa

Ada tiga aktivitas utama dalam proses rekayasa persyaratan:

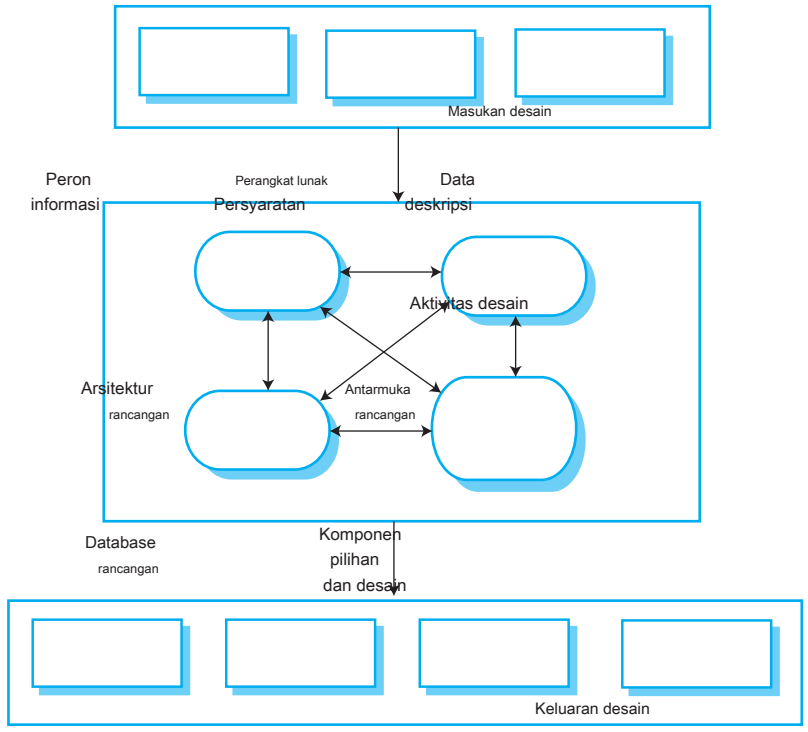
1. *Elisitasi dan analisis kebutuhan* Ini adalah proses mendapatkan persyaratan sistem melalui observasi sistem yang ada, diskusi dengan pengguna dan pengadaan potensial, analisis tugas, dan sebagainya. Ini mungkin melibatkan pengembangan satu atau lebih model dan prototipe sistem. Ini membantu Anda memahami sistem yang akan ditentukan.

2. *Spesifikasi kebutuhan* Spesifikasi kebutuhan adalah kegiatan menerjemahkan informasi yang dikumpulkan selama analisis kebutuhan ke dalam dokumen yang menetapkan seperangkat persyaratan. Dua jenis persyaratan dapat disertakan dalam dokumen ini. Persyaratan pengguna adalah pernyataan abstrak dari persyaratan sistem untuk pelanggan dan pengguna akhir sistem; persyaratan sistem adalah penjelasan yang lebih rinci tentang fungsionalitas yang akan disediakan.

3. *Validasi persyaratan* Kegiatan ini memeriksa persyaratan realisme, konsistensi, dan kelengkapan. Selama proses ini, kesalahan dalam dokumen persyaratan pasti ditemukan. Ini kemudian harus dimodifikasi untuk memperbaiki masalah ini.

Analisis kebutuhan berlanjut selama definisi dan spesifikasi, dan persyaratan baru muncul selama proses. Oleh karena itu, kegiatan analisis, definisi, dan spesifikasi saling terkait.

Dalam metode agile, spesifikasi kebutuhan bukanlah aktivitas yang terpisah tetapi dilihat sebagai bagian dari pengembangan sistem. Persyaratan ditentukan secara informal untuk setiap kenaikan sistem tepat sebelum kenaikan tersebut dikembangkan. Persyaratan ditentukan sesuai dengan prioritas pengguna. Permintaan yang muncul berasal dari pengguna yang merupakan bagian dari atau bekerja sama dengan tim pengembangan.



Gambar 2.5 Model umum
ile
proses desain

2.2.2 Desain dan implementasi perangkat lunak

Tahap implementasi pengembangan perangkat lunak adalah proses pengembangan sistem yang dapat dieksekusi untuk pengiriman ke pelanggan. Terkadang ini melibatkan aktivitas terpisah dari desain perangkat lunak dan pemrograman. Namun, jika pendekatan agile untuk pengembangan digunakan, desain dan implementasi saling terkait, tanpa dokumen desain formal yang dihasilkan selama proses tersebut. Tentu saja, software tersebut masih dirancang, tetapi desain tersebut dicatat secara informal di papan tulis dan notebook pemrogram.

Sebuah desain perangkat lunak adalah deskripsi dari struktur perangkat lunak yang akan diimplementasikan, model data dan struktur yang digunakan oleh sistem, antarmuka antara komponen sistem dan, terkadang, algoritma yang digunakan. Desainer tidak langsung sampai pada desain yang sudah jadi tetapi mengembangkan desain secara bertahap. Mereka menambahkan detail saat mengembangkan desain mereka, dengan pengulangan konstan untuk memodifikasi desain sebelumnya.

Gambar 2.5 adalah model abstrak dari proses desain yang menunjukkan input ke proses desain, aktivitas proses, dan output proses. Aktivitas proses desain saling terkait dan saling bergantung. Informasi baru tentang desain terus dihasilkan, dan ini memengaruhi keputusan desain sebelumnya. Oleh karena itu, pengerjaan ulang desain tidak bisa dihindari.

Kebanyakan antarmuka perangkat lunak dengan sistem perangkat lunak lain. Sistem lain ini termasuk sistem operasi, database, middleware, dan sistem aplikasi lainnya. Ini membentuk 'platform perangkat lunak', lingkungan tempat perangkat lunak akan dijalankan. Informasi tentang platform ini merupakan masukan penting untuk proses desain, karena desainer harus memutuskan cara terbaik untuk mengintegrasikannya dengan lingkungannya. Jika sistem mengolah data yang ada, maka deskripsi data tersebut dapat dimasukkan dalam spesifikasi platform. Jika tidak, deskripsi data harus menjadi masukan untuk proses desain sehingga organisasi data sistem dapat didefinisikan.

Aktivitas dalam proses perancangan berbeda-beda, bergantung pada jenis sistem yang dikembangkan. Misalnya, sistem real-time memerlukan tahap tambahan dari desain waktu tetapi mungkin tidak menyertakan database, jadi tidak ada desain database yang terlibat. Gambar 2.5 menunjukkan empat aktivitas yang mungkin menjadi bagian dari proses desain untuk sistem informasi:

1. *Desain arsitektur,* di mana Anda mengidentifikasi keseluruhan struktur sistem, komponen utama (terkadang disebut subsistem atau modul), hubungannya, dan bagaimana mereka didistribusikan.

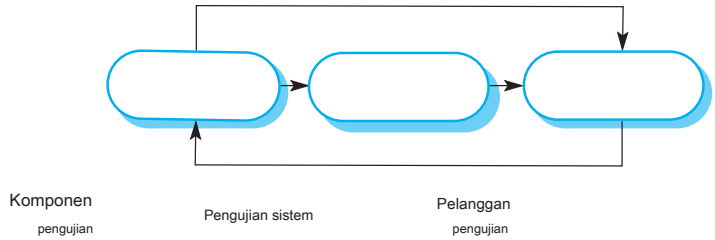
2. *Desain database,* di mana Anda mendesain struktur data sistem dan bagaimana ini akan direpresentasikan dalam database. Sekali lagi, pekerjaan di sini bergantung pada apakah database yang ada akan digunakan kembali atau database baru yang akan dibuat.

3. *Desain antarmuka,* tempat Anda menentukan antarmuka antara komponen sistem. Spesifikasi antarmuka ini harus jelas. Dengan antarmuka yang tepat, suatu komponen dapat digunakan oleh komponen lain tanpa harus mengetahui cara penerapannya. Setelah spesifikasi antarmuka disetujui, komponen dapat dirancang dan dikembangkan secara terpisah.

4. *Pemilihan dan desain komponen,* di mana Anda mencari komponen yang dapat digunakan kembali dan, jika tidak ada komponen yang sesuai tersedia, rancang komponen perangkat lunak baru. Desain pada tahap ini dapat berupa deskripsi komponen sederhana dengan detail implementasi diserahkan kepada programmer. Alternatifnya, ini bisa berupa daftar perubahan yang harus dilakukan pada komponen yang dapat digunakan kembali atau model desain rinci yang dinyatakan dalam UML. Model desain kemudian dapat digunakan untuk menghasilkan implementasi secara otomatis.

Kegiatan tersebut menghasilkan keluaran desain, yang juga ditunjukkan pada Gambar 2.5. Untuk sistem kritis, keluaran dari proses desain adalah dokumen desain terperinci yang menetapkan deskripsi sistem yang tepat dan akurat. Jika pendekatan model-driven digunakan (Bab 5), output desain adalah diagram desain. Di mana metode pengembangan yang tangkas digunakan, keluaran dari proses desain mungkin bukan dokumen spesifikasi yang terpisah tetapi dapat direpresentasikan dalam kode program.

Pengembangan program untuk mengimplementasikan sistem mengikuti secara alami dari desain sistem. Meskipun beberapa kelas program, seperti sistem kritis-keselamatan, biasanya dirancang secara rinci sebelum implementasi dimulai, desain dan pengembangan program lebih umum disisipkan. Alat pengembangan perangkat lunak dapat digunakan untuk menghasilkan program kerangka dari desain. Ini



Gambar 2.6 Tahapan pengujian

mendefinisikan dan mengimplementasikan antarmuka, dan, dalam banyak kasus, pengembang hanya perlu menambahkan detail operasi setiap komponen program.

Pemrograman adalah aktivitas individu, dan tidak ada proses umum yang biasanya diikuti. Beberapa programmer memulai dengan komponen yang mereka pahami, kembangkan, dan kemudian beralih ke komponen yang kurang dipahami. Yang lain mengambil pendekatan yang berlawanan, meninggalkan komponen yang sudah dikenal sampai yang terakhir karena mereka tahu bagaimana mengembangkannya. Beberapa pengembang suka mendefinisikan data di awal proses dan kemudian menggunakannya untuk mendorong pengembangan program; yang lain membiarkan data tidak ditentukan selama mungkin.

Biasanya, programmer melakukan beberapa pengujian kode yang telah mereka kembangkan. Ini sering menunjukkan cacat program (bug) yang harus dihapus dari program. Menemukan dan memperbaiki cacat program disebut *debugging*. Pengujian dan debugging yang rusak adalah proses yang berbeda. Pengujian menetapkan adanya cacat. Debugging berkaitan dengan menemukan dan memperbaiki cacat ini.

Saat Anda men-debug, Anda harus menghasilkan hipotesis tentang perilaku program yang dapat diamati dan kemudian menguji hipotesis ini dengan harapan menemukan kesalahan yang menyebabkan anomali keluaran. Menguji hipotesis mungkin melibatkan penelusuran kode program secara manual. Ini mungkin memerlukan kasus uji baru untuk melokalkan masalah. Alat debugging interaktif, yang menunjukkan nilai antara variabel program dan jejak pernyataan yang dieksekusi, biasanya digunakan untuk mendukung proses debugging.

2.2.3 Validasi perangkat lunak

Validasi perangkat lunak atau, lebih umum, verifikasi dan validasi (V & V) dimaksudkan untuk menunjukkan bahwa sistem sesuai dengan spesifikasinya dan memenuhi harapan pelanggan sistem. Pengujian program, di mana sistem dijalankan dengan menggunakan data pengujian simulasi, adalah teknik validasi utama. Validasi juga dapat melibatkan proses pemeriksaan, seperti inspeksi dan tinjauan, pada setiap tahap proses perangkat lunak dari definisi persyaratan pengguna hingga pengembangan program. Namun, sebagian besar waktu dan upaya V & V dihabiskan untuk pengujian program.

Kecuali untuk program kecil, sistem tidak boleh diuji sebagai satu unit monolitik. Gambar 2.6 menunjukkan proses pengujian tiga tahap di mana komponen sistem diuji secara individual, kemudian sistem terintegrasi diuji. Untuk perangkat lunak khusus, pengujian pelanggan melibatkan pengujian sistem dengan data pelanggan nyata. Untuk produk yang dijual sebagai aplikasi, pengujian pelanggan terkadang disebut pengujian beta di mana pengguna terpilih mencoba dan mengomentari perangkat lunak.

Tahapan dalam proses pengujian adalah:

1. *Pengujian komponen* Komponen penyusun sistem diuji oleh orang yang mengembangkan sistem. Setiap komponen diuji secara independen, tanpa komponen sistem lainnya. Komponen dapat berupa entitas sederhana seperti fungsi atau kelas objek atau mungkin pengelompokan yang koheren dari entitas ini. Alat otomatisasi pengujian, seperti JUnit untuk Java, yang dapat menjalankan kembali pengujian saat versi baru komponen dibuat, biasanya digunakan (Koskela 2013).

2. *Pengujian sistem* Komponen sistem terintegrasi untuk membuat sistem yang lengkap. Proses ini berkaitan dengan menemukan kesalahan yang dihasilkan dari interaksi yang tidak terduga antara komponen dan masalah antarmuka komponen. Ini juga berkaitan dengan menunjukkan bahwa sistem memenuhi persyaratan fungsional dan non-fungsionalnya, dan menguji properti sistem yang muncul. Untuk sistem besar, ini mungkin proses multistage di mana komponen diintegrasikan untuk membentuk subsistem yang diuji secara individual sebelum subsistem ini diintegrasikan untuk membentuk sistem akhir.

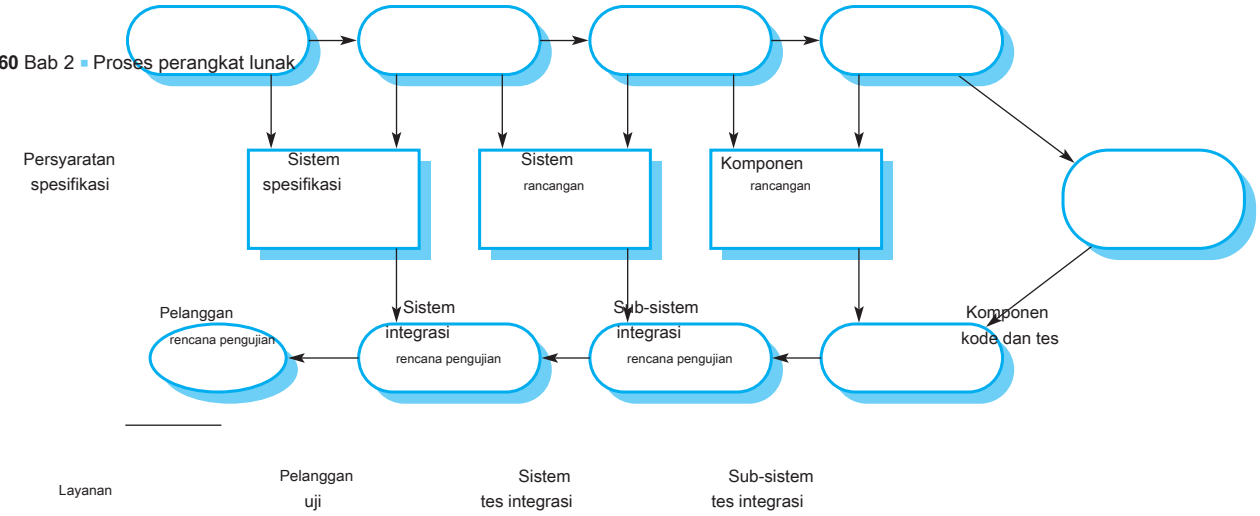
3. *Pengujian pelanggan* Ini merupakan tahap terakhir dalam proses pengujian sebelum sistem diterima untuk penggunaan operasional. Sistem diuji oleh pelanggan sistem (atau pelanggan potensial) daripada dengan data pengujian simulasi. Untuk perangkat lunak yang dibuat khusus, pengujian pelanggan dapat mengungkapkan kesalahan dan kelalaian dalam definisi persyaratan sistem, karena data nyata menjalankan sistem dengan cara yang berbeda dari data pengujian. Pengujian pelanggan juga dapat mengungkapkan masalah persyaratan di mana fasilitas sistem tidak benar-benar memenuhi kebutuhan pengguna atau kinerja sistem tidak dapat diterima. Untuk produk, pengujian pelanggan menunjukkan seberapa baik produk perangkat lunak memenuhi kebutuhan pelanggan.

Idealnya, cacat komponen ditemukan di awal proses pengujian, dan masalah antarmuka ditemukan saat sistem terintegrasi. Namun, saat cacat ditemukan, program harus di-debug, dan ini mungkin memerlukan pengulangan tahapan lain dalam proses pengujian. Kesalahan dalam komponen program, misalnya, mungkin terungkap selama pengujian sistem. Oleh karena itu, proses ini merupakan proses yang berulang dengan informasi yang diumpankan kembali dari tahap selanjutnya ke bagian proses sebelumnya.

Biasanya, pengujian komponen hanyalah bagian dari proses pengembangan normal. Pemrogram membuat data pengujian mereka sendiri dan secara bertahap menguji kode saat dikembangkan. Programmer mengetahui komponen tersebut dan oleh karena itu merupakan orang terbaik untuk menghasilkan kasus uji.

Jika pendekatan inkremental untuk pengembangan digunakan, setiap kenaikan harus diuji saat dikembangkan, dengan pengujian ini berdasarkan persyaratan untuk kenaikan tersebut. Dalam pengembangan berbasis pengujian, yang merupakan bagian normal dari proses agile, pengujian dikembangkan bersama dengan persyaratan sebelum pengembangan dimulai. Ini membantu penguji dan pengembang untuk memahami persyaratan dan memastikan bahwa tidak ada penundaan saat kasus pengujian dibuat.

Ketika proses perangkat lunak yang digerakkan oleh rencana digunakan (misalnya, untuk pengembangan sistem kritis),



Gambar 2.7 Menguji fase dalam proses perangkat lunak yang digerakkan oleh rencana

dari rencana pengujian tersebut, yang telah dikembangkan dari spesifikasi dan desain sistem. Gambar 2.7 mengilustrasikan bagaimana rencana pengujian menghubungkan antara pengujian dan kegiatan pengembangan. Ini kadang-kadang disebut model pengembangan V (putar sisinya untuk melihat V). Model V menunjukkan aktivitas validasi perangkat lunak yang sesuai dengan setiap tahap model proses waterfall.

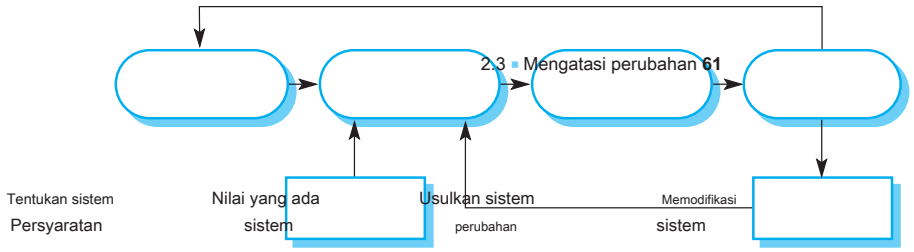
Ketika suatu sistem akan dipasarkan sebagai produk perangkat lunak, proses pengujian yang disebut pengujian beta sering digunakan. Pengujian beta melibatkan pengiriman sistem ke sejumlah pelanggan potensial yang setuju untuk menggunakan sistem itu. Mereka melaporkan masalah ke pengembang sistem. Ini mengekspos produk ke penggunaan nyata dan mendeteksi kesalahan yang mungkin tidak diantisipasi oleh pengembang produk. Setelah umpan balik ini, produk perangkat lunak dapat dimodifikasi dan dirilis untuk pengujian beta lebih lanjut atau penjualan umum.

2.2.4 Evolusi perangkat lunak

Fleksibilitas perangkat lunak adalah salah satu alasan utama mengapa semakin banyak perangkat lunak dimasukkan ke dalam sistem yang besar dan kompleks. Setelah keputusan dibuat untuk memproduksi perangkat keras, akan sangat mahal untuk melakukan perubahan pada desain perangkat keras. Namun, perubahan dapat dilakukan pada perangkat lunak kapan saja selama atau setelah pengembangan sistem. Bahkan perubahan ekstensif masih jauh lebih murah daripada perubahan terkait pada perangkat keras sistem.

Secara historis, selalu ada pemisahan antara proses pengembangan perangkat lunak dan proses evolusi perangkat lunak (pemeliharaan perangkat lunak). Orang menganggap pengembangan perangkat lunak sebagai aktivitas kreatif di mana sistem perangkat lunak dikembangkan dari konsep awal hingga sistem kerja. Namun, terkadang mereka menganggap pemeliharaan perangkat lunak membosankan dan tidak menarik. Mereka berpikir bahwa pemeliharaan perangkat lunak kurang menarik dan menantang daripada pengembangan perangkat lunak asli.

Perbedaan antara pengembangan dan pemeliharaan ini semakin tidak relevan. Sangat sedikit sistem perangkat



Gambar 2.8 Perangkat lunak evolusi sistem

akal untuk melihat pengembangan dan pemeliharaan sebagai sebuah kontinum. Daripada dua proses terpisah, lebih realistis untuk memikirkan rekayasa perangkat lunak sebagai proses evolusi (Gambar 2.8) di mana perangkat lunak terus berubah selama masa pakainya sebagai respons terhadap perubahan persyaratan dan kebutuhan pelanggan.

Mengatasi perubahan

Perubahan tidak bisa dihindari di semua proyek perangkat lunak besar. Persyaratan sistem berubah seiring bisnis menanggapi tekanan eksternal, persaingan, dan prioritas manajemen yang berubah. Saat teknologi baru tersedia, pendekatan baru untuk desain dan implementasi menjadi mungkin. Oleh karena itu, model proses perangkat lunak apa pun yang digunakan, adalah penting bahwa model itu dapat mengakomodasi perubahan pada perangkat lunak yang sedang dikembangkan.

Perubahan menambah biaya pengembangan perangkat lunak karena itu biasanya berarti bahwa pekerjaan yang telah selesai harus dikerjakan ulang. Ini disebut pengerjaan ulang. Misalnya, jika hubungan antara persyaratan dalam sistem telah dianalisis dan persyaratan baru kemudian diidentifikasi, beberapa atau semua analisis persyaratan harus diulang. Mungkin perlu untuk mendesain ulang sistem untuk memberikan persyaratan baru, mengubah program apa pun yang telah dikembangkan, dan menguji ulang sistem.

Dua pendekatan terkait dapat digunakan untuk mengurangi biaya pengerjaan ulang:

1. *Ubah antisipasi,* di mana proses perangkat lunak mencakup aktivitas yang dapat mengantisipasi atau memprediksi kemungkinan perubahan sebelum pengerjaan ulang yang signifikan diperlukan. Misalnya, sistem prototipe dapat dikembangkan untuk menunjukkan beberapa fitur utama sistem kepada pelanggan. Mereka dapat bereksperimen dengan prototipe dan menyempurnakan persyaratan mereka sebelum berkomitmen pada biaya produksi perangkat lunak yang tinggi.

2. *Ubah toleransi,* di mana proses dan perangkat lunak dirancang sedemikian rupa sehingga perubahan dapat dengan mudah dilakukan pada sistem. Ini biasanya melibatkan beberapa bentuk perkembangan tambahan. Perubahan yang diusulkan dapat diterapkan secara bertahap yang belum dikembangkan. Jika ini tidak mungkin, maka hanya satu kenaikan (bagian kecil dari sistem) yang mungkin harus diubah untuk memasukkan

Di bagian ini, saya membahas dua cara untuk mengatasi perubahan dan perubahan persyaratan sistem:

1. *Pembuatan prototipe sistem,* di mana versi sistem atau bagian dari sistem dikembangkan dengan cepat untuk memeriksa persyaratan pelanggan dan kelayakan keputusan desain. Ini adalah metode antisipasi perubahan karena memungkinkan pengguna untuk bereksperimen dengan sistem sebelum pengiriman dan menyempurnakan persyaratan mereka. Oleh karena itu, jumlah proposal perubahan persyaratan yang dibuat setelah pengiriman kemungkinan akan berkurang.

2. *Pengiriman tambahan,* di mana peningkatan sistem dikirim ke pelanggan untuk komentar dan eksperimen. Ini mendukung penghindaran perubahan dan toleransi perubahan. Ini menghindari komitmen prematur terhadap persyaratan untuk keseluruhan sistem dan memungkinkan perubahan untuk dimasukkan ke dalam peningkatan selanjutnya dengan biaya yang relatif rendah.

Pengertian refactoring, yaitu memperbaiki struktur dan organisasi suatu program, juga merupakan mekanisme penting yang mendukung toleransi perubahan. Saya membahas ini di Bab 3 (metode Agile).

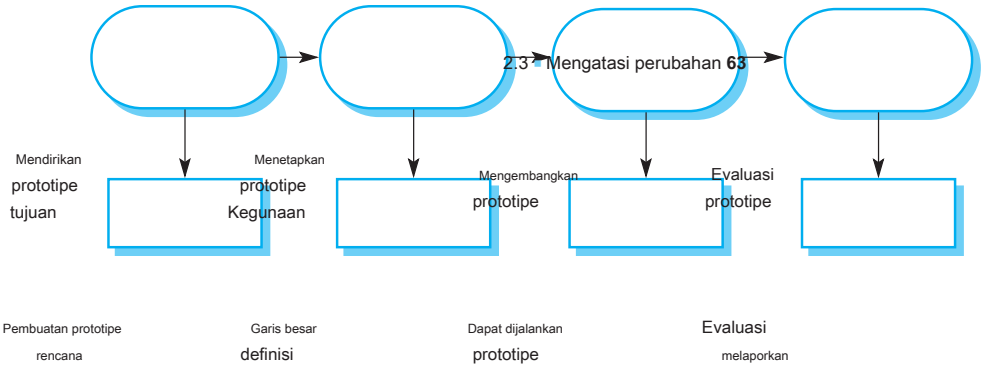
2.3.1 Pembuatan Prototipe

Prototipe adalah versi awal dari sistem perangkat lunak yang digunakan untuk mendemonstrasikan konsep, mencoba opsi desain, dan mencari tahu lebih banyak tentang masalah dan kemungkinan solusinya. Pengembangan prototipe yang cepat dan berulang sangat penting agar biaya dapat dikendalikan dan pemangku kepentingan sistem dapat bereksperimen dengan prototipe di awal proses perangkat lunak.

Prototipe perangkat lunak dapat digunakan dalam proses pengembangan perangkat lunak untuk membantu mengantisipasi perubahan yang mungkin diperlukan:

1. Dalam proses rekayasa persyaratan, prototipe dapat membantu dengan elisitasi dan validasi persyaratan sistem.
2. Dalam proses perancangan sistem, prototipe dapat digunakan untuk mengeksplorasi solusi perangkat lunak dan dalam pengembangan antarmuka pengguna untuk sistem.

Prototipe sistem memungkinkan pengguna potensial untuk melihat seberapa baik sistem mendukung pekerjaan mereka. Mereka mungkin mendapatkan ide baru untuk persyaratan dan menemukan area kekuatan dan kelemahan dalam perangkat lunak. Mereka kemudian dapat mengusulkan persyaratan sistem baru. Lebih lanjut, saat prototipe dikembangkan, hal itu dapat mengungkapkan kesalahan dan kelalaian dalam persyaratan sistem. Sebuah fitur yang dijelaskan dalam spesifikasi mungkin tampak jelas dan berguna. Namun, jika fungsi tersebut digabungkan dengan fungsi lain, pengguna sering kali menemukan bahwa tampilan awal mereka salah atau tidak lengkap. Spesifikasi sistem kemudian dapat dimodifikasi untuk mencerminkan pemahaman persyaratan yang berubah.



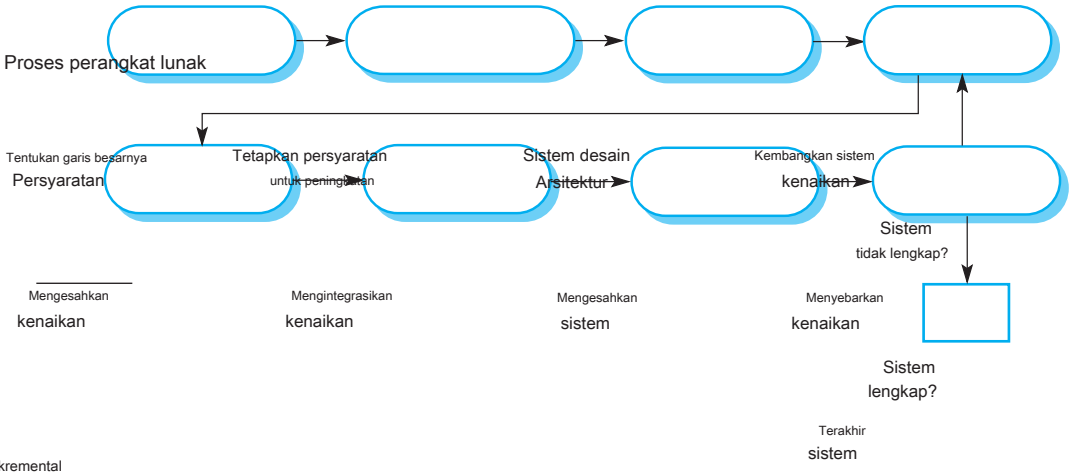
Gambar 2.9 Prototipe pengembangan

Prototipe sistem dapat digunakan saat sistem sedang dirancang untuk melakukan eksperimen desain untuk memeriksa kelayakan desain yang diusulkan. Misalnya, desain database mungkin dibuat prototipe dan diuji untuk memeriksa bahwa itu mendukung akses data yang efisien untuk kueri pengguna yang paling umum. Pembuatan prototipe cepat dengan keterlibatan pengguna akhir adalah satu-satunya cara yang masuk akal untuk mengembangkan antarmuka pengguna. Karena sifat antarmuka pengguna yang dinamis, deskripsi dan diagram tekstual tidak cukup baik untuk mengungkapkan persyaratan dan desain antarmuka pengguna.

Model proses untuk pengembangan prototipe ditunjukkan pada Gambar 2.9. Tujuan pembuatan prototipe harus dibuat eksplisit sejak awal proses. Ini mungkin untuk mengembangkan antarmuka pengguna, untuk mengembangkan sistem untuk memvalidasi persyaratan sistem fungsional, atau untuk mengembangkan sistem untuk mendemonstrasikan aplikasi kepada manajer. Prototipe yang sama biasanya tidak dapat memenuhi semua tujuan. Jika tujuan dibiarkan tidak dinyatakan, manajemen atau pengguna akhir mungkin salah memahami fungsi prototipe. Akibatnya, mereka mungkin tidak mendapatkan manfaat yang mereka harapkan dari pengembangan prototipe.

Tahap selanjutnya dalam proses ini adalah memutuskan apa yang akan dimasukkan dan, mungkin yang lebih penting, apa yang harus ditinggalkan dari sistem prototipe. Untuk mengurangi biaya pembuatan prototipe dan mempercepat jadwal pengiriman, Anda mungkin membiarkan beberapa fungsionalitas di luar prototipe. Anda dapat memutuskan untuk melonggarkan persyaratan non-fungsional seperti waktu respons dan penggunaan memori. Penanganan dan manajemen kesalahan dapat diabaikan kecuali jika tujuan prototipe adalah untuk membuat antarmuka pengguna. Standar keandalan dan kualitas program dapat diturunkan.

Tahap terakhir dari proses ini adalah evaluasi prototipe. Ketentuan harus dibuat selama tahap ini untuk pelatihan pengguna, dan tujuan prototipe harus digunakan untuk menghasilkan rencana evaluasi. Calon pengguna membutuhkan waktu untuk terbiasa dengan sistem baru dan menyesuaikan diri dengan pola penggunaan normal. Setelah mereka menggunakan sistem secara normal, mereka kemudian menemukan kesalahan dan kelalaian persyaratan. Masalah umum dengan pembuatan prototipe adalah bahwa pengguna mungkin tidak menggunakan prototipe dengan cara yang sama seperti mereka menggunakan sistem akhir. Penguji prototipe mungkin bukan pengguna sistem biasa. Mungkin tidak ada cukup waktu untuk melatih pengguna selama evaluasi prototipe. Jika prototipe lambat, evaluator dapat menyesuaikan cara kerja mereka dan menghindari fitur sistem yang memiliki waktu respons lambat. Ketika diberikan respons yang lebih baik di sistem akhir,



Gambar 2.10
Pengiriman inkremental

2.3.2 Pengiriman tambahan

Pengiriman inkremental (Gambar 2.10) adalah pendekatan pengembangan perangkat lunak di mana beberapa peningkatan yang dikembangkan dikirim ke pelanggan dan digunakan untuk digunakan di lingkungan kerja mereka. Dalam proses pengiriman tambahan, pelanggan menentukan layanan mana yang paling penting dan mana yang paling tidak penting bagi mereka. Kemudian, sejumlah kenaikan pengiriman ditentukan, dengan setiap kenaikan menyediakan subset fungsionalitas sistem. Alokasi layanan ke kenaikan bergantung pada prioritas layanan, dengan layanan prioritas tertinggi dilaksanakan dan disampaikan terlebih dahulu.

Setelah peningkatan sistem diidentifikasi, persyaratan untuk layanan yang akan diberikan pada peningkatan pertama ditentukan secara rinci dan peningkatan tersebut dikembangkan. Selama pengembangan, analisis persyaratan lebih lanjut untuk kenaikan selanjutnya dapat dilakukan, tetapi perubahan persyaratan untuk kenaikan saat ini tidak diterima.

Setelah selisih diselesaikan dan dikirimkan, itu dipasang di lingkungan kerja normal pelanggan. Mereka dapat bereksperimen dengan sistem, dan ini membantu mereka mengklarifikasi persyaratan untuk peningkatan sistem selanjutnya. Saat kenaikan baru diselesaikan, mereka diintegrasikan dengan kenaikan yang ada sehingga fungsionalitas sistem meningkat dengan setiap kenaikan yang diberikan.

Pengiriman inkremental memiliki sejumlah keuntungan:

1. Pelanggan dapat menggunakan peningkatan awal sebagai prototipe dan mendapatkan pengalaman yang menginformasikan kebutuhan mereka untuk peningkatan sistem selanjutnya. Tidak seperti prototipe, ini adalah bagian dari sistem nyata, jadi tidak ada pembelajaran ulang saat sistem lengkap tersedia.

2. Pelanggan tidak perlu menunggu sampai seluruh sistem dikirimkan sebelum mereka dapat memperoleh manfaat darinya. Kenaikan pertama memenuhi persyaratan paling kritis mereka, sehingga mereka dapat segera menggunakan perangkat lunak.

3. Proses tersebut mempertahankan manfaat dari pengembangan bertahap yang seharusnya relatif mudah untuk

4. Karena layanan dengan prioritas tertinggi disampaikan terlebih dahulu dan kemudian ditingkatkan secara bertahap, maka layanan sistem yang paling penting menerima pengujian terbanyak. Ini berarti bahwa pelanggan cenderung tidak mengalami kegagalan perangkat lunak di bagian terpenting sistem.

Namun, ada masalah dengan pengiriman inkremental. Dalam praktiknya, ini hanya berfungsi dalam situasi di mana sistem baru diperkenalkan dan evaluator sistem diberi waktu untuk bereksperimen dengan sistem baru. Masalah utama dengan pendekatan ini adalah:

1. Pengiriman berulang bermasalah ketika sistem baru dimaksudkan untuk menggantikan sistem yang sudah ada. Pengguna memerlukan semua fungsionalitas sistem lama dan biasanya tidak mau bereksperimen dengan sistem baru yang tidak lengkap. Seringkali tidak praktis untuk menggunakan sistem lama dan sistem baru secara berdampingan satu sama lain karena mereka cenderung memiliki database dan antarmuka pengguna yang berbeda.

2. Kebanyakan sistem memerlukan seperangkat fasilitas dasar yang digunakan oleh bagian sistem yang berbeda. Karena persyaratan tidak didefinisikan secara rinci hingga suatu kenaikan diterapkan, maka akan sulit untuk mengidentifikasi fasilitas umum yang diperlukan oleh semua kenaikan tersebut.

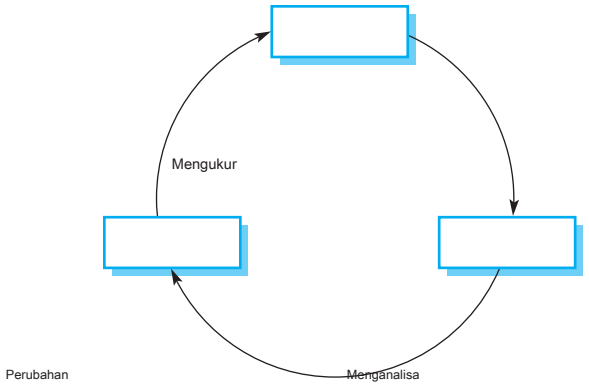
3. Inti dari proses iteratif adalah bahwa spesifikasi dikembangkan sehubungan dengan perangkat lunak. Namun, ini bertentangan dengan model pengadaan banyak organisasi, di mana spesifikasi sistem yang lengkap merupakan bagian dari kontrak pengembangan sistem. Dalam pendekatan inkremental, tidak ada spesifikasi sistem yang lengkap sampai kenaikan akhir ditentukan. Ini membutuhkan bentuk kontrak baru, yang mungkin sulit diakomodasi oleh pelanggan besar seperti lembaga pemerintah.

Untuk beberapa jenis sistem, pengembangan dan penyampaian bertahap bukanlah pendekatan terbaik. Ini adalah sistem yang sangat besar di mana pengembangan mungkin melibatkan tim yang bekerja di lokasi yang berbeda, beberapa sistem tertanam di mana perangkat lunak bergantung pada pengembangan perangkat keras, dan beberapa sistem kritis di mana semua persyaratan harus dianalisis untuk memeriksa interaksi yang dapat membahayakan keselamatan atau keamanan sistem.

Sistem yang besar ini, tentu saja, mengalami masalah yang sama, yaitu persyaratan yang tidak pasti dan berubah. Oleh karena itu, untuk mengatasi masalah ini dan mendapatkan beberapa manfaat dari pengembangan bertahap, prototipe sistem dapat dikembangkan dan digunakan sebagai platform untuk percobaan dengan persyaratan dan desain sistem. Dengan pengalaman yang didapat dari prototipe, persyaratan definitif kemudian dapat disepakati.

Peningkatan proses

Saat ini, ada permintaan konstan dari industri untuk perangkat lunak yang lebih murah dan lebih baik, yang harus dikirimkan ke tenggat waktu yang semakin ketat. Akibatnya, banyak perusahaan perangkat lunak telah beralih ke perbaikan



Gambar 2.11 Siklus perbaikan proses

kualitas perangkat lunak mereka, mengurangi biaya, atau mempercepat proses pengembangan mereka. Perbaikan proses berarti memahami proses yang ada dan mengubah proses ini untuk meningkatkan kualitas produk dan / atau mengurangi biaya dan waktu pengembangan. Saya membahas masalah umum pengukuran proses dan perbaikan proses secara rinci di web Bab 26.

Dua pendekatan yang sangat berbeda untuk proses perbaikan dan perubahan digunakan:

1. Pendekatan kematangan proses, yang berfokus pada peningkatan proses dan manajemen proyek serta memperkenalkan praktik rekayasa perangkat lunak yang baik ke dalam suatu organisasi. Tingkat kematangan proses mencerminkan sejauh mana praktik teknis dan manajemen yang baik telah diadopsi dalam proses pengembangan perangkat lunak organisasi. Tujuan utama dari pendekatan ini adalah peningkatan kualitas produk dan proses yang dapat diprediksi.

2. Pendekatan agile, yang berfokus pada pengembangan berulang dan pengurangan biaya overhead dalam proses perangkat lunak. Karakteristik utama dari metode tangkas adalah pengiriman fungsionalitas yang cepat dan respons terhadap perubahan kebutuhan pelanggan. Filosofi perbaikan di sini adalah bahwa proses terbaik adalah proses dengan overhead terendah dan pendekatan tangkas dapat mencapai ini. Saya menjelaskan pendekatan tangkas di Bab 3.

Orang-orang yang antusias dan berkomitmen pada masing-masing pendekatan ini umumnya meragukan manfaat yang lain. Pendekatan kematangan proses berakar pada pengembangan yang digerakkan oleh rencana dan biasanya membutuhkan peningkatan "overhead", dalam arti bahwa kegiatan diperkenalkan yang tidak secara langsung relevan dengan pengembangan program. Pendekatan tangkas berfokus pada kode yang sedang dikembangkan dan dengan sengaja meminimalkan formalitas dan dokumentasi.

Proses perbaikan proses umum yang mendasari pendekatan kematangan proses adalah proses siklus, seperti yang ditunjukkan pada Gambar 2.11. Tahapan dalam proses ini adalah:

1. *Pengukuran proses* Anda mengukur satu atau lebih atribut dari perangkat lunak pro-

perbaikan proses telah efektif. Saat Anda memperkenalkan perbaikan, Anda mengukur ulang atribut yang sama, yang diharapkan akan meningkat dalam beberapa cara.

2. *Analisis proses* Proses saat ini dinilai, dan kelemahan serta kemacetan proses diidentifikasi. Model proses (kadang disebut peta proses) yang menggambarkan proses dapat dikembangkan selama tahap ini. Analisis dapat difokuskan dengan mempertimbangkan karakteristik proses seperti kecepatan dan ketahanan.

3. *Proses perubahan* Perubahan proses diusulkan untuk mengatasi beberapa kelemahan proses yang teridentifikasi. Ini diperkenalkan, dan siklus dilanjutkan untuk mengumpulkan data tentang efektivitas perubahan.

Tanpa data konkret tentang suatu proses atau perangkat lunak yang dikembangkan menggunakan proses itu, tidak mungkin untuk menilai nilai perbaikan proses. Namun, perusahaan yang memulai proses perbaikan proses tidak mungkin memiliki data proses yang tersedia sebagai dasar perbaikan. Oleh karena itu, sebagai bagian dari siklus pertama perubahan, Anda mungkin harus mengumpulkan data tentang proses perangkat lunak dan mengukur karakteristik produk perangkat lunak.

Perbaikan proses adalah aktivitas jangka panjang, sehingga setiap tahapan dalam proses perbaikan bisa berlangsung beberapa bulan. Ini juga merupakan aktivitas berkelanjutan karena, apa pun proses baru yang diperkenalkan, lingkungan bisnis akan berubah dan proses baru itu sendiri harus berevolusi untuk memperhitungkan perubahan ini.

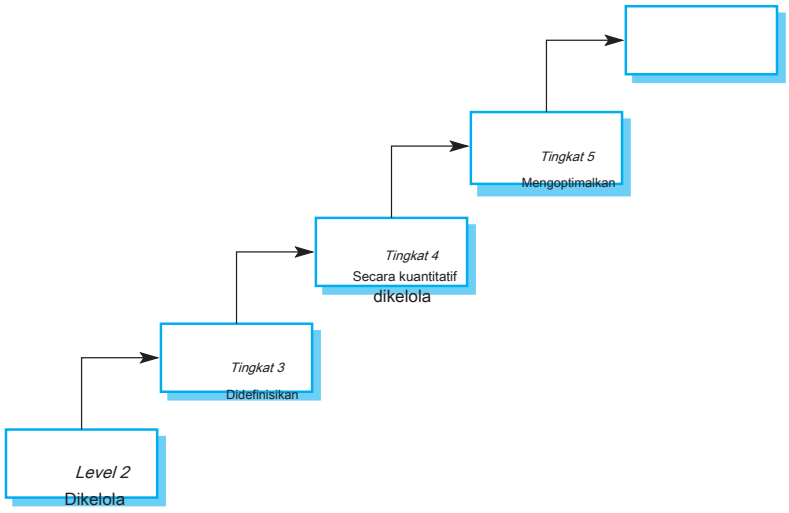
Gagasan tentang kematangan proses diperkenalkan pada akhir 1980-an ketika Software Engineering Institute (SEI) mengusulkan model kematangan kemampuan proses mereka (Humphrey 1988). Kematangan proses perusahaan perangkat lunak mencerminkan manajemen proses, pengukuran, dan penggunaan praktik rekayasa perangkat lunak yang baik di perusahaan. Ide ini diperkenalkan agar Departemen Pertahanan AS dapat menilai kapabilitas rekayasa perangkat lunak kontraktor pertahanan, dengan maksud untuk membatasi kontrak bagi kontraktor yang telah mencapai tingkat kematangan proses yang disyaratkan. Lima tingkat kematangan proses diusulkan. seperti yang ditunjukkan pada Gambar 2.12. Ini telah berevolusi dan berkembang selama 25 tahun terakhir (Chrissis, Konrad, dan Shrum 2011), tetapi ide fundamental dalam model Humphrey masih menjadi dasar penilaian kematangan proses perangkat lunak.

Tingkatan dalam model kematangan proses adalah:

1. *Awal* Sasaran yang terkait dengan area proses terpenuhi, dan untuk semua proses ruang lingkup pekerjaan yang akan dilakukan ditetapkan secara eksplisit dan dikomunikasikan kepada anggota tim.

2. *Dikelola* Pada tingkat ini, tujuan yang terkait dengan area proses yang ditetapkan, dan kebijakan organisasi yang menentukan kapan setiap proses harus digunakan. Harus ada rencana proyek yang terdokumentasi yang menentukan tujuan proyek. Manajemen sumber daya dan prosedur pemantauan proses harus ada di seluruh lembaga.

3. *Didefinisikan* Tingkat ini berfokus pada standarisasi organisasi dan penerapan proses. Setiap proyek memiliki proses yang dikelola yang disesuaikan dengan persyaratan proyek dari serangkaian proses organisasi yang ditentukan. Aset proses dan pengukuran proses harus dikumpulkan dan digunakan untuk perbaikan proses di



Gambar 2.12 Kemampuan tingkat kematangan

4. *Dikelola secara kuantitatif* Pada tingkat ini, ada tanggung jawab organisasi untuk menggunakan statistik dan metode kuantitatif lainnya untuk mengontrol subproses. Artinya, proses yang dikumpulkan dan pengukuran produk harus digunakan dalam manajemen proses.

5. *Mengoptimalkan* Pada tingkat tertinggi ini, organisasi harus menggunakan proses dan pengukuran produk untuk mendorong perbaikan proses. Tren harus dianalisis dan prosesnya disesuaikan dengan kebutuhan bisnis yang berubah.

Pekerjaan pada tingkat kematangan proses memiliki dampak besar pada industri perangkat lunak. Ini memusatkan perhatian pada proses dan praktik rekayasa perangkat lunak yang digunakan dan menyebabkan peningkatan signifikan dalam kemampuan rekayasa perangkat lunak. Namun, ada terlalu banyak biaya tambahan dalam perbaikan proses formal untuk perusahaan kecil, dan sulit untuk memperkirakan kematangan dengan proses yang gesit. Akibatnya, hanya perusahaan perangkat lunak besar yang sekarang menggunakan pendekatan yang berfokus pada kematangan ini untuk peningkatan proses perangkat lunak.

Poin-poin penting

- Proses perangkat lunak adalah aktivitas yang terlibat dalam menghasilkan sistem perangkat lunak. Model proses perangkat lunak adalah representasi abstrak dari proses ini.
- Model proses umum menggambarkan organisasi proses perangkat lunak. Contoh model umum ini mencakup model air terjun, pengembangan tambahan, serta konfigurasi dan integrasi komponen yang dapat digunakan kembali.

- Rekayasa persyaratan adalah proses mengembangkan spesifikasi perangkat lunak. Spesifikasi dimaksudkan untuk mengkomunikasikan kebutuhan sistem dari pelanggan kepada pengembang sistem.
- Proses desain dan implementasi berkaitan dengan mengubah spesifikasi persyaratan menjadi sistem perangkat lunak yang dapat dieksekusi.
- Validasi perangkat lunak adalah proses memeriksa apakah sistem sesuai dengan spesifikasinya dan memenuhi kebutuhan nyata pengguna sistem.
- Evolusi perangkat lunak terjadi ketika Anda mengubah sistem perangkat lunak yang ada untuk memenuhi persyaratan baru. Perubahan berkelanjutan, dan perangkat lunak harus berkembang agar tetap berguna.
- Proses harus mencakup aktivitas untuk mengatasi perubahan. Ini mungkin melibatkan fase pembuatan prototipe yang membantu menghindari keputusan yang buruk tentang persyaratan dan desain. Proses dapat disusun untuk pengembangan dan pengiriman berulang sehingga perubahan dapat dilakukan tanpa mengganggu sistem secara keseluruhan.
- Perbaikan proses adalah proses memperbaiki proses perangkat lunak yang ada untuk meningkatkan kualitas perangkat lunak, menurunkan biaya pengembangan, atau mengurangi waktu pengembangan. Ini adalah proses siklik yang melibatkan pengukuran, analisis, dan perubahan proses.

Bacaan lebih lanjut

"Model Proses dalam Rekayasa Perangkat Lunak". Ini adalah gambaran umum yang sangat baik dari berbagai model proses rekayasa perangkat lunak yang telah diusulkan. (W. Scacchi, *Ensiklopedia Rekayasa Perangkat Lunak,* ed. JJ Marciniak, John Wiley & Sons, 2001) http://www.ics.uci.edu/~wscacchi/ Makalah / SE-Encyc / Process-Models-SE-Encyc.pdf

Perbaikan Proses Perangkat Lunak: Hasil dan Pengalaman dari Lapangan. Buku ini adalah kumpulan makalah yang berfokus pada studi kasus perbaikan proses di beberapa perusahaan kecil dan menengah Norwegia. Ini juga mencakup pengantar yang baik untuk masalah umum perbaikan proses. (Conradi, R., Dybå, T., Sjøberg, D., dan Ulsund, T. (eds.), Springer, 2006).

"Model dan Metodologi Siklus Hidup Pengembangan Perangkat Lunak". Posting blog ini adalah ringkasan singkat dari beberapa model proses perangkat lunak yang telah diusulkan dan digunakan. Dibahas tentang kelebihan dan kekurangan masing-masing model tersebut (M. Sami, 2012). http: //melsatar.wordpress. com / 2012/03/15 / software-development-life-cycle-model-and-methodologies /

Situs web

Slide PowerPoint untuk bab ini:

www.pearsonglobaleditions.com/Sommerville

Tautan ke video pendukung:

http://software-engineering-book.com/videos/software-engineering/

Exercises

2.1. Sarankan model proses perangkat lunak generik yang paling sesuai yang dapat digunakan sebagai dasar untuk mengelola pengembangan sistem berikut. Jelaskan jawaban Anda sesuai dengan jenis sistem yang dikembangkan:

Sebuah sistem untuk mengontrol pengereman antilock di dalam mobil

Sistem realitas virtual untuk mendukung pemeliharaan perangkat lunak

Sistem akuntansi universitas yang menggantikan sistem yang ada

Sistem perencanaan perjalanan interaktif yang membantu pengguna merencanakan perjalanan dengan dampak lingkungan terendah

2.2. Pengembangan perangkat lunak tambahan bisa sangat efektif digunakan untuk pelanggan yang tidak melakukannya memiliki gagasan yang jelas tentang sistem yang dibutuhkan untuk operasi mereka. Bahas.

2.3. Pertimbangkan model proses integrasi dan konfigurasi yang ditunjukkan pada Gambar 2.3. Jelaskan kenapa penting untuk mengulangi aktivitas rekayasa persyaratan dalam prosesnya.

2.4. Sarankan mengapa penting untuk membuat perbedaan antara mengembangkan persyaratan pengguna dan mengembangkan persyaratan sistem dalam proses rekayasa persyaratan.

2.5. Dengan menggunakan contoh, jelaskan mengapa kegiatan desain desain arsitektur, desain database, desain antarmuka, dan desain komponen saling bergantung.

2.6. Jelaskan mengapa pengujian perangkat lunak harus selalu menjadi aktivitas bertahap dan bertahap. Apakah program- menjadi orang terbaik untuk menguji program yang telah mereka kembangkan?

2.7. Bayangkan bahwa pemerintah menginginkan program perangkat lunak yang membantu melacak penggunaan- dari sumber daya mineral negara yang sangat besar. Meskipun persyaratan yang diajukan oleh pemerintah tidak terlalu jelas, sebuah perusahaan perangkat lunak ditugaskan untuk mengembangkan prototipe. Pemerintah menganggap prototipe itu mengesankan, dan meminta agar diperpanjang menjadi sistem aktual yang akan digunakan. Diskusikan pro dan kontra dari pendekatan ini.

2.8. Anda telah mengembangkan prototipe sistem perangkat lunak dan manajer Anda sangat terkesan Itu. Dia mengusulkan bahwa itu harus digunakan sebagai sistem produksi, dengan fitur baru ditambahkan sesuai kebutuhan. Ini menghindari biaya pengembangan sistem dan membuat sistem segera berguna. Tulis laporan singkat untuk manajer Anda yang menjelaskan mengapa sistem prototipe biasanya tidak digunakan sebagai sistem produksi.

2.9. Sarankan dua keuntungan dan dua kerugian dari pendekatan penilaian proses dan peningkatan yang diwujudkan dalam kerangka Kematangan Kemampuan SEI.

2.10. Secara historis, pengenalan teknologi telah menyebabkan perubahan besar di pasar tenaga kerja dan, setidaknya untuk sementara, orang-orang yang mengungsi dari pekerjaan. Diskusikan apakah pengenalan otomatisasi proses yang ekstensif kemungkinan besar memiliki konsekuensi yang sama untuk insinyur perangkat lunak. Jika menurut Anda itu tidak akan terjadi, jelaskan mengapa tidak. Jika menurut Anda hal itu akan mengurangi peluang kerja, apakah etis bagi para insinyur yang terpengaruh untuk secara pasif atau aktif menolak pengenalan teknologi ini?