Pertemuan 4:
Recommended Process Model

Meuthia Rachmaniah
Departemen Ilmu Komputer, FMIPA IPB

Software Engineering: A Practitioner's Approach, 9th Ed.
Roger S. Pressman dan Bruce R. Maxim

4. Recommended Process Model

4.1 Requirements Definition
4.2 Preliminary Architectural Design
4.3 Resource Estimation
4.4 First Prototype Construction
4.5 Prototype Evaluation
4.6 Go, No Go Decision
4.7 Prototype Evolution
4.8 Prototype Release
4.9 Maintain Release Software

Rekomendasi untuk Modern Software Development Project akibat kelemahan dari pendekatan Prescriptive Software Life Cycle

1) Model proses linier akan beresiko digunakan jika tidak ada umpan balik yang cukup
2) Merencanakan pengumpulan persyaratan besar di awal tidak pernah mungkin atau diinginkan
3) Pengumpulan persyaratan di awal mungkin tidak mengurangi biaya atau mencegah selip waktu
4) Manajemen proyek yang tepat merupakan bagian integral dari pengembangan perangkat lunak
5) Dokumen harus berkembang seiringan dengan PL dan tidak boleh memperlambat dimulainya konstruksi
6) Libatkan stakeholder lebih awal dan lebih sering dalam proses pembangunan
7) Penguji (tester) harus terlibat dalam proses sebelum konstruksi perangkat lunak dimulai

Lihat kembali Pros dan Cons pada Rangkuman Pertemuan 2

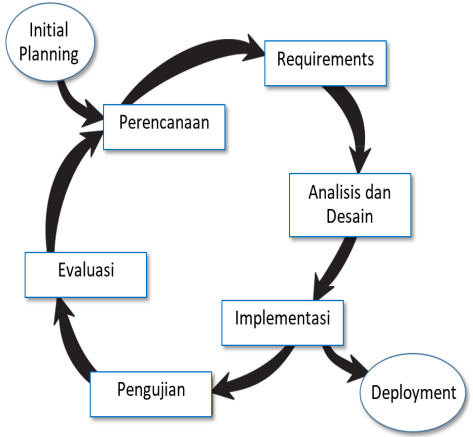
Limitasi Waterfall → Model Incremental

Prescriptive Process Model (Model Waterfal)

- Tidak dapat menerima perubahan yang mungkin perlu diperkenalkan setelah pengembang mulai membuat koding
- Umpan balik stakeholder terbatas pada awal dan akhir proyek
- Semua analisis dan desain work products diselesaikan sebelum pemrograman atau pengujian terjadi
- Hal ini membuat sulit untuk beradaptasi dengan proyek dengan persyaratan yang terus berkembang

Incremental Model (Model Prototyping, Scrum)

- Melibatkan pelanggan sejak awal sehingga mengurangi risiko produk tidak diterima pelanggan
- Godaan: stakeholder meminta banyak perubahan saat stakeholder melihat setiap protitipe dan menyadari adanya fungsi dan fitur yang ternyata diperlukan.
- Pengembang tidak merencanakan evolusi prototype dan membuat Throwaway Prototype
- Jika perubahan ditangani dengan baik, maka model incremental bagus diterapkan untuk proses yang adaptable



Model Incremental untuk Prototype Development

Model Proses Agile dan Kanban Board

Model Proses Agile (Scrum, XP)

Sangat bagus untuk mengakomodasi ketidakpastian pengetahuan (*knowledge*) dari keinginan dan problem stakeholder

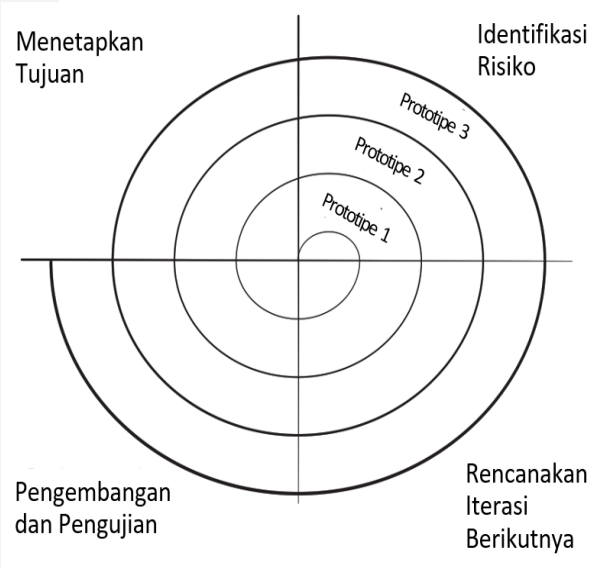
Karakteristik Utama Model Proses Agile:

- Prototipe yang diciptakan sudah didesain untuk diperluas untuk increment software ke depan
- Stakeholder terlibat di seluruh proses pengambangan
- Keperluan dokumentasi ringan; dokumentasi harus berkembang seiring dengan perkembangan PL
- Pengujian direncanakan dan dieksekusi di awal

Scrum dan Kanban

- Scrum dianggap terlalu banyak memerlukan pertemuan (*meetings*), terutama Daily Meeting
- Kanban menyediakan sistem pelacakan ringan untuk mengelola status dan prioritas dari user stories
- Scrum dan Kanban memungkinkan untuk mengendalikan pengenalan requirement baru (user story baru)
- Secara desain, tim Scrum berjumlah kecil sehingga tidak cocok untuk developers berjumlah besar; terkecuali proyek dipartisi menjadi berukuran kecil dan menangani component secara independen

Model Spiral untuk Evolusioner Prototype



Model Spiraluntuk Prototype Development

- Disebut evolusioner karena ada elemen asesmen risiko
- Bergantung pada keterlibatan stakeholder yang moderat
- Didesain untuk tim besar dan proyek besar
- Tujuan: Membuat prototype yang diperluas setiap kali proses diiterasi.
- Testing di awal adalah sangat penting
- Dokumentasi berkembang seiiring dengan diciptakannya prototype baru
- Bersifat unik karena asesmen risiko formal dibangun dan digunakan sebagai dasar untuk menentukan investasi sumberdaya untuk prototype berikutnya
- Dapat sulit dilakukan karena cakupan proyek mungkin tidak diketahui di awal proyek
- Model spiral bagus digunakan untuk adaptable process model

Rangkuman Karakteristik Model Agile - SCRUM

Mudah dikelola

Sangat bergantung pada interaksi stakeholder

Pengiriman cepat dari solusi parsial

Kemudahan mengakomodasi perubahan produk

Manajemen risiko informal

Tester terlibat secara kontinyu

Peningkatan Process berkelanjutan yang sudah built-in

Aturan minimal dan dokumentasi minimal

Tidak cocok untuk proyek besar risiko tinggi dan *mission critical*

Rangkuman Karakteristik Model Agile - SPIRAL

Memerlukan manajemen dan koordinasi proyek secara formal

Keterlibatan stakeholder yang kontinyu dalam perencanaan dan asesmen risiko

Akhir proyek tidak selalu jelas terlihat

Sulit untuk mengakomodasi perubahan produk sebelum satu iterasi prototype diselesaikan

Manajemen risiko baik

Tester terlibat di awal (tester bisa di luar tim pengembang

Peningkatan Process dilakukan di akhir proyek

Memerlukan beberapa tahapan dengan dokumentasi diselesaikan di awal

Tidak cocok untuk proyek kecil dan tidak cocok untuk proyek berisiko kecil

Diperlukan orang yang kreatif dan berpengetahuan luas melakukan RPL

Mereka mengadaptasi proses PL agar sesuai dengan produk yang mereka bangun dan memenuhi permintaan marketplace

Itulah mengapa penting bagi pengembang untuk dapat mengadaptasi proses mereka dengan cepat dan praktis untuk mengakomodasi pengetahuan baru ini

Pendekatan seperti spiral yang memiliki agility yang dibangun di setiap siklus adalah tempat yang baik untuk memulai banyak proyek PL

Pengembang mempelajari banyak hal saat mereka melanjutkan proses pengembangan

4.1 Requirements Definition

4.1 Requirements Definition

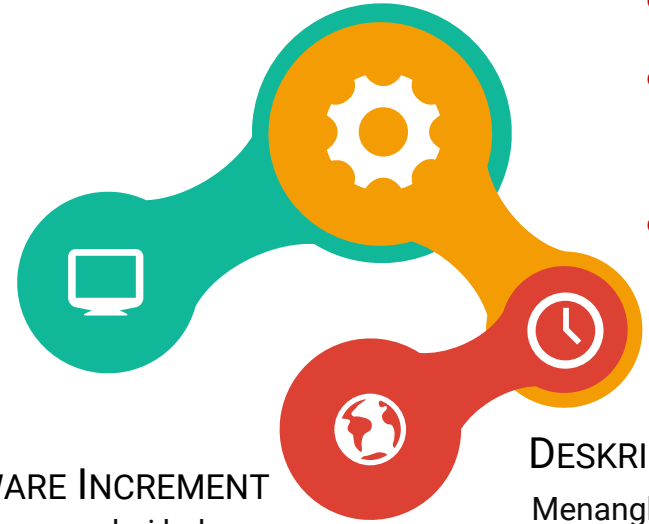
- *REQUIREMENTS ENGINEERING* (Rekayasa Persyaratan)
  - Setiap proyek PL dimulai dengan tim yang mencoba memahami masalah yang harus dipecahkan dan menentukan hasil apa yang penting bagi stakeolder
  - Hal ini termasuk memahami kebutuhan bisnis yang memotivasi proyek dan masalah teknis yang menghambatnya.
- Tim yang gagal menghabiskan cukup waktu untuk tugas ini dan mendapati bahwa proyek mereka mengandung pengerjaan ulang yang mahal, pembengkakan biaya, kualitas produk yang buruk, waktu pengiriman yang terlambat, pelanggan yang tidak puas, dan moral tim yang buruk

Best Practices untuk definisi Agile Requirements

1) Dorong partisipasi aktif stakeholder dengan mencocokkan jadwal mereka dan menghargai masukan mereka
2) Gunakan model sederhana (mis., Post-it note, sketsa cepat, user story) untuk mengurangi hambatan partisipasi
3) Luangkan waktu untuk menjelaskan teknik representasi kebutuhan Anda sebelum menggunakannya
4) Mengadopsi terminologi stakeholder, dan menghindari jargon teknis bila memungkinkan
5) Gunakan pendekatan luas-pertama untuk mendapatkan gambaran besar dari proyek yang dilakukan sebelum terjebak dalam detail
6) Izinkan tim pengembang untuk menyempurnakan (dengan masukan stakeholder) detail persyaratan “Just in Time" ketika user story dijadwalkan untuk diimplementasikan
7) Perlakukan daftar fitur yang akan diimplementasikan dalam daftar priotitas, dan terapkan user story yang paling penting terlebih dahulu
8) Berkolaborasi erat dengan stakeholder dan hanya dokumentasikan persyaratan pada tingkat yang berguna bagi semua orang saat membuat prototipe berikutnya
9) Mempertanyakan perlunya memelihara model dan dokumen yang tidak akan dirujuk di masa mendatang
10) Pastikan Anda memiliki dukungan manajemen untuk memastikan ketersediaan stakeholder dan sumber daya ketika dilakukan definisi persyaratan

Kenali 2 Realitas

1) Stakeholder tidak mungkin menggambarkan keseluruhan sistem sebelum melihat PL yang berfungsi
2) Sulit bagi pemangku kepentingan untuk menjelaskan persyaratan kualitas yang diperlukan untuk PL sebelum melihatnya beraksi



Definisikan Acceptance Criteria

- Definisikan acceptance criteria dari setiap user story
- Stakeholder perlu melihat user story yang dikoding dan dijalankan untuk mengetahui apakah itu telah diterapkan dengan benar atau tidak
- Oleh karena itu, definisi persyaratan perlu dilakukan secara iteratif dan mencakup pengembangan prototipe untuk ditinjauan stakeholder

Requirements vs Software Increment

Pengembang harus menyadari bahwa persyaratan akan ditambahkan dan disempurnakan saat increment PL dibuat

Deskripsi Stakeholer

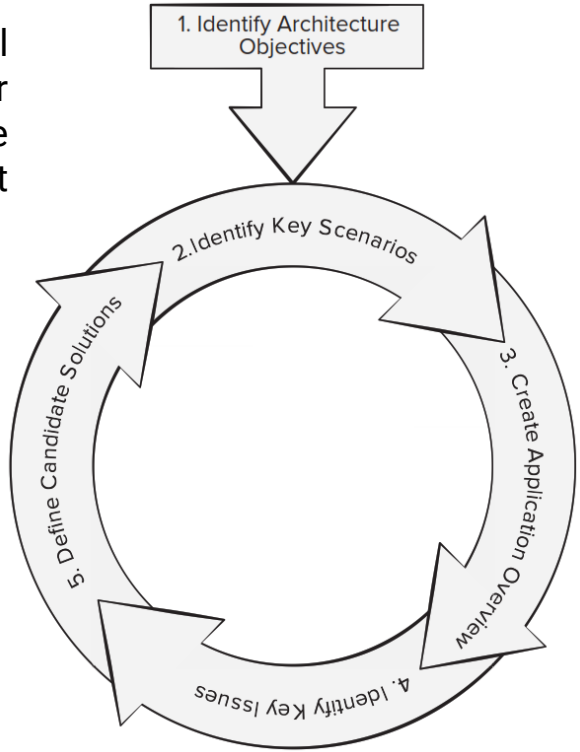
Menangkap deskripsi stakeholder tentang apa yang perlu dilakukan sistem dengan kata-kata stakeholder sendiri dalam user story adalah cara yang baik untuk memulai

- Prototipe adalah realisasi nyata dari rencana proyek yang dapat dengan mudah dirujuk oleh stakeholder ketika mencoba untuk menggambarkan perubahan yang diinginkan
- Stakeholder termotivasi untuk membahas perubahan persyaratan dalam istilah yang lebih konkret, yang meningkatkan komunikasi.
- Penting untuk diketahui bahwa prototipe memungkinkan pengembang untuk fokus pada tujuan jangka pendek dengan hanya berfokus pada perilaku pengguna yang terlihat.
- Penting untuk meninjau prototipe dengan memperhatikan kualitasnya
- Pengembang perlu menyadari bahwa menggunakan prototipe dapat meningkatkan volatilitas persyaratan jika stakeholder tidak fokus untuk mendapatkan hal yang benar pertama kali
- Ada juga risiko bahwa membuat prototipe sebelum persyaratan arsitektur perangkat lunak dipahami dengan baik dapat mengakibatkan prototipe harus dibuang, membuang-buang waktu dan sumber daya.

4.2 Preliminary Architectural Design

4.2 Preliminary Architectural Design

Architectural design for prototype development



- Keputusan desain awal harus sering dibuat saat persyaratan didefinisikan
- keputusan arsitektur perlu dialokasikan untuk peningkatan produk (sesuai gambar di kiri)
  1) Identifikasi tujuan arsitektur
  2) Identifikasi Skenarion Kunci
  3) Ciptakan overview aplikasi
  4) Identifikasi Isyu-isyu kunci
  5) Definisikan kandidat solusi
- Pemahaman awal tentang persyaratan dan pilihan arsitektur adalah kunci untuk mengelola pengembangan produk PL yang besar atau kompleks
- Persyaratan dapat digunakan untuk membuat desain arsitektur
- Eksplorasi arsitektur ketika prototipe dikembangkan akan memfasilitasi proses untuk merinci persyaratan
- Yang terbaik adalah melakukan kegiatan ini secara bersamaan untuk mencapai keseimbangan yang tepat

Empat Elemen Kunci untuk Desain Arsitektural Agile

1. Atribut Berkualitas

Fokus pada atribut kualitas utama, dan gabungkan ke dalam prototipe saat dibangun

2. Kombinasi Fitur-Infrastruktur

Saat merencanakan prototipe, ingatlah bahwa produk PL yang sukses menggabungkan fitur yang terlihat oleh pelanggan dan infrastruktur untuk mengaktifkannya

3. Arsitektur Agile

Ketahuilah bahwa arsitektur agile memungkinkan pemeliharaan koding dan kemampuan evolusi jika perhatian yang cukup diberikan pada keputusan arsitektur dan masalah kualitas terkait

4. Kelola & Sinkronisasi

Mengelola dan menyinkronkan dependensi secara berkelanjutan antara persyaratan fungsional dan arsitektur diperlukan untuk memastikan fondasi arsitektur yang berkembang akan siap tepat pada waktunya untuk peningkatan di masa mendatang

4.3 Resource Estimation

4.3 Resource Estimation

Estimasi Waktu

Salah satu aspek yang lebih kontroversial menggunakan spiral atau prototyping tangkas adalah memperkirakan waktu yang dibutuhkan untuk menyelesaikan sebuah proyek ketika tidak dapat didefinisikan sepenuhnya.

Dimulai Kalau Sudah Paham

Penting untuk dipahami sebelum Anda mulai apakah Anda memiliki peluang yang wajar untuk mengirimkan produk PL tepat waktu dan dengan biaya yang dapat diterima sebelum Anda setuju untuk mengambil proyek

Estimasi Awal Sulit

Perkiraan awal berisiko salah karena ruang lingkup proyek tidak didefinisikan dengan baik dan kemungkinan akan berubah setelah pengembangan dimulai

Estimasi di Akhir Tidak OK

Perkiraan yang dibuat ketika proyek hampir selesai tidak memberikan panduan manajemen proyek apa pun

Trik Estimasi

Triknya adalah memperkirakan waktu pengembangan perangkat lunak lebih awal berdasarkan apa yang diketahui pada saat itu dan merevisi perkiraan Anda secara teratur saat persyaratan ditambahkan atau setelah peningkatan perangkat lunak dikirimkan

Estimasi Manajer Proyek PL untuk Model Agile Spiral

- £ Dipengaruhi Jumlah Pengembang & Jumlah User Story
- £ Menggandakan jumlah pengembang hampir tidak pernah memotong waktu pengembangan menjadi setengahnya
- £ Untuk mendapatkan perkiraan yang lebih akurat, penting juga untuk mengetahui jenis proyek dan pengalaman tim

1. **Data Historis**
   Gunakan data historis, dan bekerja sebagai tim untuk mengembangkan perkiraan berapa hari yang diperlukan untuk menyelesaikan setiap cerita pengguna yang diketahui di awal proyek

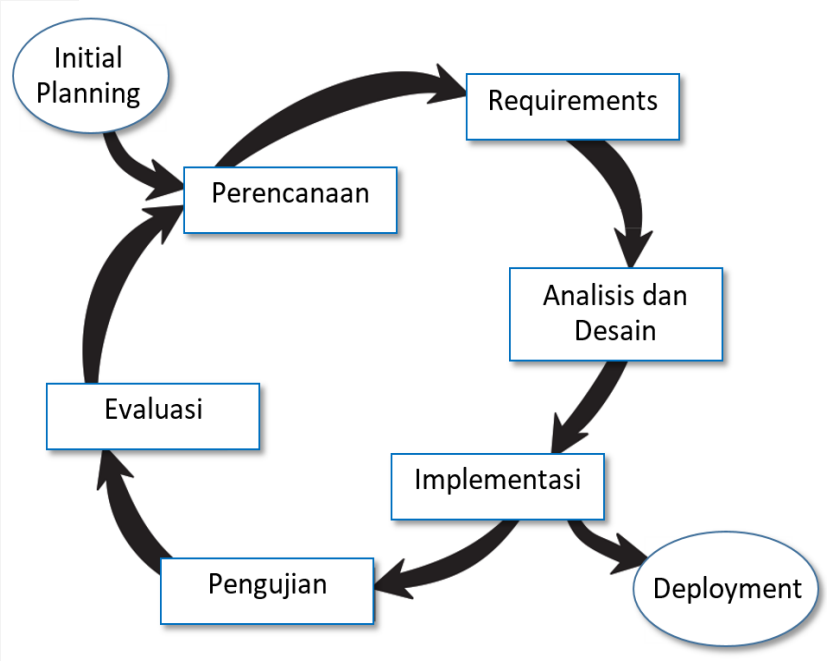
2. **Estimasi setiap Sprint**
   Atur user story secara longgar ke dalam set yang akan membentuk setiap sprint 1 yang direncanakan untuk menyelesaikan prototipe

3. **Jumlahkan Estimasi Seluruh Sprint**
   Jumlahkan jumlah hari untuk menyelesaikan setiap sprint untuk memberikan perkiraan durasi total proyek

4. **Revisi Estimasi disetujui Stakeholder**
   Merevisi perkiraan saat persyaratan ditambahkan ke proyek atau prototipe dikirimkan dan diterima oleh stakeholder

4.4 First Prototype Construction

4.4 First Prototype Construction



Model Incremental untuk Prototype Development

- Pengembang dapat menggunakan prototipe pertama untuk membuktikan bahwa desain arsitektur awal mereka adalah pendekatan yang layak untuk memberikan fungsionalitas yang diperlukan sambil memenuhi batasan kinerja pelanggan
- Untuk membuat prototipe operasional maka rekayasa persyaratan (requirements engineering), desain perangkat lunak, dan konstruksi semua berjalan secara paralel. Proses ini ditunjukkan pada gambar model incremental untuk pengembangan prototipe

Tahapan untuk Membuat Prototipe Pertama

1

Identifikasi fitur dan fungsi yang paling penting bagi stakeholder

- Ini akan membantu menentukan tujuan untuk prototipe pertama.
- Jika stakeholder dan pengembang telah membuat daftar user story yang diprioritaskan, akan lebih mudah untuk mengonfirmasi mana yang paling penting

2

Putuskan berapa banyak waktu yang akan diizinkan untuk membuat prototipe pertama

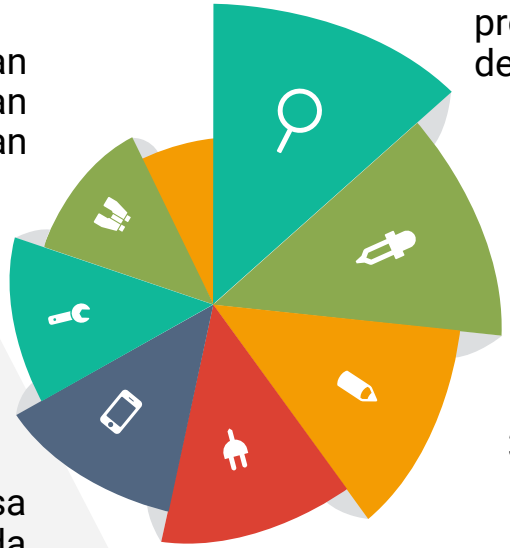
- Tim dapat memilih waktu yang tetap, mis. sprint 4-minggu, untuk mengirimkan setiap prototype
- Pengembang akan melihat perkiraan waktu dan sumber daya dan menentukan user story prioritas tinggi mana yang dapat diselesaikan dalam 4 minggu
- Tim kemudian akan mengkonfirmasi dengan para stakeholder bahwa user story yang dipilih adalah yang terbaik untuk dimasukkan dalam prototipe pertama

Pendekatan Alternatif

- Meminta para stakeholder dan pengembang bersama-sama memilih sejumlah kecil user story berprioritas tinggi untuk dimasukkan dalam prototipe pertama
- Menggunakan perkiraan waktu dan sumber daya untuk mengembangkan jadwal untuk menyelesaikan prototipe pertama

Paper Prototype untuk Membuat Prototipe Pertama

1. Transisi dari prototipe kertas ke desain PL
2. Prototipe antarmuka pengguna (user interface)
3. Buat prototipe virtual
4. Tambahkan input dan output ke prototipe Anda
5. Rekayasa algoritme Anda
6. Uji prototipe Anda
7. Prototipe dengan mempertimbangkan penerapan



- Tahap 1-7 dapat diterapkan untuk berbagai proyek PL
- Prototipe kertas untuk suatu sistem sangat murah dan dapat dilakukan di awal proses pengembangan
- Pengguna nonteknis sering dapat mengenali apa yang mereka suka atau tidak suka tentang user interface dengan sangat cepat setelah melihat sketsanya
- Komunikasi antar manusia seringkali dipenuhi dengan kesalahpahaman. Orang lupa memberi tahu satu sama lain apa yang benar-benar perlu mereka ketahui atau menganggap bahwa setiap orang memiliki pemahaman yang sama.

Menambahkan Input dan Output

- Penambahan I/O ke prototipe user interface menyediakan cara mudah untuk mulai menguji prototipe yang berkembang
- Pengujian antarmuka komponen PL harus diselesaikan sebelum menguji kode yang membentuk algoritme komponen.
- Untuk menguji algoritme itu sendiri, pengembang sering menggunakan "kerangka uji" untuk memastikan algoritme diimplementasikan berfungsi sebagaimana dimaksud.

Prototipe User Interface

- Membuat prototipe user interface sebagai bagian dari prototipe fungsional pertama adalah ide yang bijaksana
- Sistem berbasis web atau aplikasi mobile sangat bergantung pada user interface
- Membantu meningkatkan user experience

Rekayasa Algoritme

- Ialah proses mengubah ide dan sketsa menjadi kode bahasa pemrograman
- Harus mempertimbangkan kebutuhan fungsional pada user story dan kendala kinerja (eksplisit dan implisit)
- Ini adalah titik di mana fungsionalitas dukungan tambahan kemungkinan akan diidentifikasi dan ditambahkan ke lingkup proyek, jika belum ada di library code

Pengujian Prototipe

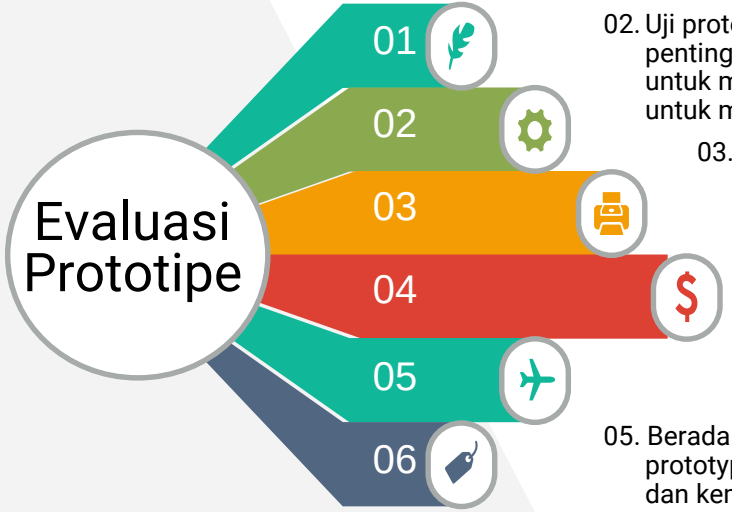
- Pengujian prototipe menunjukkan fungsionalitas yang diperlukan dan untuk mengidentifikasi cacat yang belum ditemukan sebelum menunjukkannya kepada pelanggan.
- Adalah bijaksana untuk melibatkan pelanggan dalam proses pengujian sebelum prototipe selesai untuk menghindari pengembangan fungsi yang salah
- Waktu terbaik untuk membuat kasus uji adalah selama *requirements gathering* atau ketika kasus penggunaan telah dipilih untuk implementasi

Prototipe untuk Deployment

- Membantu untuk menghindari mengambil jalan pintas yang mengarah pada pembuatan PL yang akan
- Hal ini tidak berarti bahwa setiap baris kode akan sampai ke produk PL akhir. Seperti banyak tugas kreatif, pengembangan prototipe bersifat iteratif. Draf dan revisi diharapkan

4.5 Prototype Evaluation

4.5 Evaluasi Prototype



01. Berikan perancah (*scaffolding*), mekanisme untuk umpan balik secara tidak konfrontasi dengan statement: *I like* (mendorong umpan balik positif), *I wish* (untuk berbagi ide tentang bagaimana prototipe dapat ditingkatkan/feedback negatif), *What if* (untuk eksplorasi saat membuat prototipe di iterasi mendatang)

02. Uji prototipe Anda pada orang yang tepat untuk mengevaluasi prototipe sangat penting untuk mengurangi risiko pengembangan produk yang salah. Sangat penting untuk memiliki campuran pengguna yang tepat (mis., pemula, tipikal, dan lanjutan) untuk memberi Anda umpan balik tentang prototipe

03. Ajukan pertanyaan yang tepat: menyiratkan bahwa semua stakeholder menyetujui tujuan prototype. Umpan balik mendorong proses pembuatan prototipe saat merencanakan aktivitas pengembangan produk di masa mendatang. Cobalah untuk mengajukan pertanyaan spesifik tentang fitur baru apa pun yang disertakan dalam prototipe

04. Bersikaplah netral saat menyajikan alternatif kepada pengguna. Pemrograman tanpa ego adalah filosofi pengembangan yang berfokus pada menghasilkan produk terbaik yang dapat dibuat tim untuk pengguna yang dituju. Hal-hal yang tidak berfungsi perlu diperbaiki atau dibuang

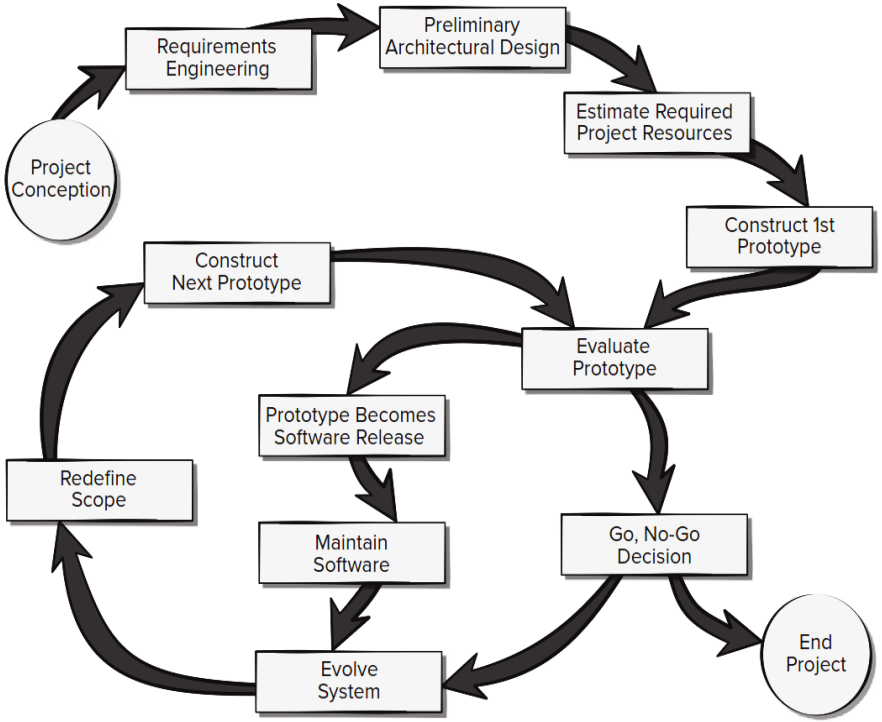
05. Beradaptasi saat menguji berarti diperlukan pola pikir yang fleksibel saat pengguna bekerja dengan prototype, artinya: mengubah rencana pengujian atau membuat perubahan cepat pada prototipe dan kemudian memulai kembali pengujian; untuk mendapatkan umpan balik yang dibutuhkan untuk membantu memutuskan apakah akan membangun prototipe berikutnya atau tidak

06. Ijinkan pengguna untuk menyumbangkan ide. Pastikan Anda memiliki cara untuk merekam saran dan pertanyaan mereka (secara elektronik atau lainnya)

4.6 Go, No Go Decision

4.6 Go, No Go Decision

- Keputusan yang sedikit berbeda berdasarkan evaluasi prototipe mungkin untuk merilis prototipe ke pengguna akhir dan memulai proses pemeliharaan
- Melewati wilayah perencanaan mengikuti proses evaluasi. Perkiraan biaya yang direvisi dan perubahan jadwal diusulkan berdasarkan apa yang ditemukan saat mengevaluasi prototipe perangkat lunak saat ini
- Tujuan dari proses asesmen risiko adalah untuk mendapatkan komitmen dari semua pemangku kepentingan dan manajemen perusahaan untuk menyediakan sumber daya yang dibutuhkan untuk membuat prototipe berikutnya.



Recommended software process model

4.7 Prototype Evolution

4.7 Prototype Evolution

- Setelah prototipe dikembangkan dan ditinjau oleh tim pengembangan dan stakeholder lainnya, saatnya untuk mempertimbangkan pengembangan prototipe berikutnya.
- Langkah pertama adalah mengumpulkan semua umpan balik dan data dari evaluasi prototipe saat ini.
- Pengembang dan stakeholder kemudian memulai negosiasi untuk merencanakan pembuatan prototipe lain.

1. New Prototype Scope

- Proses penentuan ruang lingkup prototipe baru seperti proses penentuan ruang lingkup prototipe awal.
- Pengembang akan:
  1) Memilih fitur untuk dikembangkan dalam waktu yang dialokasikan untuk sprint, atau
  2) Mengalokasikan waktu yang cukup untuk mengimplementasikan fitur yang dibutuhkan untuk memenuhi tujuan yang ditetapkan oleh pengembang dengan masukan stakeholder
- Pendekatan mana pun mengharuskan pengembang untuk mempertahankan daftar fitur atau user story yang diprioritaskan.

2. Constructing New Prototype

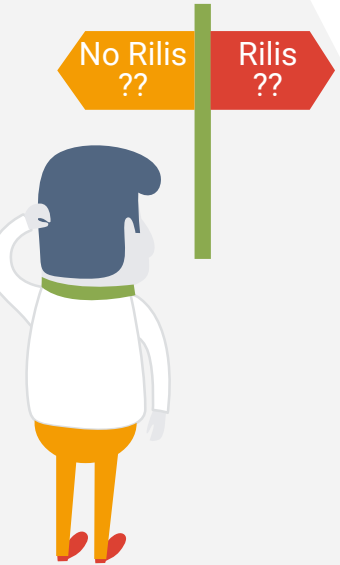
- Sebuah user story harus berisi deskripsi tentang bagaimana pelanggan berencana untuk berinteraksi dengan sistem untuk mencapai tujuan tertentu dan deskripsi tentang apa definisi penerimaan (user acceptance) pelanggan
- Tugas tim pengembangan adalah membuat komponen PL tambahan untuk mengimplementasikan user story yang dipilih untuk dimasukkan dalam prototipe baru bersama dengan kasus uji yang diperlukan
- Pengembang perlu melanjutkan komunikasi dengan semua stakeholder saat mereka membuat prototipe baru

3. Testing New Prototype

- Pengujian prototipe baru harus relatif mudah jika tim pengembangan membuat kasus uji sebagai bagian dari proses desain sebelum pemrograman selesai
- Setiap user story harus memiliki kriteria penerimaan yang melekat padanya saat dibuat.
- Acceptance statement harus memandu pembuatan kasus uji yang dimaksudkan untuk membantu memverifikasi bahwa prototipe memenuhi kebutuhan pelanggan.
- Prototipe perlu diuji terhadap cacat dan masalah kinerja

4.8 Prototype Release

4.8 Prototype Release

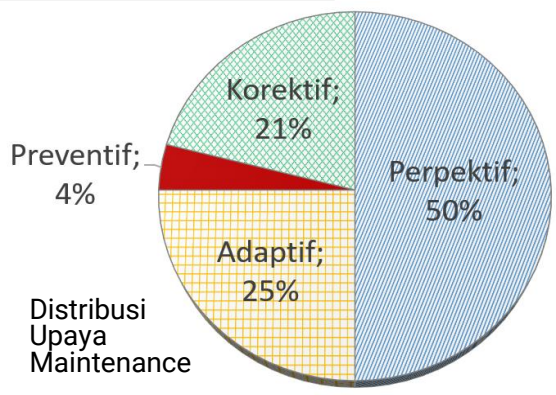


- Prototipe yang dipertimbangkan sebagai kandidat rilis harus menjalani pengujian penerimaan pengguna (acceptance testing) selain pengujian fungsional dan nonfungsional (kinerja) yang akan dilakukan selama konstruksi prototipe
- Tes penerimaan pengguna (user acceptance) didasarkan pada kriteria penerimaan (acceptance criteria) yang disepakati yang dicatat saat setiap user story dibuat dan ditambahkan ke backlog produk
- Ketika menguji kandidat rilis, tes fungsional dan nonfungsional harus diulang dengan kasus uji yang dikembangkan selama fase konstruksi prototipe incremental:
  - Tolok ukur (benchmark) kinerja tipikal berhubungan dengan waktu respons sistem, kapasitas data, atau kegunaan (usability)
  - Persyaratan nonfungsional diverifikasi untuk memastikan bahwa kandidat rilis akan berjalan di semua lingkungan run-time yang direncanakan dan di semua perangkat yang ditargetkan
  - Proses harus difokuskan pada pengujian terbatas pada kriteria penerimaan yang telah ditetapkan sebelum prototipe dibuat
- Umpan balik pengguna selama pengujian penerimaan (user acceptance) harus diatur oleh fungsi yang terlihat (user-visible function) oleh pengguna seperti yang digambarkan melalui antarmuka pengguna
  - Pengembang harus memeriksa perangkat yang dimaksud dan membuat perubahan pada layar antarmuka pengguna jika menerapkan perubahan ini tidak akan menunda rilis prototipe
- Gunakan pelacakan masalah atau sistem pelaporan bug untuk mendapatkan hasil pengujian
  - Merekam kegagalan pengujian dan
  - Mengidentifikasi kasus pengujian yang perlu dijalankan lagi untuk memverifikasi bahwa perbaikan akan memperbaiki masalah yang ditemukan dengan benar.
- Isu dan pelajaran yang didapat dari pembuatan kandidat rilis harus didokumentasikan dan dipertimbangkan oleh pengembang dan stakeholder sebagai bagian dari postmortem proyek

4.9 Maintain Release Software

4.9 Maintainance Release Software

- Pemeliharaan (maintenance) didefinisikan sebagai aktivitas yang diperlukan untuk menjaga PL tetap beroperasi setelah diterima dan dikirimkan (dirilis) ke pengguna akhir
- Pemeliharaan akan berlanjut selama masa pakai produk perangkat lunak



Distribusi Upaya Maintenance

SWEBOK (Software Engineering Book of Knowledge

Corrective Maintenance
- adalah modifikasi reaktif perangkat lunak untuk memperbaiki masalah yang ditemukan setelah perangkat lunak dikirimkan ke pengguna akhir pelanggan

Adaptive Maintenance
- adalah modifikasi reaktif perangkat lunak setelah pengiriman untuk menjaga perangkat lunak dapat digunakan dalam lingkungan pengguna akhir yang berubah

Perfective Maintenance
- adalah modifikasi proaktif perangkat lunak setelah pengiriman untuk menyediakan fitur pengguna baru, struktur kode program yang lebih baik, atau dokumentasi yang ditingkatkan

Preventive Maintenance
- adalah modifikasi proaktif perangkat lunak setelah pengiriman untuk mendeteksi dan memperbaiki kesalahan produk sebelum ditemukan oleh pengguna di lapangan

Proactive Maintenance
Pemeliharaan proaktif dapat dijadwalkan dan direncanakan.

Reactive Maintenance
Sering digambarkan sebagai pemadam kebakaran karena tidak dapat direncanakan dan harus segera ditangani untuk sistem PL yang penting bagi keberhasilan aktivitas pengguna akhir

Coding Maintenance
- Koding didokumentasi agar lebih mudah dipahami oleh orang lain untuk melakukan pekerjaan pemeliharaan
- Berguna jika PL dirancang untuk diperpanjang, agar pemeliharaan lebih mudah, selain perbaikan kerusakan darurat

Recommended Software Process Steps

Recommended Software Process Steps

1. Requirements engineering
   - Gather user stories from all stakeholders.
   - Have stakeholders describe acceptance criteria user stories.
2. Preliminary architectural design
   - Make use of paper prototypes and models.
   - Assess alternatives using nonfunctional requirements.
   - Document architecture design decisions.
3. Estimate required project resources
   - Use historic data to estimate time to complete each user story.
   - Organize the user stories into sprints.
   - Determine the number of sprints needed to complete the product.
   - Revise the time estimates as use stories are added or deleted.
4. Construct first prototype
   - Select subset of user stories most important to stakeholders.
   - Create paper prototype as part of the design process.
   - Design a user interface prototype with inputs and outputs.
   - Engineer the algorithms needed for first prototypes.
   - Prototype with deployment in mind.
5. Evaluate prototype
   - Create test cases while prototype is being designed.
   - Test prototype using appropriate users.
   - Capture stakeholder feedback for use in revision process.
6. Go, no-go decision
   - Determine the quality of the current prototype.
   - Revise time and cost estimates for completing development.
   - Determine the risk of failing to meet stakeholder expectations.
   - Get commitment to continue development.
7. Evolve system
   - Define new prototype scope.
   - Construct new prototype.
   - Evaluate new prototype and include regression testing.
   - Assess risks associated with continuing evolution.
8. Release prototype
   - Perform acceptance testing.
   - Document defects identified.
   - Share quality risks with management.
9. Maintain software
   - Understand code before making changes.
   - Test software after making changes.
   - Document changes.
   - Communicate known defects and risks to all stakeholders.

Pertemuan 4:
Recommended Process Model

Meuthia Rachmaniah
Departemen Ilmu Komputer, FMIPA IPB

Software Engineering: A Practitioner's Approach, 9th Ed.
Roger S. Pressman dan Bruce R. Maxim