1. Rekayasa Perangkat Lunak adalah : Suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal requirement capturing (analisa kebutuhan pengguna), specification (menentukan spesifikasi dari kebutuhan pengguna), desain, coding, testing sampai pemeliharaan sistem setelah digunakan. Kalimat "seluruh aspek produksi perangkat lunak" membawa implikasi bahwa Rekayasa Perangkat Lunak tidak hanya berhubungan dengan masalah teknis pengembangan perangkat lunak tetapi juga kegiatan strategis seperti manajemen proyek perangkat lunak, penentuan metode dan proses pengembangan, serta aspek teoritis, yang kesemuanya untuk mendukung terjadinya Produksi perangkat lunak . Sementara itu,**Pemrograman komputer** dapat diartikan sebagai proses menulis, menguji dan memperbaiki (debug), dan memelihara kode yang membangun sebuah program komputer dimana kode ini ditulis dalam berbagai bahasa pemrograman.

Perbedaannya dapat disederhanakan sebagai berikut:

- **Rekayasa Perangkat Lunak** mencakup seluruh proses pengembangan perangkat lunak dari awal sampai akhir, termasuk analisis, desain, pengujian, dokumentasi, dan pemeliharaan.
- **Pemrograman Komputer** hanya merupakan satu tahap dari proses tersebut, yaitu menulis kode agar perangkat lunak dapat berjalan sesuai desain dan spesifikasi.

- IRMAYANI, Deci. Rekayasa perangkat lunak. *INFORMATIKA*, 2014, 2.3: 140-148.
- RETTA, Allen Marga; ISROQMI, Asnurul; NOPRIYANTI, Tika Dwi. Pengaruh penerapan algoritma terhadap pembelajaran pemrograman komputer. *Indiktika: Jurnal Inovasi Pendidikan Matematika*, 2020, 2.2: 126-135.

2. Software Evolution merupakan aspek penting dalam Software Process karena perangkat lunak tidak pernah selesai dalam satu tahap pengembangan saja, melainkan harus terus dipelihara dan dikembangkan untuk tetap memenuhi kebutuhan pengguna yang berubah, memperbaiki kesalahan, beradaptasi dengan perubahan teknologi, serta mempertahankan kegunaan jangka panjang. Menurut Bennett dan Rajlich (2000), perangkat lunak akan mengalami evolusi secara bertahap melalui beberapa fase, mulai dari pengembangan awal, pertumbuhan, hingga akhirnya mengalami penurunan apabila tidak dilakukan pemeliharaan yang memadai.

Tanpa adanya evolusi, perangkat lunak akan cepat menjadi usang (*software aging*) dan tidak lagi memenuhi ekspektasi fungsional ataupun kualitas pengguna. Selain itu, teknologi dan standar eksternal juga terus berubah, yang mengharuskan perangkat lunak untuk beradaptasi jika ingin tetap relevan.

Contoh situasi yang memerlukan perubahan dalam software:

- **Perubahan regulasi pemerintah**, seperti kebijakan pajak baru, memaksa sistem akuntansi untuk diperbarui agar tetap patuh terhadap hukum.
- **Migrasi ke teknologi baru**, misalnya perusahaan berpindah dari server fisik ke cloud computing, sehingga sistem yang lama harus dimodifikasi agar dapat berjalan di lingkungan baru.
- **Ancaman keamanan baru**, di mana munculnya celah keamanan (vulnerability) baru mewajibkan pengembang melakukan patching untuk melindungi data pengguna.
- **Tuntutan pengguna**, misalnya pengguna menginginkan fitur baru seperti mode gelap atau integrasi dengan aplikasi lain, sehingga perangkat lunak perlu berevolusi untuk menambahkannya.

- BENNETT, Keith H.; RAJLICH, Václav T. Software maintenance and evolution: a roadmap. In: *Proceedings of the Conference on the Future of Software Engineering*. 2000. p. 73-87.

3. Keuntungan Pendekatan Low-Code dan No-Code

Pendekatan *Low-Code* dan *No-Code* menawarkan berbagai keuntungan dalam pengembangan perangkat lunak, terutama dalam konteks transformasi digital yang cepat. Menurut Abednego, Suyoto, & Julianto (2021), penggunaan platform *Low-Code* memungkinkan pengembangan aplikasi yang lebih cepat dan efisien tanpa memerlukan penulisan kode yang kompleks.

- **Peningkatan Kolaborasi**
  Memfasilitasi kolaborasi antara tim teknis dan non-teknis dalam proses pengembangan aplikasi.
- **Percepatan Pengembangan Aplikasi**
  Penggunaan antarmuka visual dan komponen yang dapat digunakan kembali mempercepat proses pengembangan aplikasi.
- **Pengurangan Biaya Pengembangan**
  Mengurangi kebutuhan akan pengembang dengan keahlian khusus, sehingga menurunkan biaya pengembangan.
- **Fleksibilitas dan Skalabilitas**
  Memungkinkan penyesuaian aplikasi dengan kebutuhan bisnis yang berubah tanpa memerlukan perubahan kode yang signifikan.

Contoh kasus : Pengembangan Aplikasi Presensi di PT Astra Sedaya Finance

PT Astra Sedaya Finance menghadapi tantangan dalam mengelola presensi karyawan selama pandemi COVID-19. Untuk mengatasi hal ini, perusahaan mengembangkan aplikasi presensi berbasis mobile menggunakan platform *Low-Code* OutSystems.

Hasil yang dicapai:

- Pengembangan aplikasi selesai dalam waktu singkat tanpa memerlukan penulisan kode yang kompleks.
- Aplikasi memfasilitasi pencatatan presensi dan pelaporan kinerja karyawan secara online melalui smartphone.
- Meningkatkan efisiensi operasional dan meminimalkan kontak fisik selama pandemi.

- AKBAR, Ghifar Maulana; IDRIS, Moh. Penerapan Low-Code Platform dalam Pengembangan Aplikasi Presensi. *AUTOMATA*, 2022, 3.2.
- ABEDNEGO, Azarya, et al. Pengembangan Aplikasi Layanan Multiguna Menggunakan Low-Code Platform (Studi Kasus: Astra Credit Companies). *Jurnal Informatika Atma Jogja*, 2021, 2.1: 10-19.

4. (k) Rational Unified Process (RUP)

Apa itu RUP?

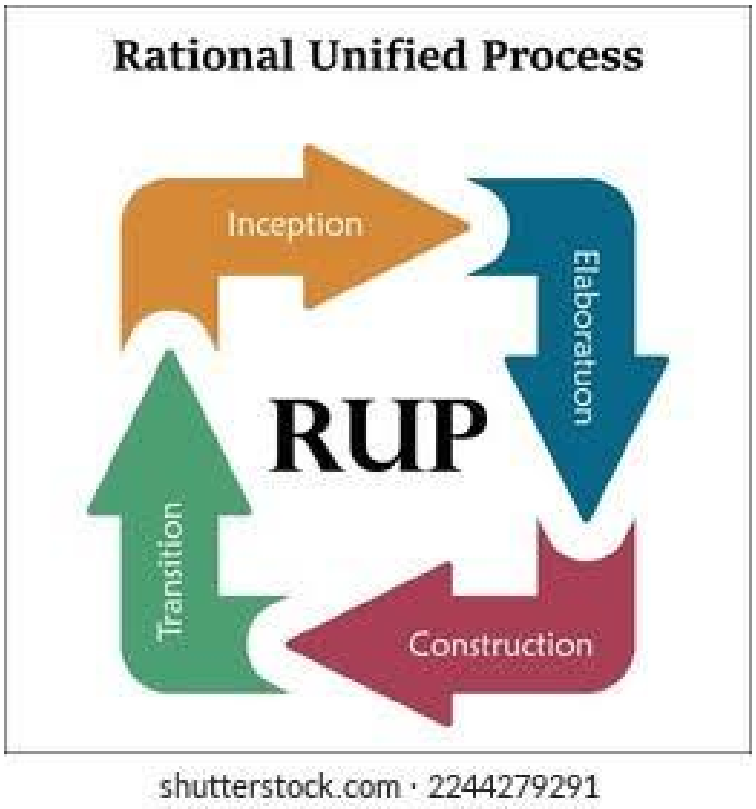
Rational Unified Process (RUP) adalah metodologi pengembangan perangkat lunak iteratif dan incremental yang dikembangkan oleh Rational Software, kemudian diakuisisi oleh IBM. RUP memadukan praktik rekayasa perangkat lunak terbaik untuk membantu tim menghasilkan perangkat lunak berkualitas tinggi yang memenuhi kebutuhan pengguna.

Menurut Kruchten (2003), RUP mengorganisasi pengembangan dalam serangkaian fase yang berfokus pada manajemen risiko dan kebutuhan perubahan.

Alur Kerja RUP

- **Inception Phase:** Mendefinisikan visi proyek, ruang lingkup, dan kebutuhan dasar.
- **Elaboration Phase:** Menguraikan sistem arsitektur dan mengatasi risiko utama.
- **Construction Phase:** Membangun komponen dan fitur sistem.
- **Transition Phase:** Menyerahkan produk akhir ke pengguna dan mendukung penggunaannya.

Diagram Alur RUP:



Prinsip-prinsip RUP

- Berorientasi pada use case dan kebutuhan pengguna.
- Pengembangan berbasis arsitektur perangkat lunak.
- Iteratif dan inkremental untuk mengurangi risiko.
- Manajemen perubahan yang terkendali.
- Pengujian dan validasi berkelanjutan..

Kelebihan RUP

- Cocok untuk proyek besar dan kompleks.
- Mengurangi risiko sejak awal proyek.
- Struktur fase yang jelas membantu pengelolaan proyek.
- Mendorong dokumentasi yang baik.

Kekurangan RUP

- Proses awal bisa memakan waktu dan biaya tinggi.
- Memerlukan pelatihan dan pemahaman mendalam.
- Bisa menjadi terlalu berat untuk proyek kecil.

(i) . Feature-Driven Development (FDD)

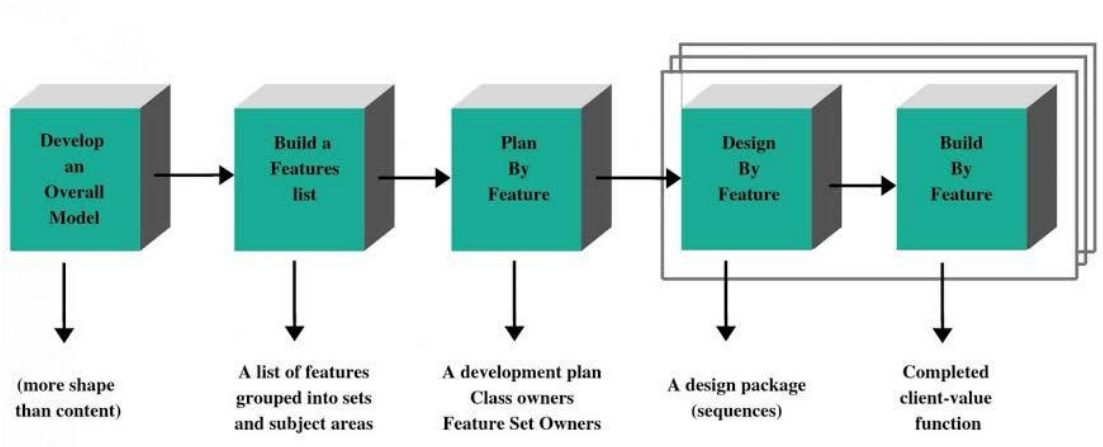
Apa itu FDD?

Feature-Driven Development (FDD) adalah framework Agile yang mengorganisasikan pembangunan sistem berdasarkan fitur-fitur kecil yang dapat dikirimkan secara bertahap.Menurut Palmer dan Felsing (2002), FDD cocok untuk proyek berskala besar dengan tim yang banyak, karena struktur kerjanya jelas dan berbasis fitur nyata yang diminta pengguna.

Alur Kerja FDD

- **Develop Overall Model:** Membuat model domain besar sistem.
- **Build Feature List:** Membuat daftar fitur kecil berdasarkan kebutuhan pengguna.
- **Plan by Feature:** Menyusun rencana pengembangan untuk setiap fitur.
- **Design by Feature:** Merancang solusi untuk masing-masing fitur secara kolaboratif.
- **Build by Feature:** Membangun dan menguji fitur secara bertahap, lalu mengintegrasikannya ke sistem.

Diagram Alur FDD:



Prinsip-prinsip FDD:

- Berbasis fitur nyata yang diinginkan pengguna.
- Mengutamakan pemodelan awal yang kuat.
- Tim kecil untuk merancang dan mengembangkan tiap fitur.
- Integrasi dan pembaruan sistem secara berkelanjutan.

Kelebihan FDD:

- Struktur kerja jelas dan terukur.
- Memudahkan pelacakan kemajuan proyek.
- Cocok untuk tim besar dengan banyak pengembang.

Kekurangan FDD:

- Kurang fleksibel dalam menghadapi perubahan cepat.
- Membutuhkan waktu besar di tahap awal untuk pemodelan.
- Tidak cocok untuk proyek kecil yang sifatnya eksploratif.

- PALMER, Steve R.; FELSING, Mac. A practical guide to feature-driven development. Pearson Education, 2001.
- KRUCHTEN, Philippe. The rational unified process: an introduction. Addison-Wesley Professional, 2004.

5. (t) . Framework: Scrum

Scrum adalah framework Agile yang digunakan untuk mengelola proyek kompleks dengan cara membaginya ke dalam iterasi pendek yang disebut Sprint.Menurut Sliger dan Broderick (2008), Scrum mendukung pengembangan adaptif, dengan tim kecil yang bekerja kolaboratif untuk mencapai hasil maksimal di setiap Sprint.

Aplikasi yang Menggunakan Scrum

Spotify adalah contoh nyata perusahaan yang menggunakan Scrum dalam pengembangan aplikasinya.

Penggunaan Scrum di Spotify

- Spotify menggunakan kerangka kerja Scrum untuk membentuk Squad, yaitu tim kecil yang otonom.
- Setiap Squad bekerja seperti tim Scrum: melakukan Sprint Planning, Daily Scrum, Sprint Review, dan Retrospective.
- Dengan Scrum, Spotify mampu merilis fitur seperti Discover Weekly, Spotify Wrapped, dan Personalized Playlists dengan cepat.
- Pendekatan Scrum membantu Spotify berinovasi lebih cepat dan responsif terhadap feedback pengguna.

- SLIGER, Michele; BRODERICK, Stacia. *The software project manager's bridge to agility*. Addison-wesley professional, 2008.
- KNIBERG, Henrik; IVARSSON, Anders. Scaling agile@ spotify. Online], UCVOF, ucvox. Files. Wordpress. Com/2012/11/113617905-scaling-Agile-spotify-11. Pdf, 2012.