

# Pertemuan 1: Perangkat Lunak dan Rekayasa Perangkat Lunak

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed., 2020*

Roger S. Pressman dan Bruce R. Maxim

Copyright © 2020 McGraw-Hill Education

# OBJECTIVES/SELF-STUDY OUTLINE

- 1) **Sifat Perangkat Lunak (PL)**
- 2) ***Defining the Discipline***
- 3) **Proses Perangkat Lunak (Software Process)**
- 4) **Praktik Rekayasa Perangkat Lunak (RPL)**
- 5) ***How It All Start***
- 6) **Rangkuman**

# QUICK LOOK

## WHAT IS IT?



- PL Komputer adalah *work product* yang dibangun dan didukung oleh professional PL selama bertahun-tahun.
  - *Work product* adalah program yang dieksekusi komputer pada ukuran dan arsitektur komputer yang beragam
  - RPL mencakup proses, koleksi metode-metode (*praktik*), dan *array of tools* untuk profesional membangun PL berkualitas tinggi

## WHO DOES IT



- Software Engineers: membangun dan mendukung PL. Software engineers menerapkan proses RPL
- Every Users: setiap orang pada industrialized world

## WHY IS IT IMPORTANT?



- RPL penting karena memungkinkan kita membangun sistem yang kompleks secara *tepat waktu* dan dengan *kualitas tinggi*
- Memaksakan *disiplin kerja dengan ketat*, namun masih memungkinkan untuk *menyesuaikan pendekatan yang sesuai dengan kebutuhan*

## WHAT ARE THE STEPS



- PL dibangun seperti Anda membangun produk yang sukses:
- menerapkan proses yang gesit (*agile*) dan *mudah beradaptasi*
  - mengarah pada *hasil berkualitas tinggi*
  - *memenuhi kebutuhan* orang-orang yang akan menggunakan produk

## WHAT IS THE WORK PRODUCT



- Sudut pandang *Software Engineers*: gugus program-program, konten (data), dan work product lainnya untuk mendukung PL komputer
- Sudut pandang *Users*: tool atau produk yang sedemikian rupa membuat dunia users menjadi lebih baik

## HOW DO I ENSURE THAT I'VE DONE IT RIGHT



- Ikuti terus kuliah dan praktikum RPL, pilih ide-ide yang dapat diterapkan pada PL yang akan dibangun, dan terapkan/gunakan PL tsb dalam pekerjaan Anda

# Tantangan Realita Pengembangan PL yang Dihadapi

## Aspek dan Minat

- PL telah menjadi aspek dari setiap kehidupan
- Peningkatan jumlah orang memiliki **minat** pada **fitur** dan **fungsi** yang terdapat pada aplikasi PL
  - *Upaya bersama (seperti konser) harus dilakukan untuk memahami masalah sebelum solusi PL dikembangkan*

## IT Requirements

- Persyaratan teknologi informasi yang dituntut oleh individu, bisnis, dan pemerintah tumbuh **semakin kompleks** setiap tahun
- **PL canggih** yang pernah diimplementasikan dalam lingkungan komputasi mandiri yang dapat diprediksi, sekarang tertanam (**embedded**) di dalam segala hal, mulai dari elektronik konsumen hingga perangkat medis hingga **kendaraan otonom**
- *Design has become a pivotal activity (Desain telah menjadi aktivitas penting)*



**PL DALAM SEGALA  
BENTUKNYA DAN DI  
SEMUA DOMAIN  
APLIKASINYA  
HARUS DIREKAYASA**

## Ketergantungan pada PL

- Individu, bisnis, dan pemerintah semakin bergantung pada PL untuk pengambilan **keputusan strategis** dan **taktis** serta **operasi** dan **control** sehari-hari
- *Software should exhibit high quality (PL harus menunjukkan kualitas tinggi)*

## Pertumbuhan Minat dan Tuntutan

- Nilai yang dirasakan dari aplikasi tertentu tumbuh, sehingga basis pengguna dan umur PL juga akan tumbuh
- Seiring bertambahnya basis pengguna dan waktu penggunaan, maka **tuntutan** untuk **adaptasi** dan peningkatan juga akan bertumbuh
- *Software should be maintainable (PL harus dapat dipelihara/maintain)*

# Rekayasa Perangkat Lunak (RPL)

## ■ Some realities:

- *Upaya bersama (seperti konser) harus dilakukan untuk memahami masalah sebelum solusi PL dikembangkan*
- *Design has become a pivotal activity (Desain telah menjadi aktivitas penting)*
- *Software should exhibit high quality (PL harus menunjukkan kualitas tinggi)*
- *Software should be maintainable (PL harus dapat dipelihara/maintain)*

## ■ The seminal definition:

- *[RPL adalah] penetapan dan penggunaan prinsip-prinsip rekayasa untuk mendapatkan PL ekonomis yang dapat diandalkan dan bekerja secara efisien pada mesin nyata (real machines)*



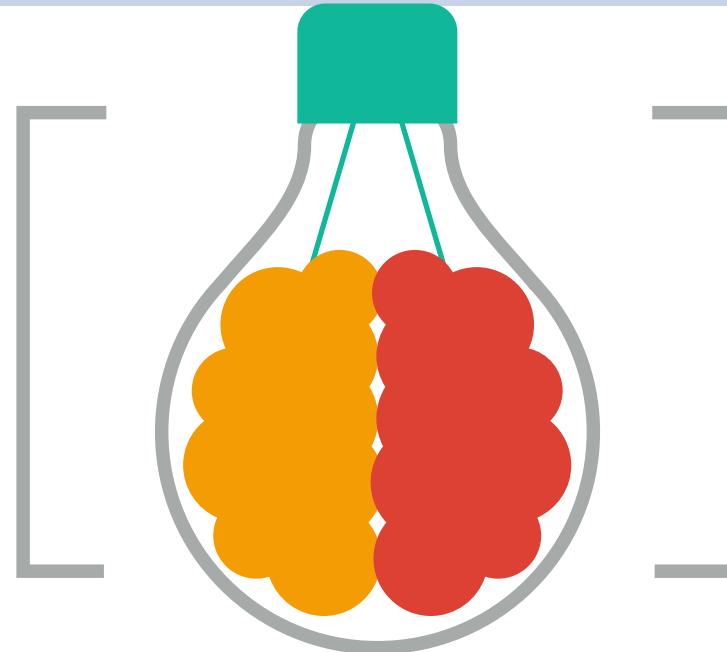
# 1) SIFAT PERANGKAT LUNAK

Dua peran PL:

- 1) Sebagai *Vehicle*
- 2) Sebagai *Produk*

## PL sebagai *Vehicle* untuk mengirimkan produk

- Bertindak sebagai basis kontrol untuk komputer (sistem operasi)
- Mengkomunikasikan informasi (jaringan)
- Menciptakan dan mengendalikan program-program lain (*software tools and environment*)



PL Komputer mengalami perubahan signifikan dalam 60 tahun terakhir:

- Kinerja hardware, arsitektur komputasi, peningkatan memory dan storage, variasi input/output
- Hasil luar biasa jika sukses, namun menimbulkan masalah besar jika gagal
- Dikembangkan oleh **tim spesialis**, bukan lagi oleh lone programmer

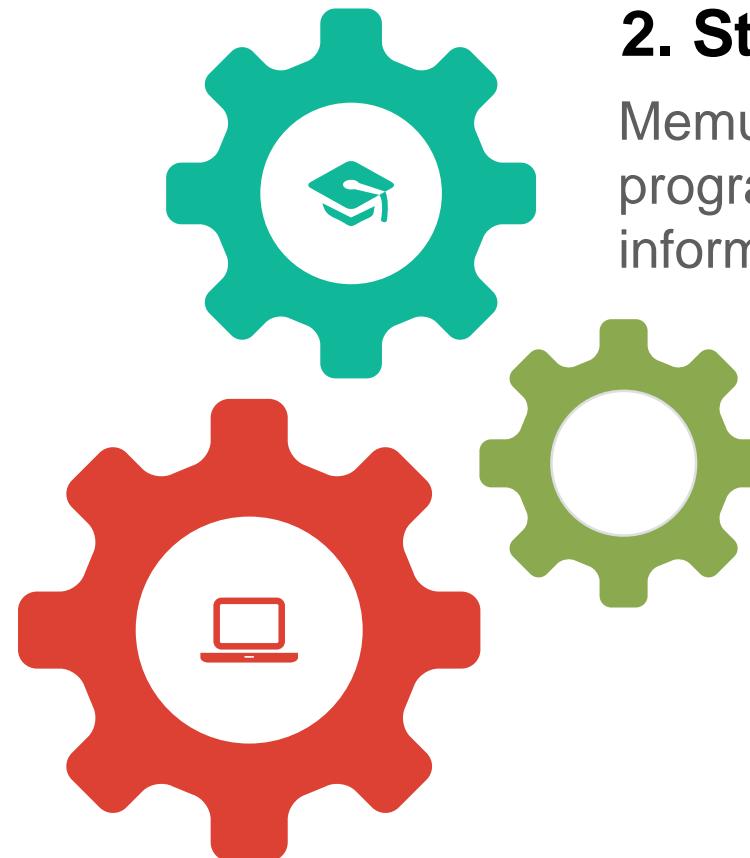
## PL sebagai Produk

- PL memberikan potensi komputasi yang diwujudkan oleh hardware komputer atau oleh jaringan komputer yg dapat diakses oleh hardware lokal
- PL terdapat dalam mobile device, desktop, cloud, computer mainframe, autonomous machine
- PL adalah transformator informasi: menghasilkan, mengelola, memperoleh, memodifikasi, menampilkan, atau mentransmisikan informasi sederhana berupa bit tunggal atau kompleks seperti representasi augmented reality yang diperoleh dari lusinan sumber independen yang dihamparkan di dunia nyata

## 1.a DEFINISI PERANGKAT LUNAK

### 1. Instruksi (Program-program Komputer)

Apabila dieksekusi menyediakan fitur-fitur, fungsi dan kinerja yang diinginkan



### 2. Struktur Data

Memungkinkan program-program untuk memanipulasi informasi secara memadai

### Informasi Deskriptif

Dalam bentuk hardcopy dan bentuk virtual yang menjelaskan operasi dan penggunaan program-program

- Karakteristik PL berbeda dari hal-hal lain yang dibuat oleh manusia.
- PL adalah logik bukan elemen sistem fisik.
- PL memiliki satu karakteristik mendasar yang membuatnya sangat berbeda dari hardware: “**PL Tidak “Usang”**”.

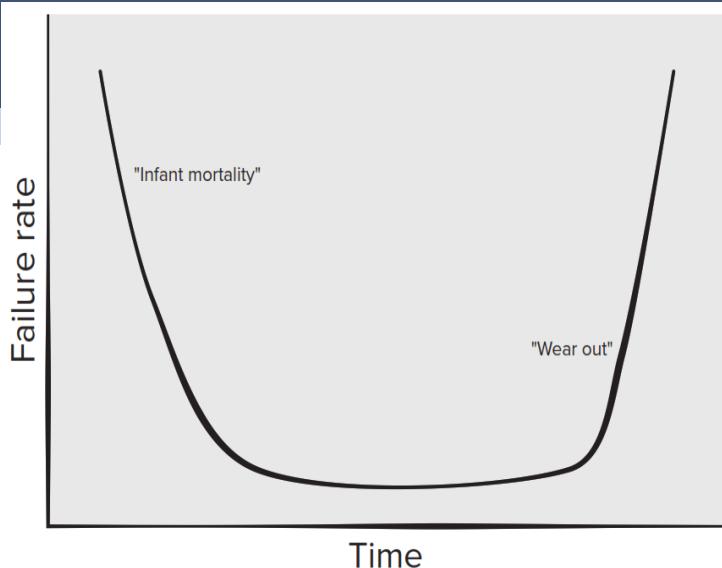


Fig.1.1 Kurva kegagalan **hardware**

- Disebut ***bathtub curve relationship***
- Kegagalan relatif tinggi di awal (cacat desain/manufaktur), lalu dikoreksi, lalu cacat menurun sampai tingkat **steady-state**
- Seiring waktu, laju gagal meningkat lagi (efek kumulatif debu, vibrasi, penyalagunaan, temperature ekstrim, dll → **WEAR OUT** (usang))

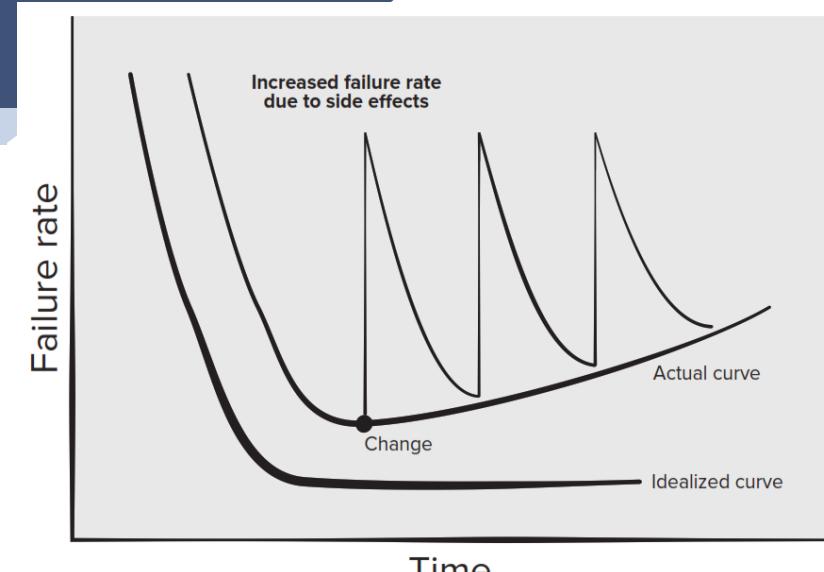


Fig.1.1 Kurva kegagalan PL (**software**)

- PL tidak rentan terhadap penyakit lingkungan yang menyebabkan hardware usang
- Secara teori, kurva tingkat kegagalan PL harus berbentuk “kurva ideal” (**idealized curve**)
- Awal program, cacat akan tinggi, lalu dikoreksi sehingga tingkat cacat mendatar
- PL mengalami **deteriorate** (menurun) sehingga perlu diubah (**change**), akan ada tingkat error (**actual curve**), dst.

# 1.b DOMAIN APLIKASI PERANGKAT LUNAK

## 7. ARTIFICIAL INTELLIGENCE SOFTWARE

- Memanfaatkan **heuristik** untuk memecahkan **problem kompleks** yang tidak dapat dilakukan dengan perhitungan biasa atau analisis langsung.
- Contoh aplikasi: robotika, sistem pengambilan keputusan, pengenalan pola (gambar dan suara), pembelajaran mesin, pembuktian teorema, dan permainan.

## 6. WEB/MOBILE APPLICATION

- **Berpusat pada jaringan** mencakup beragam aplikasi dan aplikasi berbasis browser, komputasi awan, komputasi berbasis layanan, dan perangkat lunak yang berada di perangkat seluler

## 5. PRODUCT-LINE SOFTWARE

- Terdiri dari komponen yang dapat digunakan kembali (**reusable component**) dan dirancang untuk memberikan kemampuan khusus untuk digunakan oleh banyak pelanggan yang berbeda
- Dapat difokuskan pada pasar yang terbatas dan esoterik (misalnya, produk pengendalian persediaan/*inventory control product*) atau mencoba untuk mengatasi pasar konsumen massal.



## 4. EMBEDDED SOFTWARE

- **Terdapat di dalam produk atau sistem** dan digunakan untuk mengimplementasikan dan mengontrol fitur dan fungsi untuk pengguna akhir dan untuk sistem itu sendiri
- Dapat melakukan fungsi terbatas dan esoteris (misalnya, kontrol tombol untuk oven microwave) atau menyediakan fungsi dan kemampuan kontrol yang signifikan (misalnya, fungsi digital dalam mobil seperti kontrol bahan bakar, tampilan dasbor, dan sistem penggeraman)

## 1. SYSTEM SOFTWARE

Koleksi program-program sebagai layanan (service) untuk program-program lainnya:

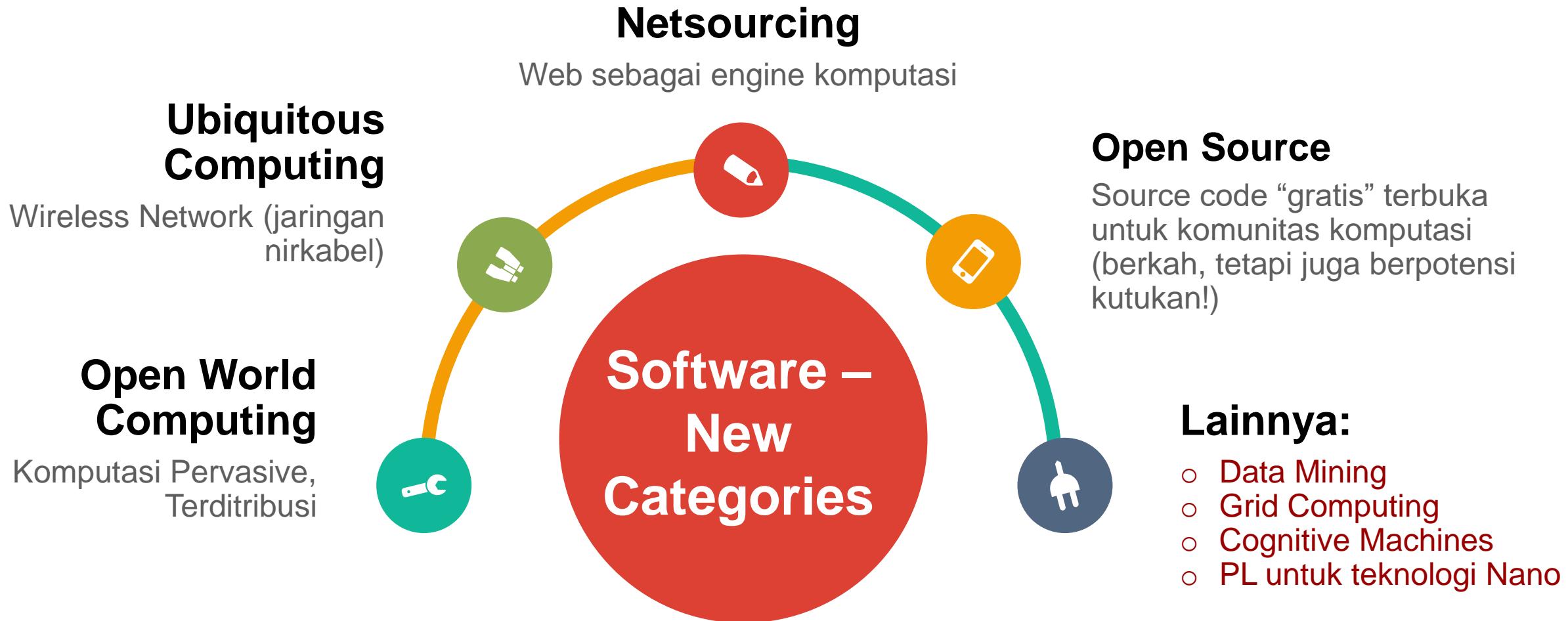
- **Determinate**: PL sistem (misalnya, kompiler, editor, dan utilitas manajemen file) memproses kompleks pada struktur informasi tertentu
- **Indeterminate**: aplikasi sistem (misalnya, komponen sistem operasi, driver, perangkat lunak jaringan, prosesor telekomunikasi) memproses sebagian besar data tak tentu

## 2. APPLICATION SOFTWARE

- Program yang **berdiri sendiri** yang memecahkan **kebutuhan bisnis tertentu**
- Memproses data bisnis atau teknis dengan cara yang memfasilitasi **operasi bisnis** atau **pengambilan keputusan manajemen/teknis**

## 3. ENGINEERING/SCIENTIFIC SOFTWARE

- Serangkaian program "**penghitungan angka**" atau **data sains** yang luas seperti astronomi hingga vulkanologi, dari **stress analysis** otomotif hingga dinamika orbital, dari desain berbantuan komputer hingga kebiasaan belanja konsumen, dan dari analisis genetik hingga meteorologi



## 1.c LEGACY PERANGKAT LUNAK

Left ??

Right ??



### Legacy Software Systems ?

- Ratusan ribu program komputer termasuk dalam salah satu dari tujuh domain aplikasi PL
- Perubahan ini dapat menciptakan efek samping tambahan yang sering ada pada perangkat lunak lawas—kualitas buruk.
- Sistem lama terkadang memiliki desain yang tidak dapat diperluas, kode yang berbelit-belit, dokumentasi yang buruk atau tidak ada sama sekali, kasus pengujian dan hasil yang tidak pernah diarsipkan, dan riwayat perubahan yang tidak dikelola dengan baik

- Dikembangkan beberapa dekade yang lalu dan telah terus dimodifikasi untuk memenuhi perubahan kebutuhan bisnis dan platform komputasi.
- Proliferasi sistem semacam itu menyebabkan sakit kepala bagi organisasi besar yang menganggapnya mahal untuk dipelihara dan berisiko untuk berkembang

### ○ What To Do?

- ***Do Nothing*** setidaknya sampai sistem warisan harus mengalami beberapa perubahan signifikan
- ***Does not need to be fixed*** jika memenuhi kebutuhan penggunanya dan berjalan dengan andal

# ALASAN LEGACY SYSTEM HARUS BEREVOLUSI



## Harus Disesuaikan

untuk memenuhi kebutuhan lingkungan atau teknologi komputasi baru



## Harus Ditingkatkan

untuk menerapkan persyaratan bisnis baru



- Merancang metodologi yang didasarkan pada gagasan evolusi, yaitu:
  - Gagasan bahwa system PL baru dapat dibangun dari yang lama
  - Semua harus berinteraksi dan bekerja sama satu sama lain

## Harus Diperluas

untuk membuatnya bekerja dengan sistem atau database lain yang lebih modern



## Harus Diarsitektur Ulang

untuk membuatnya layak dalam lingkungan komputasi yang berkembang..



## Tujuan dari RPL modern:

# Karakteristik WebApps - I

- **Network Intensiveness.** WebApp terdapat dalam network dan harus melayani kebutuhan komunitas klien yang beragam
- **Concurrency.** Sejumlah besar user dapat mengakses WebApp pada satu waktu bersamaan
- **Unpredictable Load.** Jumlah user pada WebApp dapat bervariasi magnitut besarnya dari hari ke hari
- **Performance.** Jika user WebApp harus menunggu lama (untuk akses, untuk pemrosesan di sisi server, untuk format dan display di sisi klien), maka WebApp akan ditinggalkan
- **Availability.** Walau ekspektasi tersedia 100% tidak mungkin diperoleh, namun user dari WebApp yang popular meminta akses **“24/7/365”**



# Karakteristik WebApps - II

- **Data Driven.** Fungsi primer dari WebApp ialah penggunaan hypermedia untuk teks, grafik, dan konten video
- **Content Sensitive.** Kualitas dan sifat estetika konten merupakan determinan penting untuk WebApp berkualitas
- **Continuous Evolution.** Aplikasi WebApp berevolusi secara konntinyu
- **Immediacy** (kebutuhan mendesak untuk meluncurkan PL ke pasar dengan cepat). WebApps sering menunjukkan waktu untuk memasarkan yang bisa dalam hitungan beberapa hari atau minggu
- **Security.** Karena WebApps tersedia melalui akses jaringan, sulit, jika bukan tidak mungkin, untuk membatasi populasi pengguna akhir yang dapat mengakses aplikasi
- **Aesthetic.** Bagian yang tak terbantahkan dari daya tarik WebApp adalah tampilan dan rasanya (***the look and feel***)

## 2) DEFINING THE DISCIPLINE

### ■ Definisi RPL menurut IEEE:

- 1) Penerapan pendekatan yang sistematis, disiplin, terukur untuk pengembangan, pengoperasian, dan pemeliharaan PL; yaitu, penerapan rekayasa ke PL.
- 2) Studi tentang pendekatan seperti pada (1)

# Lapisan-lapisan pada RPL



## Fokus pada Kualitas

- Komitmen organisasi pada kualitas
- Total Quality Management (TQM) atau Six Sigma



## Proses

- Adalah perekat yang menyatukan lapisan teknologi dan memungkinkan pengembangan PL komputer yang rasional dan tepat waktu
- Mendefinisikan kerangka kerja (framework) yang harus ditetapkan untuk pengiriman yang efektif dari teknologi RPL
- Membentuk dasar untuk pengendalian manajemen proyek PL dan menetapkan konteks di mana metode teknis diterapkan, produk kerja/work product (model, dokumen, data, laporan, formulir, dll.) diproduksi, tonggak (benchmark) pencapaian ditetapkan, kualitas dipastikan, dan perubahan dikelola dengan baik



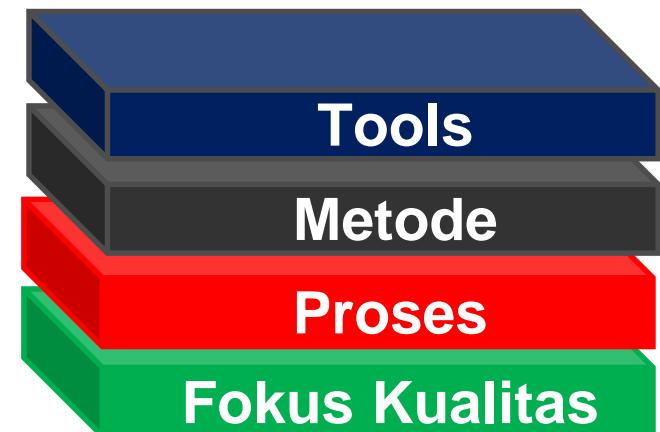
## Metode

- Petunjuk teknis untuk membangun PL.
- Metode mencakup beragam tugas yang mencakup komunikasi, analisis persyaratan (requirements), pemodelan desain, konstruksi program, pengujian, dan dukungan.
- Metode RPL bergantung pada seperangkat prinsip dasar yang mengatur setiap bidang teknologi dan mencakup aktivitas pemodelan dan teknik deskriptif lainnya



## Tools (Perangkat Bantu)

- Memberikan dukungan otomatis atau semi-otomatis untuk proses dan metode.
- Ketika tools terintegrasi sehingga informasi yang dibuat oleh satu tool dapat digunakan oleh tool lain, sebuah sistem untuk mendukung pengembangan PL , yang disebut RPL berbantuan komputer, dibuat



### 3) PROSES PERANGKAT LUNAK

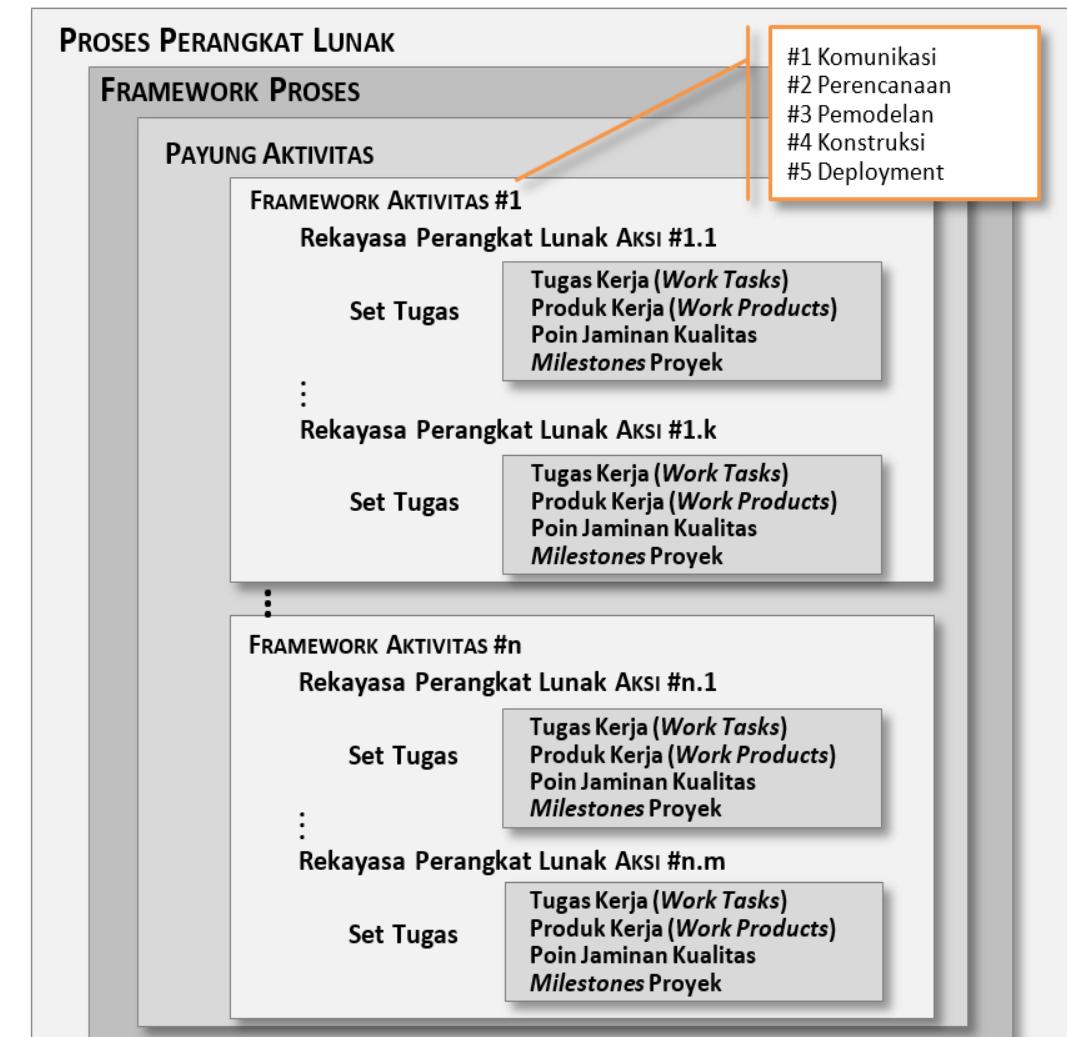
○ **Proses** adalah koleksi aktivitas, aksi (*actions*), dan tugas (*tasks*) yang dilakukan saat suatu produk kerja (*work product*) diciptakan

- **Aktivitas** adalah upaya mencapai tujuan luas (mis. Komunikasi dengan stakeholders) dan menerapkannya (terlepas dari domain aplikasi, ukuran proyek, kompleksitas upaya, derajat ketelitiannya)
- **Aksi/Actions** (mis.desain arsitektural) terdiri dari gugus tugas (*set of tasks*) yang menghasilkan produk kerja (*work product*) utama
- **Tugas/Tasks** berfokus pada tujuan kecil, tetapi terdefinisi dengan baik (mis. melakukan tes unit) yang menghasilkan hasil yang nyata

Sebuah proses bukanlah resep kaku untuk bagaimana membangun PL komputer. Sebaliknya, ini adalah pendekatan yang dapat disesuaikan yang memungkinkan orang yang melakukan pekerjaan (tim PL) untuk memilih dan memilih rangkaian tindakan dan tugas kerja yang sesuai.

### 3.a FRAMEWOK PROSES

- Menetapkan dasar untuk proses RPL yang lengkap dengan mengidentifikasi sejumlah kecil **framework aktivitas** yang berlaku untuk semua proyek PL, terlepas dari ukuran atau kompleksitasnya
- Mencakup serangkaian kegiatan payung (*Umbrella Activities*) yang berlaku di seluruh proses PL
- Framework Proses generik untuk RPL mencakup lima kegiatan: **Komunikasi, Perencanaan, Pemodelan, Konstruksi, dan Deployment**



# Framework Proses Generik



## Komunikasi

- Komunikasi dan kolaborasi dengan **konsumen** dan **stakeholder**
- Tujuan: untuk memahami tujuan stakeholder untuk proyek dan untuk mengumpulkan persyaratan yang membantu menentukan fitur dan fungsi PL



## Perencanaan

- Membuat rencana proyek PL— mendefinisikan pekerjaan RPL dengan menggambarkan tugas teknis yang akan dilakukan, risiko yang mungkin terjadi, sumber daya yang akan diperlukan, produk kerja yang akan diproduksi, dan jadwal kerja



## Pemodelan

- Model: Membuat "sketsa" sesuatu sehingga Anda akan memahami gambaran besarnya—seperti apa tampilannya secara arsitektur, bagaimana bagian-bagian penyusunnya cocok, dan banyak karakteristik lainnya.
- RPL: model untuk lebih memahami persyaratan PL dan desain yang akan mencapai persyaratan tersebut



## Konstruksi

- Apa yang Anda desain harus dibangun.
- Aktivitas ini menggabungkan pembuatan koding (baik manual atau otomatis) dan pengujian yang diperlukan untuk mengungkap kesalahan dalam koding



## Deployment

- PL (sebagai entitas yang lengkap atau sebagai tambahan yang diselesaikan sebagian) dikirimkan ke pelanggan yang mengevaluasi produk yang dikirimkan dan memberikan umpan balik berdasarkan evaluasi

## ■ Penerapan lima framework aktivitas generik:

- ▷ Pengembangan program sederhana dan kecil
- ▷ Penciptaan aplikasi web
- ▷ Rekayasa sistem berbasis komputer yang besar dan komplek

## ■ Banyak proyek PL melakukan framework aktivitas secara iteratif seiring dengan berkembangnya proyek

- ▷ Komunikasi, Perencanaan, Pemodelan, Konstruksi, dan Deployment diterapkan berulang kali pada sejumlah iterasi proyek
- ▷ Setiap iterasi menghasilkan *software increment* berupa subset dari keseluruhan fitur dan fungsi PL

## 3.b UMBRELLA ACTIVITIES

Aktivitas Framework Proses RPL dilengkapi oleh sejumlah Umbrella Activities

### PENGUKURAN

Mendefinisikan dan mengumpulkan proses, proyek, dan ukuran produk yang membantu tim dalam memberikan PL yang memenuhi kebutuhan pemangku kepentingan;

dapat digunakan bersama dengan semua framework aktivitas dan umbrella activities lainnya

### REVIEW TEKNIS

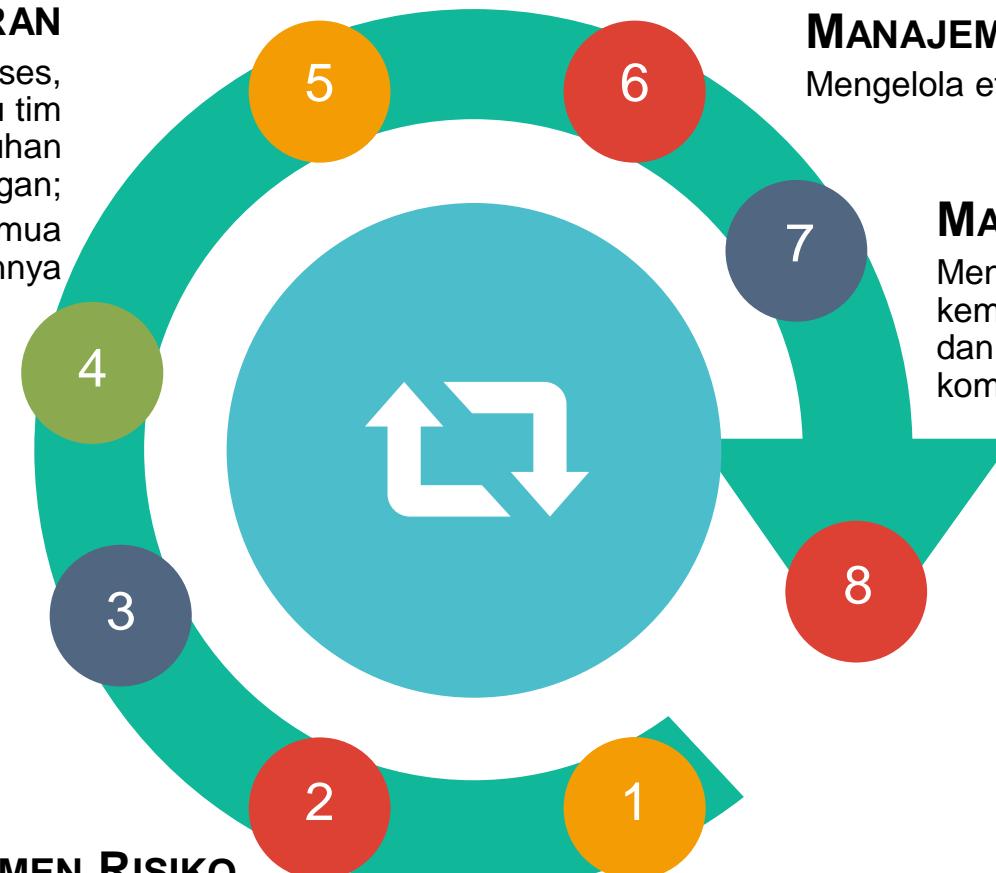
Menilai produk kerja RPL dalam upaya untuk mengungkap dan menghilangkan kesalahan sebelum disebarluaskan ke aktivitas berikutnya

### JAMINAN KUALITAS PL (SOFTWARE QUALITY ASSURANCE)

Mendefinisikan dan melakukan aktivitas yang diperlukan untuk memastikan kualitas PL

### MANAJEMEN RISIKO

Mengkaji risiko yang dapat mempengaruhi outcome proyek atau kualitas produk



### PELACAKAN DAN KONTROL PROYEK PL

Tim PL menilai kemajuan terhadap rencana proyek dan mengambil tindakan yang diperlukan untuk mempertahankan jadwal

### MANAJEMEN KONFIGURASI PL

Mengelola efek perubahan selama proses PL

### MANAJEMEN REUSABILITY

Mendefinisikan kriteria untuk penggunaan kembali produk kerja (termasuk komponen PL) dan menetapkan mekanisme untuk mencapai komponen reusable

### PREPARASI DAN PRODUKSI WORK PRODUCT

Melibati aktivitas yang diperlukan untuk membuat produk kerja (*work product*) seperti model, dokumen, log, formulir, dan daftar

### 3.c ADAPTASI PROSES RPL



# 4) PRAKTIK RPL - a) Esensi Praktik RPL

## 1. Understand the Problem (Komunikasi dan Analisis)

- Siapa yang memiliki kepentingan dalam pemecahan masalah? Artinya, siapa pemangku kepentingannya?
- Apa yang tidak diketahui? Data, fungsi, dan fitur apa yang diperlukan untuk menyelesaikan masalah dengan benar?
- Bisakah masalah dikotak-kotakkan? Apakah mungkin untuk merepresentasikan masalah yang lebih kecil yang mungkin lebih mudah dipahami?
- Bisakah masalah direpresentasikan secara grafis? Bisakah model analisis dibuat?

## 2. Rencanakan Solusi (Pemodelan dan Desain PL)

- Pernahkah Anda melihat masalah serupa sebelumnya? Apakah ada pola yang dapat dikenali dalam solusi potensial? Apakah ada PL yang mengimplementasikan data, fungsi, dan fitur yang diperlukan?
  - Apakah masalah serupa sudah terpecahkan? Jika demikian, apakah elemen solusi dapat digunakan kembali?
  - Bisakah submasalah didefinisikan? Jika demikian, apakah solusi mudah terlihat untuk submasalah?
- Dapatkah Anda mewakili solusi dengan cara yang mengarah pada implementasi yang efektif? Bisakah model desain dibuat?



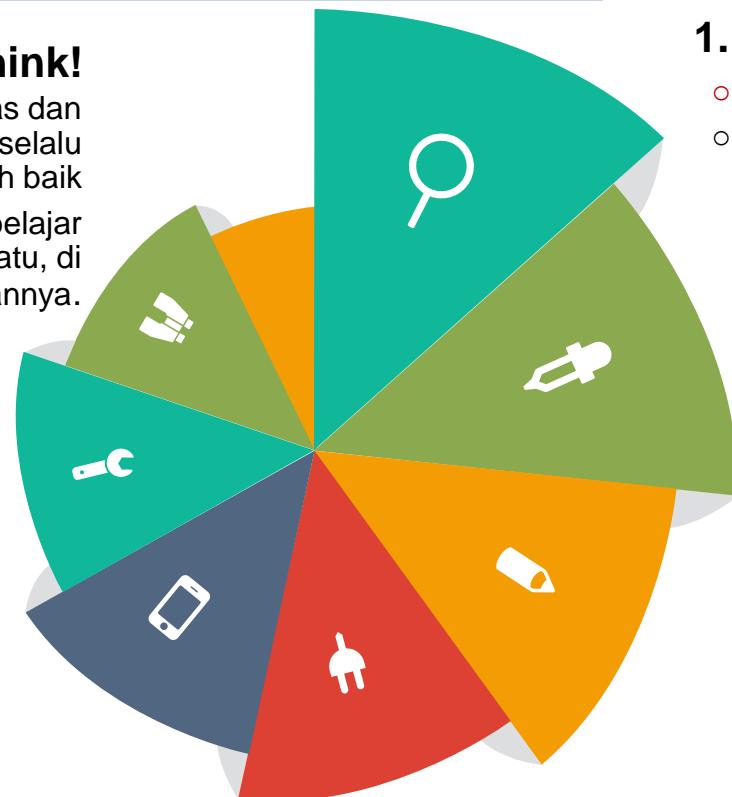
## 3. Jalankan Rencana (Code Generation)

- Apakah solusi sesuai dengan rencana? Apakah source code dapat dilacak ke model desain?
- Apakah setiap komponen bagian dari solusi terbukti benar? Apakah desain dan kode telah ditinjau, atau lebih baik, apakah bukti kebenaran telah diterapkan pada algoritme?

## Periksa Akurasi Hasil (Testing and QA)

- Apakah mungkin untuk menguji setiap komponen bagian dari solusi? Apakah strategi pengujian yang masuk akal telah diterapkan?
- Apakah solusi menghasilkan hasil yang sesuai dengan data, fungsi, dan fitur yang diperlukan? Apakah PL telah divalidasi terhadap semua persyaratan pemangku kepentingan?

## 4) PRAKTIK RPL - b) Prinsip-prinsip Umum Hooker



### 7. Think!

- Menempatkan pemikiran yang jelas dan lengkap sebelum bertindak hampir selalu menghasilkan hasil yang lebih baik
- Efek samping dari berpikir adalah belajar mengenali ketika Anda tidak tahu sesuatu, di mana Anda dapat meneliti jawabannya.

### 6. Plan Ahead for Reuse

- Reuse save time and effort. Reuse coding and desain merupakan benefit dari teknologi object-oriented
- Perencanaan ke depan untuk reusable mengurangi biaya dan meningkatkan nilai komponen reusable dan sistem di mana komponen-komponen tergabung

### 5. Be Open to the Future

- Spesifikasi berubah pada saat itu juga dan platform hardware sudah usang hanya beberapa bulan, masa pakai PL biasanya diukur dalam bulan, bukan tahun
- Selalu tanyakan "bagaimana jika", dan persiapkan semua kemungkinan jawaban dengan menciptakan sistem yang memecahkan masalah umum, bukan hanya masalah spesifik

### 4. What Your Produce, Others Will Consume

- Selalu tentukan, rancang, dokumentasikan, dan implementasikan dengan mengetahui bahwa orang lain harus memahami apa yang Anda lakukan
- Potensi user sangat besar. Saat desain, pertimbangkan implementer. Saat Koding, pertimbangkan SDM yang akan memelihara dan melakukan debug

### 1. The Reason It All Exist

- To provide value to its users
- "Does this add real value to the system?" If the answer is no, don't do it.

### 2. KISS (Keep It Simple, Stupid!)

- All design should be as simple as possible, but no simpler
- Desain yang lebih elegan biasanya yang lebih sederhana
- Seringkali dibutuhkan banyak pemikiran dan penggeraan berulang kali untuk menyederhanakan desain

### 3. Maintain the Vision

- Visi yang jelas sangat penting untuk keberhasilan proyek PL
- Memiliki arsitek yang dapat memegang visi dan menegakkan kepatuhan membantu memastikan kesuksesan proyek PL

## 5) How It ALL START

### ■ *SafeHome:*

#### ■ **Setiap proyek PL dipicu oleh beberapa kebutuhan bisnis —**

- kebutuhan untuk memperbaiki cacat pada aplikasi yang ada;
- kebutuhan akan kebutuhan untuk mengadaptasi 'legacy system' dengan lingkungan bisnis yang berubah;
- kebutuhan untuk memperluas fungsi dan fitur aplikasi yang ada, atau
- kebutuhan untuk menciptakan produk, layanan, atau sistem baru.

## 6) RANGKUMAN

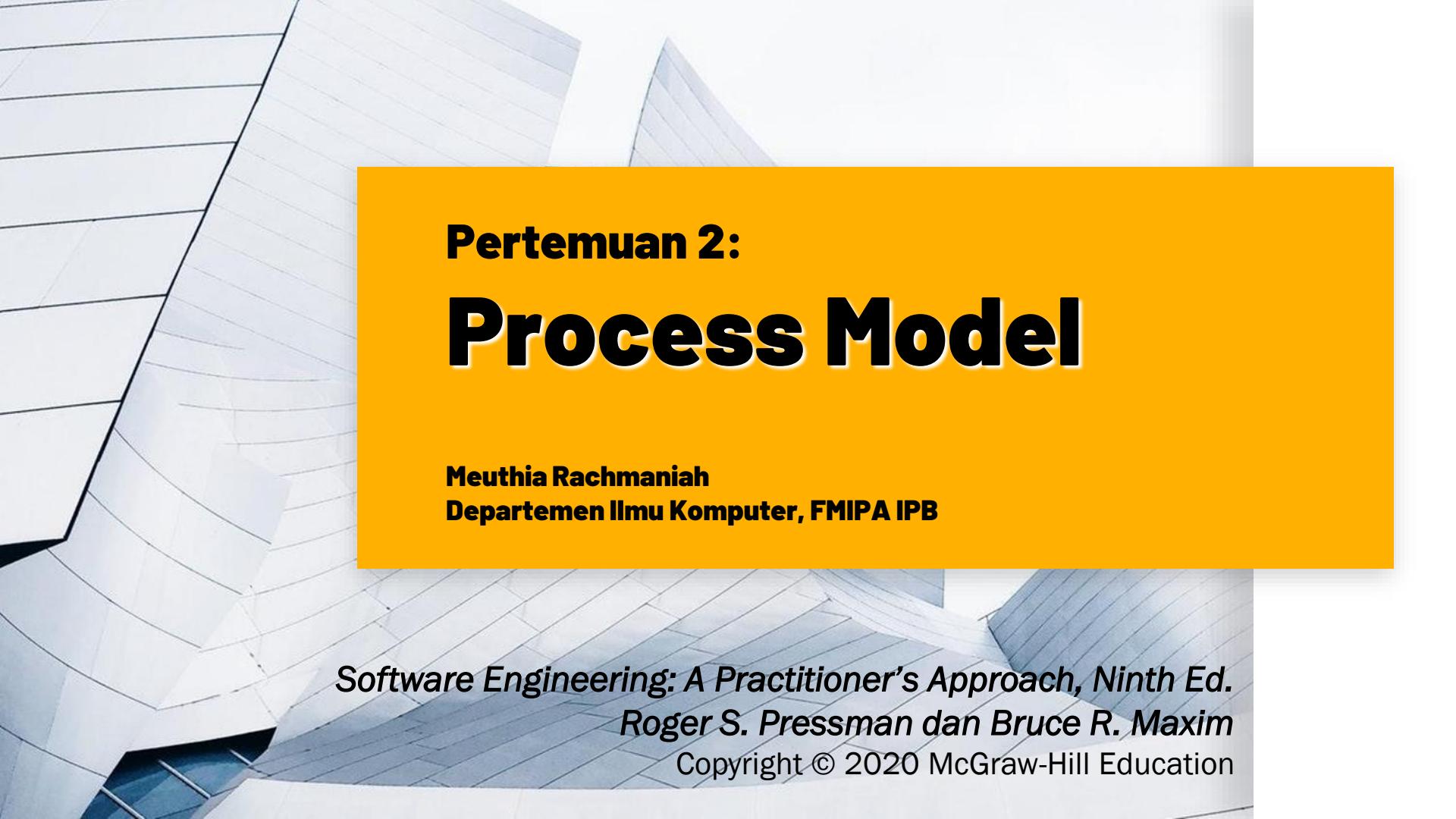
- PL adalah elemen kunci dalam evolusi sistem dan produk berbasis komputer dan salah satu teknologi terpenting di panggung dunia.
- Selama 60 tahun terakhir, PL telah berevolusi dari pemecahan masalah khusus dan alat analisis informasi menjadi industri itu sendiri.
- Namun masih terdapat kesulitan mengembangkan PL berkualitas tinggi tepat waktu dan sesuai anggaran.
- Perangkat lunak — program, data, dan informasi deskriptif — menangani beragam bidang teknologi dan aplikasi.
- PL lama terus menghadirkan tantangan khusus bagi mereka yang harus memeliharanya.
- RPL meliputi proses, metode, dan tools yang memungkinkan sistem berbasis komputer yang kompleks untuk dibangun secara tepat waktu dan berkualitas.
- Proses PL menggabungkan lima framework aktivitas — komunikasi, perencanaan, pemodelan, konstruksi, dan penyebaran (*deployment*) — yang berlaku untuk semua proyek PL.
- Praktik RPL adalah aktivitas pemecahan masalah yang mengikuti seperangkat prinsip inti.
- Saat Anda mempelajari lebih lanjut tentang RPL, Anda akan mulai memahami mengapa prinsip-prinsip ini harus dipertimbangkan saat memulai proyek PL apa pun

# Chapter 1: Perangkat Lunak dan Rekayasa Perangkat Lunak

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioners Approach*  
9<sup>th</sup> Edition, 2020

Roger S. Pressman  
Bruce R. Maxim

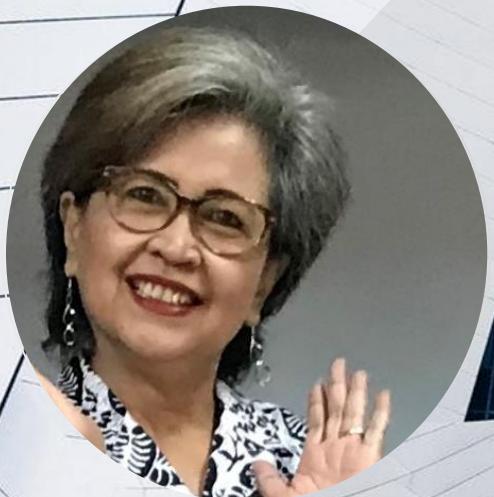


## **Pertemuan 2:**

# **Process Model**

**Meuthia Rachmaniah**  
**Departemen Ilmu Komputer, FMIPA IPB**

*Software Engineering: A Practitioner's Approach, Ninth Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education



# Hello!

Ir. Meuthia Rachmaniah, M.Sc.  
*Associate Professor*  
NIP 195907111984032006

Computer Science Department  
Software Engineering & Information Sciences  
[meuthiara@apps.ipb.ac.id](mailto:meuthiara@apps.ipb.ac.id)

SATYALANCANA KARYA SATYA 20 TAHUN  
SATYALANCANA KARYA SATYA 30 TAHUN



# Cakupan Materi

- 2.1 Model Proses Generik
- 2.2 Mendefinisikan *Framework Activity*
- 2.3 Mengidentifikasi Gugus Tugas (*Task Set*)
- 2.4 Penilaian dan Peningkatan Proses  
*(Process Assessment and Improvement)*
- 2.5 Model Proses Preskriptif
- 2.6 Product and Process
- 2.7 Rangkuman

# Quick Look

## WHAT IS IT?

- Saat membangun produk atau sistem, penting untuk mengikuti serangkaian langkah yang dapat diprediksi (peta jalan) yang membantu Anda mengirimkan produk berkualitas tinggi tepat waktu
- Peta jalan ini disebut "Software Process."



## WHO DOES IT

- Perekayasa PL menyesuaikan proses dengan kebutuhan klien dan kemudian mengikutinya
- Orang yang meminta PL juga memiliki peran dalam proses mendefinisikan, membangun, dan mengujinya



## HOW DO I ENSURE THAT I'VE DONE IT RIGHT

- Kualitas, ketepatan waktu, dan kelangsungan hidup jangka panjang dari produk yang dibangun adalah indikator terbaik dari keberhasilan proses yang digunakan



## WHAT ARE THE STEPS

- Proses yang Anda adopsi bergantung pada perangkat lunak yang Anda buat.
- Suatu proses mungkin sesuai untuk membuat perangkat lunak untuk sistem avionik pesawat tetapi mungkin tidak berfungsi dengan baik untuk pembuatan aplikasi seluler atau video game



## WHY IS IT IMPORTANT?

- Sebuah proses memberikan stabilitas, kontrol, dan organisasi pada suatu aktivitas sehingga tidak menjadi kacau.
- Proses RPL modern harus AGILE
- Harus mencakup aktivitas, kontrol, dan work product yang untuk tim proyek dan produk yang akan diproduksi



## WHAT IS THE WORK PRODUCT

- Work product ialah program, dokumen, dan data yang dihasilkan dari aktivitas dan task rekayasa yang dicakup dalam proses

# Social Learning Process (Proses Pembelajaran Sosial)

## Dialog

Untuk mengubah pengetahuan menjadi PL, diperlukan dialog antara pengguna dan perancang, antara perancang dan tools untuk membawa pengetahuan ke dalam PL



## Perangkat Lunak

PL adalah pengetahuan yang diwujudkan yang awalnya tersebar, tacit, dan tidak lengkap

## Rekayasa Perangkat Lunak

RPL pada dasarnya adalah proses pembelajaran sosial berulang, dan hasilnya adalah "kapital perangkat lunak" (Software Capital)

# Definisi Proses Perangkat Lunak

Proses  
Perangkat Lunak

Proses PL mendefinisikan pendekatan yang diambil saat PL direkayasa



## Framework

Framework adalah aktivitas, aksi(action), dan tugas (task) yang diperlukan untuk membangun PL berkualitas tinggi

## Proses PL vs RPL

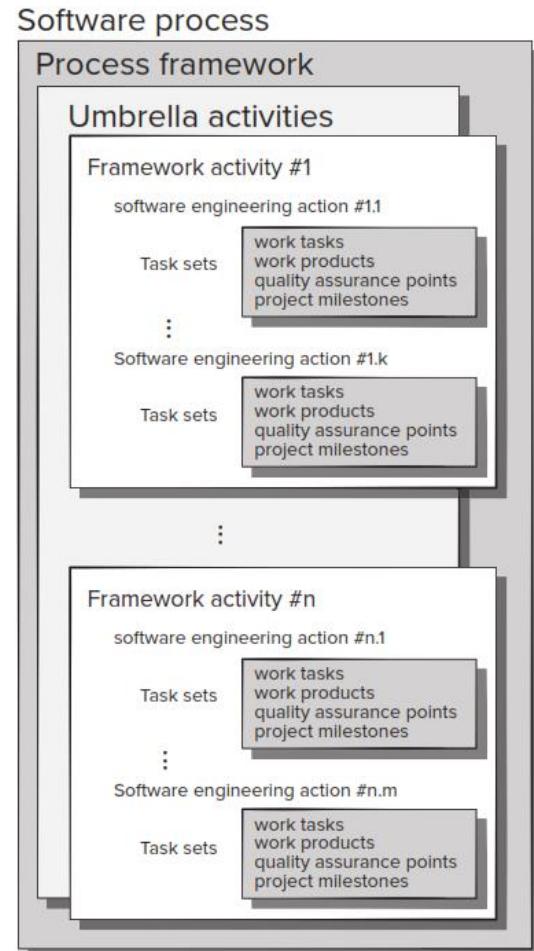
Proses PL tidak sama dengan RPL, yang juga mencakup teknologi yang mengisi proses—metode teknis dan tool otomatis (automated tools)



## 2.1 Model Proses Generik

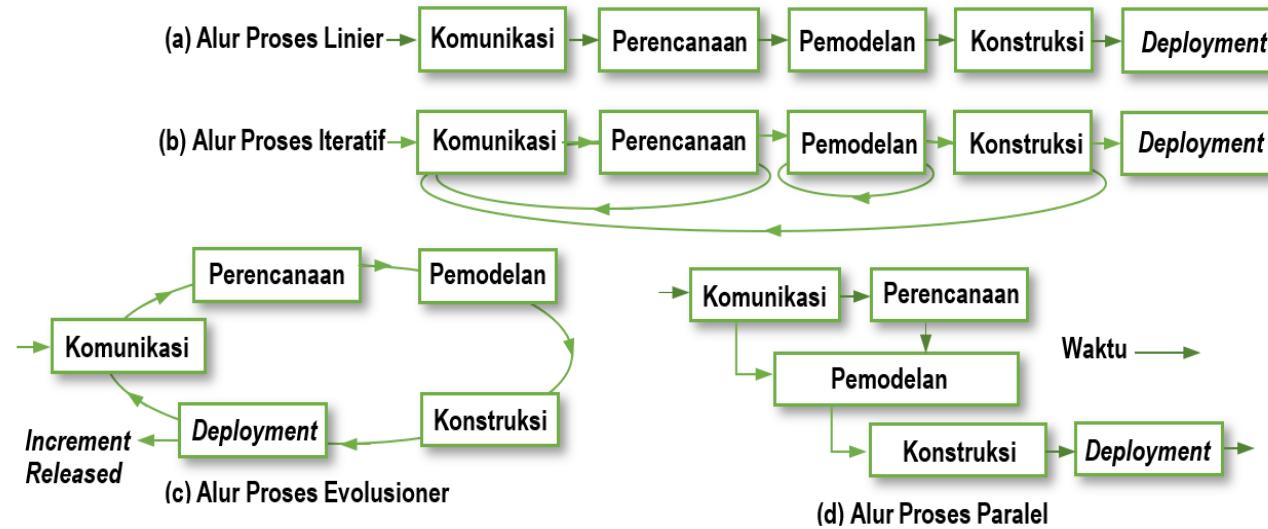
# Model Proses Generik

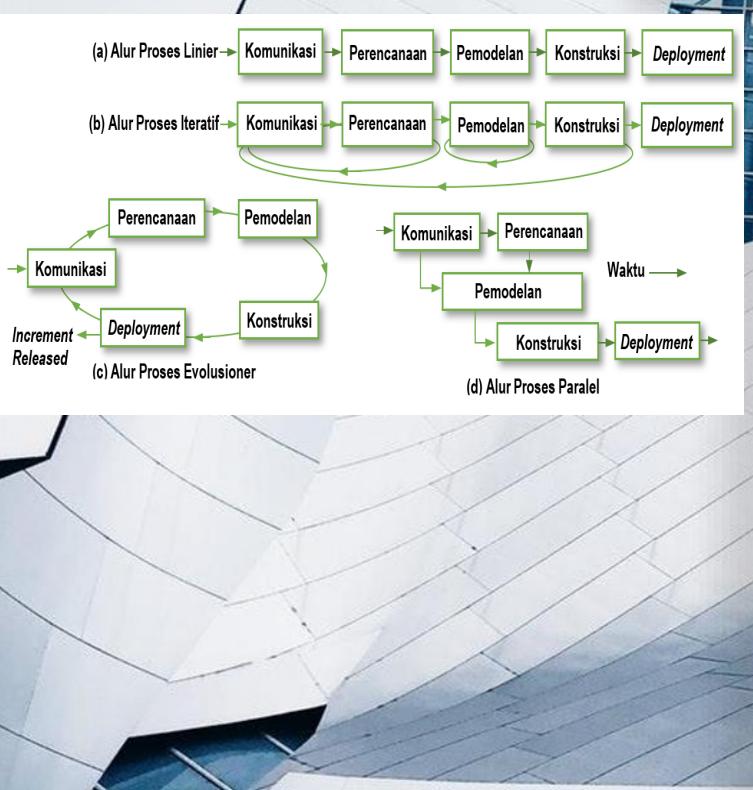
- Framework Proses generik untuk RPL mendefinisikan 5 aktivitas framework:
  - Komunikasi, Perencanaan, Pemodelan, Konstruksi, dan Penerapan (Deployment)
- Gugus Umbrella Activities yang terdiri dari:
  - *Project Tracking and Control, Manajemen Risiko, Quality Assurance, Manajemen Konfigurasi, Review Teknis, dan lainnya* yang diterapkan di seluruh proses
- Alur Proses (Process Flow):
  - Adalah bagaimana Framework Activity dan actions dan tasks yang terdapat dalam setiap aktivitas dikelola sesuai urutan dan waktunya



# Alur Proses (Process Flow)

Alur Proses (Process Flow) menjelaskan bagaimana Framework Activity dan Action serta Task yang terjadi dalam setiap Framework Activity diatur sesuai dengan urutan dan waktu





## ○ Alur Proses Linear

- Mengeksekusi setiap aktivitas dari lima aktivitas secara berurutan

## ○ Alur Proses Iteratif

- Mengulang satu atau lebih aktivitas sebelum berlanjut ke aktivitas berikutnya

## ○ Alur Proses Evolusioner

- Mengeksekusi aktivitas secara sirkular
- Setiap sirkular menuju ke versi perangkat lunak yang lebih lengkap

## ○ Alur Proses Paralel

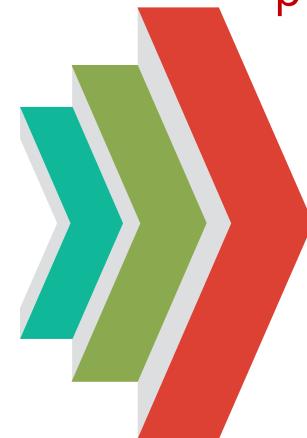
- Mengeksekusi satu atau lebih aktivitas secara paralel dengan aktivitas lainnya
- Misal: aspek dari pemodelan perangkat lunak dilakukan secara paralel dengan aspek lain dari perangkat lunak)



## 2.2 Mendefinisikan Framework Activity



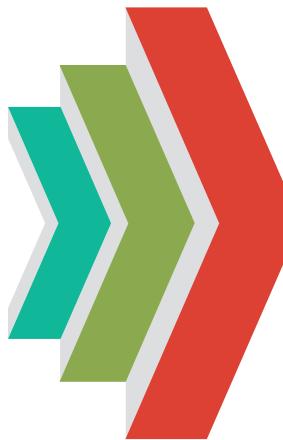
Tim PL akan  
membutuhkan  
**lebih banyak**  
**informasi** secara  
signifikan sebelum  
dapat menjalankan  
salah satu dari  
aktivitas dari  
proses PL dengan  
benar



- Harus dapat **menjawab**  
**pertanyaan:**
  - Action apa yang sesuai untuk Framework Activity, mengingat
    - **Sifat problem** yang harus dipecahkan,
    - **Karakteristik** orang yang melakukan pekerjaan, dan
    - **Stakeholder** yang mensponsori proyek?

## Small Project

- Untuk proyek PL kecil yang diminta oleh satu orang (di lokasi remote) dengan kebutuhan sederhana dan langsung,
  - Aktivitas komunikasi mungkin mencakup sedikit lebih dari panggilan telepon atau email dengan stakeholder yang meminta proyek



**Aktivitas:** Komunikasi

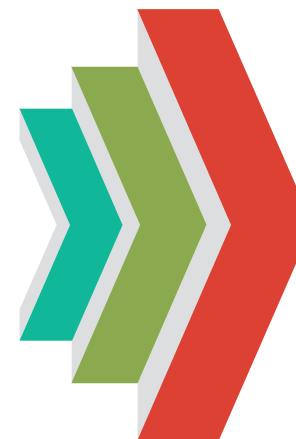
**Action:** Percakapan telepon

**Tasks:**

1. Kontak stakeholder dengan telepon
2. Diskusikan kebutuhan / persyaratan (*requirements*) dan buat catatan
3. Kelola catatan menjadi pernyataan tertulis singkat tentang kebutuhan
4. Email stakeholder agar direviu dan disetujui

## More Complex Project:

- Stakeholder banyak
- Setiap stakeholder berbeda kebutuhan/*requirements* (terkadang berbenturan), maka aktivitas Komunikasi terdiri dari 6 (enam) actions.
- Masing-masing Action mungkin memiliki banyak Task dan dalam beberapa kasus sejumlah Work Product yang berbeda



**Aktivitas:** Komunikasi  
**Action:**

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation

**Tasks:**

- 1.1 ...1.n (untuk Action 1)
- 1.2 ...2.n (untuk Action 2)

...



## 2.3 Mengidentifikasi Gugus Tugas (Tasks Set)



“

- Satu gugus task (tugas) mendefinisikan pekerjaan aktual yang harus dilakukan untuk mencapai tujuan dari Action
  - Daftar Task untuk dicapai
  - Daftar Work Product untuk dihasilkan
  - Daftar penjaminan mutu (*quality assurance*) yang harus diterapkan
  - *Project Milestone* yang dihasilkan
- **Proyek berbeda akan meminta task set yang berbeda**
  - Artinya, Action harus disesuaikan dengan kebutuhan spesifik proyek PL dan karakteristik tim proyek

# Task Set untuk **Simple Project** Banyak Stakeholder

## TASK SET

Action:  
ELICITATION

Aktivitas:  
KOMUNIKASI

- Komunikasi adalah salah satu aktivitas dari Framework Process
- Aktivitas Komunikasi memiliki lebih dari satu Action
  - Disebut juga sebagai *Requirement Gathering*
  - Tujuan Elicitation - memahami apa yang diinginkan berbagai stakeholder dari PL yang akan dibangun

1. Buat daftar stakeholder proyek
2. Undang para stakeholder ke informal meeting
3. Mintalah daftar fitur dan fungsi yang diinginkan setiap stakeholder
4. Diskusikan requirement dan buat daftar requirement final
5. Prioritisasi requirements
6. Catat area ketidakpastian

# Task Set untuk **BIG Project**

## TASK SET

Action:  
ELICITATION

Aktivitas:  
KOMUNIKASI

- Komunikasi adalah salah satu aktivitas dari Framework Process
- Aktivitas Komunikasi memiliki lebih dari satu Action
  - Disebut juga sebagai Requirement Gathering
  - Tujuan Elicitation - memahami apa yang diinginkan berbagai stakeholder dari PL yang akan dibangun

1. Buat daftar stakeholder proyek
2. Wawancarai setiap stakeholder secara terpisah untuk menentukan keseluruhan keinginan dan kebutuhan
3. Buat daftar awal fungsi dan fitur berdasarkan masukan stakeholder
4. Jadwalkan serangkaian pertemuan spesifikasi aplikasi yang difasilitasi
5. Melakukan pertemuan (meeting)
6. Hasilkan skenario pengguna informal sebagai bagian dari setiap pertemuan

7. Perbaiki skenario pengguna berdasarkan umpan balik stakeholder
8. Buat daftar revisi requirement stakeholder
9. Gunakan teknik penyebaran fungsi kualitas untuk memprioritaskan kebutuhan
10. Package requirement agar dapat dikirimkan secara bertahap
11. Perhatikan kendala dan batasan yang akan ditempatkan pada sistem
12. Diskusikan metode untuk memvalidasi sistem

- Task Set slide #16 dan #17 memenuhi "pengumpulan kebutuhan" (requirement gathering), tetapi sangat berbeda kedalaman dan tingkat formalitasnya.
- Tim PL memilih Task Set yang memungkinkan untuk mencapai tujuan untuk setiap Action dan tetap mempertahankan KUALITAS dan tetap AGILE



## 2.4 Penilaian dan Peningkatan Proses

*(Process Assessment and Improvement)*

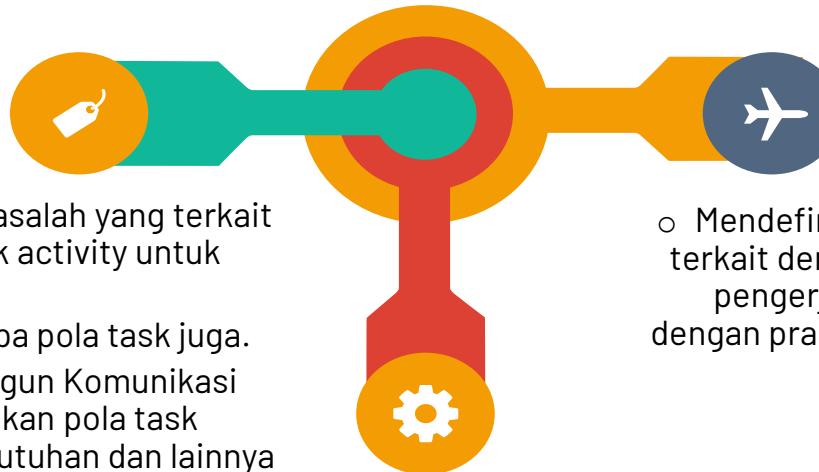
# Assesment dan Process Pattern

- Pola proses (process pattern) harus digabungkan dengan praktik RPL yang solid
- Proses dan aktivitas PL harus dinilai dengan menggunakan ukuran numerik atau analitik PL (metrik)
  - proses dapat dinilai untuk memastikan telah memenuhi serangkaian kriteria proses dasar yang telah terbukti penting untuk RPL yang sukses
- Pola Proses (Process Pattern)
  - Menggambarkan masalah terkait proses yang dihadapi selama pekerjaan RPL
  - Mengidentifikasi lingkungan di mana masalah telah dihadapi, dan
  - Menyarankan satu atau lebih solusi yang telah terbukti mengatasi masalah tersebut

# Tipe-tipe Process Pattern

## STAGE PATTERN

- Mendefinisikan masalah yang terkait dengan framework activity untuk proses tersebut.
- Mencakup beberapa pola task juga.
- Misalnya, Membangun Komunikasi akan menggabungkan pola task Pengumpulan kebutuhan dan lainnya



## TASK PATTERN

- Mendefinisikan masalah yang terkait dengan action RPL atau penggeraan task dan relevan dengan praktik RPL yang sukses

## PHASE PATTERN

- Mendefinisikan urutan framework activity yang terjadi dengan proses, bahkan ketika keseluruhan aliran aktivitas bersifat iteratif
- Contohnya termasuk Sprial Model atau Prototyping

# Contoh Process Pattern

- Menjelaskan pendekatan yang dapat diterapkan ketika stakeholder memiliki gagasan umum tentang apa yang harus dilakukan tetapi tidak yakin dengan kebutuhan PL tertentu
- **PATTERN NAME.** kebutuhan (Requirement) Tidak Jelas
- **INTENT.** Pola ini menggambarkan pendekatan untuk membangun model yang dapat dinilai secara iteratif oleh stakeholder dalam upaya mengidentifikasi atau memantapkan kebutuhan PL
- **TYPE.** Phase pattern
- **INITIAL CONTEXT.** Kondisi yang harus dipenuhi (1) Stakeholder telah diidentifikasi; (2) Moda komunikasi antara stakeholder dan tim PL telah ditetapkan; (3) Masalah PL utama yang harus dipecahkan telah diidentifikasi oleh para stakeholder; (4) Pemahaman awal tentang ruang lingkup proyek, kebutuhan bisnis dasar, dan kendala proyek telah dikembangkan
- **PROBLEM.** Requirements samar-samar atau tidak ada. pemangku kepentingan tidak yakin dengan apa yang mereka inginkan.
- **SOLUTION.** Deskripsi proses pembuatan prototipe akan disajikan di sini.
- **RESULTING CONTEXT.** Prototipe PL yang mengidentifikasi kebutuhan dasar. (moda interaksi, fitur komputasi, fungsi pemrosesan) disetujui oleh stakeholder. Setelah ini, 1. Prototipe ini dapat berkembang melalui serangkaian peningkatan menjadi PL produksi atau 2. Prototipe dapat dibuang.
- **RELATED PATTERNS.** CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.

# Penilaian dan Peningkatan Proses

(Process Assessment and Improvement)

01

## SCAMPI

- Standard CMMI Assessment Method for Process Improvement (SCAMPI)
- Menyediakan model penilaian proses lima langkah yang menggabungkan lima fase: mulai, mendiagnosa, menetapkan, bertindak, dan belajar

02

## SPICE

- SPICE—The SPICE (ISO/IEC15504) Standard
- Mendefinisikan satu set kebutuhan untuk penilaian proses PL.
- Maksud dari standar ini adalah untuk membantu organisasi dalam mengembangkan evaluasi objektif dari kemanjuran setiap proses PL yang ditentukan

03

## CBA IPI

- CMM-Based Appraisal for Internal Process Improvement (CBA IPI)
- Menyediakan teknik diagnostik untuk menilai kematangan relatif dari organisasi PL;
- Menggunakan SEI CMM sebagai dasar penilaian

04

## ISO 9001:2000

- ISO 9001:2000 for Software
- Standar generik yang berlaku untuk organisasi mana pun yang ingin meningkatkan kualitas keseluruhan produk, sistem, atau layanan yang disediakannya.
- Standar ini secara langsung dapat diterapkan pada organisasi dan perusahaan PL



## 2.5 Model Proses Preskriptif

- 2.5.1 Model Waterfall
- 2.5.2 Model Incremental
- 2.5.3 Model Proses Prototyping
- 2.5.4 Model Proses Evolusioner
- 2.5.5 Model Concurrent
- 2.5.5 Model Proses Unified

# Model Proses Preskriptif



Model proses preskriptif mendefinisikan satu set elemen proses yang telah ditentukan sebelumnya dan alur kerja proses yang dapat diprediksi



Model proses preskriptif meminta untuk struktur dan ketertiban dalam pengembangan PL



Kegiatan dan task terjadi secara berurutan dengan pedoman yang ditetapkan untuk kemajuan



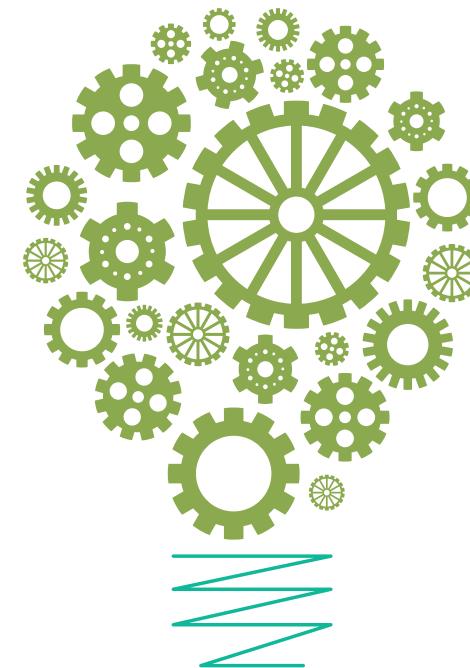
Setiap model proses juga menentukan alur proses (juga disebut alur kerja) – yaitu, cara elemen proses saling terkait satu sama lain



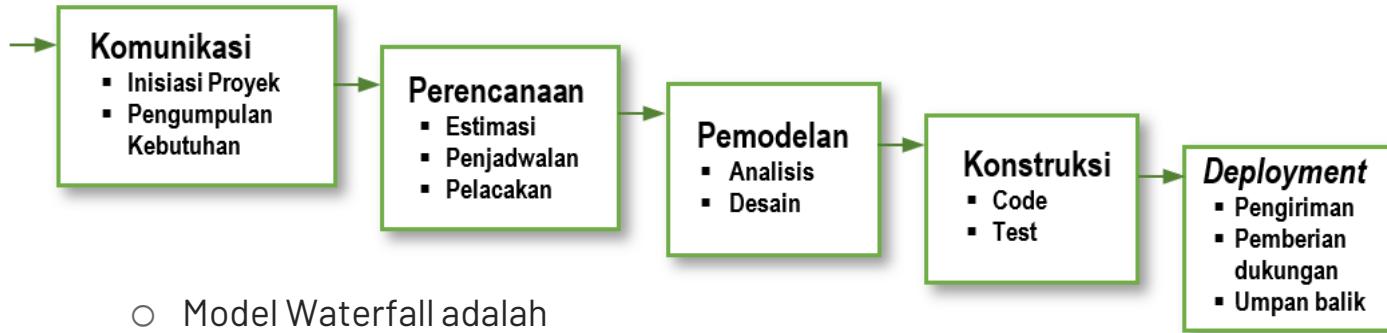
Disebut "preskriptif" karena meresepkan satu set elemen proses - aktivitas framework, action RPL, task, produk kerja, jaminan kualitas, dan mekanisme kontrol perubahan untuk setiap proyek

# Model Proses Preskriptif

- 2.5.1 Model Waterfall
  - a. Model Waterfall
  - b. Model V
- 2.5.2 Model Incremental
- 2.5.3 Model Proses Prototyping
- 2.5.4 Model Proses Evolusioner
  - a. Prototyping
  - b. Spiral Model
- 2.5.5 Model Concurrent
- 2.5.6 Model Proses Unified



## 2.5.1.a Model Waterfall



- Model Waterfall adalah **paradigma tertua** pada rekayasa perangkat lunak
- Model waterfall dapat berfungsi sebagai model proses pada situasi di mana **kebutuhan telah jelas** ditetapkan sehingga pekerjaan dapat dilanjutkan hingga tahap penyelesaian secara linier
- Model Waterfall dipilih apabila kebutuhan-kebutuhan **terdefinisi baik** dan **cukup stabil**, dan mengarah ke tipe **alur proses linier**, yaitu dimulai dengan komunikasi, perencanaan, pemodelan, konstruksi, dan diakhiri dengan *deployment*

# Permasalahan pada Model Waterfall

## Klien Sulit Menyatakan Requirements

Klien seringkali **sulit menyatakan semua kebutuhan** secara eksplisit. Padahal model waterfall membutuhkan kepastian pada saat awal proyek pengembangan sistem dimulai



## Jarang Linier dan Sulit Iterasi

Dalam realitanya, pengembangan sistem jarang mengikuti aliran sekuensial yang diusulkan model. Meskipun model linier dapat mengakomodasi iterasi, namun ini dilakukan secara tidak langsung, sehingga perubahan tak langsung ini dapat menyebabkan kebingungan saat tim proyek melanjutkan

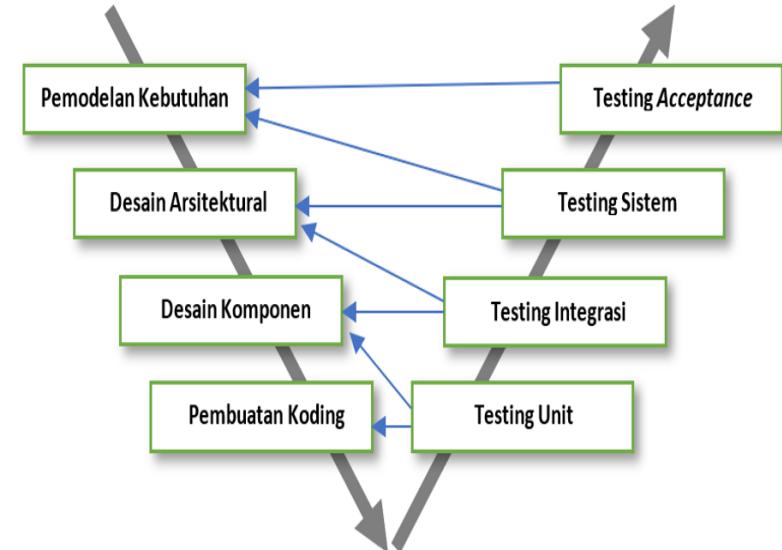


## Blocking pada Aktivitas Sebelumnya

Terjadi *blocking state*, yaitu beberapa anggota tim proyek harus menunggu anggota tim lainnya untuk menyelesaikan tasks. Padahal, waktu yang dihabiskan untuk menunggu bisa melebihi waktu yang dihabiskan untuk pekerjaan produktif. *Blocking state* cenderung lebih umum terjadi di awal dan akhir proses sekuensial linier

## 2.5.1.b Model V

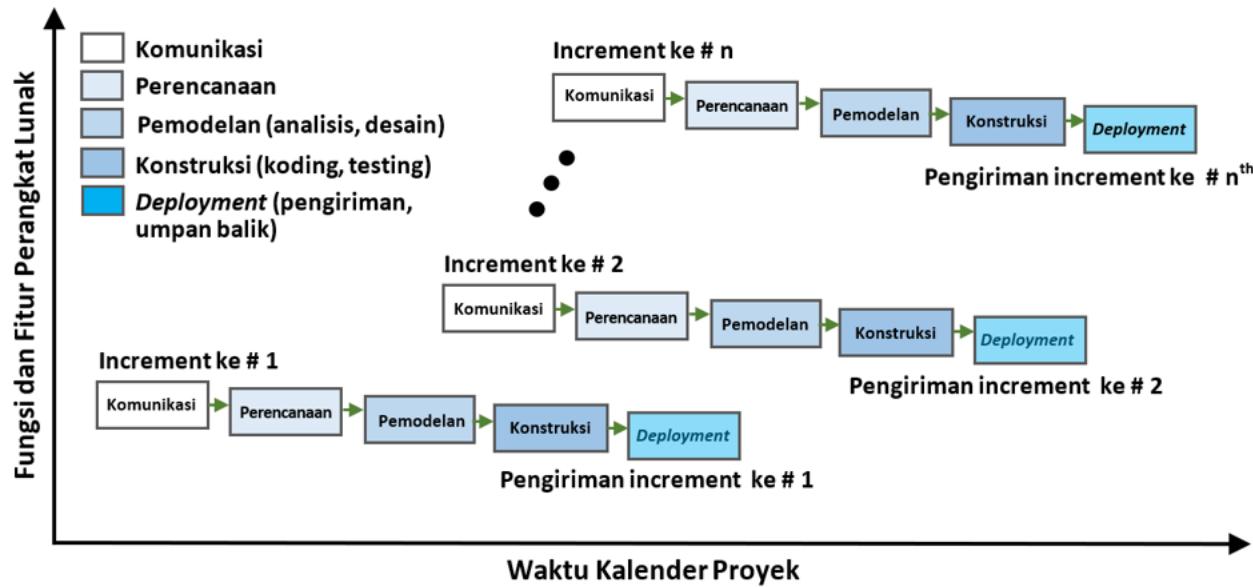
- Variasi model Waterfall yang menggambarkan hubungan action jaminan kualitas dengan action yang terkait dengan komunikasi, pemodelan, dan pengaktifan konstruksi kode awal
- Tim pertama-tama bergerak ke sisi kiri V untuk memperbaiki kebutuhan masalah.
- Setelah kode dibuat, tim bergerak ke sisi kanan V, melakukan serangkaian tes yang memvalidasi setiap model yang dibuat saat tim bergerak ke sisi kiri.
- Pada kenyataannya, tidak ada perbedaan mendasar antara siklus hidup klasik dan model V.
- Dalam hal ini, model-V menyediakan cara untuk memvisualisasikan bagaimana verifikasi dan action validasi diterapkan pada awal pengembangan sistem



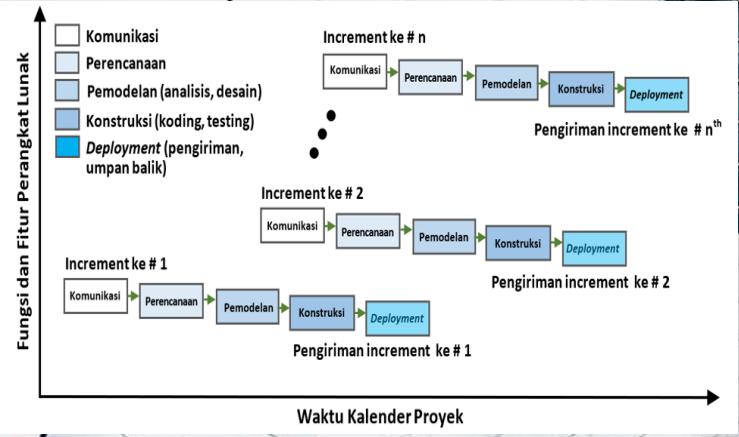
Perangkat Lunak yang Dapat Dieksekusi

## 2.5.2 Model Incremental

- Model Proses *Incremental* dipilih ketika kebutuhan awal didefinisikan dengan cukup baik, tetapi keseluruhan ruang lingkup upaya pengembangan menghalangi proses yang murni linier
- Terdapat kebutuhan yang mendesak untuk memperluas rangkaian fungsi baru yang terbatas ke rilis sistem yang lebih baru.



## 2.5.2 Model Incremental

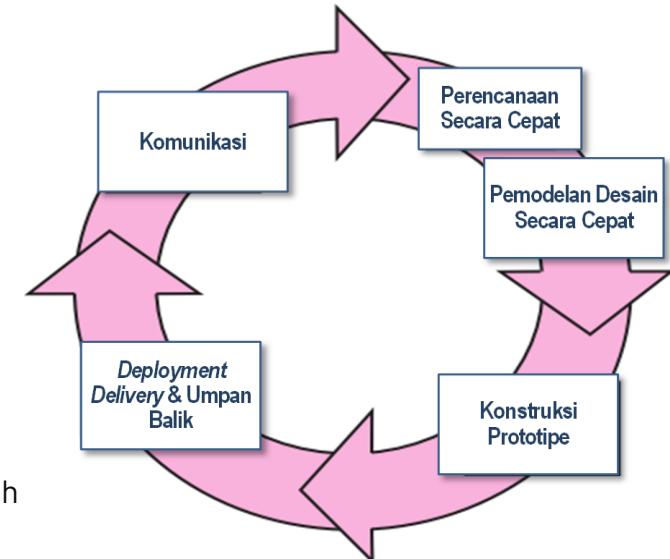


- Model Proses *Incremental* menggabungkan elemen aliran proses linier dan paralel
- Model *Incremental* menerapkan sekvens linear dalam *staggered fashion* sesuai waktu kalender berlangsung.
- Setiap urutan linier menghasilkan *deliverable increment* perangkat lunak dan dapat disampaikan kepada klien.
- *Increment* pertama sering merupakan produk inti dengan banyak fitur tambahan.
- Klien menggunakan dan mengevaluasi *increment* pertama.
- Klien juga memberikan umpan balik dengan lebih banyak modifikasi untuk lebih memenuhi kebutuhan.
- Aktivitas yang sama berlangsung sampai dengan *increment* ke-n

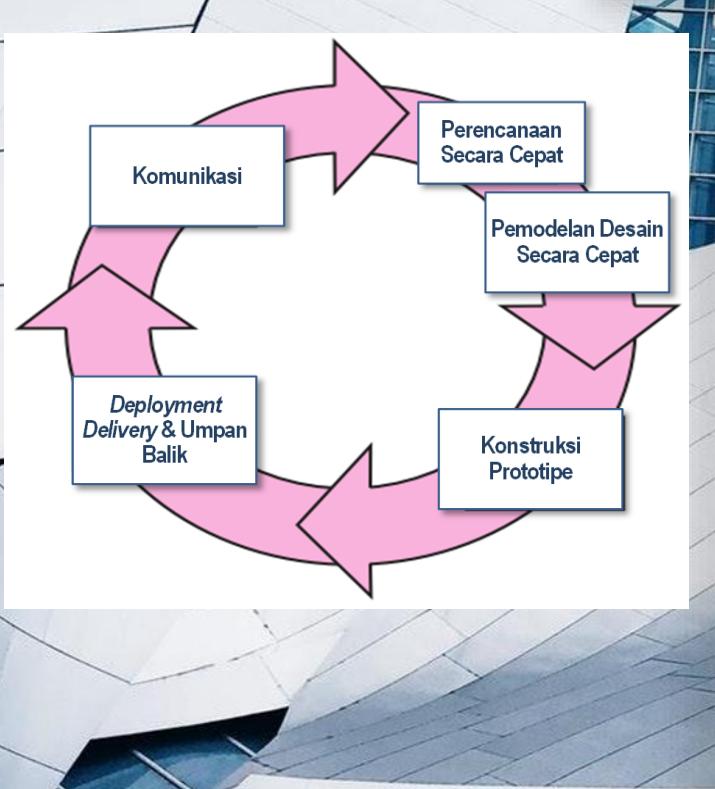
## 2.5.3 Prototyping Paradigm

### ❑ KAPAN DIGUNAKAN:

- Pelanggan menetapkan serangkaian tujuan umum tetapi tidak mengidentifikasi kebutuhan terperinci untuk fungsi dan fitur, atau
- Pengembang mungkin tidak yakin dengan efisiensi suatu algoritme, bentuk yang harus diambil oleh interaksi komputer manusia
- ❑ Baik pemangku kepentingan dan insinyur perangkat lunak menyukai paradigma prototyping.
- ❑ Pengguna dapat merasakan sistem yang sebenarnya, dan pengembang dapat segera membangun sesuatu.
- ❑ Namun, para insinyur dapat membuat kompromi untuk membuat prototipe bekerja dengan cepat.
- ❑ Pilihan yang kurang ideal dapat diadopsi selamanya setelah Anda terbiasa
- ❑ Iterasi akan terjadi sampai berbagai kebutuhan stakeholder dipenuhi. Pada saat bersamaan, tim pengembang juga akan lebih memahami apa yang perlu dilakukan



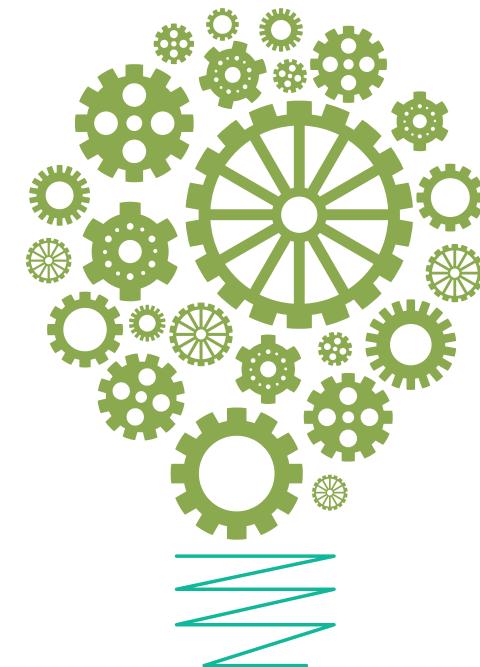
## Tahapan Prototyping Paradigm



- ❑ **KOMUNIKASI.** Meeting dengan stakeholder untuk menentukan tujuan, mengidentifikasi kebutuhan apa pun yang diketahui, garis besar area di mana definisi lebih lanjut adalah wajib
- ❑ **PERENCANAAN CEPAT.** Rencana cepat untuk pembuatan prototipe dan pemodelan (desain cepat) terjadi
- ❑ **PEMODELAN DESAIN CEPAT.** Berfokus pada representasi dari aspek-aspek perangkat lunak yang akan terlihat oleh pengguna akhir. (antarmuka/interface dan keluaran)
- ❑ **KONSTRUKSI PROTOTIPE.** Konstruksi prototipe yang akan digunakan dan dievaluasi
- ❑ **DEPLOYMENT DELIVERY & UMPAM BALIK.** Komentar pemangku kepentingan akan digunakan untuk menyempurnakan kebutuhan (requirements)

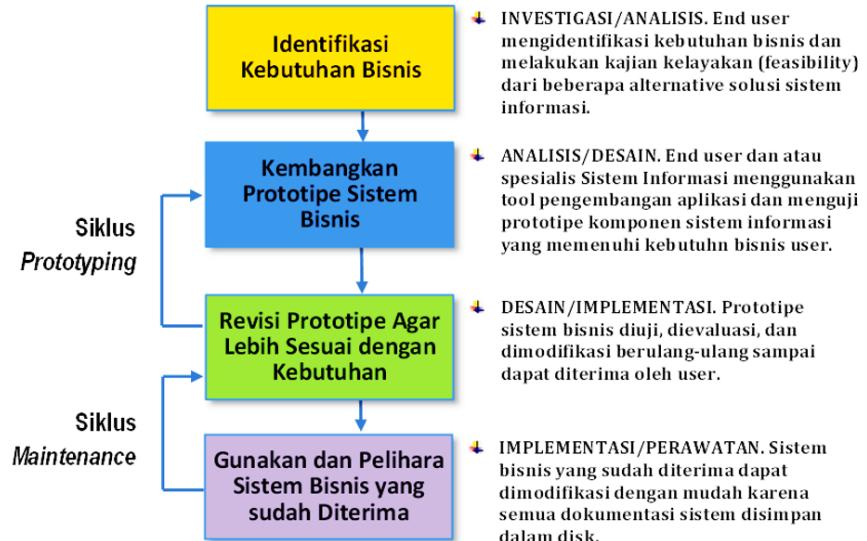
## 2.5.4. Model Proses Evolusioner

- Sistem PL berkembang dari waktu ke waktu akibat kebutuhan sering berubah sebagai hasil pengembangan. Dengan demikian, garis lurus ke produk akhir yang lengkap tidak mungkin. Namun, versi terbatas harus dikirimkan untuk memenuhi tekanan persaingan
- Biasanya satu set produk inti atau kebutuhan sistem dipahami dengan baik, tetapi detail dan ekstensinya belum ditentukan
- Anda memerlukan model proses yang telah dirancang secara eksplisit untuk mengakomodasi produk yang berkembang dari waktu ke waktu
- Dilakukan iteratif yang memungkinkan Anda untuk mengembangkan versi PL yang semakin lengkap
- Dua jenis diperkenalkan, yaitu **model Prototyping** dan **Model Spiral**

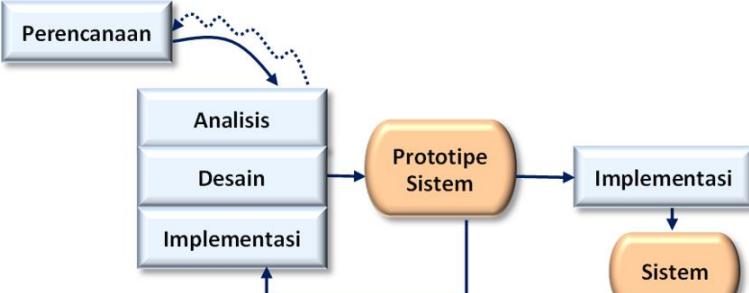


## 2.5.4.a Model Proses Evolusioner: PROTOTYPING

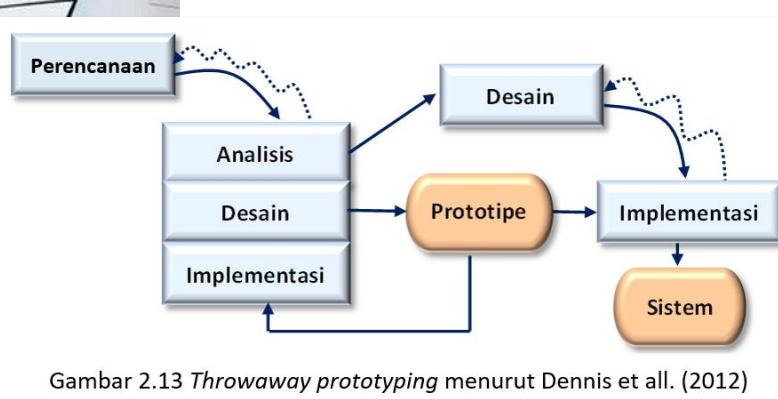
- Prototype dapat dibangun dengan lebih cepat dengan menggunakan fragmen program atau tool (misal: *report generator* dan *windows manager*).
- Prototype dapat juga dianggap sebagai “*the first system*” yang pada akhirnya tidak akan digunakan, atau disebut “*throw away*”.
- Namun, karena prototype terus diperbaiki sesuai iterasinya, maka pada akhirnya secara **EVOLUSIONER PROTOTYPE** akan menjadi sistem yang sebenarnya (*actual system*).



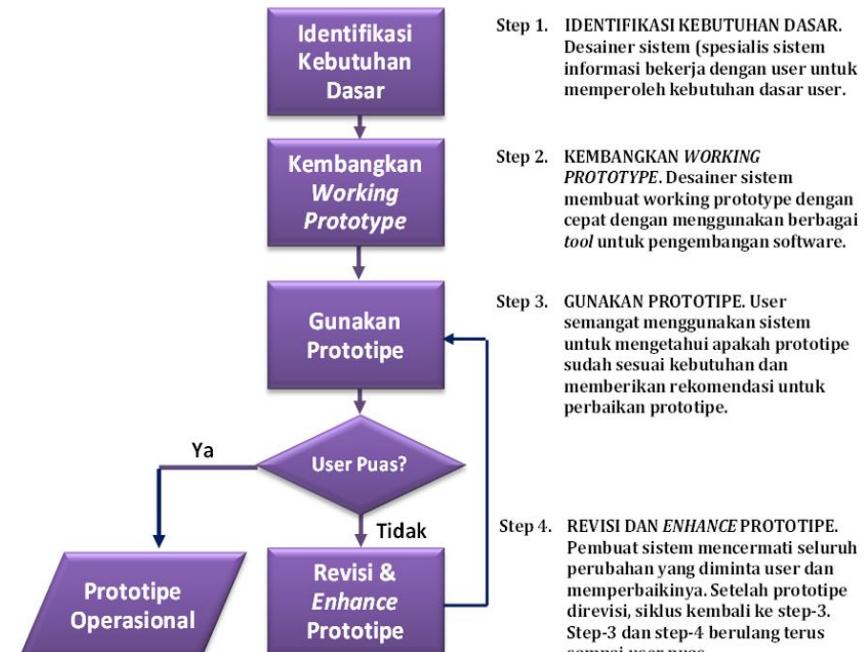
Gambar 2.10 Prototipe Pengembangan aplikasi dengan prototyping, dimana prototyping mengkombinasikan tahapan dalam SDLC (O'Brien & Marakas, 2011)



Gambar 2.11 Prototipe Sistem menurut Dennis et all. (2012)



Gambar 2.13 Throwaway prototyping menurut Dennis et all. (2012)



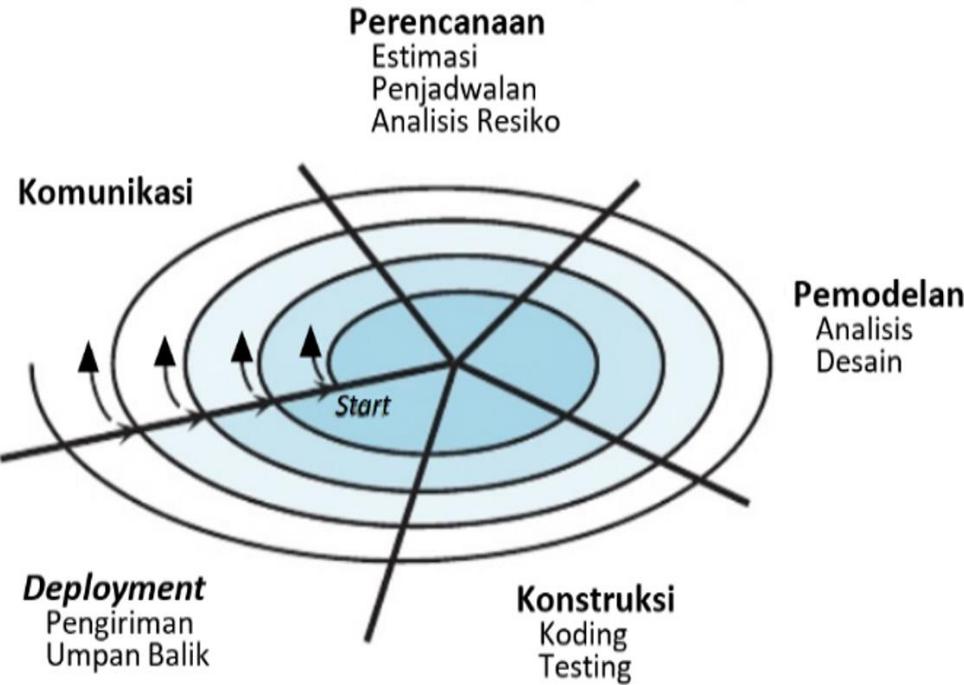
Gambar 2.12 Prototipe Sistem menurut Laudon & Laudon (2014)

## 2.5.4.b Model Proses Evolusioner: SPIRAL MODEL

- Menggabungkan sifat iteratif prototyping dengan aspek terkontrol dan sistematis dari model Waterfall dan merupakan generator model proses yang digerakkan oleh risiko yang digunakan untuk memandu rekayasa serentak multi-stakeholder dari sistem intensif PL.
- Dua fitur pembeda utama:
  1. Pendekatan siklus untuk secara bertahap menumbuhkan tingkat definisi dan implementasi sistem sambil mengurangi tingkat risikonya.
  2. Serangkaian **anchor point milestone** untuk memastikan komitmen stakeholder terhadap solusi sistem yang layak dan saling memuaskan
- Serangkaian rilis evolusi disampaikan. Selama iterasi awal, rilis mungkin berupa model atau prototipe. Selama iterasi selanjutnya, versi yang lebih lengkap dari sistem rekayasa diproduksi

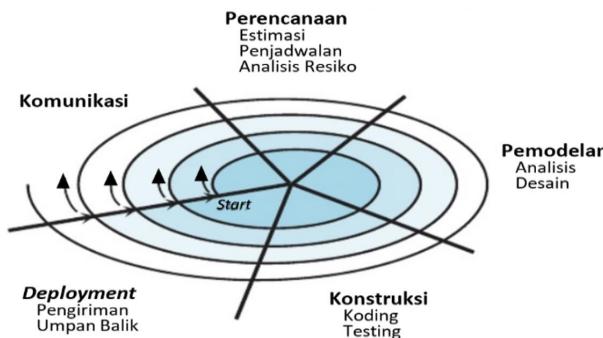


## 2.5.4.b Model Proses Evolusioner: SPIRAL MODEL



- Sirkuit pertama searah jarum jam mungkin menghasilkan **spesifikasi produk**;
- Lintasan berikutnya di sekitar spiral dapat digunakan untuk mengembangkan **prototipe** dan kemudian versi **perangkat lunak** yang semakin canggih.
- Setiap lulus menghasilkan penyesuaian terhadap rencana proyek.
- Biaya dan jadwal disesuaikan berdasarkan umpan balik.
- Juga, jumlah iterasi akan disesuaikan oleh manajer proyek

## 2.5.4.b Model Proses Evolusioner: SPIRAL MODEL

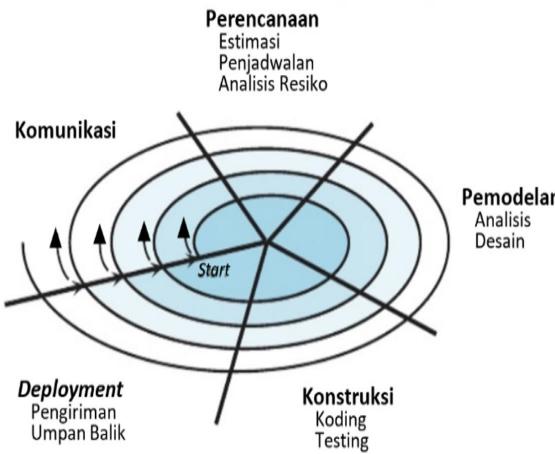


Baik untuk mengembangkan sistem skala besar saat PL berkembang seiring dengan adanya kemajuan proses dan risiko harus dipahami dan ditanggapi dengan benar. Prototyping digunakan untuk mengurangi risiko



Namun, mungkin sulit untuk meyakinkan pelanggan bahwa itu dapat dikontrol karena menuntut keahlian penilaian risiko yang cukup besar

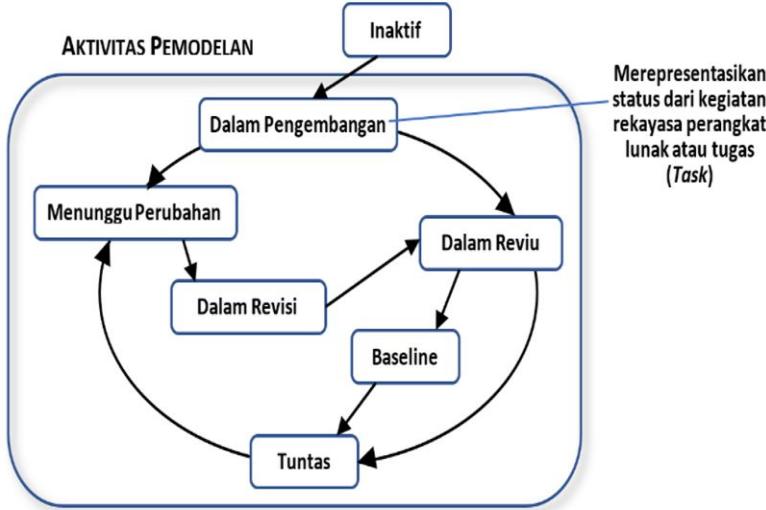
# Tiga Kekhawatiran tentang Proses Evolusi



- 1) Pembuatan prototipe menimbulkan masalah bagi perencanaan proyek karena jumlah siklus yang tidak pasti yang diperlukan untuk membangun produk
- 2) Tidak menetapkan kecepatan maksimum evolusi. Jika evolusi terjadi terlalu cepat, tanpa periode relaksasi, dapat dipastikan proses tersebut akan jatuh ke dalam chaos. Di sisi lain jika kecepatannya terlalu lambat maka produktivitas bisa terpengaruh
- 3) Proses PL harus difokuskan pada fleksibilitas dan ekstensibilitas daripada kualitas tinggi. Kita harus memprioritaskan kecepatan pengembangan di atas nol cacat. Memperluas pengembangan untuk mencapai kualitas tinggi dapat mengakibatkan keterlambatan pengiriman produk ketika peluang yang muncul telah menghilang

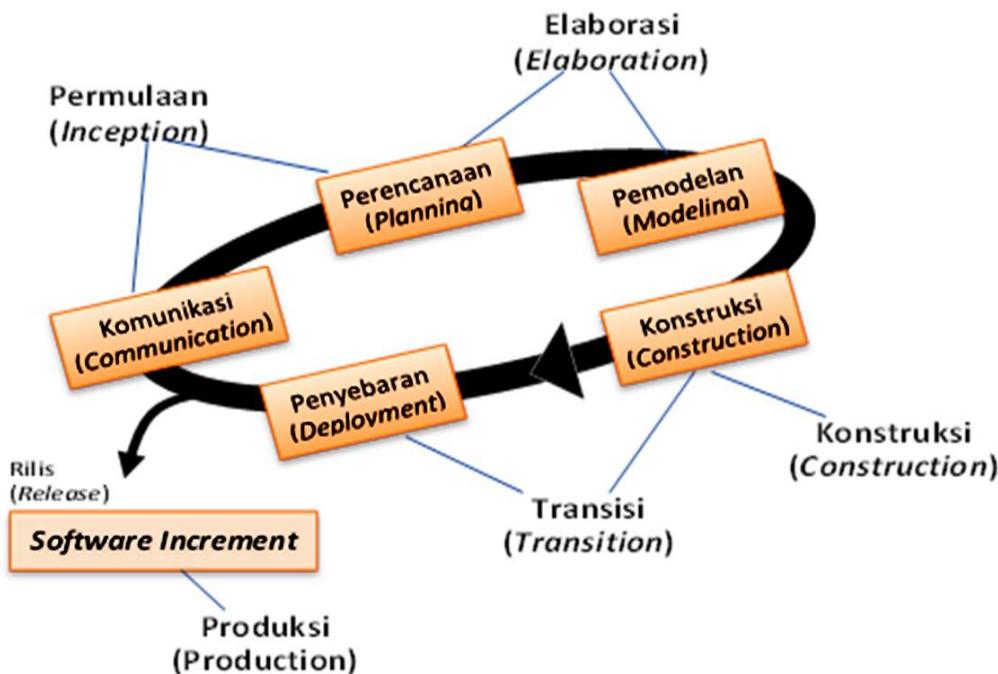
## 2.5.5 Model Concurrent

- ❖ Model konkuren memungkinkan tim PL untuk melakukan elemen iteratif dan konkuren dari salah satu model proses.
  - Misalnya, aktivitas pemodelan yang ditentukan untuk model spiral diselesaikan dengan menerapkan satu atau lebih action berikut: pembuatan prototipe, analisis, dan desain
- ❖ Gambar di samping menunjukkan pemodelan pada salah satu state pada waktu tertentu. Misalnya,
  - aktivitas komunikasi telah menyelesaikan iterasi pertama dan dalam state menunggu perubahan.
  - Aktivitas pemodelan berada dalam state inaktif, dan sekarang membuat transisi ke state pengembangan.
  - Jika konsumen menunjukkan perubahan dalam kebutuhan, aktivitas pemodelan bergerak dari state pengembangan yang sedang dikembangkan ke state menunggu perubahan



- **Model konkuren dapat diaplikasi pada seluruh tipe pengembangan PL dan memberikan gambaran akurat dari status terkini pada proyek.**
- **Model konkuren mendefinisikan jaringan proses, jadi tidak membatasi kegiatan RPL, actions dan tasks ke urutan events.**
- **Setiap aktivitas, aksi, atau tugas pada jaringan ada bersamaan dengan aktivitas-aktivitas, actions dan tasks lainnya.**
- **Events yang dihasilkan pada satu titik akan memicu transisi di antara state**

## 2.5.6 Model Proses Unified (Unified Process-UP)



Unified Process (UP) adalah proses perangkat lunak ‘*use-case driven, architecture-centric, iteratif, dan incremental*’ yang selaras dengan *Unified Modeling Language* (UML) untuk memodelkan dan mengembangkan sistem berorientasi objek secara iteratif dan *incremental*. UP menerapkan fase-fase **aktivitas generik**, yaitu komunikasi, perencanaan, pemodelan, konstruksi, dan delivery.

# Work Products UP

## Fase Awal

- Dokumen Visi
- Modal Use-Case awal
- Glossary Proyek Awal
- Case Bisnis Awal
- Penilaian Resiko Awal
- Rencana Proyek, Fase-fase, dan Iterasi
- Model Bisnis, kalau diperlukan
- Satu atau lebih Prototipe1

## Fase Elaborasi

- Model Use-Case
- Tambahan Kebutuhan, termasuk Kebutuhan Non Fungsional
- Model Analisis
- Deskripsi Arsitektur Perangkat Lunak
- Prototipe Arsitektur yang Dapat Dieksekusi
- Model Desain Pendahuluan
- Daftar Risiko yang Direvisi
- Rencana Proyek, termasuk Rencana Iterasi, alur kerja milestone yang sudah diadaptasi, dan teknis Produk Kerja
- User Manual Awal

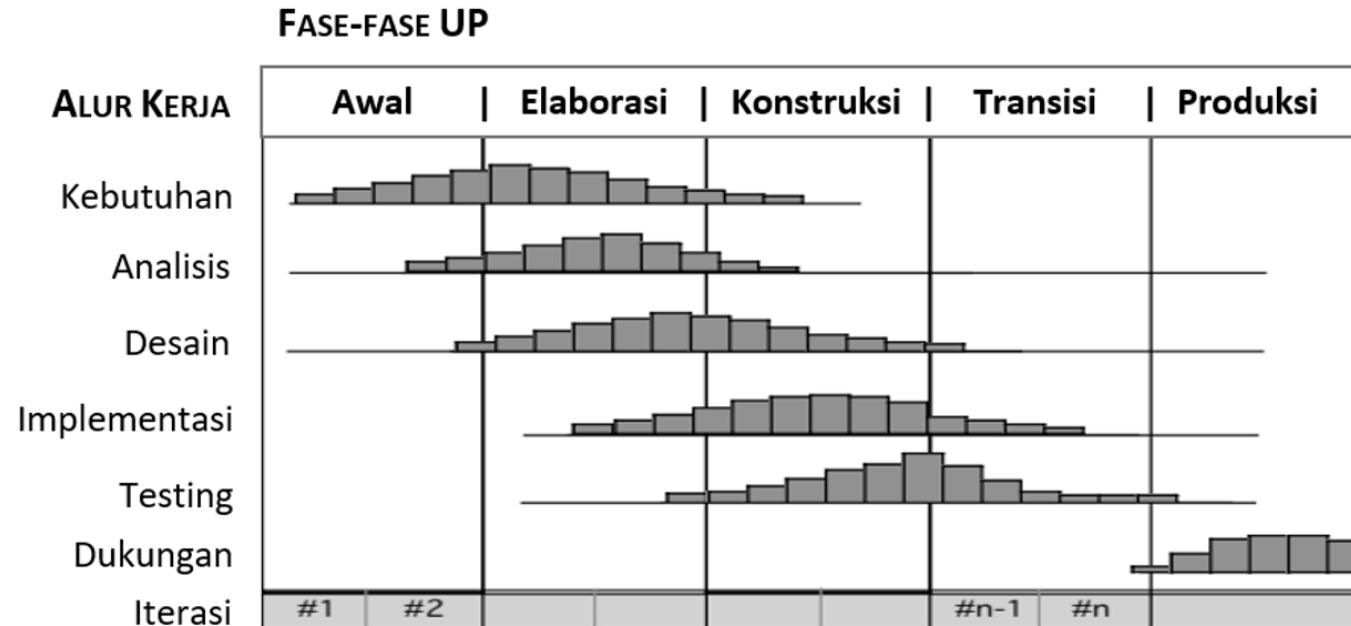
## Fase Konstruksi

- Model Desain
- Perangkat lunak Komponen
- Perangkat lunak Increment Terintegrasi
- Rencana dan Prosedur Testing
- Test Cases
- Dokumentasi Pendukung: User Manual, Manual Instalasi, Deskripsi Increment Saat Ini

## Fase Transisi

- Perangkat Lunak Increment yang diserahkan
- Laporan Beta Test
- Umpan Balik Pengguna secara Umum

# Fase-fase UP dan alur kerja secara staggered concurrency



Ada kemungkinan bahwa pada saat fase konstruksi, fase transisi, dan fase produksi dilaksanakan ternyata pekerjaan untuk perangkat lunak *increment* berikutnya sudah dimulai. Hal ini berarti bahwa fase-fase UP tidak terjadi secara berurutan namun secara bersamaan (*staggered concurrency*).



## 2.6 Produk dan Proses

# Kekuatan dan Kelemahan Model Proses

- Pada kenyataannya tidak ada proses yang sempurna untuk setiap proyek
- Biasanya tim PL mengadaptasi satu atau lebih model proses yang dibahas dalam 2.5 atau model proses agile yang akan dibahas di Pertemuan 3 untuk memenuhi kebutuhan mereka akan proyek yang ada.
- Jika prosesnya lemah, produk akhir pasti akan menderita.
- Tetapi ketergantungan yang berlebihan pada proses juga berbahaya.
- Orang memperoleh (atau lebih) kepuasan dari proses kreatif seperti yang mereka lakukan dari produk akhir.
- Sebagai seorang profesional perangkat lunak yang kreatif, Anda juga harus mendapatkan kepuasan dari prosesnya seperti halnya produk akhir.
- Dualitas produk dan proses merupakan salah satu elemen penting dalam menjaga orang-orang kreatif tetap terlibat karena rekayasa perangkat lunak terus berkembang



## 2.7 Rangkuman



Model Proses	PROS	CONS
WATERFALL	<ul style="list-style-type: none"><li>❖ Mudah dipahami dan direncanakan</li><li>❖ Berfungsi untuk proyek kecil yang dipahami dengan baik</li><li>❖ Analisis dan pengujian sangat mudah</li></ul>	<ul style="list-style-type: none"><li>❑ Itu tidak mengakomodasi perubahan dengan baik.</li><li>❑ Pengujian terjadi di akhir proses</li><li>❑ Persetujuan pelanggan ada di akhir</li></ul>
PROTOTYPING	<ul style="list-style-type: none"><li>❖ Ada pengurangan dampak perubahan requirements</li><li>❖ Pelanggan terlibat lebih awal dan sering</li><li>❖ Bekerja dengan baik untuk proyek-proyek kecil</li><li>❖ Kemungkinan penolakan produk berkurang</li></ul>	<ul style="list-style-type: none"><li>❑ Keterlibatan pelanggan dapat menyebabkan penundaan</li><li>❑ Mungkin ada godaan untuk "mengirimkan" prototype</li><li>❑ Pekerjaan hilang dalam throwaway prototipe</li><li>❑ Sulit untuk merencanakan dan mengelola</li></ul>



Model Proses	PROS	CONS
SPIRAL	<ul style="list-style-type: none"><li>❖ Ada keterlibatan pelanggan yang berkelanjutan</li><li>❖ Risiko pengembangan dikelola</li><li>❖ Sangat cocok untuk proyek besar dan kompleks</li><li>❖ Bekerja dengan baik untuk produk yang dapat diperluas</li></ul>	<ul style="list-style-type: none"><li><input type="checkbox"/> Kegagalan analisis risiko dapat merusak proyek</li><li><input type="checkbox"/> Proyek mungkin sulit untuk dikelola</li><li><input type="checkbox"/> Membutuhkan tim pengembang yang ahli</li></ul>
UNIFIED PROCESS	<ul style="list-style-type: none"><li>❖ Dokumentasi kualitas ditekankan</li><li>❖ Ada keterlibatan pelanggan yang berkelanjutan</li><li>❖ Mengakomodasi perubahan kebutuhan</li><li>❖ Bekerja dengan baik untuk pemeliharaan proyek</li></ul>	<ul style="list-style-type: none"><li><input type="checkbox"/> Use Case tidak selalu akurat</li><li><input type="checkbox"/> Memiliki integrasi peningkatan perangkat lunak yang rumit</li><li><input type="checkbox"/> Fase yang tumpang tindih dapat menyebabkan masalah</li><li><input type="checkbox"/> Membutuhkan tim pengembang yang ahli</li></ul>

# Rangkuman



## Model Proses Generik



Mencakup set dari framework dan umbrella activities, actions, dan set work tasks



Masing-masing dari berbagai model proses dapat dijelaskan dengan aliran proses yang berbeda—deskripsi tentang bagaimana framework activities, actions, dan tasks diatur secara berurutan dan kronologis.



Pola proses dapat digunakan untuk memecahkan masalah umum yang dihadapi sebagai bagian dari proses perangkat lunak

# Rangkuman



## Model Proses Preskriptif



Telah diterapkan selama bertahun-tahun dalam upaya untuk membawa ketertiban dan struktur untuk pengembangan perangkat lunak



Masing-masing model ini menyarankan aliran proses yang agak berbeda, tetapi semuanya melakukan rangkaian aktivitas kerangka umum yang sama:

komunikasi, perencanaan,  
pemodelan, konstruksi, dan  
penerapan (deployment)

# Rangkuman



## Model Proses Sekuensial



Waterfall adalah paradigm rekayasa perangkat lunak tertua



Menyarankan aliran proses linier yang sering tidak konsisten dengan realitas modern (misalnya akibat perubahan terus-menerus, sistem yang berkembang, garis waktu yang ketat) di dunia perangkat lunak



Memiliki penerapan dalam situasi di mana kebutuhan (*requirements*) didefinisikan dengan baik dan stabil

# Rangkuman

## Model Proses Incremental



Bersifat iteratif dan menghasilkan versi PL yang berfungsi dengan cukup cepat



Model proses evolusioner mengenali sifat iteratif, inkremental dari sebagian besar proyek RPL dan dirancang untuk mengakomodasi perubahan



Model evolusioner, seperti pembuatan prototipe dan model spiral, menghasilkan produk kerja tambahan (atau versi PL yang berfungsi) dengan cepat



Model-model ini dapat diadopsi untuk diterapkan di semua aktivitas RPL—mulai dari pengembangan konsep hingga pemeliharaan sistem jangka panjang

# Rangkuman

## Unified Process



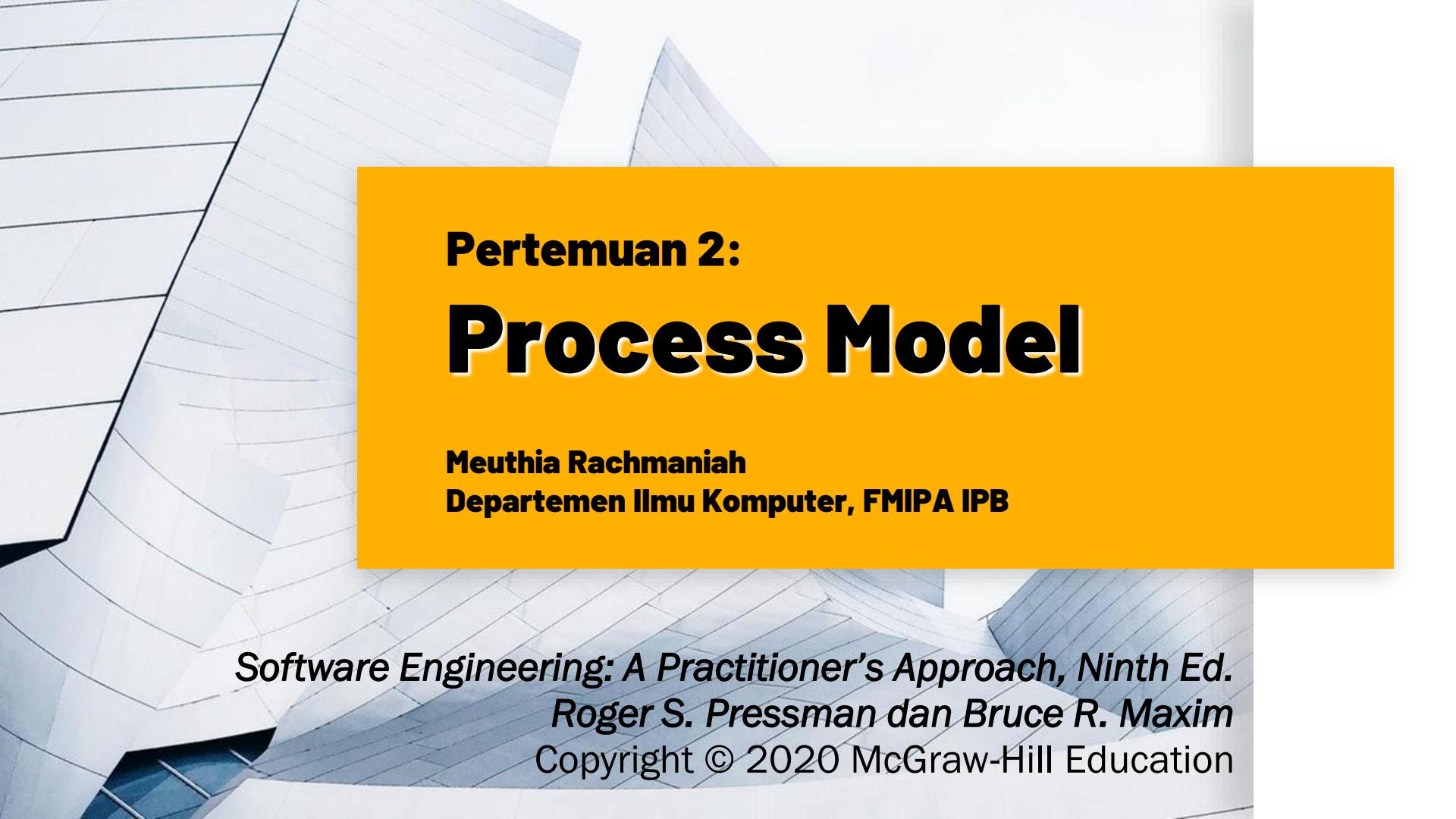
Proses perangkat lunak "use case-driven, arsitektur-sentrис, berulang, dan inkremental" yang dirancang sebagai framework untuk metode dan tools UML



Menerapkan aktivitas generik: komunikasi, perencanaan, pemodelan, konstruksi, dan deployment



Fase-fase UP: Inception (komunikasi dan perencanaan), Elaborasi (perencanaan dan pemodelan), Konstruksi, Transisi (konstruksi dan deployment), serta production

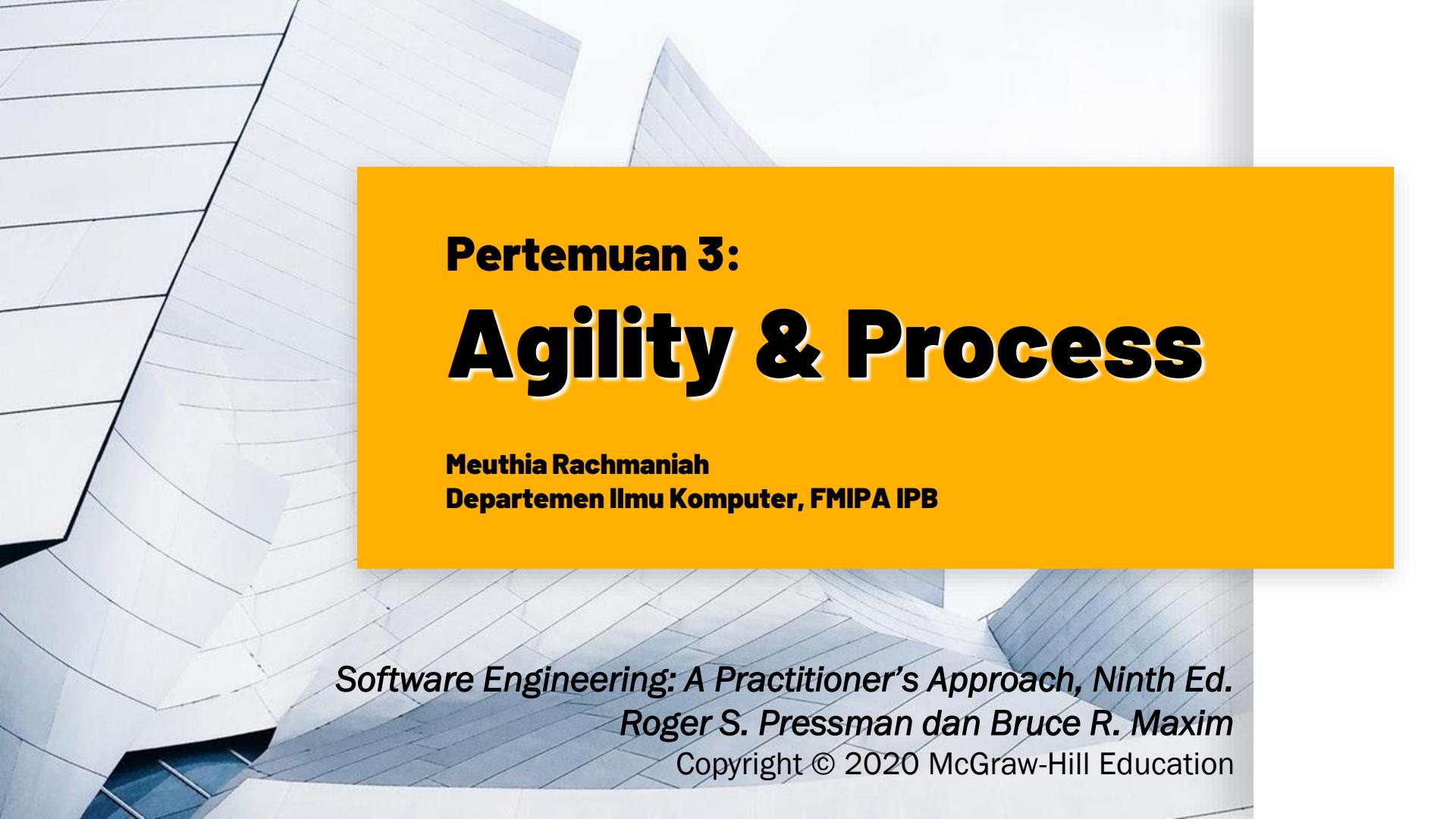


## **Pertemuan 2:**

# **Process Model**

**Meuthia Rachmaniah**  
**Departemen Ilmu Komputer, FMIPA IPB**

*Software Engineering: A Practitioner's Approach, Ninth Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education

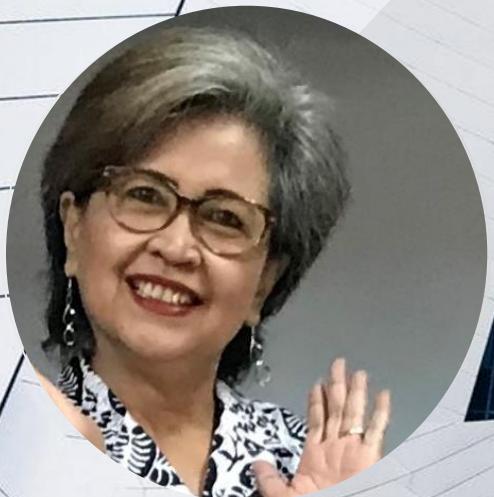


# **Pertemuan 3:**

# **Agility & Process**

**Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB**

*Software Engineering: A Practitioner's Approach, Ninth Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education



# Hello!

Ir. Meuthia Rachmaniah, M.Sc.  
*Associate Professor*  
NIP 195907111984032006

Computer Science Department  
Software Engineering & Information Sciences  
[meuthiara@apps.ipb.ac.id](mailto:meuthiara@apps.ipb.ac.id)

SATYALANCANA KARYA SATYA 20 TAHUN  
SATYALANCANA KARYA SATYA 30 TAHUN



# Cakupan Materi



- 3.1 What is Agility
- 3.2 Agility and Cost of Change
- 3.3 What is an Agile Process
- 3.4 Scrum
- 3.5 Framework Agile Lainnya
  - 3.5.1 Framework XP
  - 3.5.2 Kanban
  - 3.5.3 DevOps
- 3.6 Rangkuman

# Quick Look

## WHAT IS IT?

- RPL Agile mengkombinasikan filosofi dan set pedoman pengembangan
- Filosofi RPL Agile mengutamakan customer satisfaction dan pengiriman awal dari increment PL; tim proyek yang kecil, sangat bermotivasi; metode informal; work products RPL minimal; dan kesederhanaan seluruh pengembangan
- Pedoman pengembangan menekankan penyampaian pada analisis dan desain (walaupun kegiatan ini tidak dianjurkan)



## WHO DOES IT

- Software Engineer dan stakeholder proyek lainnya (manajer, pelanggan, pengguna akhir) bekerja sama dalam tim agile—tim yang mengatur dan mengendalikan diri dan nasibnya sendiri
- Tim agile memupuk komunikasi dan kolaborasi di antara semua yang terlibat di dalamnya



## WHAT ARE THE STEPS

- Framework Aktivitas—komunikasi, perencanaan, pemodelan, konstruksi, dan deployment ada. Tetapi mereka berubah menjadi kumpulan tugas minimal yang mendorong tim proyek menuju konstruksi dan pengiriman



## WHY IS IT IMPORTANT?

- Lingkungan bisnis modern menelurkan sistem berbasis komputer dan produk PL yang serba cepat dan selalu berubah
- RPL agile merupakan alternatif yang masuk akal untuk RPL konvensional. Telah ditunjukkan untuk memberikan sistem yang sukses dengan cepat



## WHAT IS THE WORK PRODUCT

- Work product paling penting ialah ‘software increment’ yang operasional yang diserahkan ke konsumen tepat waktu
- Dokumen paling penting ialah user story dan kasus uji terkait



## How Do I ENSURE THAT I'VE DONE IT RIGHT

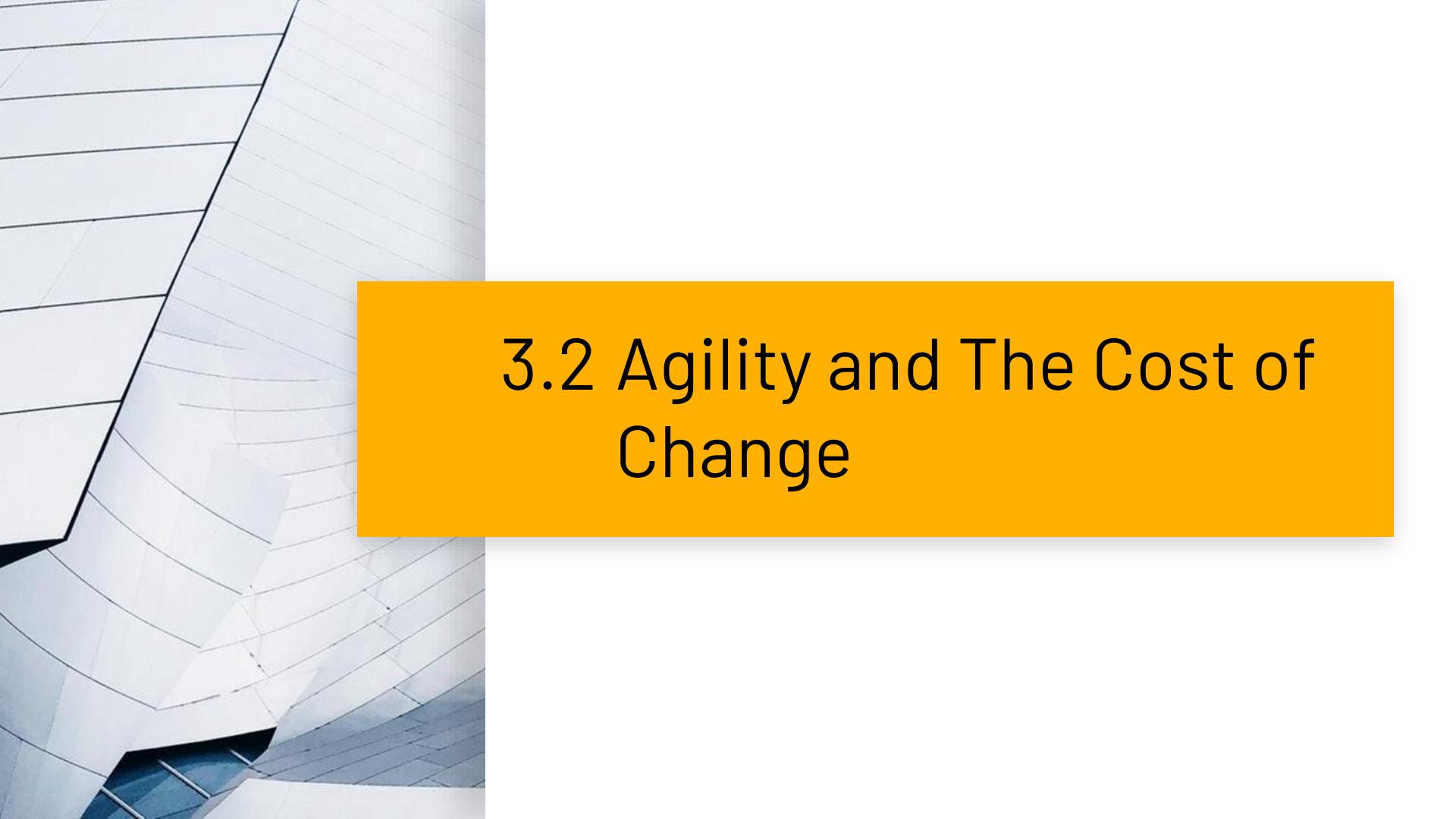
- Jika tim agile memenuhi bahwa prosesnya berhasil, dan tim menghasilkan peningkatan PL yang dapat dikirimkan yang memuaskan pelanggan, maka Anda telah melakukannya dengan benar



## 3.1 What Is Agility

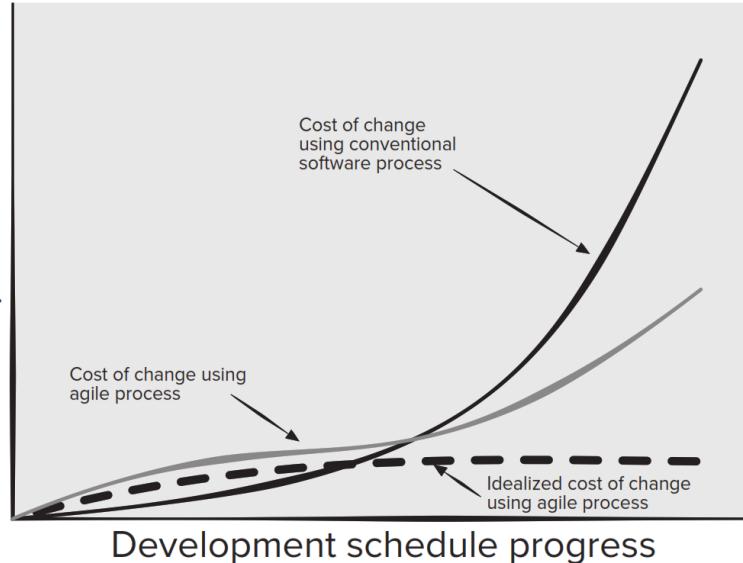
## 3.1 What Is Agility (*Kelincahan*)

- Respons yang **efektif** terhadap perubahan
- Mendorong **struktur** dan sikap tim yang membuat **komunikasi** (antara anggota tim, antara teknologi dan pebisnis, dan antara perekayasa PL dan manajer) lebih mudah
- Menekankan **pengiriman cepat** PL yang **beroperasi** dan menekankan pentingnya **work products menengah** (tidak selalu merupakan hal yang baik)
- Mengadopsi **pelanggan** sebagai **bagian dari tim pengembangan** dan bekerja untuk menghilangkan **sikap "kita dan mereka"** yang terus meliputi banyak proyek perangkat lunak
- Mengakui bahwa **perencanaan** di dunia dan bahwa rencana proyek **tidak pasti memiliki batasnya** harus fleksibel
- Agility dapat diterapkan pada proses PL apa pun. Namun penting agar:
  - proses dirancang sedemikian rupa sehingga memungkinkan tim proyek untuk **menyesuaikan tugas dan merampingkannya**,
  - melakukan **perencanaan** dengan cara yang memahami **fluiditas pendekatan pengembangan agile**,
  - menghilangkan semua kecuali **work products** yang paling yang penting dan menjaganya tetap ramping,
  - menekankan strategi pengiriman **increment** yang memberikan PL yang berfungsi kepada pelanggan secepat mungkin untuk jenis produk dan lingkungan operasional



## 3.2 Agility and The Cost of Change

## Perubahan biaya sebagai fungsi waktu dalam pengembangan



- Proses PL **konvensional** adalah bahwa biaya perubahan meningkat secara **nonlinier** seiring berjalannya proyek.
  - Tetapi jika di tengah pengujian validasi, stakeholder meminta perubahan fungsional utama yang memerlukan modifikasi desain arsitektur, konstruksi komponen baru, perubahan komponen lain yang sudah ada, pengujian baru dan sebagainya. Maka biaya meningkat dengan cepat
- Proses **agile** yang dirancang dengan baik “meratakan” kurva biaya perubahan, memungkinkan tim PL untuk mengakomodasi **perubahan** di akhir proyek PL **tanpa dampak biaya dan waktu** yang dramatis



## 3.3 What Is An Agile Process

# Karakteristik Agile Process



1. **Kesulitan Memprediksi Kebutuhan**
  - o Sulit untuk memprediksi sebelumnya kebutuhan PL mana yang akan bertahan dan mana yang akan berubah.
  - o Sama sulitnya untuk memprediksi bagaimana **prioritas pelanggan akan berubah** seiring berjalannya proyek
2. Pada banyak PL, **Desain dan Konstruksi Disisipkan**
  - o Artinya, kedua kegiatan tersebut harus **dilakukan secara bersamaan** sehingga model desain terbukti saat dibuat.
  - o Sulit untuk memprediksi berapa banyak desain yang diperlukan sebelum konstruksi digunakan untuk membuktikan desain
3. Dari sudut pandang perencanaan: **analisis, desain, konstruksi, dan pengujian tidak dapat diprediksi** seperti yang kita inginkan

### 3.3.1 Duabelas Prinsip Agility

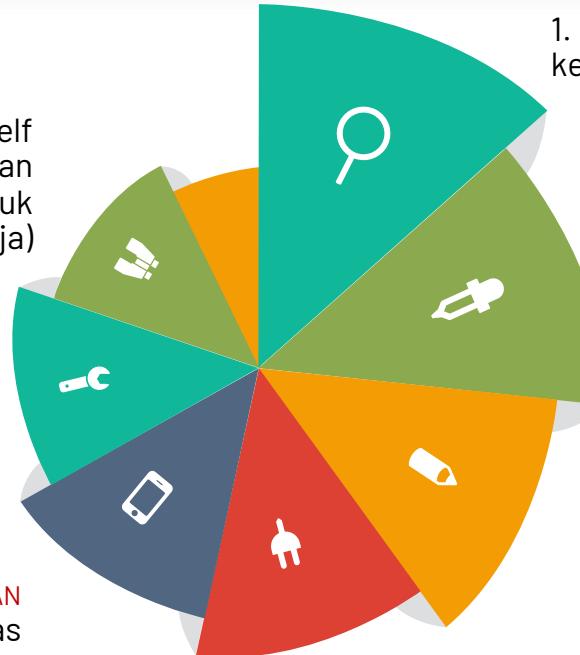
- 
1. Prioritas tertinggi adalah untuk memuaskan pelanggan melalui pengiriman PL yang berharga secara dini dan berkelanjutan
  2. Menyambut perubahan kebutuhan, walaupun terlambat dalam pengembangan. Proses agile memanfaatkan perubahan untuk keunggulan kompetitif pelanggan
  3. Kirimkan PL yang berfungsi sesering mungkin, dari beberapa minggu hingga beberapa bulan, dengan preferensi pada skala waktu yang lebih pendek
  4. Pebisnis dan pengembang harus bekerja sama setiap hari selama proyek berlangsung
  5. Bangun proyek di sekitar individu yang termotivasi. Beri mereka lingkungan dan dukungan yang mereka butuhkan, dan percayai mereka untuk menyelesaikan pekerjaan
  6. Metode penyampaian informasi yang paling efisien dan efektif ke dan di dalam tim pengembangan adalah percakapan tatap muka
  7. Perangkat lunak yang berfungsi adalah ukuran utama kemajuan
  8. Proses agile mempromosikan pembangunan berkelanjutan. Sponsor, pengembang, dan pengguna harus dapat mempertahankan kecepatan konstan tanpa batas
  9. Perhatian terus-menerus pada keunggulan teknis dan desain yang baik meningkatkan agility
  10. Kesederhanaan – seni memaksimalkan jumlah pekerjaan yang tidak dilakukan – sangat penting
  11. Arsitektur, kebutuhan, dan desain terbaik muncul dari tim yang mengatur diri sendiri (self organizing)
  12. Secara berkala, tim merefleksikan bagaimana menjadi lebih efektif, kemudian menyelaraskan dan menyesuaikan perilakunya sesuai dengan itu

# Human Factors

7. **MENGATUR DIRI SENDIRI** (self organization( diri untuk pekerjaan yang dilakukan, proses untuk lingkungan lokalnya, jadwal kerja)

6. **SALING PERCAYA DAN MENGHORMATI**

5. **KEMAMPUAN PEMECAHAN PROBLEM YANG FUZZY.** (Ambiguitas dan perubahan konstan, problem hari ini mungkin bukan problem besok!)



1. **KOMPETENSI.** (bakat, keterampilan, pengetahuan)!
2. **FOKUS BERSAMA.** (memberikan peningkatan PL yang berfungsi )
3. **KOLABORASI.** (rekan dan stakeholder)
4. **KEMAMPUAN MEMBUAT KEPUTUSAN.** (kebebasan untuk mengendalikan nasibnya sendiri)

## 3.3.2 The Politics of Agile Development

- **PRO AGILITY:** "Metodologi tradisional adalah sekelompok orang yang tidak tahu apa-apa yang lebih suka menghasilkan dokumentasi tanpa cacat daripada sistem kerja yang memenuhi kebutuhan bisnis"
- **PRO KONVENTIONAL:** "Ahli metodologi yang ringan, eh, 'gesit' adalah sekelompok peretas yang dimuliakan yang akan mendapat kejutan besar ketika mereka mencoba meningkatkan mainan mereka ke dalam perangkat lunak tingkat perusahaan."



Tidak ada yang menentang agility. Pertanyaan agility yang sebenarnya adalah:

- Apa cara terbaik untuk mencapainya?
- Ingatlah bahwa PL yang **berfungsi** itu penting, tetapi jangan lupa bahwa itu juga harus menunjukkan berbagai **atribut kualitas** termasuk **keandalan, kegunaan, dan kemudahan perawatan**.
- Bagaimana Anda membangun PL yang **memenuhi kebutuhan pelanggan saat ini** dan menunjukkan **karakteristik kualitas** yang memungkinkannya **diperluas dan ditingkatkan** untuk memenuhi **kebutuhan pelanggan dalam jangka panjang**?

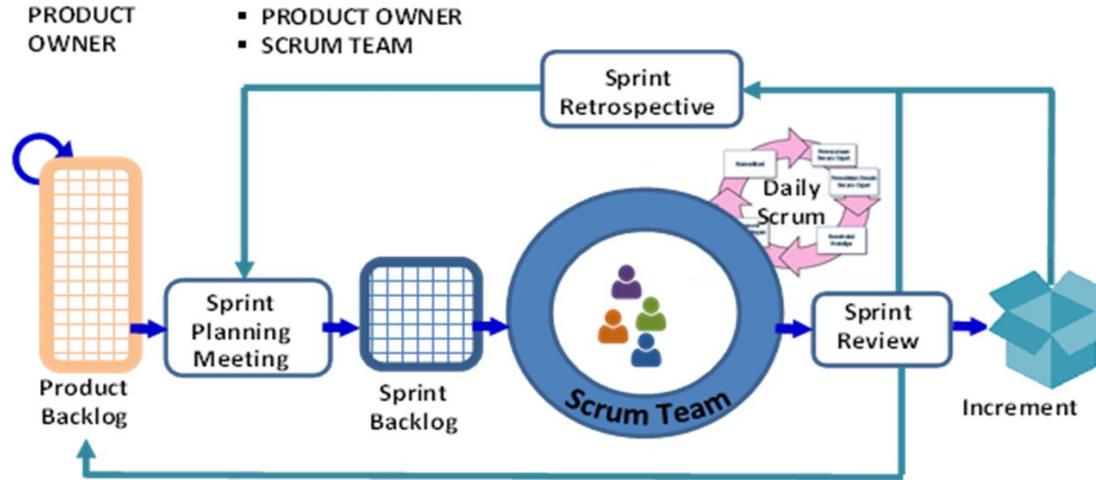
The background features a close-up view of a modern building's exterior with a white, curved, ribbed facade. A large, solid yellow rectangular box is positioned in the center-right area of the slide. The text "3.4 SCRUM" is centered within this yellow box.

## 3.4 SCRUM

## 3.4 SCRUM

- Scrum merupakan **metode Agile** untuk manajemen proyek yang dikembangkan oleh Ken Schwaber dengan tujuan untuk secara dramatis meningkatkan produktivitas dari tim.
- Scrum menerapkan metode empirisme ilmiah.
- Scrum mengantikan pendekatan algoritmik terprogram secara **heuristik**, yang **memperhatikan orang dan organisasi** untuk **mengatasi ketidakpastian** dan **memecahkan problem yang kompleks**.
- Gambar pada slide 15 mewakili *Scrum in Action* seperti yang dijelaskan oleh Schwaber & Sutherland (2012) dalam buku *Software in 30 Days*.
- Pada Scrum terdapat lima pertemuan untuk membahas *backlog refinement meeting*, *sprint planning meeting*, *daily scrum meeting* (15 menit *standup*), *sprint review meeting*, dan *sprint retrospective meeting*.
- Gambar pada slide 16 menjelaskan alur proses Scrum yang dielaborasi Pressman dan Maxim (2020)

# Framework Scrum Schwaber & Beedle, 2012



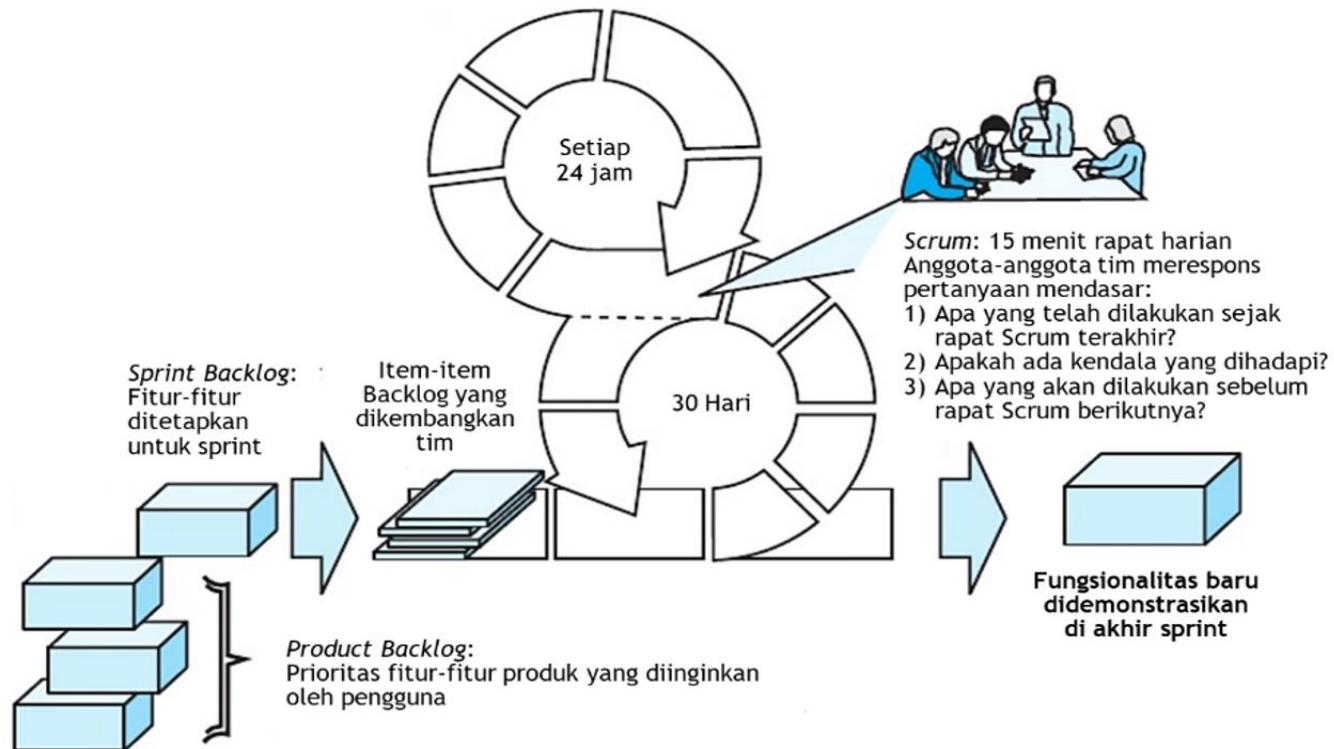
*Product Owner* membuat **product backlog** (daftar tugas yang perlu diprioritaskan dalam sebuah proyek). *Product owner* adalah **klien**, yaitu orang yang memesan perangkat lunak.

*Product Backlog* adalah daftar tugas yang **diprioritaskan** dalam sebuah proyek

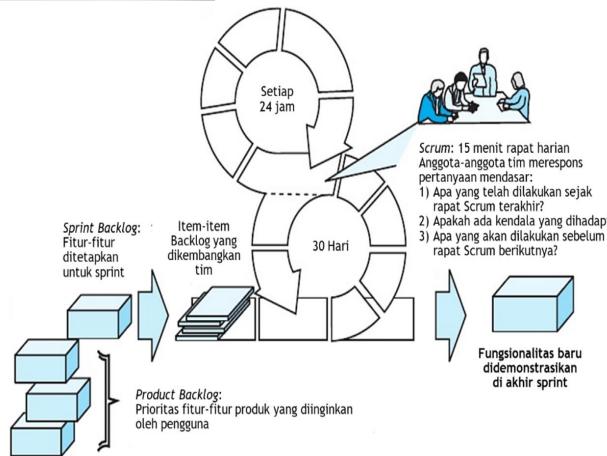
**Scrum Master** ialah menjadi **fasilitator** dan menjembatani **komunikasi** dan membuat **pelaporan**, namun tidak menentukan jadwal atau tugas.

**Scrum Team** adalah grup kecil para **pengembang**, biasanya berjumlah antara **5-9 orang** yang saling bekerja sama untuk menghasilkan PL

# Framework Scrum Pressman & Maxim 2020



### 3.4.1 Scrum Teams and Artifacts



- **Scrum Team:** Terdiri atas **Product Owner**, **Scrum Master**, **3-6 orang Development Team**
- **Scrum Artifact:** **Product Backlog**, **Sprint Backlog**, dan **Code Increment**
- Pengembangan dilanjutkan dengan memecah proyek menjadi **serangkaian periode pengembangan prototipe incremental** selama **2 hingga 4 minggu** yang disebut **Sprint**
- **Product Backlog** adalah daftar prioritas kebutuhan produk atau fitur yang memberikan nilai bisnis bagi pelanggan. Item dapat ditambahkan ke backlog kapan saja dengan persetujuan **Product Owner** dan persetujuan tim pengembang
- **Sprint Backlog** adalah subset dari item product backlog yang dipilih oleh tim produk untuk diselesaikan sebagai **code increment** pada **sprint aktif (current sprint active)**
- **Code Increment** adalah **union** dari semua item backlog produk yang diselesaikan di **sprint sebelumnya** dan semua item backlog yang harus **diselesaikan di sprint saat ini**
- **Scrum Master:** adalah **fasilitator** untuk seluruh anggota Scrum Team
  - Menjalankan **Daily Scrum Meeting** dan bertanggung jawab untuk menghilangkan hambatan yang diidentifikasi oleh anggota tim selama rapat
  - **Melatih anggota tim pengembangan** untuk **saling membantu** menyelesaikan **Sprint Task** ketika mereka memiliki waktu yang tertentu
  - Membantu **Product Owner** menemukan teknik untuk **mengelola item product backlog** dan membantu memastikan bahwa **item backlog** dinyatakan dalam istilah yang **jelas dan ringkas**

## 3.4.2 Sprint Planning Meeting

### Prior to Beginning



Setiap **tim pengembang** bekerjasama dengan **Product Owner** dan seluruh **stakeholder** lainnya untuk mengembangkan **item-item pada product backlog**



### Product Backlog Diranking

- Product backlog diranking berdasarkan kepentingan **kebutuhan bisnis** pemilik dan **kompleksitas** tugas RPL (**pemrograman** dan **pengujian**) yang diperlukan untuk menyelesaiakannya
- Terkadang dihasilkan **identifikasi fitur yang hilang padahal diperlukan** untuk memberikan fungsionalitas yang diperlukan kepada pengguna akhir



### Sebelum Memulai Sprint

- **Product Owner** menyatakan **tujuan pengembangan increment** yang akan diselesaikan di Sprint mendatang
- Scrum Master dan tim pengembangan **memilih item** yang akan dipindahkan ke **sprint backlog**
- **Tim pengembangan** menentukan apa yang dapat disampaikan pada increment dalam batasan waktu yang dialokasikan untuk sprint dan, dengan **Scrum Master**, pekerjaan apa yang diperlukan untuk mengirimkan increment
- Tim pengembangan memutuskan **peran mana yang dibutuhkan** dan bagaimana peran tersebut **perlu diisi**

### 3.4.3 Daily Scrum Meeting



- Daily Scrum Meeting ialah **event 15-menit** yang **dijadwalkan setiap awal hari kerja** agar anggota tim melakukan **sinkronisasi aktivitas** dan membuat rencana untuk **24 jam**
- Daily Scrum Meeting dihadiri oleh **Scrum Master** dan **terkadang Product Owner**
- Scrum master **memimpin rapat** dan **menilai tanggapan** dari setiap orang



Tiga pertanyaan kunci diajukan dan dijawab oleh semua anggota tim:

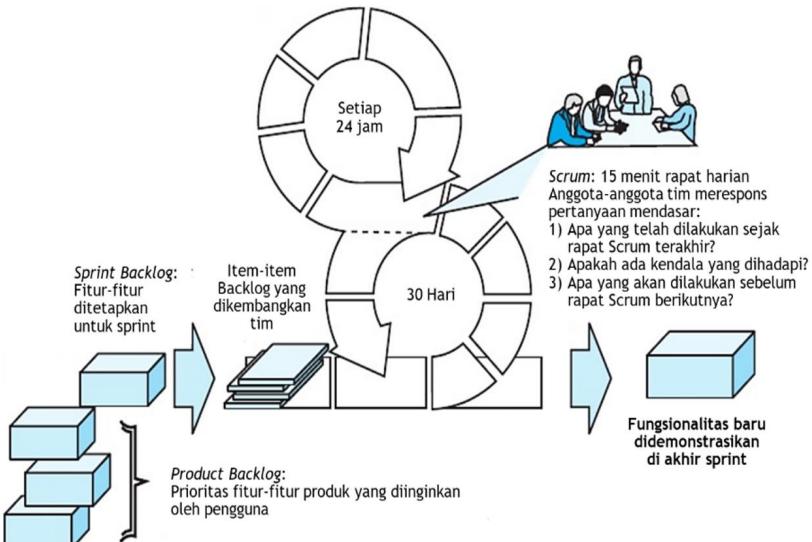
- 1) **Apa yang Anda lakukan** sejak pertemuan tim terakhir?
- 2) **Apa kendala** yang Anda hadapi?
- 3) **Apa yang Anda rencanakan** untuk dicapai pada pertemuan tim berikutnya?



- Daily Scrum Meeting membantu tim untuk **mengungkap potensi problem** sedini mungkin
  - Daily Scrum Meeting **bukan pertemuan pemecahan problem** (untuk pemecahan problem dilakukan secara off-line dan hanya melibatkan pihak-pihak yang terkena dampak)
- Daily Scrum Meeting mengarah pada "**sosialisasi pengetahuan**" , jadi mempromosikan struktur tim yang mengatur diri sendiri
  - Beberapa tim menggunakan pertemuan ini untuk **menyatakan item sprint backlog Complete atau Done**
  - Ketika tim menganggap semua item sprint backlog **Complete**, tim dapat memutuskan untuk menjadwalkan **demo** dan **meninjau increment** yang diselesaikan dengan **Product Owner**

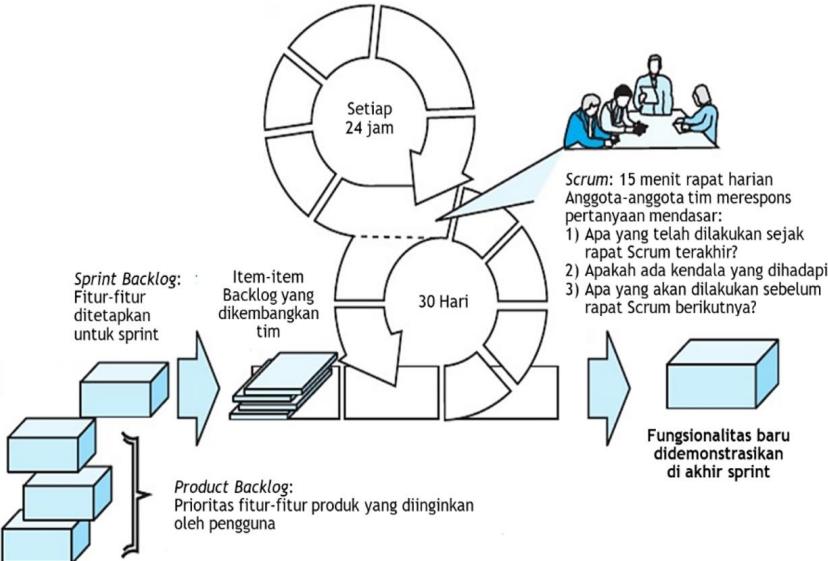


### 3.4.4 Sprint Review Meeting



- Terjadi pada akhir sprint ketika tim pengembang telah menilai increment selesai.
- Seringkali dibatasi sebagai **4 jam meeting** untuk sprint **4 minggu**.
- Scrum Master, tim pengembangan, Product Owner, dan Stakeholder terpilih menghadiri tinjauan ini.
- Aktivitas utama adalah **demo increment** PL yang diselesaikan **dari suatu sprint**.

### 3.4.5 Sprint Retrospective



- Scrum Master memimpin **3 jam** meeting (utk 4-minggu sprint) dengan **Tim Pengembang**:
  - Apa yang **berjalan baik** di sprint?
  - Apa yang **bisa ditingkatkan**?
  - Apa yang akan dilakukan tim untuk **dingkatkan di sprint berikutnya**
- Tim merencanakan cara untuk meningkatkan kualitas produk dengan mengadaptasi definisi "**selesai**".



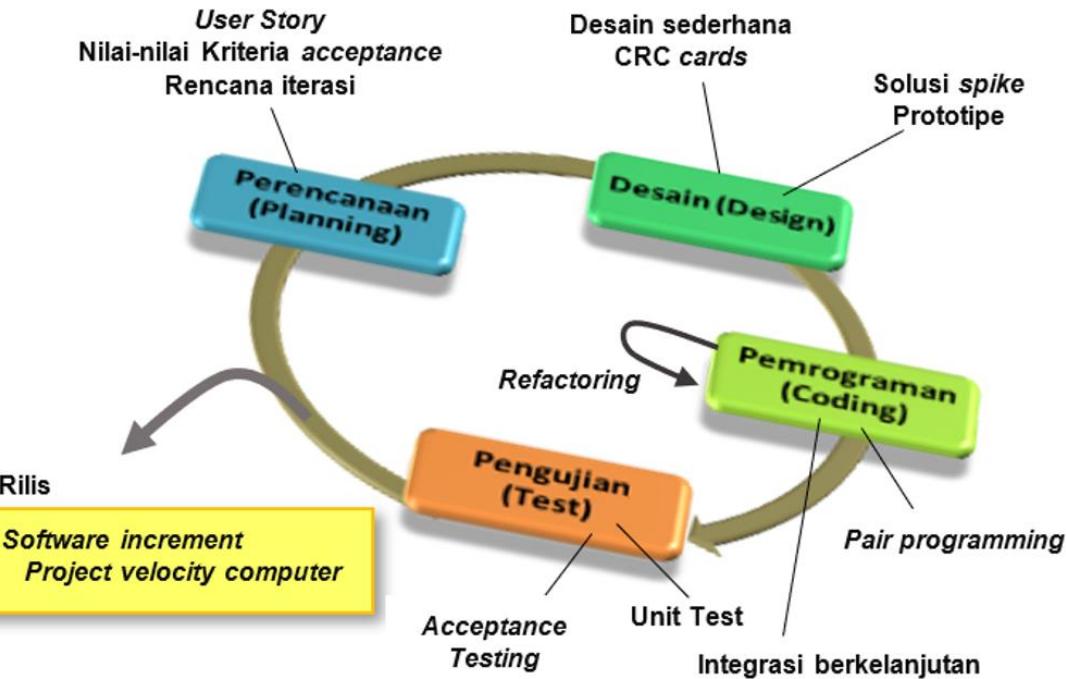
## 3.5 Framework Agile Lainnya

3.5.1 Framework XP

3.5.2 Kanban

3.5.3 DevOps

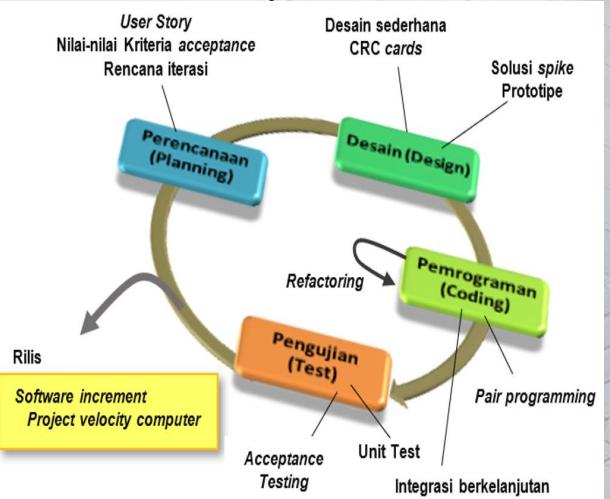
## 3.5.1 Extreme Programming (XP) Framework



- Extreme Programming (XP) mengutamakan **kepuasan konsumen** dan **kerja tim dengan komunikasi, kesederhanaan, dan keberanian** sebagai nilai-nilai utama
- Pengembang berkomunikasi dengan **pengguna** dan programmer
- Desain dijaga tetap **sederhana** dan bebas kesalahan
- Testing yang dilakukan **di awal** seringkali menyediakan **umpan balik** sehingga pengembang berani mengubah kebutuhan dan teknologi
- Tim proyek dijaga berukuran kecil, **sekitar 10 orang**

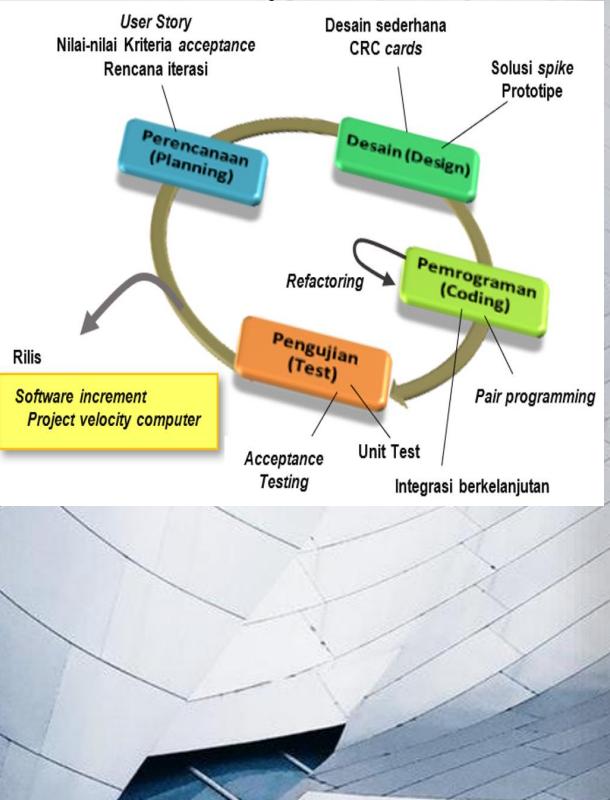
# Perencanaan

- Disebut juga **Planning Game**, diawali aktivitas kebutuhan **listening** menghasilkan **user stories**
- **User Stories**: **output** yang diperlukan, **fitur**, dan **fungsionalitas**.
- Setiap user story dibuat pada **Index Card**, diberi **value** (prioritas setiap user story) serta **cost** (ukuran minggu pengembangan, bila > 3 minggu user story di-split (dipecah menjadi beberapa user story)
- **Komitmen**: **user story yang dicakup**, **delivery date**, info lain terkait proyek) dalam 3 cara:
  - i. Seluruh user story diimplementasi dalam beberapa minggu,
  - ii. Prioritas tinggi didahulukan,
  - iii. User story paling berisiko diimplementasikan pertama kali
- Customer dan developer menentukan jumlah **Release** (rilis), yaitu **increment** PL berikutnya yang disertai dengan penghitungan **Project Velocity**



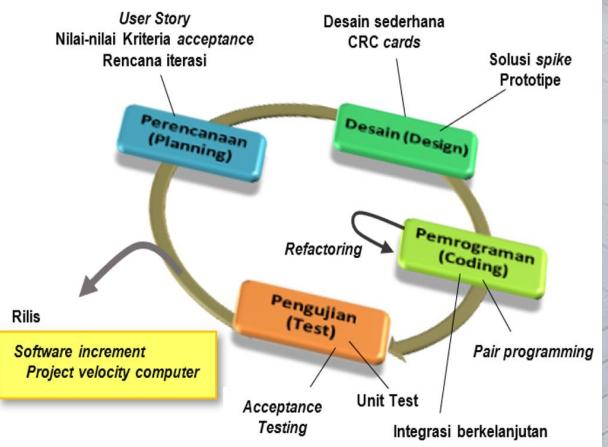
# Desain

- Mengikuti prinsip KIS, yaitu *Keep It Simple* (tetap sederhana).
- Menyediakan panduan untuk implementasi sesuai yang tertulis pada story.
  - Desain dengan penambahan fungsionalitas ekstra (karena pengembang berasumsi nantinya akan diperlukan) **tidak disarankan** untuk dilakukan.
  - XP menganjurkan penggunaan kartu *class responsibility collaborator* (CRC) sebagai mekanisme efektif terkait PL dalam konteks berbasis obyek.
  - Kartu CRC **mengidentifikasi** dan menata **class** berbasis obyek yang relevan dengan **software increment**. Kartu CRC ialah **satu-satunya produk desain** yang dihasilkan sebagai bagian dari proses XP.
  - Jika kesulitan mendesain suatu *story*, maka **langsung saja dibuat prototipe operasionalnya**, atau disebut sebagai *Spike Solution*, yaitu prototipe desain yang langsung diimplementasikan dan dievaluasi.
    - Tujuan: untuk memperkecil resiko saat implementasi dan untuk memvalidasi estimasi orisinal dari *story* yang sulit didesain.



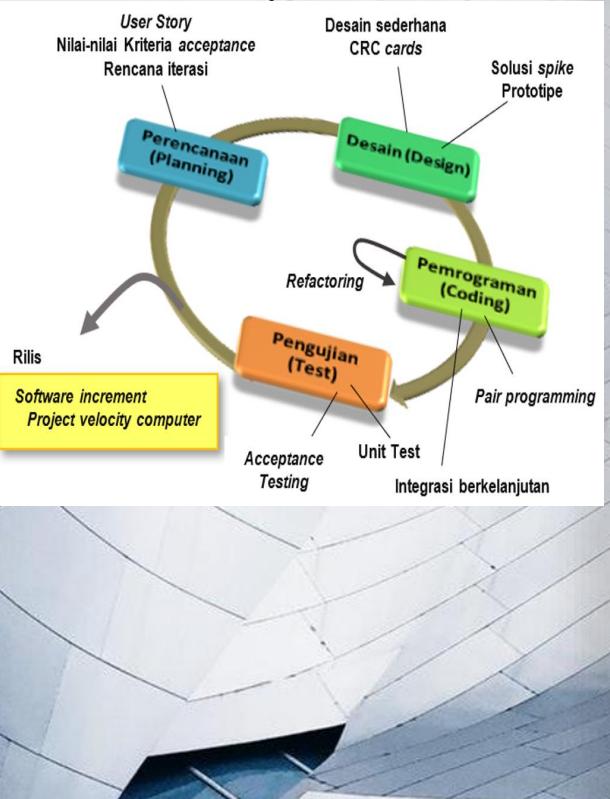
# Koding

- Konsep kunci dari aktivitas koding ialah ***pair programming***, yaitu dua orang bekerja sama pada satu komputer untuk membuat koding dari suatu story.
- Hal ini dilakukan agar terjadi **mekanisme pemecahan problem dan jaminan mutu secara *real time*** (saat koding dibuat langsung direview) dan agar pengembang fokus pada permasalahannya.
  - Konsep pemrograman parallel - dua programer bekerja untuk tugas yang sama **pada komputer yang sama**; salah seorang programer membuat program sementara programer satunya lagi melakukan navigasi/menjadi pengamat.
  - **Programer pengamat** memeriksa koding secara strategis dalam garis besar PL sementara **programer yang satunya lagi (*the driver*)** lebih menangani tugas individual.
  - Pada dasarnya setiap orang memiliki peran berbeda. Misalnya satu orang memikirkan detail koding dari suatu desain sementara satu orang lainnya memastikan standar-standar koding dipenuhi.
- Hasil koding ***pair programmer*** **diintegrasikan** dengan hasil koding **dari *pair programmer* lainnya**.
  - Integrasi dilakukan secara **harian** oleh tim integrase atau ***pair programmer*** yang bertanggung jawab melakukan integrasi.
  - Strategi integrasi secara kontinyu dilakukan agar terhindar dari problem kesesuaian (***compatibility***) dan antar muka (***interfacing***) dan menyediakan lingkungan testing untuk menemukan error di awal



# Refactoring

- Refactoring ialah teknik konstruksi yang merupakan **metode untuk optimisasi desain**.
  - Refactoring adalah proses mengubah sistem PL yaitu tidak mengubah perilaku eksternal namun memperbaiki struktur internalnya sedemikian rupa.
  - Refactoring adalah suatu disiplin untuk merapikan koding (dan modifikasi/ menyederhanakan desain internal) yang meminimisasi peluang adanya kesalahan (*bug*).
- Pada dasarnya *refactoring* sebenarnya **memperbaiki desain dari koding** setelah koding tersebut dibuat.
- Selain kartu CRC dan *spike solution*, desain XP secara virtual tidak menggunakan notasi dan jumlah desainnya sedikit. Oleh karenanya **desain XP** sering dianggap sebagai **artefak sementara (*transient artifact*)** sehingga saat konstruksi berlangsung harus **scara kontinyu dimodifikasi**.
- **Tujuan** dari *refactoring* ialah **mengendalikan modifikasi** dengan cara menganjurkan **perubahan kecil pada desain** yang pada akhirnya secara **radikal** akan **memperbaiki desain itu sendiri**. Namun penting diperhatikan bahwa upaya yang dibutuhkan untuk *refactoring* akan bertambah secara dramatis sesuai penambahan ukuran dari aplikasi.
- **Pada XP, desain berlangsung baik sebelum maupun sesudah koding dilakukan.** Refactoring artinya ialah bahwa desain terjadi secara kontinyu selama sistem dikonstruksikan. Dalam kenyataannya, aktivitas konstruksi akan memberikan panduan bagi tim XP untuk memperbaiki desain.

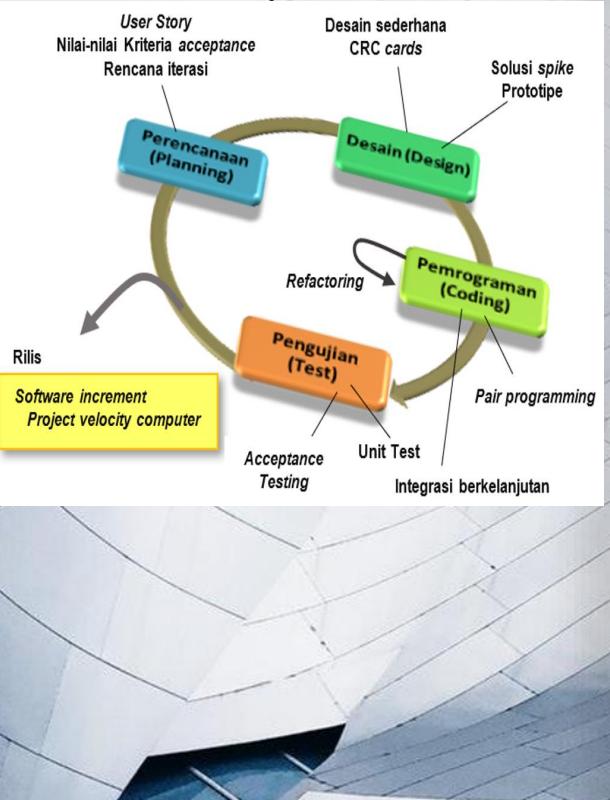


# Pengujian

- Pembuatan unit test sebelum koding dilakukan merupakan elemen kunci dari pendekatan XP.
- **Unit test** yang dibuat harus diimplementasikan dengan *framework* yang memungkinkan otomasi (sehingga bisa dieksekusi dengan mudah dan berulang kali).
- Setelah setiap individual *unit test* ditata ke dalam "*universal testing suite*", maka testing integrasi dan validasi dilakukan secara harian. Dengan demikian, tim XP bisa mengetahui indikasi kontinyu terkait kemajuan pengembangan dan bisa segera mengetahui apabila ada problem yang terjadi.
- **Acceptance test** pada XP, disebut juga *customer test*, dilakukan oleh konsumen dengan fokus pada keseluruhan fitur dan fungsionalitas sistem yang *visible* dan bisa direview oleh konsumen.
- **Acceptance test** dikembangkan dari berbagai *user story* yang telah diimplementasikan sebagai bagian dari rilis perangkat lunak.

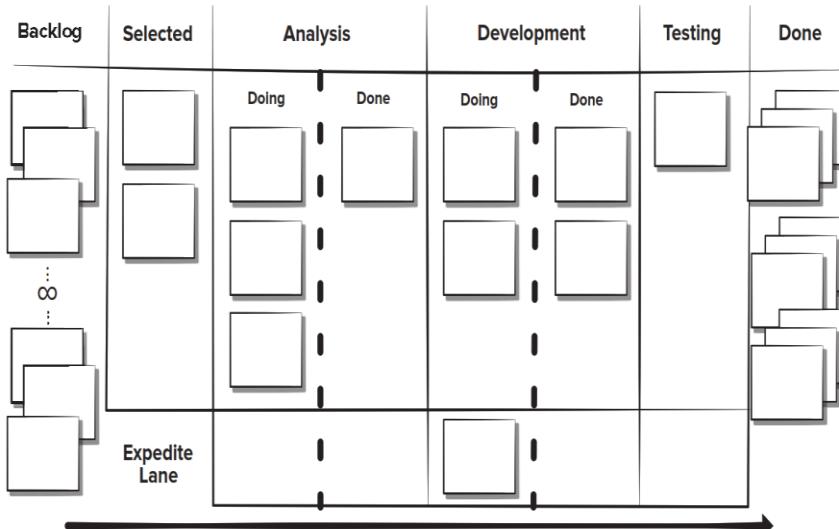
# Rilis

- Setelah rilis pertama (disebut juga *software increment*) diserahkan, maka tim XP menghitung *project velocity*, yaitu **jumlah story yang selesai diimplementasikan pada rilis yang pertama**.
- **Project Velocity** digunakan untuk:
  - i. Estimasi tanggal penyerahan dan jadwal rilis berikutnya dan
  - ii. Menentukan apakah terjadi janji yang berlebih-lebihan (*overcommitment*) untuk seluruh *story* pada keseluruhan proyek pengembangan.



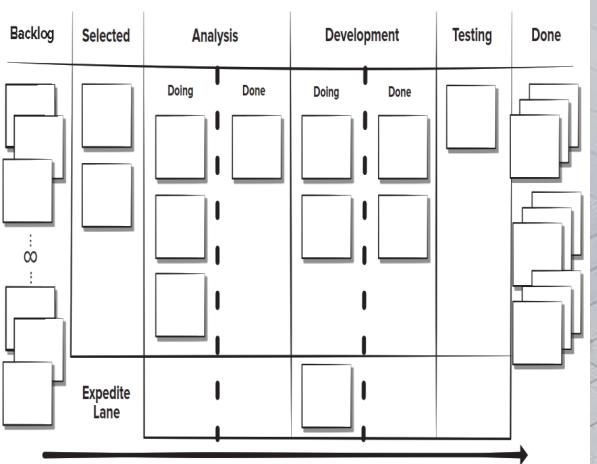
## 3.5.2 Metode Kanban

### Kanban Board



- Merupakan *lean methodology*, yaitu metode untuk meningkatkan setiap proses atau workflow
- Berfokus pada *change management* dan *service delivery*
- *Change management* mendefinisikan proses di mana perubahan yang diminta diintegrasikan ke dalam sistem berbasis perangkat lunak.
- *Service Delivery* didorong dengan berfokus pada pemahaman kebutuhan dan harapan pelanggan
- Anggota tim mengelola pekerjaan dan diberi kebebasan untuk **mengatur diri sendiri** untuk menyelesaiannya.
- Kebijakan akan berkembang **sesuai kebutuhan** untuk meningkatkan hasil
- Kanban berasal dari **Toyota** yang merupakan praktik-praktik rekayasa industrial yang **diadaptasikan** ke pengembangan PL

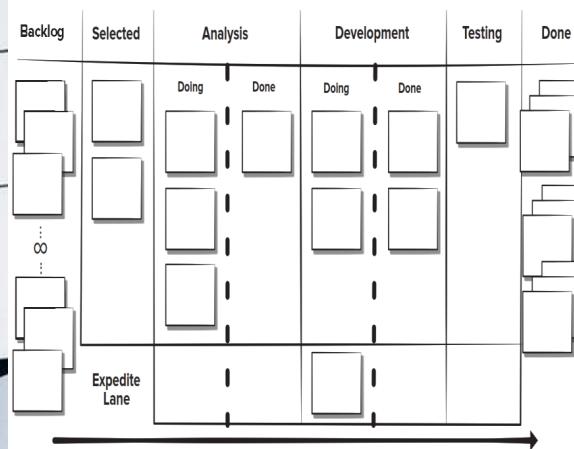
## Kanban Board



- Team meetings untuk Kanban **mirip** seperti yang ada dalam framework **Scrum**. Jika Kanban digunakan untuk proyek yang sudah ada, maka **tidak semua item** akan masuk di kolom **backlog**.
- Pengembang perlu menempatkan kartu(cards) di kolom proses tim dengan **bertanya** pada diri sendiri:
  - Di mana mereka sekarang? Dari mana mereka berasal? Kemana mereka pergi?
- Dasar dari Kanban Standup Meeting ialah task yang disebut '**Walking The Board**'
  - Pemimpin meeting **dirotasi** secara **harian**
  - Anggota tim mengidentifikasi item yang **hilang** dari Kanban Board yang sedang dikerjakan dan **menambahkannya** ke Kanban Board
  - Setiap item dikerjakan sampai '**Done**'
  - Tujuannya adalah untuk mengerjakan item **bernilai bisnis tinggi** terlebih dahulu
  - Tim melihat alur dan mencoba mengidentifikasi **hambatan** pada penyelesaian dengan melihat **beban kerja** dan **risiko**
- Saat **Restrospective Meeting Mingguan** dilakukan process measurement.
  - Tim mempertimbangkan di mana **perbaikan proses** mungkin diperlukan dan **mengusulkan perubahan** untuk diterapkan.
  - Kanban dapat dengan mudah digabungkan dengan praktik pengembangan **agile lainnya** untuk memperkaya disiplin proses

# Six Core Practice Metode Kanban

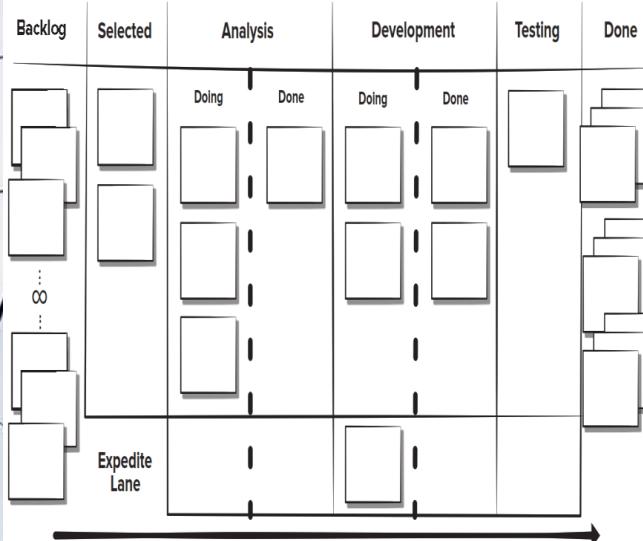
Kanban Board



- 1) **Visualisi Workflow** menggunakan Kanban Board.  
Kanban Board terdiri atas **kolom-kolom** yang merepresentasikan tahap pengembangan dari setiap **elemen fungsionalitas PL**. Kartu-kartu (**Cards**) pada Kanban Board dapat berisi **1 user story** atau **defect** yang ditemukan. Tim bergerak dari '**To Do**' ke '**Doing**', dan '**Done**'
- 2) **Jumlah Work in Progress (WIP)** pada suatu waktu tertentu dibatasi  
Pengembang didorong untuk **menyelesaikan** suatu task sebelum bergerak ke task berikutnya. Hal ini untuk **mengurangi lead time**, **meningkatkan kualitas kerja**, dan **meningkatkan kemampuan tim** untuk lebih sering menghasilkan fungsionalitas PL kepada stakeholder
- 3) **Pengelolaan Workflow**  
Mengelola workflow untuk mengurangi pemborosan dengan memahami **nilai workflow** saat ini, **menganalisis** tempat yang terhenti, **menentukan perubahan**, dan kemudian **menerapkan perubahan**

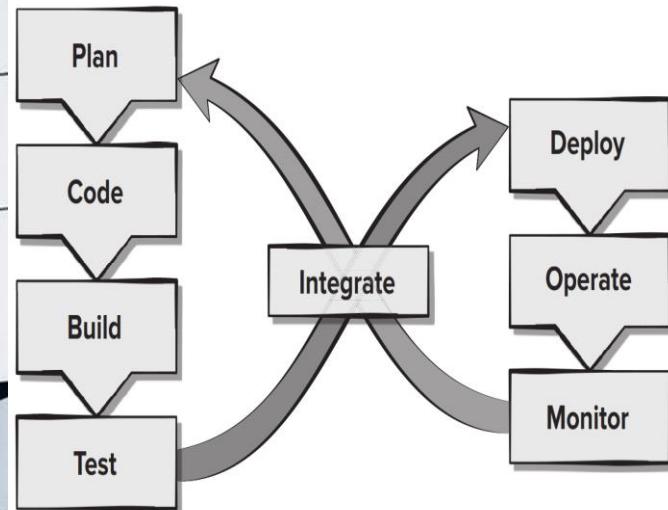
# Six Core Practice Metode Kanban

Kanban Board



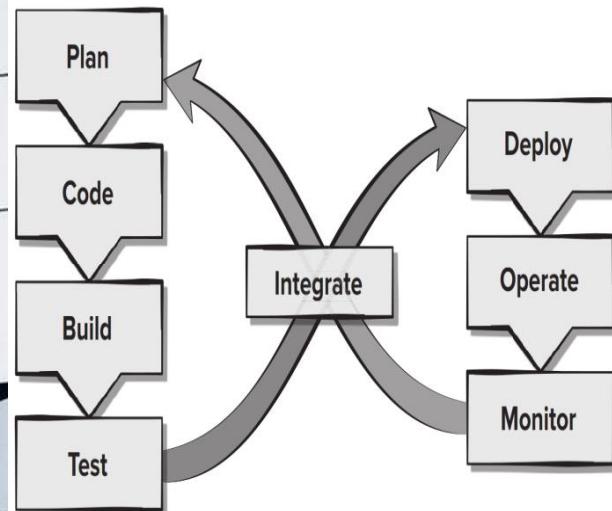
- 4) Membuat **kebijakan proses yang eksplisit**  
Misal, menuliskan alasan mendasar mengapa suatu item dipilih untuk dikerjakan dan kriteria yang digunakan untuk definisi ‘Done’
- 5) Berfokus pada peningkatan terus menerus (**continuous improvement**)  
dengan membuat **loop umpan balik** di mana perubahan diperkenalkan berdasarkan data proses dan efek dari perubahan pada proses diukur setelah perubahan dibuat.
- 6) Melakukan perubahan proses secara **kolaboratif** dan **melibatkan** semua **anggota tim** dan **stakeholder** lainnya sesuai kebutuhan

### 3.5.3 DevOps



- diciptakan oleh **Patrick DeBois** untuk mengkombinasikan **Development** dan **Operations**
- Menerapkan prinsip-prinsip **agile** dan **lean development** di seluruh software supply chain
- **Workflow DevOps** (gambar di kiri) melibatkan **5 tahapan dalam loop** terus menerus sampai dihasilkan produk yang diinginkan
- DevOps meningkatkan **pengalaman pelanggan** dengan bereaksi cepat terhadap **perubahan kebutuhan atau keinginan** mereka
- Hal ini dapat **meningkatkan loyalitas merek** dan meningkatkan pangsa pasar.
- Pendekatan ramping (**lean**) seperti DevOps dapat memberi organisasi peningkatan **kapasitas untuk berinovasi** dengan **mengurangi penggeraan ulang** dan memungkinkan **peralihan ke aktivitas nilai bisnis yang lebih tinggi**
- Produk tidak menghasilkan uang sebelum konsumen memiliki akses ke produk tersebut, dan DevOps dapat memberikan **waktu deployment yang lebih cepat** ke platform produksi

# Lima Tahapan DevOps



- 1) **Continuous Development.** Deliverable dari PL dibagi dan dikembangkan menjadi beberapa sprint dengan increment yang dikirimkan ke anggota jaminan kualitas tim pengembangan untuk pengujian
- 2) **Continuous Testing.** Tools automasi testing digunakan untuk membantu anggota tim menguji beberapa increment koding secara bersamaan untuk memastikan bebas dari cacat sebelum integrase dilakukan
- 3) **Continuous Integration.** Potongan kode dengan fungsionalitas baru ditambahkan ke kode yang ada dan ke lingkungan run-time dan kemudian diperiksa untuk memastikan tidak ada kesalahan setelah penerapan
- 4) **Continuous Deployment.** Kode terintegrasi disebarluaskan(diinstall) ke lingkungan produksi, yang mungkin mencakup beberapa situs secara global yang perlu disiapkan untuk menerima fungsionalitas baru
- 5) **Continuous Monitoring.** Staf operasi yang merupakan anggota tim pengembangan membantu meningkatkan kualitas PL dengan memantau kinerjanya di lingkungan produksi dan secara proaktif mencari kemungkinan problem sebelum pengguna menemukannya



## 3.6 Rangkuman



Model Proses	PROS	CONS
SCRUM	<ul style="list-style-type: none"><li>❖ Product Owner menetapkan prioritas</li><li>❖ Tim memiliki pengambilan keputusan</li><li>❖ Dokumentasinya ringan</li><li>❖ Mendukung pembaruan yang sering</li></ul>	<ul style="list-style-type: none"><li><input type="checkbox"/> Sulit untuk mengendalikan biaya perubahan</li><li><input type="checkbox"/> Mungkin tidak cocok untuk tim besar</li><li><input type="checkbox"/> Membutuhkan anggota tim yang ahli</li></ul>
EXTREME PROGRAMMING (XP)	<ul style="list-style-type: none"><li>❖ Menekankan keterlibatan pelanggan</li><li>❖ Menetapkan rencana dan jadwal yang rasional</li><li>❖ Ada komitmen pengembang yang tinggi untuk proyek tersebut</li><li>❖ Kemungkinan penolakan produk berkurang</li></ul>	<ul style="list-style-type: none"><li><input type="checkbox"/> Ada godaan untuk "mengirimkan" prototype</li><li><input type="checkbox"/> Membutuhkan pertemuan yang sering tentang biaya yang meningkat</li><li><input type="checkbox"/> Memungkinkan untuk perubahan yang berlebihan</li><li><input type="checkbox"/> Ada ketergantungan pada anggota tim yang sangat terampil</li></ul>



Model Proses	PROS	CONS
KANBAN	<ul style="list-style-type: none"><li>❖ Memiliki kebutuhan anggaran dan waktu yang lebih rendah</li><li>❖ memungkinkan pengiriman produk lebih awal</li><li>❖ Kebijakan proses dituliskan</li><li>❖ Ada perbaikan proses yang berkelanjutan</li></ul>	<ul style="list-style-type: none"><li>❑ Keterampilan kolaborasi tim menentukan kesuksesan</li><li>❑ Analisis bisnis yang buruk dapat merusak proyek</li><li>❑ Fleksibilitas dapat menyebabkan pengembang kehilangan focus</li><li>❑ Keengganan pengembang untuk menggunakan pengukuran</li></ul>
DEVOPS	<ul style="list-style-type: none"><li>❑ Ada pengurangan waktu untuk deployment koding</li><li>❑ Tim terdiri atas pengembang dan staf operasi</li><li>❑ Tim memiliki kepemilikan proyek ujung ke ujung (end-to-end)</li><li>❑ Ada pemantauan proaktif dari produk yang dilakukan deploy</li></ul>	<ul style="list-style-type: none"><li>❑ Ada tekanan untuk mengerjakan koding lama dan baru</li><li>❑ Ada ketergantungan besar pada tools automasi agar efektif</li><li>❑ Deployment dapat mempengaruhi lingkungan produksi</li><li>❑ Membutuhkan tim pengembang yang ahli</li></ul>

# Rangkuman

## Model Proses Agile



Dalam ekonomi modern, kondisi pasar berubah dengan cepat, kebutuhan pelanggan dan pengguna akhir berkembang, dan ancaman persaingan baru muncul tanpa peringatan.



Praktisi harus mendekati RPL dengan cara yang memungkinkan mereka untuk tetap agile – untuk menentukan proses *lean* yang dapat bermanuver, adaptif, yang dapat mengakomodasi kebutuhan bisnis modern.



Filosofi agile untuk RPL menekankan empat prinsip utama:

- i. Pentingnya tim yang mengatur diri sendiri yang memiliki kendali atas pekerjaan yang mereka lakukan,
- ii. Komunikasi dan kolaborasi antara anggota tim dan antara praktisi dan pelanggan mereka,
- iii. Pengakuan bahwa perubahan mewakili peluang, dan
- iv. Penekanan pada pengiriman cepat PL yang memuaskan pelanggan



Kenyataannya adalah tidak ada metode agile yang sempurna untuk setiap proyek. Pengembang agile bekerja dalam tim mandiri dan diberdayakan untuk membuat model proses mereka sendiri.

# Rangkuman

## SCRUM



Scrum menekankan penggunaan seperangkat pola proses PL yang telah terbukti efektif untuk proyek dengan garis waktu yang ketat, kebutuhan yang berubah, dan kekritisan bisnis



Tidak ada alasan mengapa tim Scrum tidak dapat mengadopsi penggunaan bagan Kanban untuk membantu mengatur rapat perencanaan harianya

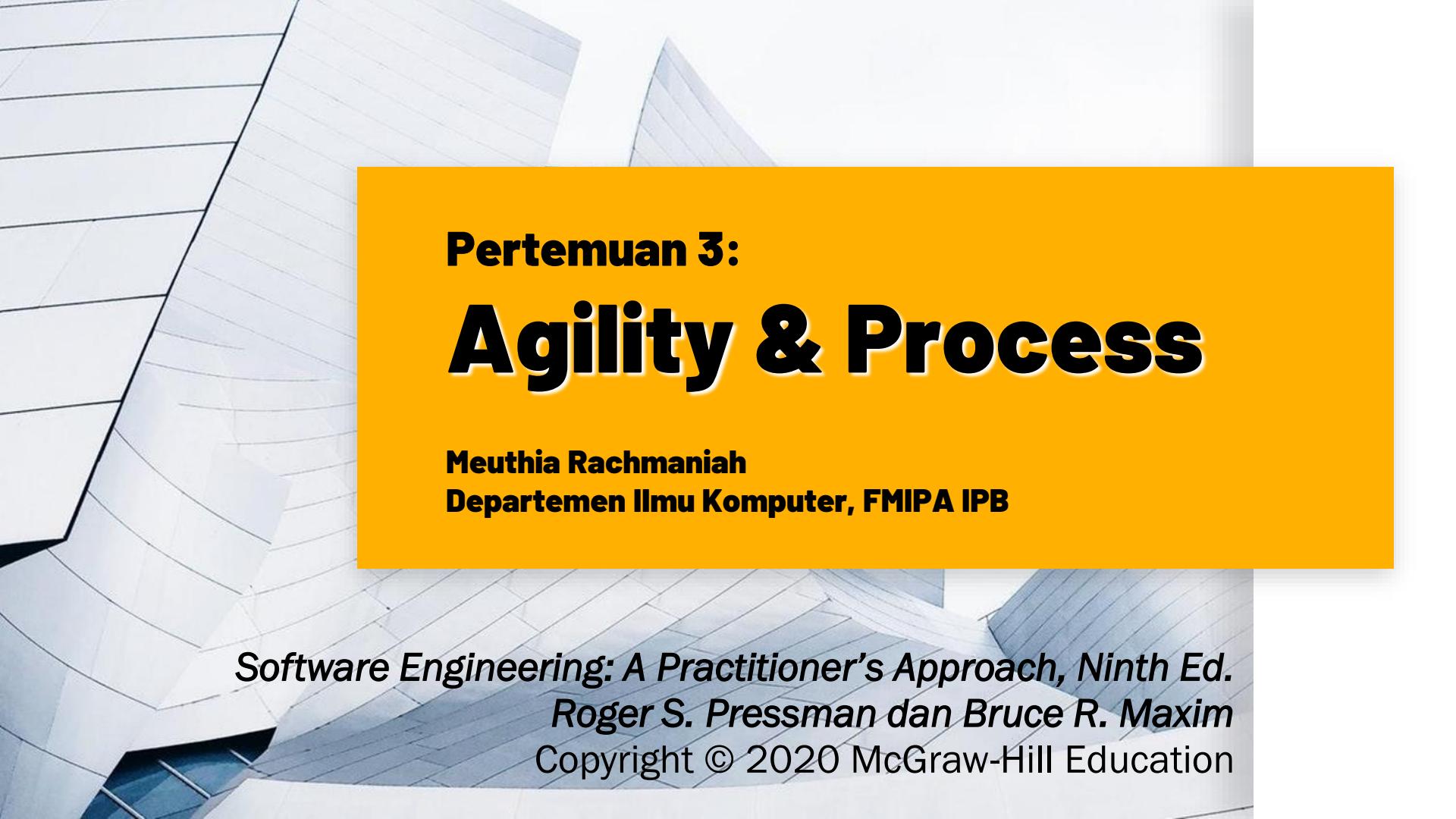
## Extreme Programming



Extreme programming (XP) diatur di sekitar empat framework aktivitas—perencanaan, desain, pengkodean, dan pengujian—XP menyarankan sejumlah teknik inovatif dan kuat yang memungkinkan tim agile membuat rilis PL yang sering menghadirkan fitur dan fungsionalitas yang telah dijelaskan dan kemudian diprioritaskan oleh stakeholder



Tidak ada yang mencegah tim XP untuk menggunakan teknik DevOps untuk mengurangi waktu penerapannya



# **Pertemuan 3:**

# **Agility & Process**

**Meuthia Rachmaniah**  
**Departemen Ilmu Komputer, FMIPA IPB**

*Software Engineering: A Practitioner's Approach, Ninth Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education

# Pertemuan 4: Recommended Process Model

Meuthia Rachmaniah

Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*

*Roger S. Pressman dan Bruce R. Maxim*

Copyright © 2020 McGraw-Hill Education

# 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

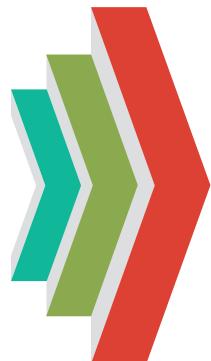
## Rekomendasi untuk Modern Software Development Project akibat kelemahan dari pendekatan Prescriptive Software Life Cycle

- 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

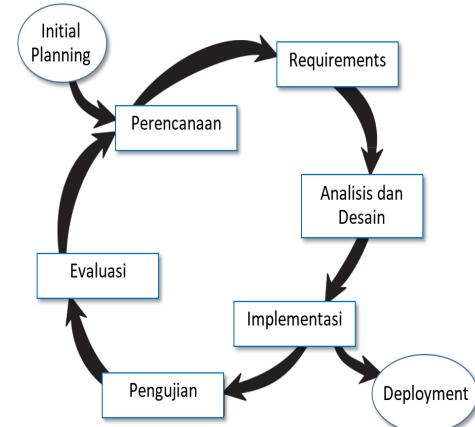
## Prescriptive Process Model (Model Waterfall)

- 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 prototipe 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:

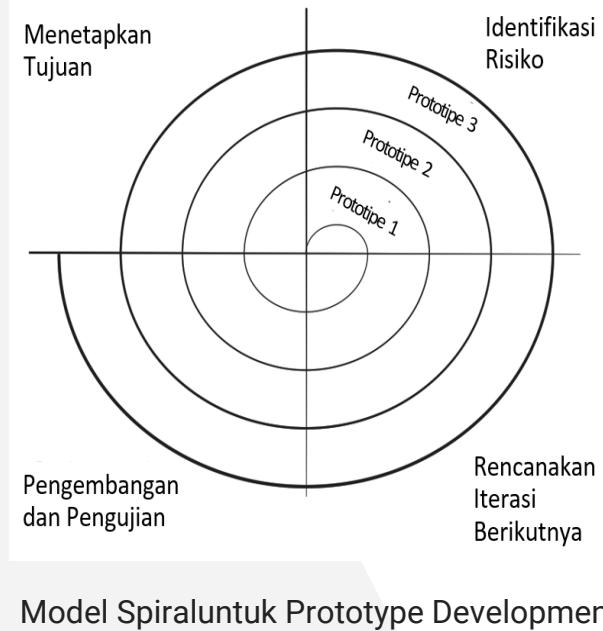
- ❖ Prototipe yang diciptakan sudah didesain untuk diperluas untuk increment software ke depan
- ❖ Stakeholder terlibat di seluruh proses pengembangan
- ❖ Keperluan dokumentasi ringan; dokumentasi harus berkembang seiring dengan perkembangan PL
- ❖ Pengujian direncanakan dan dieksekusi di awal



### 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



- ❖ 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 seiring 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

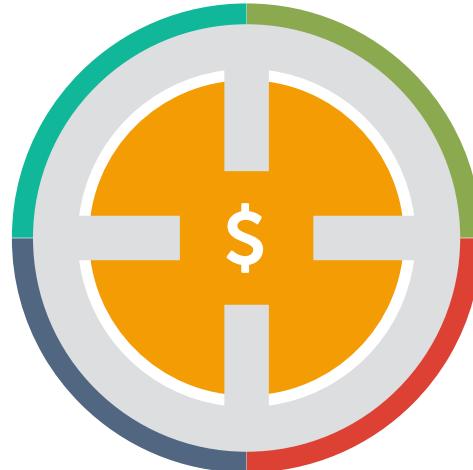


# Rangkuman Karakteristik Model Agile - SPIRAL



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 stakeholder
  - ▶ 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 pengrajaan ulang yang mahal, pembengkakan biaya, kualitas produk yang buruk, waktu pengiriman yang terlambat, pelanggan yang tidak puas, dan moral tim yang buruk

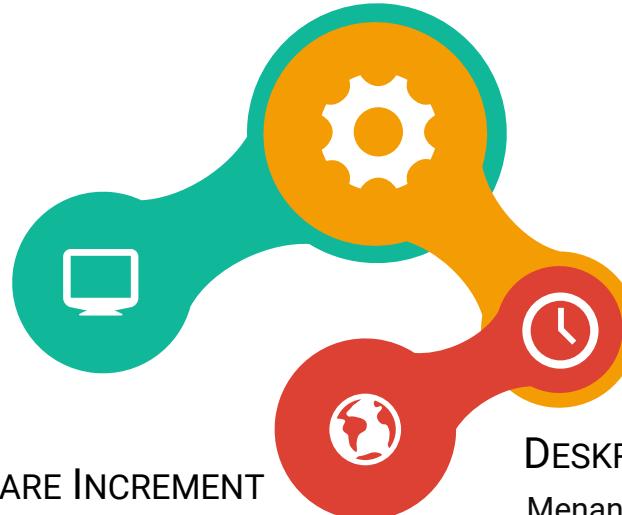


# Best Practices untuk definisi Agile Requirements

- 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 menggunakan
- 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 prioritas, 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



## REQUIREMENTS VS SOFTWARE INCREMENT

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

## 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

## DESKRIPSI STAKEHOLDER

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

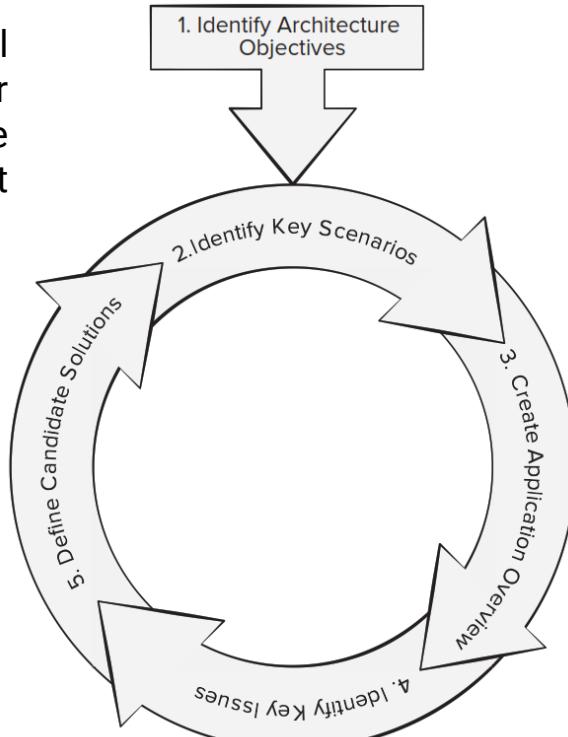


Ada juga risiko bahwa membuat prototipe sebelum persyaratan arsitektur perangkat lunak dipahami dengan baik dapat mengakibatkan prototipe harus dibuang, membuang-buang waktu dan sumber daya.

## 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 Skenario Kunci
  - 3) Ciptakan overview aplikasi
  - 4) Identifikasi Isu-isu 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

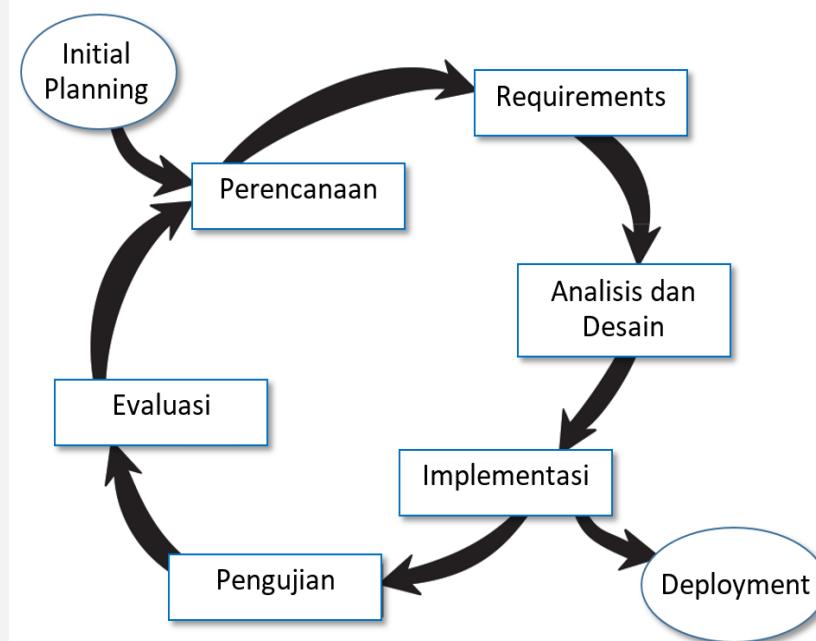
4

## 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



- 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

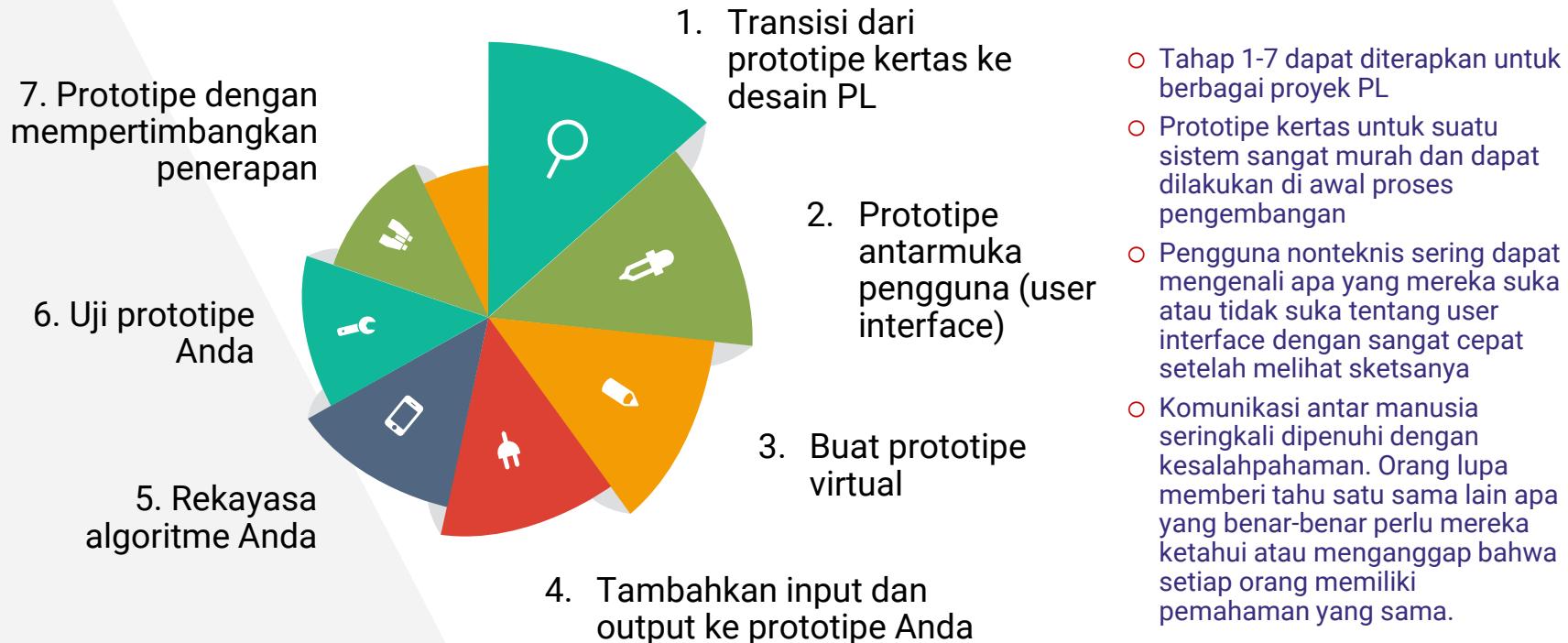
Putuskan berapa banyak waktu yang akan diizinkan untuk membuat prototipe pertama

- 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



## 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

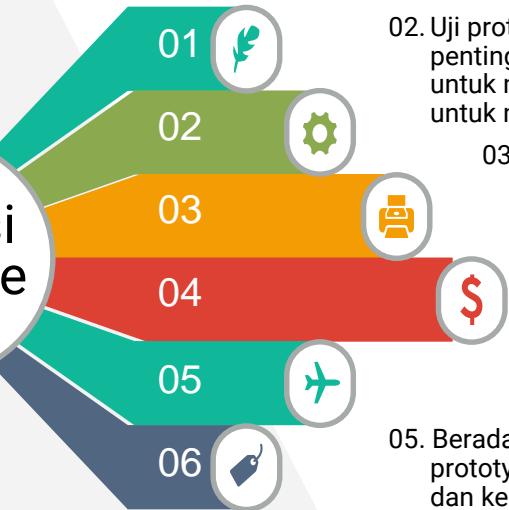
## Prototipe untuk Deployment

- 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

## Evaluasi Prototipe

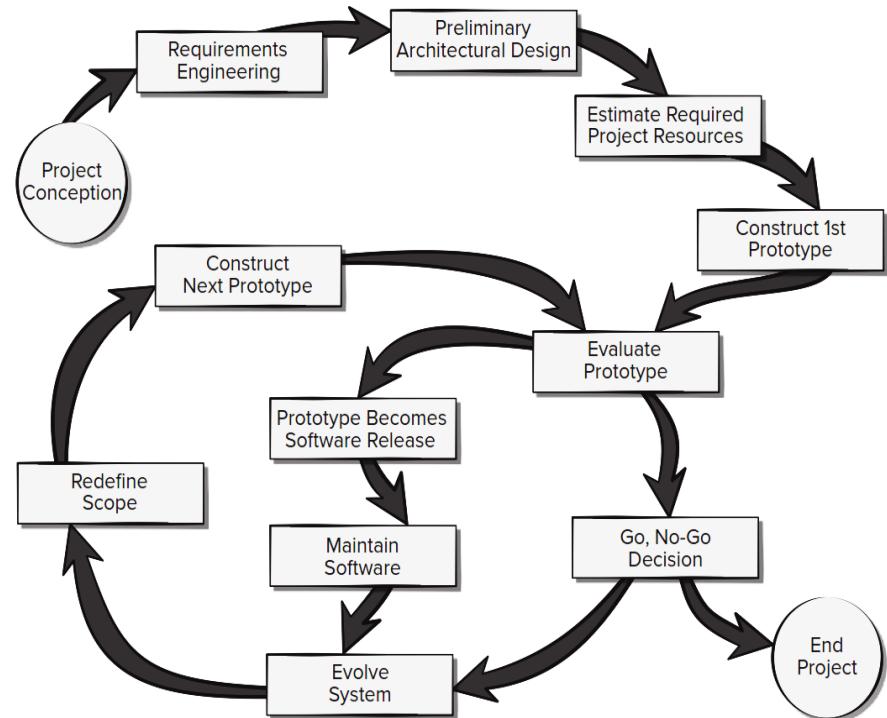


01. Berikan perancah (*scaffolding*), mekanisme untuk umpan balik secara tidak konfrontasi dengan statement: *I like* (mendorong umpan balik positif), *I wish* (untuk berbagi ide tentang bagaimana prototipe dapat ditingkatkan/feedback negatif), *What if* (untuk eksplorasi saat membuat prototipe di iterasi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

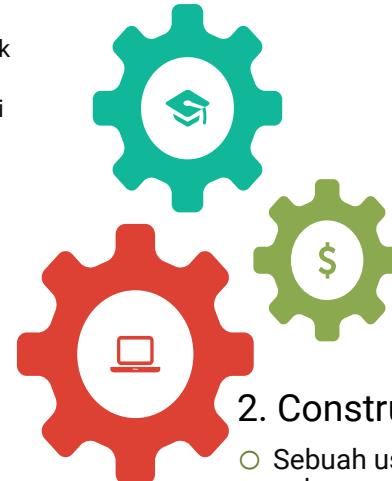
# 4.7 Prototype Evolution

# 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.

## 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



## 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

# 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 dilanjut 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.

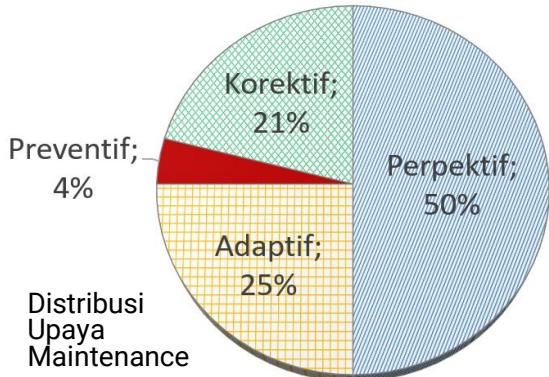


Isu dan pelajaran yang didapat dari pembuatan kandidat rilis harus didokumentasikan dan dipertimbangkan oleh pengembang dan stakeholder sebagai bagian dari postmortem proyek

# 4.9 Maintain Release Software

# 4.9 Maintenance 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



## 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 user 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.
6. Go, no-go decision
  - Test prototype using appropriate users.
  - Capture stakeholder feedback for use in revision process.
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, 9<sup>th</sup> Ed.*

*Roger S. Pressman dan Bruce R. Maxim*

Copyright © 2020 McGraw-Hill Education

# Pertemuan 5: Human Aspects Software Engineering

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education

# 5. Human Aspects Software Engineering

5.1 Karakteristik Software Engineer

5.2 Psikologi Software Engineering

5.3 The Software Team

5.4 Team Structures

5.5 Dampak Sosial Media

5.6 Global Teams

# QUICK LOOK

## WHAT IS IT?

- Pada akhirnya, orang membuat perangkat lunak computer
- Aspek manusia dari RPL sering kali berkaitan dengan keberhasilan proyek sebagai teknologi terbaru dan terhebat



## WHO DOES IT

- Individu dan tim melakukan pekerjaan RPL
- Dalam beberapa kasus, satu orang memiliki banyak tanggung jawab, tetapi dalam sebagian besar upaya PL tingkat industri, tim orang yang melakukan pekerjaan



## WHY IS IT IMPORTANT?

- Tim PL akan berhasil hanya jika dinamika tim benar
- Sangat penting bagi software engineer dalam tim untuk bermain baik dengan kolega mereka dan dengan stakeholder produk lainnya



## WHAT ARE THE STEPS

- Pertama, Anda perlu mencoba meniru karakteristik pribadi software engineer yang sukses
- Selanjutnya, Anda harus menghargai psikologi kompleks pekerjaan RPL sehingga Anda dapat menavigasi jalan Anda melalui proyek tanpa bahaya
- Kemudian, Anda perlu memahami struktur dan dinamika tim PL
- Terakhir, Anda harus menghargai dampak media sosial, cloud, dan perangkat bantu (tools) kolaboratif lainnya



## WHAT IS THE WORK PRODUCT

- Wawasan yang lebih baik tentang orang, proses, dan produk



## HOW DO I ENSURE THAT I'VE DONE IT RIGHT

- Luangkan waktu untuk mengamati bagaimana software engineer yang sukses melakukan pekerjaan mereka, dan sesuaikan pendekatan Anda untuk memanfaatkan kekuatan yang mereka proyeksikan

# 5.1 Karakteristik Software Engineer

# 5.1 Karakteristik Software Engineer



## SOFTWARE ENGINEER?

- ❑ Menguasai hal-hal teknis, mempelajari keterampilan yang diperlukan untuk memahami problem, merancang solusi yang efektif, membangun perangkat lunak, dan mengujinya dalam upaya mengembangkan produk dengan kualitas setinggi mungkin
- ❑ Mengelola perubahan, berkomunikasi dengan stakeholder, dan menggunakan perangkat bantu (tools) yang sesuai sesuai kebutuhan
- ❑ Aspek manusia yang akan membuat Anda efektif sebagai Software Engineer
  - ❑ **Tujuh ciri** yang hadir ketika seorang Software Engineer menunjukkan **perilaku "superprofesional"**

# Tujuh Ciri Software Engineer

## 7. Pragmatic

- Dia mengakui bahwa RPL bukanlah aturan dogmatis yang harus diikuti, melainkan disiplin yang dapat disesuaikan berdasarkan keadaan yang ada

## 6. Attention Detail

- Dia dengan hati-hati mempertimbangkan kriteria yang lebih luas (misalnya, kinerja, biaya, kualitas) yang telah ditetapkan untuk produk dan proyek dalam membuat keputusan teknis harianya.

## 5. Heightened Sense of Fairness

- Dia dengan senang hati berbagi pujian dengan rekan-rekannya.
- Dia berusaha menghindari konflik kepentingan dan tidak pernah bertindak untuk menyabotase pekerjaan orang lain

## 4. Resilience under Pressure

- Tekanan datang dalam berbagai bentuk—perubahan persyaratan dan prioritas, stakeholder yang menuntut, dan manajer yang sombong
- Seorang Software Engineer yang efektif mengelola tekanan sehingga kinerjanya tidak terganggu



## 1. Individual Responsibility

- Dorongan untuk memenuhi janjinya kepada rekan kerja, stakeholder, dan manajemennya
- Ini menyiratkan bahwa dia akan melakukan apa yang perlu dilakukan, ketika itu perlu dilakukan dalam upaya utama untuk mencapai hasil yang sukses

## 2. Acute Awareness

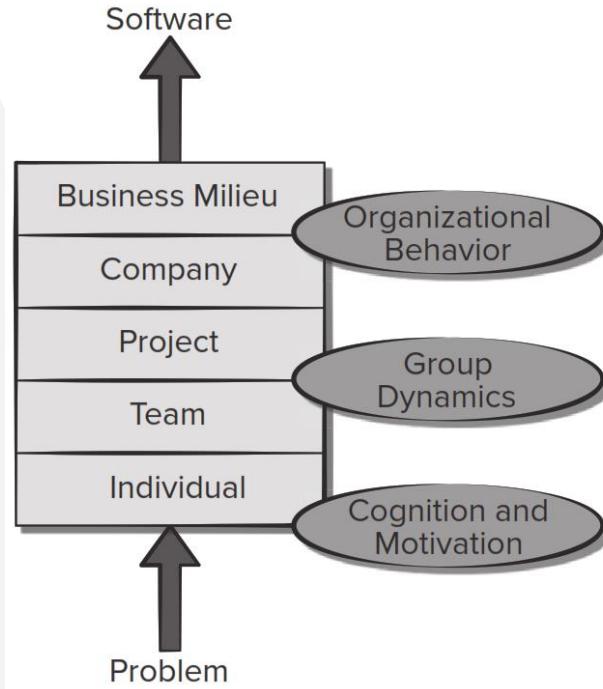
- Memiliki kesadaran yang tinggi akan kebutuhan anggota tim lainnya, stakeholder yang meminta perubahan pada solusi PL yang ada, dan manajer yang memiliki kendali keseluruhan atas proyek
- Mengamati lingkungan tempat orang bekerja dan menyesuaikan perilakunya dengan mempertimbangkan keduanya

## 3. Brutally Honest

- Jika dia melihat desain yang cacat, dia menunjukkan kekurangannya dengan cara yang konstruktif tetapi jujur
- Jika diminta untuk mengubah fakta tentang jadwal, fitur, kinerja, atau karakteristik produk atau proyek lainnya, dia memilih untuk bersikap realistik dan jujur

## 5.2 Psikologi Software Engineering

# 5.2 Psikologi Software Engineering



## COGNITION AND MOTIVATION

- Psikologi RPL berfokus pada pengenalan problem yang harus dipecahkan, keterampilan pemecahan problem yang diperlukan untuk menyelesaiannya, dan motivasi untuk menyelesaikan solusi dalam batasan yang ditetapkan oleh lapisan luar dalam model



## GROUP DYNAMICS

- Dinamika kelompok (Group Dynamics) menjadi faktor yang mendominasi.
- Struktur tim dan faktor sosial mengatur kesuksesan
- Komunikasi kelompok, kolaborasi, dan koordinasi sama pentingnya dengan keterampilan anggota tim individu



## ORGANIZATIONAL BEHAVIOR

- mengatur tindakan perusahaan (company) dan tanggapannya terhadap lingkungan bisnis (business mileu)

## 5.3 The Software Team

# 5.3 The Software Team

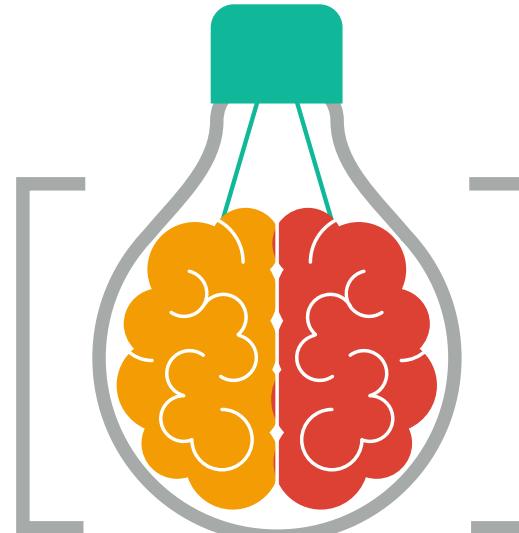
- Kekompakan tim PL
  - “Yelled Team” lebih produktif dan termotivasi
  - Sense of Purpose (Rasa yang sama pada Tujuan)
  - Sense of Involvement (Rasa Keterlibatan)
  - Sense of Trust (Rasa Mempercayai)
  - Sense of Improvement (Rasa Peningkatan)
  - Mengkombinasikan beragam keterampilan yang berbeda

Team Toxicity (Jackman)	Cara Menghindarinya
1) Suasana kerja yang hiruk pikuk	Tim harus memiliki akses ke semua informasi yang diperlukan untuk melakukan pekerjaan itu. Tujuan dan sasaran utama, setelah ditetapkan, tidak boleh diubah kecuali benar-benar diperlukan.
2) Frustrasi tinggi yang menyebabkan gesekan di antara anggota tim	Diberikan sebanyak mungkin tanggung jawab untuk pengambilan keputusan
3) Proses PL ‘terfragmentasi atau kurang terkoordinasi’	Memahami produk yang akan dibangun dan orang-orang yang melakukan pekerjaan dan dengan mengizinkan tim untuk memilih model proses.
4) Definisi peran yang tidak jelas dalam tim PL	Menetapkan mekanisme akuntabilitasnya sendiri (tinjauan teknis adalah cara terbaik untuk mencapai hal ini) dan menentukan serangkaian pendekatan korektif ketika seorang anggota tim gagal melakukan
5)“Paparan kegagalan yang terus menerus dan berulang”	Membangun teknik berbasis tim untuk umpan balik dan pemecahan masalah (problem solving)

Tim PL sering bergumul dengan sifat manusia yang berbeda dari anggotanya

## Perbedaan Sifat Manusia

- Beberapa orang mengumpulkan informasi secara intuitif, menyaring konsep luas dari fakta yang berbeda
  - Yang lain memproses informasi secara linier, mengumpulkan dan mengatur detail kecil dari data yang diberikan
- Beberapa anggota tim merasa nyaman membuat keputusan hanya ketika argumen yang logis dan teratur disajikan
- Yang lain intuitif, bersedia membuat keputusan berdasarkan "perasaan".
  - Beberapa bekerja keras untuk menyelesaikan sesuatu jauh sebelum tanggal pencapaian, sehingga menghindari stres saat mendekati tanggal, sementara yang lain diberi energi oleh terburu-buru untuk membuat tenggat waktu menit terakhir.



Pengakuan perbedaan manusia, bersama dengan pedoman lain yang disajikan di bagian ini, memberikan kemungkinan yang lebih tinggi untuk menciptakan tim yang jeli

## 5.4 Team Structures

## 5.4 Team Structures



- ▶ Struktur tim "terbaik" tergantung pada gaya manajemen organisasi Anda, jumlah orang yang akan mengisi tim dan tingkat keahlian mereka, dan tingkat kesulitan masalah secara keseluruhan
- ▶ Tujuh faktor proyek yang harus dipertimbangkan ketika merencanakan struktur tim RPL:
  - 1) Kesulitan problem yang akan dipecahkan
  - 2) "Ukuran" dari program yang dihasilkan dalam baris kode atau titik fungsi
  - 3) Waktu tim akan tetap bersama (tim seumur hidup)
  - 4) Sejauh mana problem dapat dimodulasi
  - 5) Keperluan kualitas dan keandalan sistem yang akan dibangun
  - 6) Kekakuan tanggal pengiriman
  - 7) Tingkat sosialisasi (komunikasi) yang diperlukan untuk proyek

## 5.4 Team Structures



- ▶ Selama dekade terakhir, pengembangan PL agile telah disarankan sebagai penangkal banyak problem yang mengganggu pekerjaan proyek PL
- ▶ Filosofi agile mendorong kepuasan pelanggan dan pengiriman PL tambahan awal, tim proyek kecil bermotivasi tinggi, metode informal, produk kerja RPL minimal, dan kesederhanaan pengembangan secara keseluruhan
- ▶ Tim agile, tim proyek kecil yang bermotivasi tinggi dan berkompetensi serta berkolaborasi, mengadopsi banyak karakteristik tim proyek PL yang sukses

## 5.5 Dampak Sosial Media

# 5.5 Dampak Sosial Media

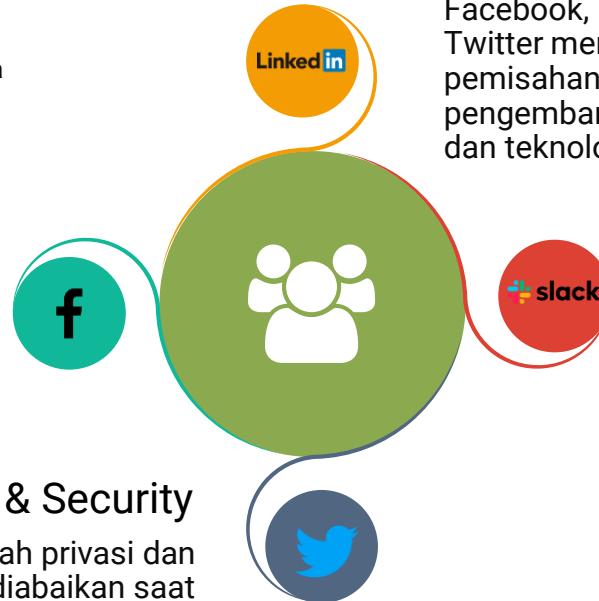
- Sebagian besar pekerjaan yang dilakukan oleh Software Engineer mungkin milik majikan mereka dan pengungkapannya bisa sangat berbahaya
- Oleh karena itu, manfaat media sosial yang berbeda harus ditimbang terhadap ancaman pengungkapan informasi pribadi yang tidak terkendali

## Specialized Private Network

Jaringan pribadi khusus (specialized private network) yang dibangun di atas paradigma jejaring sosial dapat digunakan dalam suatu organisasi

## Privacy & Security

Perhatikan bahwa masalah privasi dan keamanan tidak boleh diabaikan saat menggunakan media sosial untuk pekerjaan RPL



## Social Networking Tools

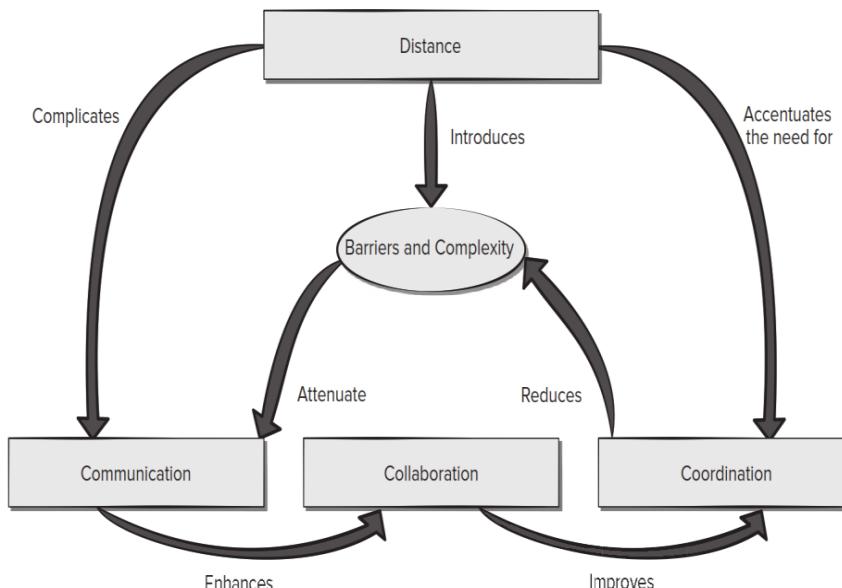
Facebook, LinkedIn, Slack, Twitter memungkinkan tingkat pemisahan koneksi antara pengembang perangkat lunak dan teknologi terkait

## “Friends”

Memungkinkan “teman” di situs jejaring sosial untuk belajar tentang teman dari teman yang mungkin memiliki pengetahuan atau keahlian terkait dengan domain aplikasi atau masalah yang harus dipecahkan

## 5.6 Global Teams

# 5.6 Global Teams



Faktor yang mempengaruhi tim Global Software Development (GSD)

## Globalisasi

Selama beberapa dekade terakhir, semakin banyak produk perangkat lunak utama telah dibangun oleh tim perangkat lunak yang sering berlokasi di berbagai negara

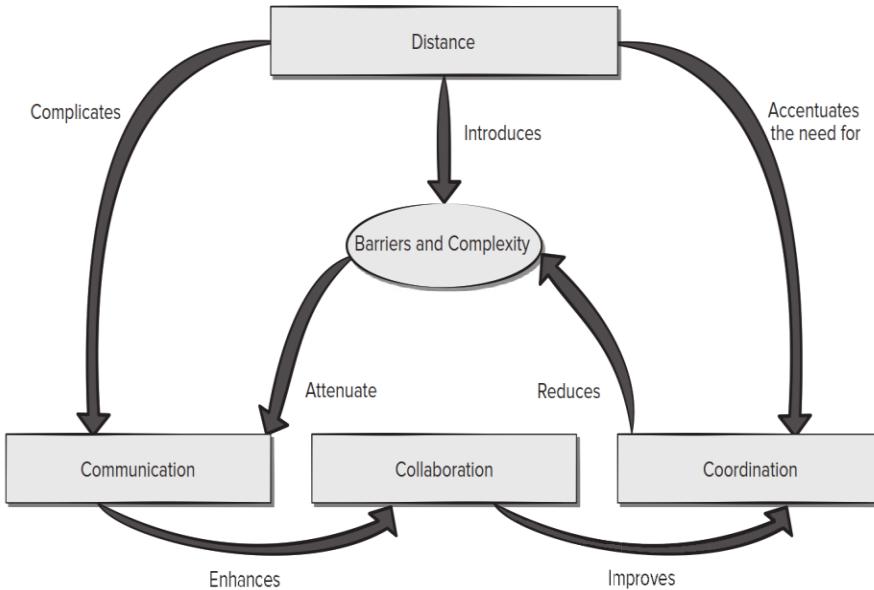
## Tim Global Software Development (GSD)

- Memiliki tantangan unik yang mencakup koordinasi, kolaborasi, komunikasi, dan pengambilan keputusan khusus
- Pendekatan koordinasi, kolaborasi, dan komunikasi dipengaruhi oleh struktur tim yang telah dibentuk

## Media Sosial & Komunikasi Elektronik

Media sosial dan komunikasi elektronik sangat berguna untuk pengembangan PL global

## 5.6 Global Teams



Faktor yang mempengaruhi tim Global Software Development (GSD)

### Dampak jarak

- Jarak memperumit komunikasi, tetapi, pada saat yang sama, menekankan (accentuate) perlunya koordinasi.
- Jarak juga memperkenalkan hambatan dan kompleksitas yang dapat didorong oleh perbedaan budaya
- Hambatan dan kompleksitas melemahkan (attenuate) komunikasi (yaitu, rasio signal-to-noise menurun)
- Masalah yang melekat dalam dinamika ini dapat mengakibatkan proyek menjadi tidak stabil

### Empat Faktor Pembuatan Keputusan

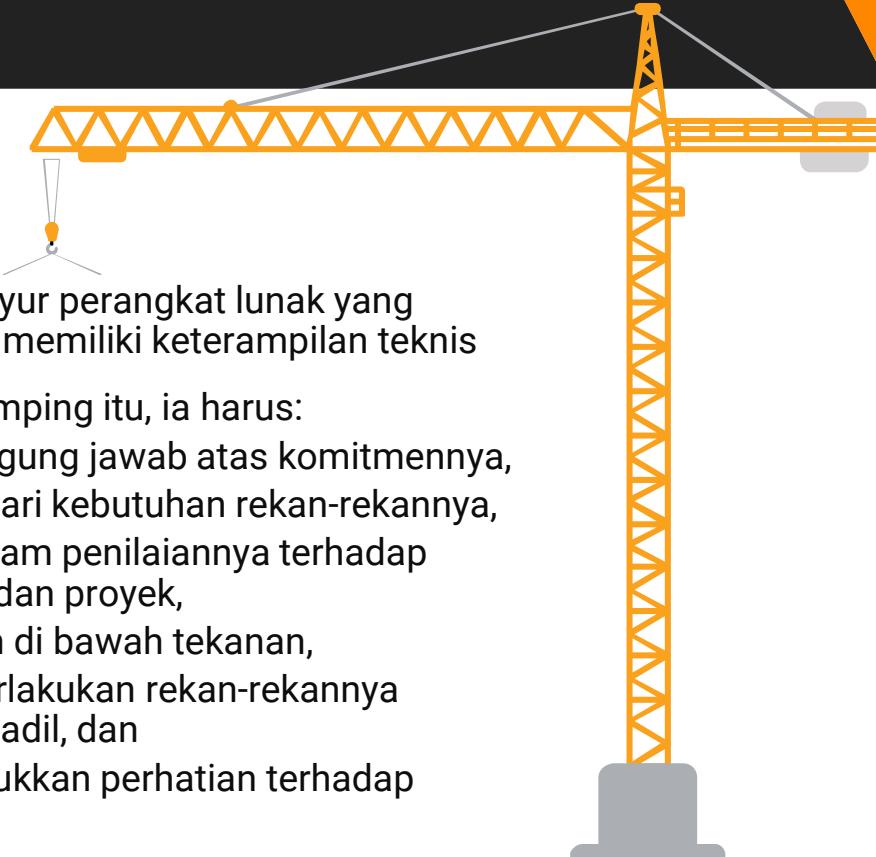
- 1) Kompleksitas dari problem
- 2) Ketidakpastian dan risiko yang terkait dengan keputusan
- 3) Hukum konsekuensi yang tidak diinginkan (yaitu, keputusan terkait pekerjaan memiliki efek yang tidak diinginkan pada tujuan proyek lain)
- 4) Pandangan yang berbeda dari masalah yang mengarah pada kesimpulan yang berbeda tentang jalan ke depan

## 5.7 Rangkuman

# Rangkuman



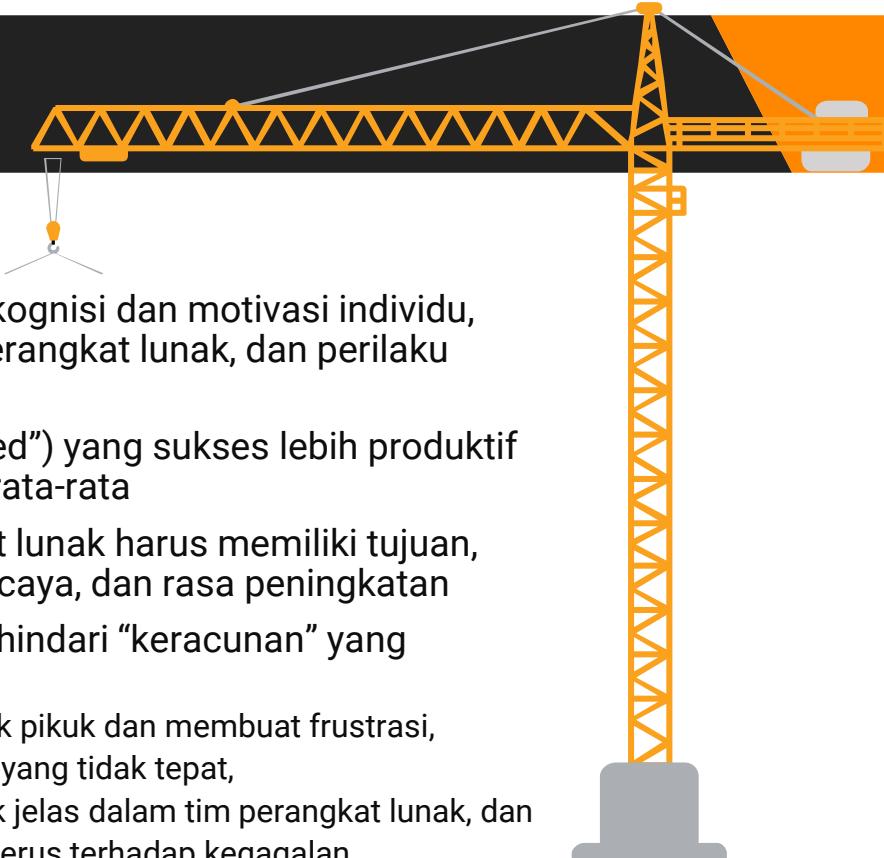
- Seorang insinyur perangkat lunak yang sukses harus memiliki keterampilan teknis
- Namun, di samping itu, ia harus:
  - bertanggung jawab atas komitmennya,
  - menyadari kebutuhan rekan-rekannya,
  - jujur dalam penilaianya terhadap produk dan proyek,
  - tangguh di bawah tekanan,
  - memperlakukan rekan-rekannya dengan adil, dan
  - menunjukkan perhatian terhadap detail.



## Rangkuman



- Psikologi RPL mencakup kognisi dan motivasi individu, dinamika kelompok tim perangkat lunak, dan perilaku organisasi perusahaan
- Tim perangkat lunak (“jelled”) yang sukses lebih produktif dan termotivasi daripada rata-rata
- Agar efektif, tim perangkat lunak harus memiliki tujuan, rasa keterlibatan, rasa percaya, dan rasa peningkatan
- Selain itu, tim harus menghindari “keracunan” yang ditandai dengan:
  - suasana kerja yang hiruk pikuk dan membuat frustrasi,
  - proses perangkat lunak yang tidak tepat,
  - definisi peran yang tidak jelas dalam tim perangkat lunak, dan
  - keterpaparan terus-menerus terhadap kegagalan.



## Rangkuman



- Tim agile menganut filosofi agile (gesit) dan umumnya memiliki otonomi yang lebih besar daripada tim PL yang lebih konvensional dengan peran anggota yang kaku dan kontrol manajemen eksternal
- Tim agile menekankan komunikasi, kesederhanaan, umpan balik, keberanian, dan rasa hormat



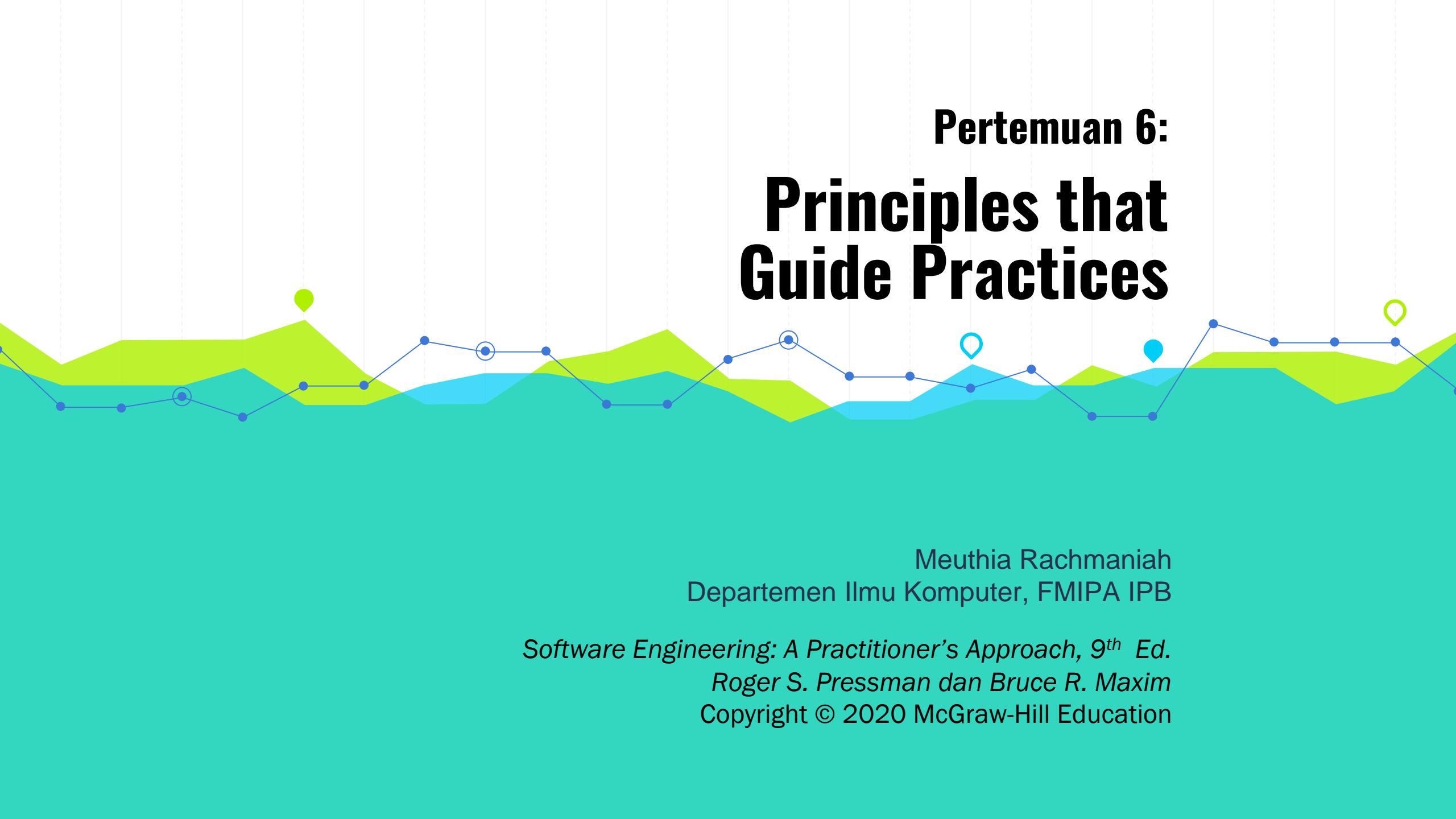
- Alat media sosial menjadi bagian integral dari banyak proyek PL, menyediakan layanan yang meningkatkan komunikasi dan kolaborasi untuk tim PL
- Media sosial dan komunikasi elektronik sangat berguna untuk pengembangan PL global di mana pemisahan geografis dapat memicu hambatan untuk RPL yang sukses



# Pertemuan 5: Human Aspects Software Engineering

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
*Roger S. Pressman dan Bruce R. Maxim*  
Copyright © 2020 McGraw-Hill Education



# Pertemuan 6: Principles that Guide Practices

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
Roger S. Pressman dan Bruce R. Maxim  
Copyright © 2020 McGraw-Hill Education

# 6. Principles that Guide Practices

## ● 6.1 CORE PRINCIPLES

- 6.1.1 Principles that Guide Process (Prinsip yang Memandu Proses)
- 6.1.2 Principles that Guide Practices (Prinsip yang Memandu Praktik)

## ● 6.2 PRINCIPLES THAT GUIDE EACH FRAMEWORK ACTIVITY

- 6.2.1 Communication Principles (Prinsip Komunikasi)
- 6.2.2 Planning Principles (Prinsip Perencanaan)
- 6.2.3 Modeling Principles (Prinsip Pemodelan)
- 6.2.4 Construction Principles (Prinsip Konstruksi)
- 6.2.5 Deployment Principles (Prinsip Deployment)

## ● 6.3 RANGKUMAN

# QUICK LOOK

## WHAT IS IT?



- Praktik RPL ialah array dari prinsip-prinsip, konsep-konsep, metode-metode, dan perangkat bantu (tools) yang harus Anda pertimbangkan ketika PL direncanakan dan dikembangkan
- Prinsip-prinsip yang memandu praktik membentuk fondasi untuk melakukan RPL

## WHO DOES IT



- Praktisi (Software Engineer) dan manajer melakukan berbagai tugas RPL

## WHY IS IT IMPORTANT?



- Proses PL memberi semua orang yang terlibat dalam pembuatan sistem atau produk berbasis komputer dengan peta jalan (road map) untuk mencapai tujuan yang sukses
- Praktik PL memberi Anda detail yang Anda perlukan seperti untuk mengemudi di sepanjang jalan
- Proses PL membantu untuk memahami konsep-konsep dan prinsip-prinsip
- Dalam konteks RPL, praktik adalah apa yang Anda lakukan hari demi hari saat PL berkembang dari ide menjadi kenyataan

## WHAT ARE THE STEPS



- Empat elemen praktik berlaku terlepas dari model proses yang dipilih:
  - Prinsip, konsep, metode, dan perangkat bantu (tools)
  - Perangkat bantu (tools) mendukung penerapan metode

## WHAT IS THE WORK PRODUCT



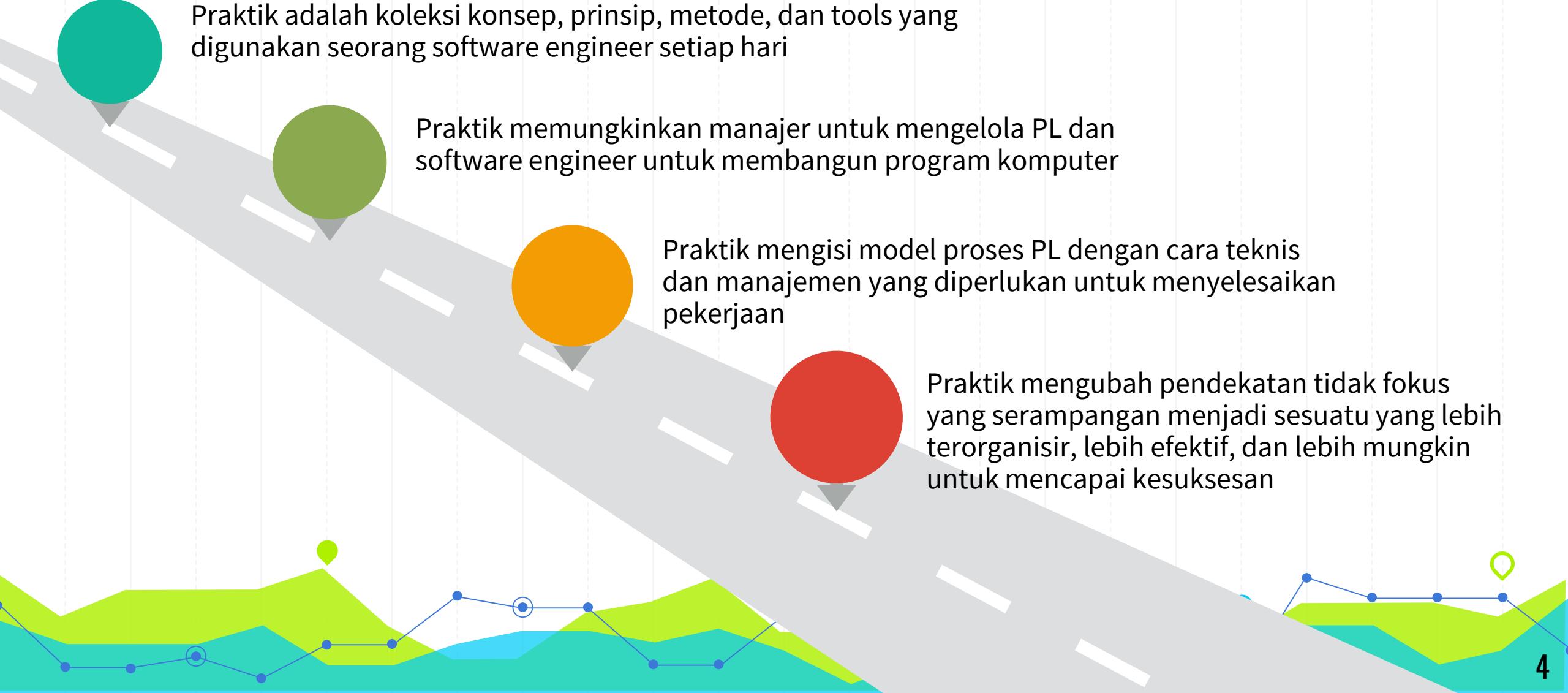
- Praktik meliputi kegiatan teknis yang menghasilkan semua produk kerja (work product) yang didefinisikan oleh model proses PL yang telah dipilih

## How Do I ENSURE THAT I'VE DONE IT RIGHT



- Pertama, memiliki pemahaman yang kuat tentang prinsip-prinsip yang berlaku untuk pekerjaan (misalnya, desain) yang Anda lakukan saat ini
- Kemudian, pastikan bahwa Anda telah memilih metode yang sesuai untuk pekerjaan itu, pastikan bahwa Anda memahami cara menerapkan metode tersebut, menggunakan perangkat bantu (tools) otomatis saat sesuai untuk tugas tersebut, dan bersikeras tentang perlunya teknik untuk memastikan kualitas produk kerja yang dihasilkan
- Anda juga harus cukup gesit (agile) untuk membuat perubahan pada rencana dan metode Anda sesuai kebutuhan

# What is Software Engineering “Practice”

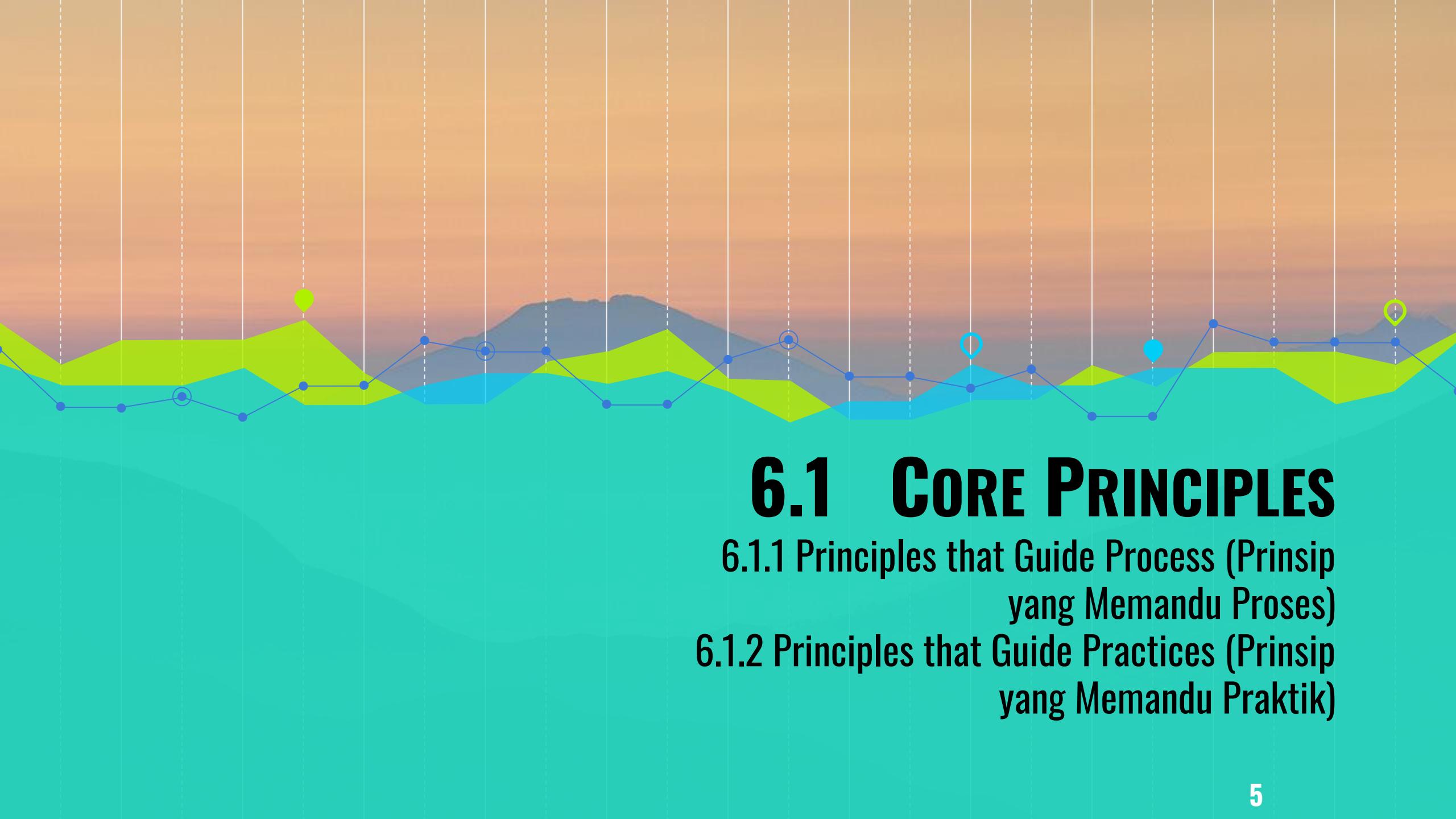


Praktik adalah koleksi konsep, prinsip, metode, dan tools yang digunakan seorang software engineer setiap hari

Praktik memungkinkan manajer untuk mengelola PL dan software engineer untuk membangun program komputer

Praktik mengisi model proses PL dengan cara teknis dan manajemen yang diperlukan untuk menyelesaikan pekerjaan

Praktik mengubah pendekatan tidak fokus yang serampangan menjadi sesuatu yang lebih terorganisir, lebih efektif, dan lebih mungkin untuk mencapai kesuksesan



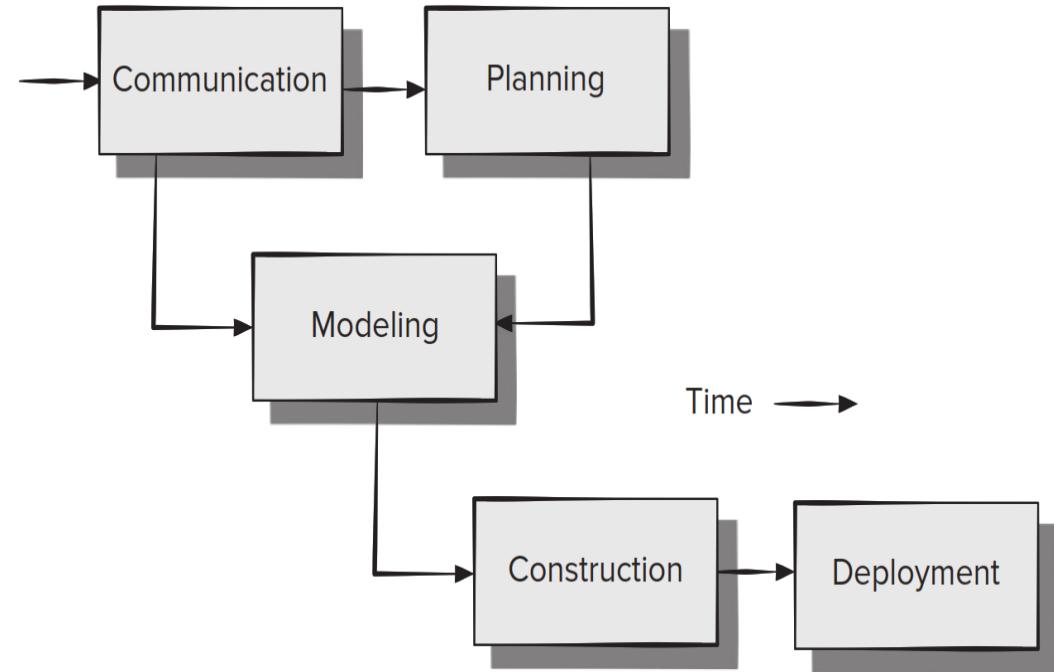
## **6.1 CORE PRINCIPLES**

**6.1.1 Principles that Guide Process (Prinsip yang Memandu Proses)**

**6.1.2 Principles that Guide Practices (Prinsip yang Memandu Praktik)**

## 6.1.1 Principles that Guide PROCESS (Prinsip yang Memandu PROSES)

- Setiap proyek itu unik, dan setiap tim itu unik
  - Itu berarti Anda harus menyesuaikan proses Anda agar sesuai dengan kebutuhan Anda.
  - Terlepas dari model proses yang diadopsi tim Anda, proses berisi elemen dari framework proses generik
- Framework proses yang disederhanakan ialah seperti gambar di kanan ini



Framework Proses yang  
Disederhanakan

# Delapan prinsip inti dapat diterapkan pada FRAMEWORK PROSES dan untuk setiap PROSES PERANGKAT LUNAK

## ● **Prinsip 1. Harus Agile**

- Apakah model proses yang Anda pilih bersifat preskriptif atau agile, prinsip dasar pengembangan agile harus mengatur pendekatan Anda
- Setiap aspek pekerjaan yang Anda lakukan harus menekankan tindakan ekonomi—buat pendekatan teknis Anda sesederhana mungkin, pertahankan produk kerja yang Anda hasilkan sesingkat mungkin, dan buat keputusan secara lokal bila memungkinkan

## ● **Prinsip 2. Fokus pada Kualitas di Setiap langkah**

- Kondisi keluar untuk setiap aktivitas proses, tindakan, dan tugas harus fokus pada kualitas produk kerja yang telah dihasilkan

## ● **Prinsip 3. Selalu Siap Beradaptasi**

- Proses bukanlah pengalaman religius, dan dogma tidak memiliki tempat di dalamnya
- Bila perlu, sesuaikan pendekatan Anda dengan kendala yang ditimbulkan oleh problem, orang-orang, dan proyek itu sendiri

## ● **Prinsip 4. Bangun Tim yang Efektif**

- Proses dan praktik rekayasa perangkat lunak itu penting, tetapi intinya adalah manusia
- Bangun tim yang mengatur diri sendiri yang saling percaya dan menghormati

**Delapan prinsip inti dapat diterapkan pada framework proses dan untuk setiap proses perangkat lunak**

● **Prinsip 5. Bangun mekanisme komunikasi & koordinasi**

- Proyek gagal karena informasi penting tidak sampai dan/atau stakeholder gagal mengoordinasikan upaya mereka untuk menciptakan produk akhir yang sukses
- Ini adalah masalah manajemen, dan mereka harus ditangani

● **Prinsip 6. Kelola perubahan**

- Pendekatannya bisa formal atau informal, tetapi mekanisme harus ditetapkan untuk mengelola cara perubahan diminta, dinilai, disetujui, dan diimplementasikan

● **Prinsip 7. Kaji risiko**

- Banyak hal bisa salah ketika PL sedang dikembangkan
- Sangat penting bagi Anda untuk membuat rencana darurat
- Beberapa dari rencana darurat ini akan menjadi dasar untuk tugas-tugas rekayasa keamanan

● **Prinsip 8. Ciptakan Work Product dan pertimbangkan orang lain**

- Buat hanya produk kerja yang memberikan nilai untuk aktivitas, tindakan, atau tugas proses lainnya.
- Setiap produk kerja yang dihasilkan sebagai bagian dari praktik rekayasa perangkat lunak akan diteruskan ke orang lain.
- Pastikan bahwa produk kerja memberikan informasi yang diperlukan tanpa ambiguitas atau kelalaian

## 6.1.2 Principles that Guide PRACTICES (Prinsip yang Memandu PRAKTIK)

- Praktik RPL memiliki satu tujuan utama:
  - untuk memberikan PL operasional yang tepat waktu, berkualitas tinggi, yang berisi fungsi dan fitur yang memenuhi kebutuhan semua stakeholder
- Untuk mencapai tujuan ini, Anda harus mengadopsi seperangkat prinsip inti yang memandu pekerjaan teknis Anda
  - Prinsip-prinsip ini memiliki manfaat terlepas dari metode analisis dan desain yang Anda terapkan, teknik konstruksi (misalnya, bahasa pemrograman, perangkat bantu otomatis) yang Anda gunakan, atau pendekatan verifikasi dan validasi yang Anda pilih

# Delapan Prinsip Inti Sangat Mendasar bagi PRAKTIK RPL



## Prinsip 1: Divide and conquer

- Dinyatakan dengan cara yang lebih teknis, analisis dan desain harus selalu menekankan pemisahan perhatian/kekhawatiran (*Separation of Concerns*-SoCs)
- Masalah besar lebih mudah dipecahkan jika dibagi lagi menjadi kumpulan elemen (atau masalah/concerns)



## Prinsip 2: Memahami Penggunaan Abstraksi

- Abstraksi adalah penyederhanaan beberapa elemen kompleks dari sistem yang digunakan untuk mengkomunikasikan makna dalam satu frase
- Saat menggunakan spreadsheet abstraksi, pahami apa itu spreadsheet, struktur umum konten yang disajikan spreadsheet, dan fungsi umum yang dapat diterapkan
- Menggunakan banyak tingkat abstraksi yang berbeda, masing-masing menyampaikan atau menyiratkan makna yang harus dikomunikasikan
- Dalam pekerjaan analisis dan desain, tim PL biasanya memulai dengan model yang mewakili abstraksi tingkat tinggi (misalnya, spreadsheet) dan perlahan menyempurnakan model tersebut ke tingkat abstraksi yang lebih rendah (misalnya, kolom atau fungsi SUM)



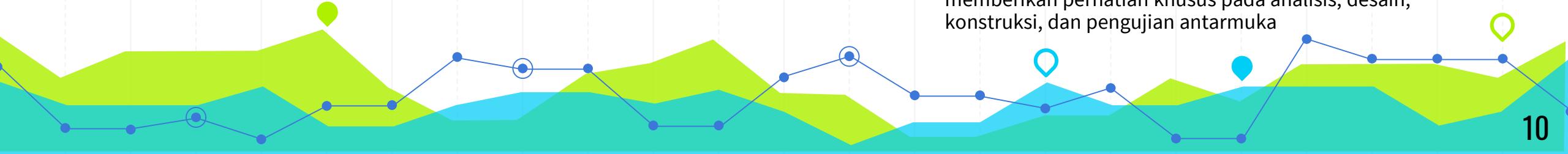
## Prinsip 3: Berusaha untuk konsistensi

- Ketika membuat model analisis, mengembangkan desain PL, menghasilkan kode sumber, atau membuat kasus uji, prinsip konsistensi menunjukkan bahwa konteks yang familiar membuat PL lebih mudah digunakan
- Sebagai contoh, pertimbangkan desain antarmuka pengguna untuk aplikasi seluler:
  - Penempatan opsi menu yang konsisten, penggunaan skema warna yang konsisten, dan penggunaan ikon yang dapat dikenali secara konsisten membantu menciptakan pengalaman pengguna (*user experience*) yang sangat efektif



## Prinsip 4: Fokus pada Transfer Informasi

- PL adalah tentang transfer informasi—from database ke pengguna akhir, dari sistem warisan ke WebApp, dari pengguna akhir ke antarmuka pengguna grafis (GUI), dari sistem operasi ke aplikasi, dari satu komponen PL ke komponen lainnya—daftarnya hampir tidak ada habisnya
- Dalam setiap kasus, informasi mengalir melintasi antarmuka, dan ini berarti ada peluang untuk kesalahan, kelalaian, atau ambiguitas
- Implikasi dari prinsip ini adalah Anda harus memberikan perhatian khusus pada analisis, desain, konstruksi, dan pengujian antarmuka



# Delapan Prinsip Inti Sangat Mendasar bagi PRAKTIK RPL



## Prinsip 5: Bangun PL yang Menunjukkan Modularitas Efektif

- Pemisahan masalah (Prinsip 1) menetapkan filosofi untuk PL
- Modularitas menyediakan mekanisme untuk mewujudkan filosofi
- Sistem kompleks apa pun dapat dibagi menjadi modul (komponen), tetapi praktik RPL yang baik menuntut lebih banyak
- Modularitas harus efektif. Artinya, setiap modul harus fokus secara eksklusif pada satu aspek sistem yang dibatasi dengan baik
- Modul harus saling berhubungan dengan cara yang relatif sederhana ke modul lain, ke sumber data, dan aspek lingkungan lainnya



## Prinsip 6: Carilah Pola

- Software Engineer menggunakan pola sebagai sarana untuk membuat katalog dan menggunakan kembali solusi untuk problem yang mereka temui di masa lalu
- Penggunaan pola desain ini dapat diterapkan pada rekayasa sistem dan masalah integrasi sistem yang lebih luas, dengan memungkinkan komponen dalam sistem yang kompleks berkembang secara independen



## Prinsip 7: Jika Memungkinkan, Tunjukkan Problem dan Solusinya dari Beberapa Perspektif yang Berbeda

- Ketika sebuah problem dan solusinya diperiksa dari perspektif yang berbeda, kemungkinan besar wawasan yang lebih besar akan tercapai dan kesalahan serta kelalaian akan terungkap
- Unified Modeling Language (UML) menyediakan sarana untuk menggambarkan solusi problem dari berbagai sudut pandang



## Prinsip 8: Ingatlah Bahwa Seseorang Akan Memelihara PL

- Dalam jangka panjang, PL akan dikoreksi saat cacat ditemukan, disesuaikan dengan perubahan lingkungannya, dan ditingkatkan saat stakeholder meminta lebih banyak kemampuan
- Kegiatan pemeliharaan ini dapat difasilitasi jika praktik RPL yang solid diterapkan di seluruh proses PL

## **6.2 PRINCIPLES THAT GUIDE EACH FRAMEWORK ACTIVITY**

- 6.2.1 Communication Principles (Prinsip Komunikasi)**
- 6.2.2 Planning Principles (Prinsip Perencanaan)**
- 6.2.3 Modeling Principles (Prinsip Pemodelan)**
- 6.2.4 Construction Principles (Prinsip Konstruksi)**
- 6.2.5 Deployment Principles (Prinsip Deployment)**

## 6.2.1 Communication Principles (Prinsip KOMUNIKASI)

- Perbedaan Customer dan End User
- Dalam beberapa kasus, customer (pelanggan) dan pengguna akhir (end user) mungkin satu dan sama, tetapi untuk banyak proyek tidak demikian
- Customers adalah seseorang/grup yang :
  - 1) awalnya meminta PL yang akan dibangun,
  - 2) mendefinisikan tujuan bisnis keseluruhan untuk PL,
  - 3) menyediakan persyaratan produk dasar, dan
  - 4) mengoordinasikan pendanaan untuk proyek
- End Users adalah seseorang/grup yang:
  - 1) Akan benar-benar menggunakan PL yang dibuat untuk mencapai beberapa tujuan bisnis
  - 2) Akan menentukan detail operasional PL sehingga tujuan bisnis dapat tercapai

## 6.2.1 Communication Principles (Prinsip KOMUNIKASI)

- Persyaratan pelanggan harus dikumpulkan melalui aktivitas komunikasi
- Pelanggan memiliki problem yang mungkin dapat diselesaikan dengan solusi berbasis komputer
- Anda menanggapi permintaan bantuan pelanggan
- Komunikasi telah dimulai. Tapi jalan dari komunikasi menuju pemahaman seringkali berliku



# Sepuluh Prinsip KOMUNIKASI



## Prinsip 1. Dengarkan

- Pastikan Anda memahami sudut pandang pihak lain, tahu sedikit tentang kebutuhannya, dan kemudian mendengarkan
- Cobalah untuk fokus pada kata-kata pembicara, daripada merumuskan respons Anda terhadap kata-kata itu
- Mintalah klarifikasi jika ada sesuatu yang tidak jelas, dan hindari interupsi terus-menerus
- Jangan pernah menjadi kontroversial dalam kata-kata atau tindakan Anda (misalnya, memutar mata atau menggelengkan kepala) saat seseorang berbicara



## Prinsip 2. Persiapkan Diri sebelum Anda berkomunikasi

- Luangkan waktu untuk memahami problem sebelum Anda bertemu dengan orang lain
- Jika perlu, lakukan riset untuk memahami jargon domain bisnis
- Jika Anda memiliki tanggung jawab untuk mengadakan rapat, siapkan agenda sebelum rapat



## Prinsip 3. Seseorang Harus Memfasilitasi Kegiatan

- Setiap pertemuan komunikasi harus memiliki seorang pemimpin (fasilitator) untuk:
  - 1) Menjaga percakapan bergerak ke arah yang produktif,
  - 2) Menengahi setiap konflik yang terjadi, dan
  - 3) Memastikan bahwa prinsip-prinsip lain diikuti.



## Prinsip 4. Komunikasi Face-to-Face adalah Terbaik

- Bekerja lebih baik ketika beberapa representasi lain dari informasi yang relevan hadir.
- Mis., seorang peserta dapat membuat gambar atau dokumen "strawman" yang berfungsi sebagai fokus diskusi



## Prinsip 5. Buat Catatan dan Dokumentasikan Keputusan

- Hal-hal memiliki cara untuk jatuh ke dalam celah
- Seseorang yang berpartisipasi dalam komunikasi harus berfungsi sebagai "perekam" dan menuliskan semua poin dan keputusan penting

# Sepuluh Prinsip KOMUNIKASI



## Prinsip 6. Upayakan Kolaborasi

- Kolaborasi dan konsensus terjadi ketika pengetahuan kolektif anggota tim digunakan untuk menggambarkan fungsi atau fitur produk atau system
- Setiap kolaborasi kecil berfungsi untuk membangun kepercayaan di antara anggota tim dan menciptakan tujuan bersama untuk tim



## Prinsip 7. Tetap Fokus; Diskusi Dimodularisasi

- Semakin banyak orang yang terlibat dalam komunikasi apa pun, semakin besar kemungkinan diskusi akan terpental dari satu topik ke topik berikutnya
- Fasilitator harus menjaga percakapan tetap modular, meninggalkan satu topik hanya setelah diselesaikan (namun, lihat Prinsip 9)



## Prinsip 8. Jika Ada yang Tidak Jelas, Buatlah Gambar

- Komunikasi verbal hanya berjalan sejauh ini.
- Sebuah sketsa atau gambar seringkali dapat memberikan kejelasan ketika kata-kata gagal untuk melakukan pekerjaan itu



## Prinsip 9. (a) Setelah Anda menyetujui sesuatu, lanjutkan. (b) Jika Anda tidak dapat menyetujui sesuatu, lanjutkan. (c) Jika fitur atau fungsi tidak jelas dan tidak dapat diklarifikasi sekarang, lanjutkan

- Komunikasi, seperti aktivitas rekayasa perangkat lunak lainnya, membutuhkan waktu
- Daripada mengulangi tanpa henti, orang-orang yang berpartisipasi harus menyadari bahwa banyak topik memerlukan diskusi (lihat Prinsip 2) dan bahwa "move on" terkadang merupakan cara terbaik untuk mencapai komunikasi yang agile

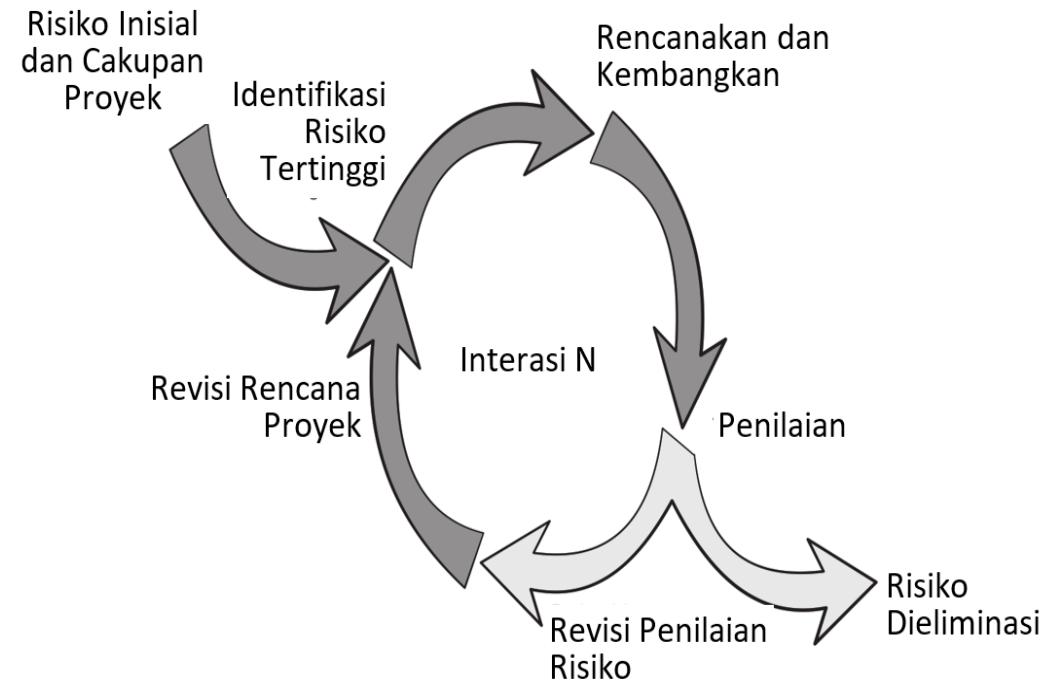


## Prinsip 10. Negosiasi bukanlah kontes atau 'game'. Ini bekerja paling baik ketika kedua belah pihak menang

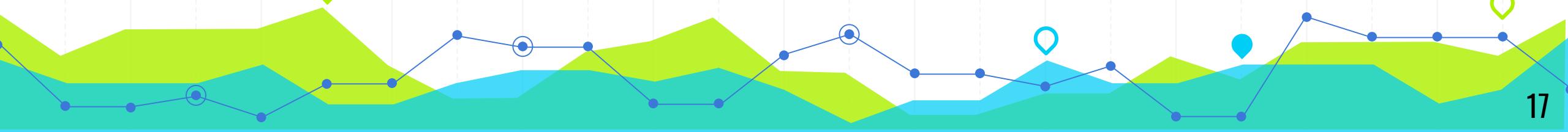
- Ada banyak contoh di mana Anda dan stakeholder lainnya harus menegosiasikan fungsi dan fitur, prioritas, dan tanggal pengiriman
- Jika tim telah bekerja sama dengan baik, semua pihak memiliki tujuan yang sama
- Namun, negosiasi akan menuntut kompromi dari semua pihak

## 6.2.2 Planning Principles (Prinsip PERENCANAAN)

- Aktivitas perencanaan mencakup serangkaian praktik manajemen dan teknis yang memungkinkan tim PL untuk menentukan peta jalan saat bergerak menuju tujuan strategis dan tujuan taktisnya
- Tim PL yang baik harus merencanakan pendekatannya. Seringkali perencanaan dilakukan berulang
- Dua Filosofi Perencanaan:
  - **Minimalis.** Perubahan sering meniadakan kebutuhan akan rencana terperinci
  - **Tradisionalis.** Rencana tersebut memberikan peta jalan yang efektif dan semakin detail, semakin kecil kemungkinan tim akan tersesat
- Seperti kebanyakan hal dalam hidup, perencanaan harus agile dan dilakukan secukupnya, cukup untuk memberikan panduan yang berguna bagi tim—tidak lebih, tidak kurang



Perencanaan Iteratif



# Sepuluh Prinsip PERENCANAAN



## Prinsip 1. Pahami Cakupan Proyek

- Tidak mungkin menggunakan peta jalan jika Anda tidak tahu ke mana Anda akan pergi
- Lingkup memberi tim perangkat lunak tujuan



## Prinsip 2. Libatkan Stakeholder dalam Kegiatan Perencanaan

- Stakeholder menentukan prioritas dan menetapkan kendala proyek
- Untuk mengakomodasi kenyataan ini, Software Engineer harus sering menegosiasikan urutan pengiriman, garis waktu, dan masalah terkait proyek lainnya



## Prinsip 3. Ketahuilah Bahwa Perencanaan Itu Berulang

- Rencana proyek tidak pernah terukir di atas batu. Saat pekerjaan dimulai, kemungkinan besar segalanya akan berubah. Rencananya perlu disesuaikan
- Model proses inkremental yang berulang mencakup waktu untuk merevisi rencana setelah pengiriman setiap peningkatan perangkat lunak berdasarkan umpan balik yang diterima dari pengguna



## Prinsip 4. Estimasi Berdasarkan Apa Yang Anda Ketahui

- Maksud dari estimasi adalah untuk memberikan indikasi upaya, biaya, dan durasi tugas, berdasarkan pemahaman tim saat ini tentang pekerjaan yang harus dilakukan.
- Jika informasi tidak jelas atau tidak dapat diandalkan, perkiraan akan sama tidak dapat diandalkan



## Prinsip 5. Pertimbangkan Risiko Saat Anda Menentukan Rencana

- Jika Anda telah mengidentifikasi risiko yang memiliki dampak tinggi dan probabilitas tinggi, perencanaan kontinjensi diperlukan
- Rencana proyek (termasuk jadwal) harus disesuaikan untuk mengakomodasi kemungkinan terjadinya satu atau beberapa risiko ini



## Prinsip 6. Realistik

- Orang tidak bekerja 100 persen setiap hari. Perubahan akan terjadi
- Bahkan Software Engineer terbaik pun membuat kesalahan
- Ini dan kenyataan lainnya harus dipertimbangkan saat rencana proyek ditetapkan

# Sepuluh Prinsip PERENCANAAN



## Prinsip 7. Sesuaikan Granularitas Saat Menentukan Rencana

- Istilah granularitas mengacu pada detail yang dengannya beberapa elemen perencanaan diwakili atau dilakukan
- Rencana granularitas tinggi memberikan detail tugas kerja yang signifikan yang direncanakan dalam waktu yang relatif singkat (sehingga pelacakan dan kontrol sering terjadi)
- Rencana granularitas rendah memberikan tugas kerja yang lebih luas yang direncanakan dalam periode waktu yang lebih lama
- Secara umum, granularitas bergerak dari tinggi ke rendah saat garis waktu proyek menjauh dari tanggal saat ini
- Aktivitas yang tidak akan terjadi selama berbulan-bulan tidak memerlukan granularitas yang tinggi (terlalu banyak yang dapat berubah)



## Prinsip 8. Tentukan Bagaimana Anda Ingin Memastikan Kualitas

- Rencana tersebut harus mengidentifikasi bagaimana tim PL bermaksud untuk memastikan kualitas
- Jika tinjauan teknis akan dilakukan, maka harus dijadwalkan
- Jika pair programming akan digunakan selama konstruksi, maka harus didefinisikan secara eksplisit dalam rencana



## Prinsip 9. Jelaskan Bagaimana Anda Berniat untuk Mengakomodasi Perubahan

- Perubahan yang tidak terkendali dapat meniadakan bahkan perencanaan terbaik
- Anda harus mengidentifikasi bagaimana perubahan diakomodasi saat pekerjaan RPL berlangsung. Misalnya, dapatkah pelanggan meminta perubahan kapan saja?
- Jika diminta perubahan, apakah tim wajib segera menerapkannya?
- Bagaimana dampak dan biaya perubahan dinilai?

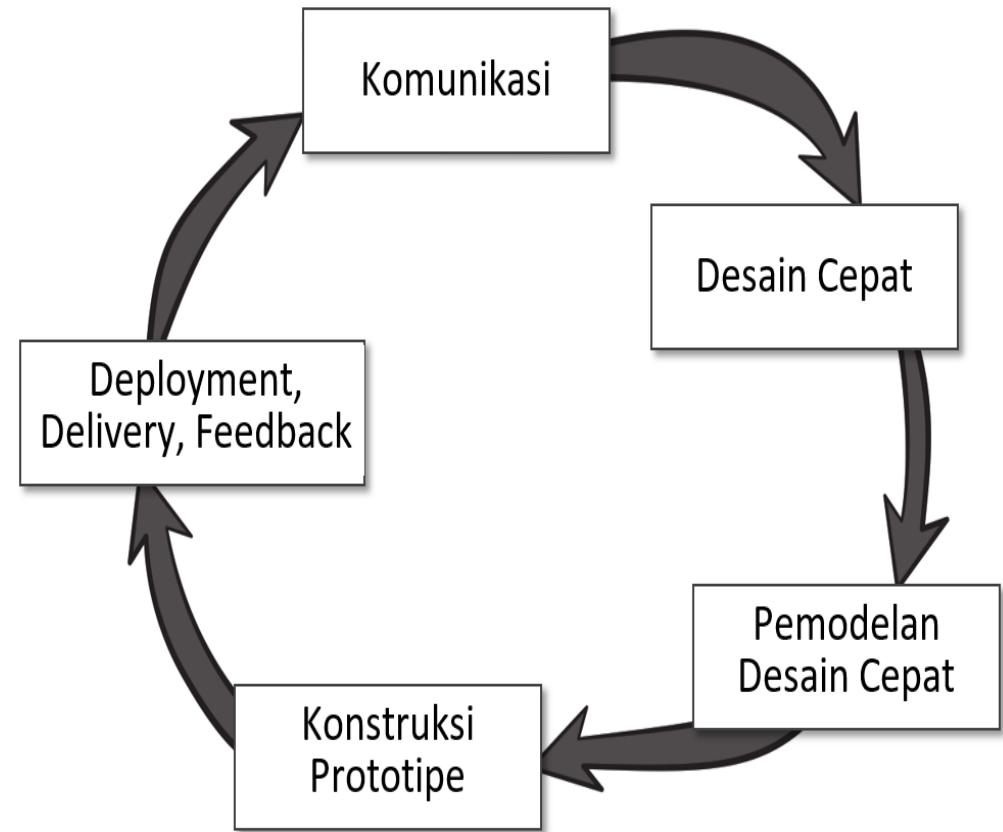


## Prinsip 10. Lacak Rencana Sesering Mungkin, dan Buat Penyesuaian Sesuai Kebutuhan

- Proyek PL jatuh terlambat jadwal satu hari pada suatu waktu
- Oleh karena itu, masuk akal untuk melacak kemajuan setiap hari, mencari area masalah dan situasi di mana pekerjaan yang dijadwalkan tidak sesuai dengan pekerjaan yang sebenarnya dilakukan
- Saat terjadi selip jadwal, maka rencana akan disesuaikan

## 6.2.3 Modeling Principles (Prinsip PEMODELAN)

- Untuk mendapatkan pemahaman yang lebih baik tentang entitas sebenarnya yang akan dibangun
- Harus mampu mewakili informasi yang ditransformasikan PL, arsitektur dan fungsi yang memungkinkan transformasi terjadi, fitur yang diinginkan pengguna, dan perilaku sistem saat transformasi berlangsung
- Harus mencapai tujuan ini pada tingkat abstraksi yang berbeda—pertama menggambarkan PL dari sudut pandang pelanggan dan kemudian mewakili perangkat lunak pada tingkat yang lebih teknis
- Gambar di kanan menunjukkan bagaimana pemodelan dapat digunakan dalam desain PL agile



Peran Pemodelan Software

## 6.2.3 Modeling Principles (Prinsip PEMODELAN)

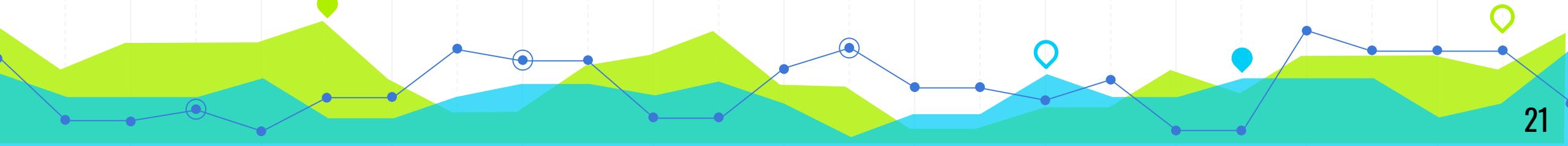
### Requirements Models/ Analysis Models

Merepresentasikan kebutuhan pelanggan dengan menggambarkan perangkat lunak dalam tiga domain yang berbeda: domain informasi, domain fungsional, dan domain perilaku



### Design Models

Merepresentasikan karakteristik perangkat lunak yang membantu praktisi untuk membangunnya secara efektif: arsitektur, antarmuka pengguna, dan detail tingkat komponen



# Sepuluh Prinsip PEMODELAN



## Prinsip 1. Tujuan utama tim perangkat lunak adalah membangun PL, bukan membuat model

- Agilitas berarti memberikan PL kepada pelanggan dalam waktu secepat mungkin
- Model yang mewujudkan hal ini layak untuk dibuat, tetapi model yang memperlambat proses atau memberikan sedikit wawasan baru harus dihindari



## Prinsip 2. “Perjalanan Ringan” - Jangan Membuat Lebih Banyak Model Daripada yang Anda Butuhkan

- Setiap model yang dibuat harus selalu up to date saat terjadi perubahan
- Lebih penting lagi, setiap model baru membutuhkan waktu yang mungkin dihabiskan untuk konstruksi (pengkodean dan pengujian)
- Oleh karena itu, buat hanya model-model yang membuatnya lebih mudah dan lebih cepat untuk membangun perangkat lunak



## Prinsip 3. Berusaha Keras untuk Menghasilkan Model Paling Sederhana yang Akan Menggambarkan Problem atau PL

- Jangan membangun PL secara berlebihan
- Dengan menjaga model tetap sederhana, PL yang dihasilkan juga akan sederhana
- Hasilnya adalah PL yang lebih mudah untuk diintegrasikan, lebih mudah untuk diuji, dan lebih mudah untuk dipelihara (diubah)
- Model sederhana lebih mudah dipahami dan dikritik oleh anggota tim PL, menghasilkan bentuk umpan balik berkelanjutan yang mengoptimalkan hasil akhir



## Prinsip 4. Bangun model dengan Cara yang Membuatnya Dapat Diubah

- Asumsikan model Anda akan berubah, tetapi dalam membuat asumsi ini jangan asal-asalan (Hal ini karena model persyaratan mungkin tidak cukup lengkap, yang berarti desain (model desain) akan selalu kehilangan fungsi dan fitur penting)



## Prinsip 5. Mampu Menyatakan secara Eksplisit Tujuan dari Setiap Model yang Dibuat

- Setiap kali Anda membuat model, tanyakan pada diri sendiri mengapa Anda melakukannya
- Jika Anda tidak dapat memberikan pbenaran yang kuat untuk keberadaan model, jangan habiskan waktu untuk itu

# Sepuluh Prinsip PEMODELAN



## Prinsip 6. Sesuaikan Model yang Dikembangkan dengan Sistem yang ada

- Mungkin perlu untuk mengadaptasi notasi atau aturan model ke aplikasi; misalnya, aplikasi video game mungkin memerlukan teknik pemodelan yang berbeda dari PL tertanam real-time untuk cruise control adaptif di mobil



## Prinsip 7. Cobalah untuk Membuat Model yang Berguna, Lupakan Membangun Model yang Sempurna

- Saat membangun persyaratan dan model desain, seorang Software Engineer mencapai titik pengembalian yang semakin berkurang. Artinya, upaya yang diperlukan untuk membuat model yang lengkap dan konsisten secara internal mungkin tidak sebanding dengan manfaatnya
- Iterasi tanpa henti untuk membuat model "sempurna" tidak memenuhi kebutuhan akan agility



## Prinsip 8. Jangan menjadi dogmatis tentang sintaks model. Jika konten berhasil Dikomunikasikan, representasi adalah hal yang kedua

- Karakteristik paling penting dari model ini adalah untuk mengkomunikasikan informasi yang memungkinkan tugas rekayasa perangkat lunak berikutnya.
- Jika model berhasil melakukan ini, sintaks yang salah dapat dimaafkan



## Prinsip 9. Jika insting Anda memberi tahu Anda bahwa suatu model tidak benar meskipun tampaknya baik-baik saja di atas kertas, Anda mungkin punya alasan untuk khawatir

- Jika Anda seorang software engineer yang berpengalaman, percayalah pada insting Anda
- Pekerjaan PL mengajarkan banyak pelajaran - beberapa di antaranya di tingkat bawah sadar.
- Jika ada sesuatu yang memberi tahu Anda bahwa model desain pasti akan gagal (meskipun Anda tidak dapat membuktikannya secara eksplisit), Anda memiliki alasan untuk menghabiskan waktu tambahan untuk memeriksa model atau mengembangkan model yang berbeda



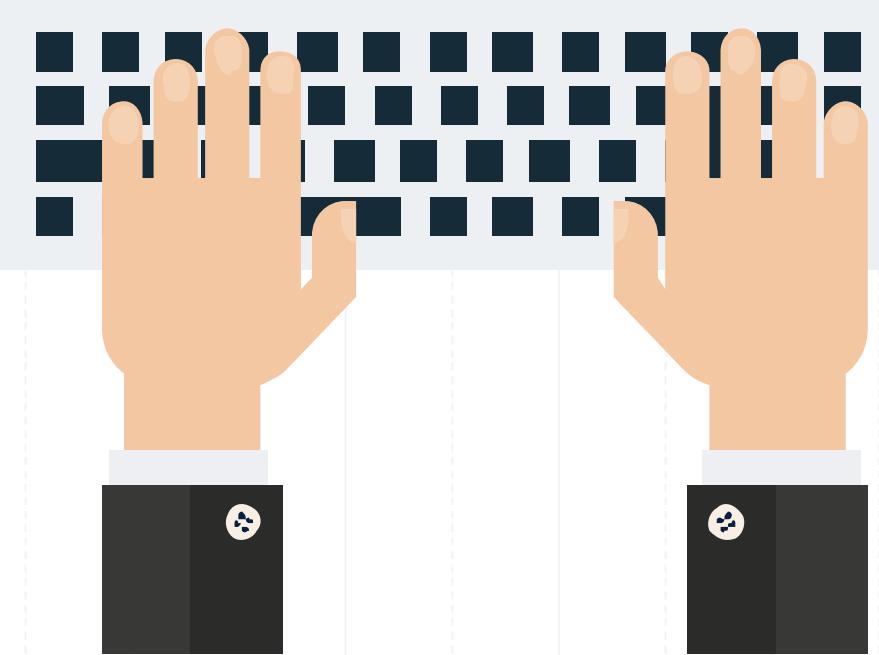
## Prinsip 10. Dapatkan Umpan Balik Sesegera Mungkin

- Maksud dari model apapun adalah untuk mengkomunikasikan informasi. Ia harus berdiri sendiri.
- Asumsikan bahwa Anda tidak akan berada di sana untuk menjelaskan modelnya. Setiap model harus ditinjau oleh anggota tim perangkat lunak
- Maksud dari tinjauan ini adalah untuk memberikan umpan balik yang dapat digunakan untuk memperbaiki kesalahan pemodelan, mengubah salah tafsir, dan menambahkan fitur atau fungsi yang dihilangkan secara tidak sengaja.

## 6.2.4 Construction Principles (Prinsip KONSTRUKSI)

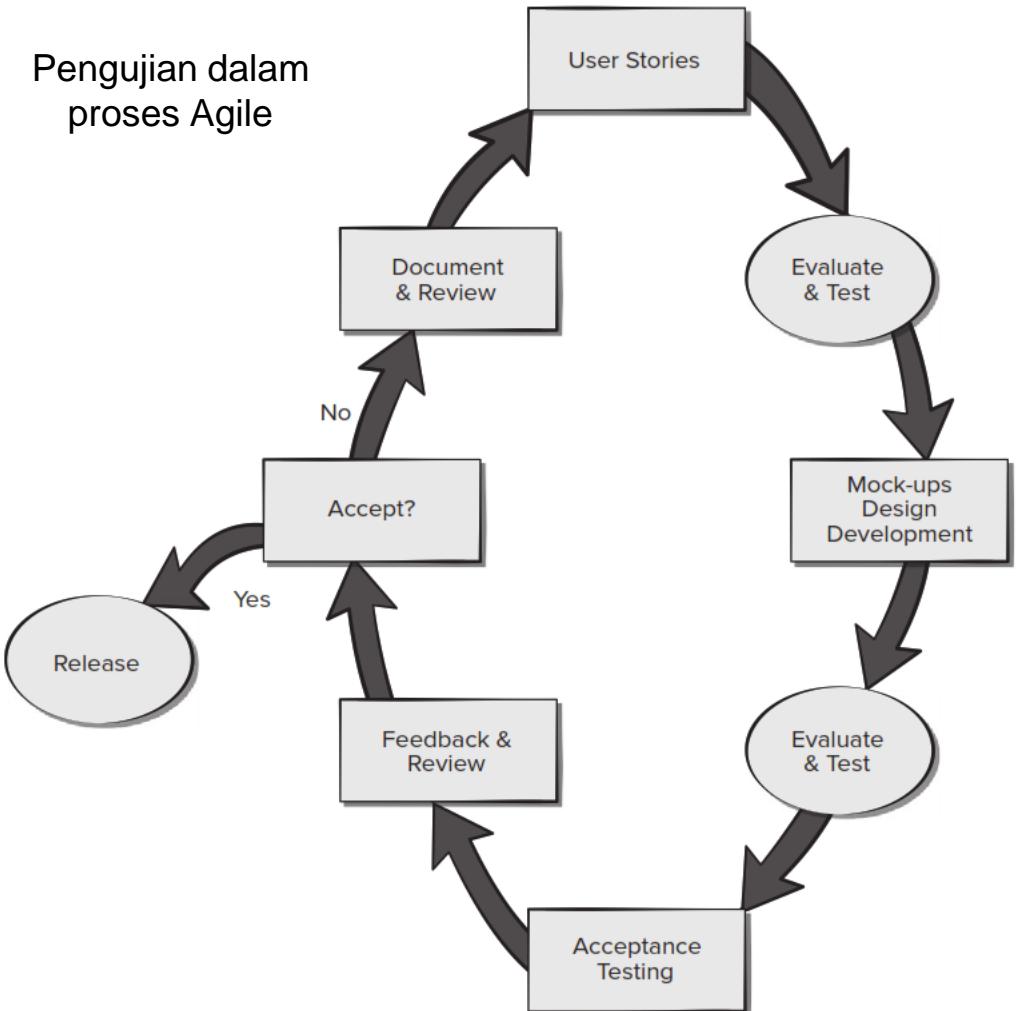
- Aktivitas konstruksi mencakup serangkaian tugas pengkodean dan pengujian yang mengarah ke PL operasional yang siap dikirim ke pelanggan atau pengguna akhir
- Prinsip Konstruksi juga mencakup Prinsip Testing
- Dalam pekerjaan RPL modern, pengkodean dapat berupa:
  - 1) Pembuatan langsung kode sumber (source code) bahasa pemrograman,
  - 2) Pembuatan source code secara otomatis menggunakan desain perantara seperti representasi komponen yang akan dibangun, atau
  - 3) Otomatisasi pembuatan coding yang dapat dieksekusi menggunakan bahasa pemrograman generasi keempat (mis., Unreal4 Blueprints)

Blueprints is a visual scripting tool created by Epic Games  
(<https://docs.unrealengine.com/latest/INT/Engine/Blueprints/>)



## 6.2.4 Construction Principles (Prinsip KONSTRUKSI)

Pengujian dalam proses Agile



- Jenis-jenis pengujian (testing)
  - 1) **Unit Testing** (Pengujian Unit). Fokus awal pengujian adalah pada tingkat komponen,
  - 2) **Integration Testing** (Pengujian Integrasi). Dilakukan saat sistem dibangun,
  - 3) **Validation Testing** (Pengujian Validasi). Menilai apakah persyaratan telah dipenuhi untuk sistem yang lengkap (atau peningkatan PL)
  - 4) **Acceptance Testing** (Pengujian Penerimaan). Dilakukan oleh pelanggan dalam upaya untuk menjalankan semua fitur dan fungsi yang diperlukan
- Gambar di kiri menunjukkan pengujian dan desain kasus uji ditempatkan dalam proses yang agile

## 6.2.4.1 Prinsip-prinsip KODING

- Prinsip-prinsip yang memandu tugas pengkodean sangat selaras dengan gaya pemrograman, bahasa pemrograman, dan metode pemrograman
- Ada beberapa prinsip dasar yang dapat diikuti:
  - A. Prinsip Preparasi (Preparation Principles)
  - B. Prinsip Koding
  - C. Prinsip Validasi (Validation Principles)

### A. Prinsip Preparasi (Preparation Principles)

Sebelum Anda menulis satu baris kode, pastikan Anda

- Prinsip 1.** Pahami masalah yang Anda coba selesaikan
- Prinsip 2.** Pahami prinsip dan konsep desain dasar
- Prinsip 3.** Pilih bahasa pemrograman yang memenuhi kebutuhan perangkat lunak yang akan dibangun dan lingkungan di mana ia akan beroperasi
- Prinsip 4.** Pilih lingkungan pemrograman yang menyediakan alat yang akan membuat pekerjaan Anda lebih mudah
- Prinsip 5.** Buat satu set unit test yang akan diterapkan setelah komponen yang Anda kodekan selesai

## 6.2.4.1 Prinsip-prinsip KODING

### B. Prinsip Koding

Saat Anda mulai menulis kode,  
pastikan Anda:

- Prinsip 6.** Batasi algoritme Anda dengan mengikuti praktik pemrograman terstruktur
- Prinsip 7.** Pertimbangkan penggunaan pair programming
- Prinsip 8.** Pilih struktur data yang akan memenuhi kebutuhan desain
- Prinsip 9.** Pahami arsitektur PL dan buat antarmuka yang konsisten dengannya

### C. Prinsip Validasi

Setelah menyelesaikan coding pass pertama, pastikan Anda:

- Prinsip 10.** Lakukan penelusuran koding bila perlu
- Prinsip 11.** Lakukan pengujian unit dan perbaiki kesalahan yang Anda temukan
- Prinsip 12.** Refactor coding untuk meningkatkan kualitasnya

## 6.2.4.1 Prinsip-prinsip **TESTING**

- Tujuan Testing menurut Glen Myers
  - 1) Testing adalah proses mengeksekusi program dengan maksud untuk menemukan error
  - 2) Kasus uji yang baik adalah kasus yang memiliki probabilitas tinggi untuk menemukan error yang belum ditemukan
  - 3) Tes yang berhasil adalah tes yang mengungkap error yang belum ditemukan
- Tujuan Anda adalah merancang tes yang secara sistematis mengungkap kelas error yang berbeda dan melakukannya dengan waktu dan usaha yang minimal
- Pengujian menunjukkan bahwa fungsi PL tampaknya bekerja sesuai dengan spesifikasi dan persyaratan perilaku dan kinerja tampaknya telah terpenuhi
- Data yang dikumpulkan saat pengujian dilakukan memberikan indikasi keandalan PL yang baik dan beberapa indikasi kualitas PL
- Testing tidak dapat menunjukkan tidak adanya kesalahan dan cacat; Testing hanya dapat menunjukkan bahwa ada kesalahan dan cacat perangkat lunak (Gambar 6.6)



## 6.2.4.1 Prinsip-prinsip TESTING



### Prinsip 1. Semua tes harus dapat dilacak ke kebutuhan pelanggan

- Tujuan testing PL adalah untuk mengungkap error
- Oleh karena itu, cacat yang paling parah (dari sudut pandang pelanggan) adalah yang menyebabkan program gagal memenuhi persyaratannya



### Prinsip 2. Testing harus direncanakan jauh sebelum testing dimulai

- Perencanaan testing dapat dimulai segera setelah model persyaratan (requirements model) selesai
- Definisi rinci dari kasus uji dapat dimulai segera setelah model desain telah solid
- Oleh karena itu, semua testing dapat direncanakan dan dirancang sebelum koding apa pun dibuat



### Prinsip 3. Prinsip Pareto berlaku untuk testing perangkat lunak

- Dalam konteks ini, prinsip Pareto menyiratkan bahwa 80 persen dari semua kesalahan yang ditemukan selama pengujian kemungkinan akan dapat dilacak hingga 20 persen dari semua komponen program
- Masalahnya, tentu saja, adalah mengisolasi komponen yang dicurigai ini dan mengujinya secara menyeluruh



### Prinsip 4. Testing harus dimulai "dari yang kecil" dan berlanjut ke testing "dalam skala besar"

- Tes pertama yang direncanakan dan dilaksanakan umumnya fokus pada komponen individu
- Saat testing berlangsung, fokus beralih ke mencari kesalahan dalam kelompok komponen yang terintegrasi dan akhirnya di seluruh sistem



### Prinsip 5. Testing menyeluruh tidak mungkin dilakukan

Setiap kali Anda membuat model, tanyakan pada diri sendiri mengapa Anda melakukannya

- Jumlah permutasi jalur bahkan untuk program berukuran sedang sangat besar
- Untuk alasan ini, tidak mungkin untuk mengeksekusi setiap kombinasi jalur selama testing
- Walaupun demikian, hal ini dimungkinkan dilakukan semua, untuk cukup menutupi logika program dan untuk memastikan bahwa semua kondisi dalam desain tingkat komponen telah dilaksanakan

## 6.2.4.1 Prinsip-prinsip TESTING



**Prinsip 6.** Terapkan ke setiap modul dalam sistem upaya testing yang sepadan dengan ekspektasi densitas kesalahan

- Ini sering merupakan modul terbaru atau yang paling tidak dipahami oleh pengembang



**Prinsip 7.** Teknik testing statis dapat memberikan hasil yang tinggi

- Lebih dari 85 persen cacat PL berasal dari dokumentasi PL (persyaratan, spesifikasi, penelusuran kode, dan manual pengguna)
- Mungkin ada manfaatnya untuk menguji dokumentasi sistem



**Prinsip 8.** Lacak cacat dan cari pola cacat yang ditemukan dengan testing

- Cacat total yang ditemukan merupakan indikator kualitas PL yang baik
- Jenis cacat yang ditemukan dapat menjadi ukuran stabilitas PL yang baik
- Pola cacat yang ditemukan dari waktu ke waktu dapat memperkirakan jumlah cacat yang diharapkan



**Prinsip 9.** Sertakan kasus uji yang menunjukkan PL berperilaku dengan benar

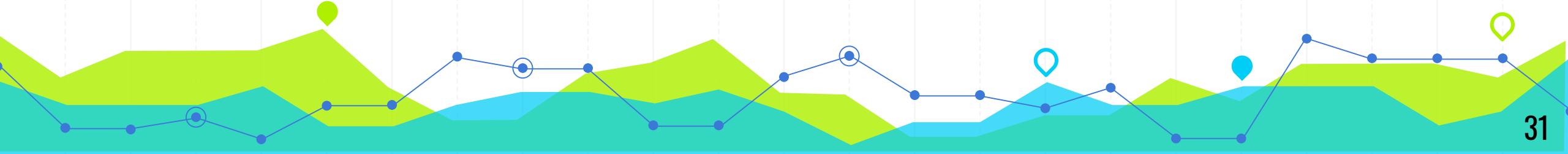
- Saat komponen PL dipertahankan atau diadaptasi, interaksi tak terduga menyebabkan efek samping yang tidak diinginkan pada komponen lain
- Penting untuk memiliki satu set kasus uji regresi yang siap untuk memeriksa perilaku sistem setelah perubahan dilakukan pada produk PL

## 6.2.5 Deployment Principles (Prinsip Deployment)

- Aktivitas deployment terdiri atas: delivery, support, dan feedback
- Model proses PL modern bersifat evolusioner atau inkremental, sehingga deployment terjadi tidak hanya sekali, tetapi beberapa kali saat PL bergerak mengarah ke penyelesaian
- Setiap siklus pengiriman (delivery) memberi pelanggan dan pengguna akhir peningkatan PL operasional yang menyediakan fungsi dan fitur yang dapat digunakan
- Setiap siklus dukungan (support) menyediakan dokumentasi dan bantuan manusia untuk semua fungsi dan fitur yang diperkenalkan selama semua siklus deployment
- Setiap siklus umpan balik (feedback) menyediakan tim PL dengan panduan penting yang menghasilkan modifikasi pada fungsi, fitur, dan pendekatan yang diambil untuk increment berikutnya
- Pengiriman increment PL merupakan tonggak penting untuk setiap proyek perangkat lunak
- Aksi deployment tipikal diilustrasikan pada Gambar 6.7



Gambar 6.7 Deployment Actions



## 6.2.5 Deployment Principles (Prinsip Deployment)



### Prinsip 1. Ekspektasi pelanggan untuk perangkat lunak harus dikelola

- Terlalu sering, pelanggan mengharapkan lebih dari yang dijanjikan tim untuk delivery, dan kekecewaan segera terjadi
- Ini menghasilkan umpan balik yang tidak produktif dan merusak moral tim
- Dalam bukunya tentang mengelola ekspektasi, Naomi Karten menyatakan:
  - "Titik awal untuk mengelola ekspektasi adalah menjadi lebih teliti tentang apa yang Anda komunikasikan dan caranya."
  - Seorang software engineer harus berhati-hati dalam mengirim pesan yang bertentangan kepada pelanggan (mis., menjanjikan lebih dari yang dapat Anda berikan secara wajar dalam periode waktu yang disediakan atau memberikan lebih dari yang Anda janjikan untuk satu increment perangkat lunak dan kemudian kurang dari yang dijanjikan untuk yang berikutnya)



### Prinsip 2. Paket pengiriman (delivery) lengkap harus dirakit dan diuji

- Semua PL yang dapat dijalankan, file data dukungan, dokumen dukungan, dan informasi relevan lainnya harus dirakit dan diuji beta secara menyeluruh dengan pengguna sebenarnya
- Semua skrip penginstalan dan fitur operasional lainnya harus dijalankan secara menyeluruh di semua konfigurasi komputasi yang mungkin (yaitu, perangkat keras, sistem operasi, perangkat periferal, pengaturan jaringan)



### Prinsip 3. Regimen dukungan harus ditetapkan sebelum PL dikirimkan

- Pengguna akhir mengharapkan respon dan informasi yang akurat ketika pertanyaan atau masalah muncul
- Jika dukungan bersifat ad hoc, atau lebih buruk lagi, tidak ada, pelanggan akan langsung merasa tidak puas
- Dukungan harus direncanakan, bahan pendukung harus disiapkan, dan mekanisme pencatatan yang tepat harus ditetapkan sehingga tim PL dapat melakukan penilaian kategoris dari jenis dukungan yang diminta

## 6.2.5 Deployment Principles (Prinsip Deployment)



### Prinsip 4. Instruksional yang memadai harus disediakan untuk pengguna akhir

- Tim PL memberikan lebih dari PL itu sendiri
- Alat bantu pelatihan (training aids) yang sesuai (jika diperlukan) harus dikembangkan; pedoman pemecahan masalah (troubleshooting) harus disediakan
- Bila perlu, deskripsi "apa yang berbeda tentang increment perangkat lunak ini" harus dipublikasikan/disampaikan



### Prinsip 5. Software buggy harus diperbaiki dulu, dikirim nanti

- Di bawah tekanan waktu, beberapa organisasi perangkat lunak memberikan peningkatan kualitas rendah dengan peringatan kepada pelanggan bahwa bug "akan diperbaiki pada rilis berikutnya." Ini adalah kesalahan
- Ada pepatah dalam bisnis PL: "*Pelanggan akan lupa bahwa Anda mengirimkan produk berkualitas tinggi beberapa hari terlambat, tetapi mereka tidak akan pernah melupakan masalah yang disebabkan oleh produk berkualitas rendah. Perangkat lunak berkualitas rendah mengingatkan mereka setiap hari.*"





## 6.3 Rangkuman

# Rangkuman

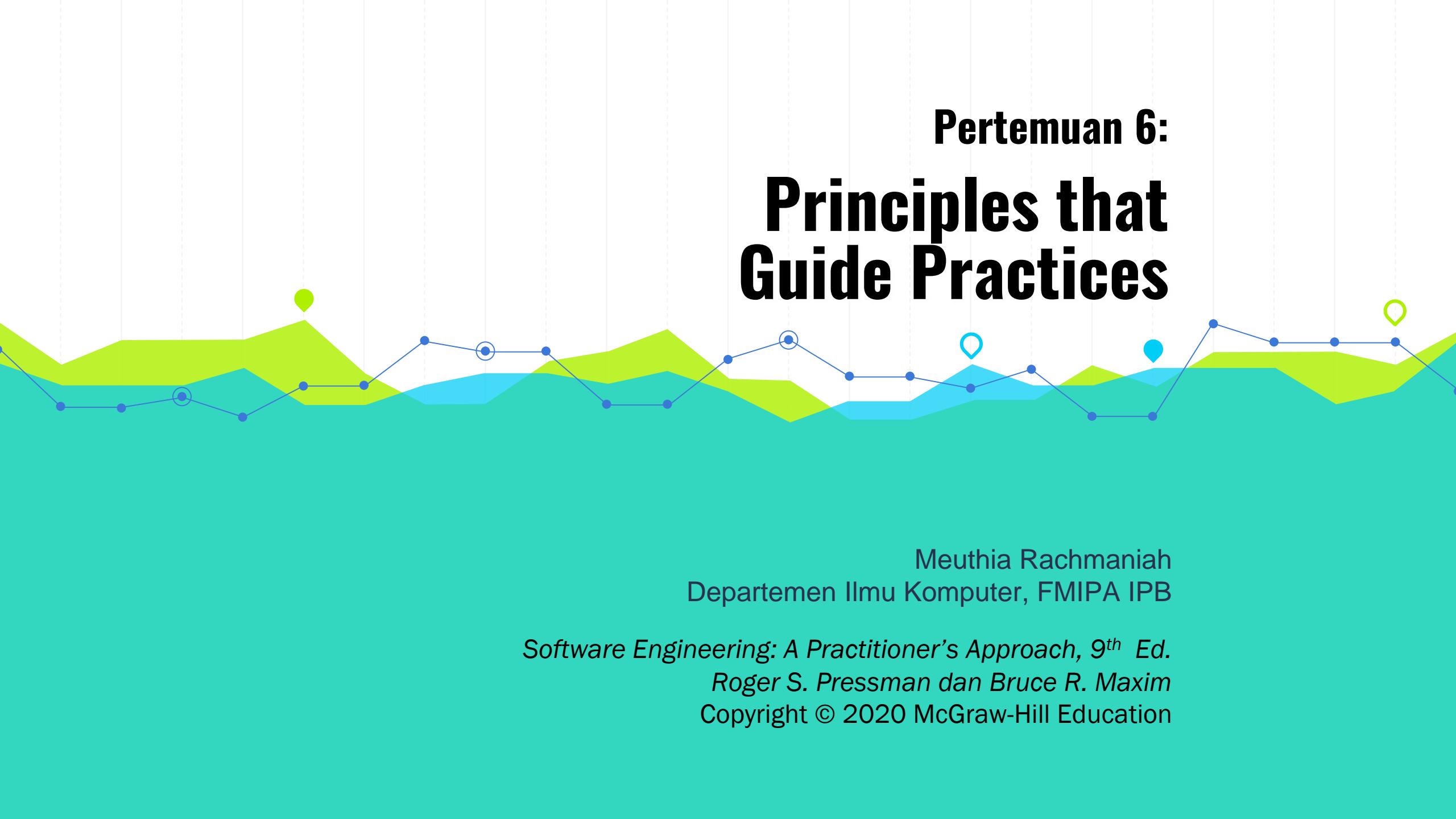
- Praktik RPL mencakup prinsip, konsep, metode, dan alat yang diterapkan oleh insinyur perangkat lunak selama proses perangkat lunak.
- Setiap proyek RPL berbeda.
- Namun, seperangkat prinsip umum berlaku untuk proses dan praktik setiap framework aktivitas terlepas dari proyek atau produknya
- Serangkaian prinsip inti membantu dalam penerapan proses PL yang bermakna dan pelaksanaan metode RPL yang efektif.
- Pada tingkat proses, prinsip-prinsip inti membangun landasan filosofis yang memandu tim PL saat menavigasi melalui proses PL.
- Pada tingkat praktik, prinsip inti menetapkan kumpulan nilai dan aturan yang berfungsi sebagai panduan saat Anda menganalisis problem, merancang solusi, menerapkan dan menguji solusi, dan akhirnya menerapkan perangkat lunak di komunitas pengguna.
- Prinsip-prinsip komunikasi berfokus pada kebutuhan untuk mengurangi kebisingan dan meningkatkan bandwidth saat percakapan antara pengembang dan pelanggan berlangsung.
- Kedua belah pihak harus berkolaborasi agar komunikasi yang terbaik terjadi

## Rangkuman...

- Prinsip-prinsip perencanaan memberikan pedoman untuk membangun peta terbaik untuk perjalanan ke sistem atau produk yang telah selesai.
- Rencana tersebut dapat dirancang hanya untuk peningkatan PL tunggal, atau dapat ditentukan untuk keseluruhan proyek.
- Apapun, itu harus membahas apa yang akan dilakukan, siapa yang akan melakukannya, dan kapan pekerjaan itu akan selesai
- Prinsip pemodelan berfungsi sebagai dasar untuk metode dan notasi yang digunakan untuk membuat PL.
- Pemodelan meliputi analisis dan desain, menggambarkan representasi PL yang semakin menjadi lebih rinci.
- Maksud dari model ini adalah untuk memperkuat pemahaman tentang pekerjaan yang harus dilakukan dan untuk memberikan bimbingan teknis kepada mereka yang akan mengimplementasikan PL
- Konstruksi menggabungkan siklus pengkodean dan pengujian di mana kode sumber untuk komponen dihasilkan dan diuji.
- Prinsip pengkodean mendefinisikan tindakan umum yang harus terjadi sebelum kode ditulis, saat dibuat, dan setelah selesai.
- Meskipun ada banyak prinsip pengujian, hanya satu yang dominan: Pengujian adalah proses mengeksekusi program dengan maksud untuk menemukan error

## Rangkuman...

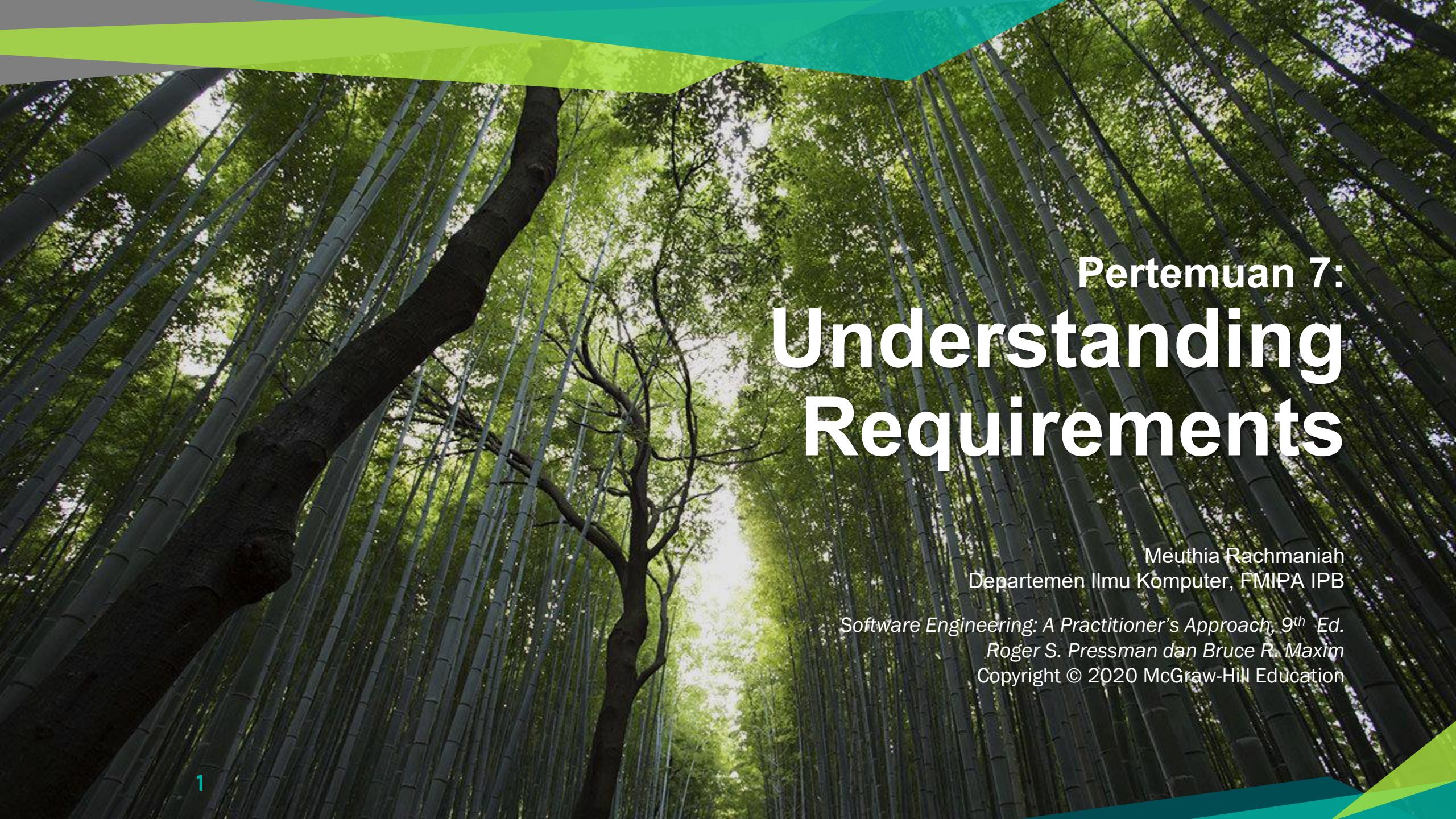
- **Deployment** terjadi karena setiap increment PL disajikan kepada pelanggan dan mencakup pengiriman (delivery), dukungan (support), dan umpan balik (feedback).
- Prinsip-prinsip utama untuk pengiriman mempertimbangkan pengelolaan harapan pelanggan dan menyediakan pelanggan dengan informasi dukungan yang sesuai untuk perangkat lunak.
- Dukungan menuntut persiapan terlebih dahulu.
- Umpan balik memungkinkan pelanggan untuk menyarankan perubahan yang memiliki nilai bisnis dan memberikan masukan kepada pengembang untuk siklus RPL berulang berikutnya



# Pertemuan 6: Principles that Guide Practices

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
Roger S. Pressman dan Bruce R. Maxim  
Copyright © 2020 McGraw-Hill Education



# Pertemuan 7: Understanding Requirements

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
Roger S. Pressman dan Bruce R. Maxim  
Copyright © 2020 McGraw-Hill Education



# Hello!

Ir. Meuthia Rachmaniah, M.Sc.  
Associate Professor  
NIP 195907111984032006

Computer Science Department  
Software Engineering & Information Sciences  
[meuthiara@apps.ipb.ac.id](mailto:meuthiara@apps.ipb.ac.id)

SATYALANCANA KARYA SATYA 20 TAHUN  
SATYALANCANA KARYA SATYA 30 TAHUN

# 7. Understanding Requirements

- 7.1 Requirements Engineering
- 7.2 Establishing the Groundwork
- 7.3 Requirements Gathering
- 7.4 Developing Use Cases
- 7.5 Building the Analysis Model
- 7.6 Negotiating Requirements
- 7.7 Requirements Monitoring
- 7.8 Validating Requirements
- 7.9 Rangkuman

# QUICK LOOK

## WHAT IS IT?



- Sebelum Anda memulai pekerjaan teknis apa pun, ada baiknya Anda membuat serangkaian kebutuhan untuk tugas-tugas teknik.
- Dengan menetapkan serangkaian kebutuhan, Anda akan memperoleh pemahaman tentang apa dampak bisnis dari PL, apa yang diinginkan pelanggan, dan bagaimana pengguna akhir akan berinteraksi dengan PL.

## WHO DOES IT



- Software engineer dan stakeholder proyek lainnya (manajer, pelanggan, dan pengguna akhir) semuanya berpartisipasi dalam rekayasa kebutuhan.

## WHY IS IT IMPORTANT?



- Untuk memahami apa yang diinginkan pelanggan sebelum Anda mulai merancang dan membangun sistem berbasis komputer.
  - Membangun program komputer elegan yang memecahkan masalah yang salah tidak membantu siapa pun.



## WHAT ARE THE STEPS

- Inception - tugas yang mendefinisikan ruang lingkup dan sifat problem yang akan dipecahkan.
- Elitisasi - tugas yang membantu stakeholder menentukan apa yang diperlukan
- Elaborasi - kebutuhan dasar disempurnakan dan dimodifikasi.
- Negosiasi - Saat pemangku kepentingan mendefinisikan masalah, negosiasi terjadi (apa prioritasnya, apa yang esensial, kapan dibutuhkan?).
- Spesifikasi - problemnya ditentukan dalam beberapa cara
- Validasi – problem ditinjau atau divalidasi untuk memastikan bahwa pemahaman Anda tentang problem dan pemahaman stakeholder tentang problem tersebut bertepatan.



## WHAT IS THE WORK PRODUCT

- Rekayasa kebutuhan menyediakan semua pihak pemahaman tertulis tentang problem.
- Produk kerja dapat mencakup: skenario penggunaan, daftar fungsi dan fitur, dan model kebutuhan.



## HOW DO I ENSURE THAT I'VE DONE IT RIGHT

- Work product rekayasa kebutuhan adalah: ditinjau dengan stakeholder untuk memastikan bahwa semua orang berada di halaman yang sama.
- Sebuah kata peringatan: Bahkan setelah semua pihak setuju, segalanya akan berubah, dan mereka akan terus berubah sepanjang proyek

- ◆ Customer: "*Saya tahu Anda pikir Anda mengerti apa yang saya katakan, tetapi apa yang Anda tidak mengerti adalah apa yang saya katakan bukanlah apa yang saya maksudkan.*"
- ◆ Selalu, ini terjadi di akhir proyek, setelah komitmen tenggat waktu dibuat, reputasi dipertaruhkan, dan uang serius dipertaruhkan.
- ◆ Kita semua yang telah bekerja di bisnis sistem dan perangkat lunak selama lebih dari beberapa tahun telah mengalami mimpi buruk ini, namun, hanya sedikit dari kita yang belajar untuk menghilangkannya.
- ◆ Kita berjuang ketika kita mencoba untuk mendapatkan kebutuhan dari pelanggan kita. Kita kesulitan memahami informasi yang kita peroleh.
- ◆ Kita sering mencatat kebutuhan dengan cara yang tidak teratur, dan kita menghabiskan terlalu sedikit waktu untuk memverifikasi apa yang kita rekam.
- ◆ Kita mengizinkan perubahan untuk mengendalikan kita, daripada membangun mekanisme untuk mengontrol perubahan.
- ◆ Singkatnya, kita gagal membangun fondasi yang kokoh untuk sistem atau PL. Masing-masing problem ini menantang. Ketika digabungkan, prospeknya menakutkan bahkan bagi manajer dan praktisi yang paling berpengalaman sekalipun. Tapi solusi memang ada.

# 7.1 Requirements Engineering

- 7.1.1 Inception
- 7.1.2 Elicitation
- 7.1.3 Elaboration
- 7.1.4 Negotiation
- 7.1.5 Specification
- 7.1.6 Validation
- 7.1.7 Requirements Management

# 7.1 Requirements Engineering (Rekayasa Kebutuhan)

- 7.1.1 Inception
- 7.1.2 Elicitation
- 7.1.3 Elaboration
- 7.1.4 Negotiation
- 7.1.5 Specification
- 7.1.6 Validation
- 7.1.7 Requirements Management

Penting untuk diperhatikan bahwa beberapa tasks (7.2.1 s.d. 7.17) terjadi secara paralel dan semuanya disesuaikan dengan kebutuhan proyek

- ◆ Argumen Pengembang PL berpendapat:
  - ◆ bahwa segala sesuatunya akan menjadi jelas saat mereka membangun,
  - ◆ bahwa stakeholder proyek akan dapat memahami kebutuhan hanya setelah memeriksa iterasi awal PL,
  - ◆ bahwa segala sesuatunya berubah begitu cepat sehingga setiap upaya untuk memahami kebutuhan secara rinci adalah buang-buang waktu, itu intinya adalah menghasilkan program kerja, dan yang lainnya adalah sekunder.
- ◆ Apa yang membuat argumen ini menggiurkan adalah karena mengandung unsur kebenaran. Tetapi setiap argumen cacat dan dapat menyebabkan proyek perangkat lunak gagal.

# 7.1 Rekayasa Kebutuhan

- 7.1.1 Inception
- 7.1.2 Elicitation
- 7.1.3 Elaboration
- 7.1.4 Negotiation
- 7.1.5 Specification
- 7.1.6 Validation
- 7.1.7 Requirements Management

Penting untuk diperhatikan bahwa beberapa tasks (7.2.1 s.d. 7.17) terjadi secara paralel dan semuanya disesuaikan dengan kebutuhan proyek

- ◆ Requirements Engineering (Rekayasa Kebutuhan) adalah istilah untuk spektrum tasks dan teknik yang luas yang mengarah pada pemahaman tentang kebutuhan.
- ◆ Perspektif Proses PL tentang Rekayasa Kebutuhan:
  - ◆ adalah tindakan RPL utama yang dimulai selama aktivitas komunikasi dan berlanjut ke aktivitas pemodelan.
  - ◆ menetapkan dasar yang kokoh untuk desain dan konstruksi. Tanpa itu, PL yang dihasilkan memiliki kemungkinan besar untuk tidak memenuhi kebutuhan pelanggan.
  - ◆ harus disesuaikan dengan kebutuhan proses, proyek, produk, dan orang yang melakukan pekerjaan.
- ◆ Penting untuk disadari bahwa setiap tasks dilakukan secara berulang karena tim proyek dan stakeholder terus berbagi informasi tentang problem mereka masing-masing.

## 7.1 .1 Insepsi (Inception)

- ◆ Secara umum, sebagian besar proyek dimulai dengan kebutuhan bisnis yang teridentifikasi atau ketika pasar atau layanan baru yang potensial ditemukan
- ◆ Membangun pemahaman dasar tentang masalah, orang-orang yang menginginkan solusi, dan sifat solusi yang diinginkan.
- ◆ Komunikasi antara semua stakeholder dan tim PL perlu dibangun selama task ini untuk memulai kolaborasi yang efektif



## 7.1.2 Elisitasi (Elicitation)

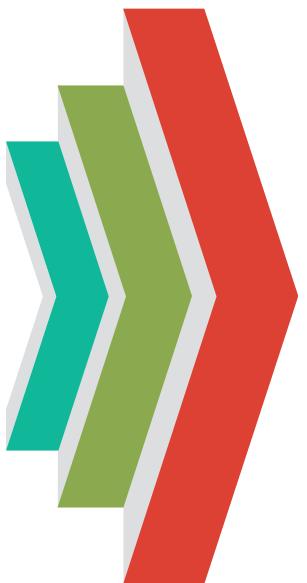
- ◆ Tanyakan kepada pelanggan, pengguna, dan orang lain apa tujuan sistem atau produk, apa yang harus dicapai, bagaimana sistem atau produk cocok dengan kebutuhan bisnis, dan akhirnya, bagaimana sistem atau produk akan digunakan sehari-hari
- ◆ Untuk memahami tujuan bisnis.
  - ◆ Tujuan adalah tujuan jangka panjang yang harus dicapai oleh sistem atau produk.
  - ◆ Tujuan dapat berhubungan dengan masalah fungsional atau nonfungsional (misalnya, keandalan, keamanan, kegunaan)
- ◆ Tugas Anda adalah melibatkan stakeholder dan mendorong mereka untuk membagikan tujuan mereka dengan jujur.
- ◆ Setelah tujuan ditangkap, Anda membuat mekanisme prioritas dan membuat alasan desain untuk arsitektur potensial (yang memenuhi tujuan stakeholder)
- ◆ Agility merupakan aspek penting dari requirement engineering.:
  - ◆ Maksud dari elisitasi adalah untuk mentransfer ide dari stakeholder ke tim PL dengan lancar dan tanpa penundaan.
  - ◆ Sangat mungkin bahwa kebutuhan baru akan terus muncul saat pengembangan produk berulang terjadi

## 7.1.3 Elaborasi (Elaboration)

- ◆ Berfokus pada pengembangan model kebutuhan yang disempurnakan yang mengidentifikasi berbagai aspek fungsi PL, perilaku, dan informasi
- ◆ Didorong oleh pembuatan dan penyempurnaan skenario pengguna (user scenario) yang diperoleh selama elisitasi
- ◆ Skenario ini menggambarkan bagaimana pengguna akhir (dan aktor lain) akan berinteraksi dengan system
- ◆ Setiap user scenario diuraikan untuk mengekstrak kelas analisis—entitas domain bisnis yang terlihat oleh pengguna akhir
- ◆ Atribut dari setiap kelas analisis didefinisikan, dan layanan yang dibutuhkan oleh setiap kelas diidentifikasi.
- ◆ Hubungan dan kolaborasi antar kelas diidentifikasi.
- ◆ Elaborasi adalah hal yang baik, tetapi Anda perlu tahu kapan harus berhenti.
- ◆ Kuncinya adalah untuk menggambarkan problem dengan cara yang menetapkan dasar yang kuat untuk desain dan kemudian melanjutkan.
- ◆ Jangan terobsesi dengan detail yang tidak perlu

## 7.1.4 Negosiasi (Negotiation)

- ◆ Bukan hal yang aneh bagi pelanggan dan pengguna untuk meminta lebih dari yang dapat dicapai, mengingat sumber daya bisnis yang terbatas.
- ◆ Juga relatif umum bagi pelanggan atau pengguna yang berbeda untuk mengusulkan kebutuhan yang bertentangan, dengan alasan bahwa versi mereka "penting untuk kebutuhan khusus kami."



- ◆ Konflik-konflik ini perlu didamaikan melalui proses negosiasi.
- ◆ Pelanggan, pengguna, dan stakeholder lainnya diminta untuk membuat peringkat kebutuhan dan kemudian mendiskusikan konflik dalam prioritas.
- ◆ Seharusnya tidak ada pemenang dan tidak ada pecundang dalam negosiasi yang efektif.
- ◆ Kedua belah pihak menang, karena "kesepakatan" yang dapat dijalani keduanya telah dipadatkan.
- ◆ Anda harus menggunakan pendekatan berulang yang memprioritaskan kebutuhan, menilai biaya dan risikonya, dan mengatasi konflik internal.
- ◆ Dengan cara ini, kebutuhan dihilangkan, digabungkan, dan/atau dimodifikasi sehingga masing-masing pihak mencapai beberapa ukuran kepuasan

## 7.1.5 Spesifikasi (Specification)

Dalam konteks sistem berbasis komputer (dan perangkat lunak), istilah spesifikasi memiliki arti yang berbeda bagi orang yang berbeda.



Spesifikasi dapat berupa dokumen tertulis, sekumpulan model grafis, model matematika formal, kumpulan skenario penggunaan, prototipe, atau kombinasi dari semuanya.



Beberapa menyarankan bahwa "template standar" harus dikembangkan dan digunakan untuk spesifikasi, dengan alasan bahwa ini mengarah pada persyaratan yang disajikan secara konsisten dan karena itu lebih mudah dipahami.



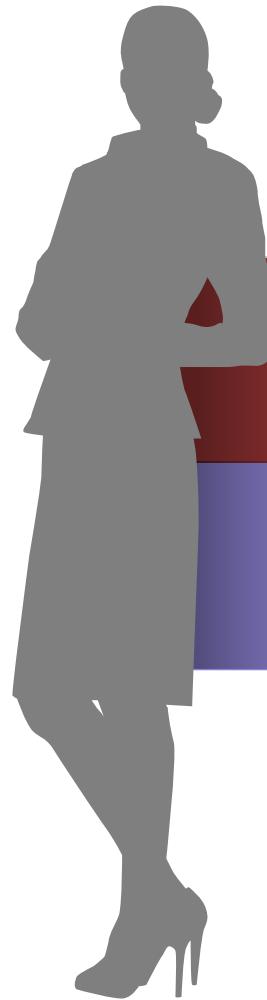
Skenario penggunaan (usage scenario) mungkin adalah yang diperlukan untuk produk atau sistem yang lebih kecil yang berada dalam lingkungan teknis yang dipahami dengan baik

- Namun, terkadang perlu untuk tetap fleksibel ketika spesifikasi akan dikembangkan.
- Formalitas dan format spesifikasi bervariasi menurut ukuran dan kompleksitas PL yang akan dibangun.
  - Untuk sistem besar, dokumen tertulis, yang menggabungkan deskripsi bahasa alami dan model grafis, mungkin merupakan pendekatan terbaik.

Templat dokumen formal software requirements specification tersedia di [https://web.cs.dal.ca/~hawkey/3130/srs\\_template-ieee.doc](https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc)



## 7.1.6 Validasi (Validation)



Kualitas  
Work Product

Technical  
Review



### KUALITAS WORK PRODUCT

- Pada validasi, produk kerja yang dihasilkan selama rekayasa kebutuhan adalah **dinilai kualitasnya**.
- Perhatian utama selama validasi kebutuhan adalah **konsistensi**.
- Gunakan **model analisis** untuk memastikan bahwa kebutuhan telah dinyatakan **konsisten**:
  - Validasi kebutuhan **memeriksa spesifikasi** untuk memastikan bahwa semua kebutuhan PL telah dinyatakan dengan jelas;
  - bahwa inkonsistensi, kelalaian, dan kesalahan telah **dideteksi** dan **diperbaiki**;
  - bahwa produk kerja **sesuai dengan standar** yang ditetapkan untuk proses, proyek, dan produk.



### TECHNICAL REVIEW

- **Mekanisme utama** validasi kebutuhan adalah tinjauan teknis (Technical review)
- Tim review termasuk **software engineer**, pelanggan (**customer**), pengguna (**end users**), dan **stakeholder** lain memvalidasi kebutuhan, yaitu:
  - Memeriksa spesifikasi mencari kesalahan dalam konten atau interpretasi,
  - area di mana **clarifikasi** mungkin diperlukan,
  - **informasi yang hilang**,
  - **inkonsistensi** (masalah utama ketika produk atau sistem besar direkayasa),
  - **kebutuhan yang saling bertentangan**, atau
  - **kebutuhan yang tidak realistik** (tidak dapat dicapai).

Ilustrasi beberapa masalah yang terjadi selama validasi persyaratan, perhatikan dua kebutuhan berikut yang tampaknya tidak berbahaya:



- Kebutuhan pertama terlalu kabur untuk diuji atau dinilai oleh pengembang.
- Apa sebenarnya yang dimaksud dengan "ramah pengguna"?
- Untuk memvalidasinya, itu harus dikuantifikasi atau dikualifikasikan dalam beberapa cara.

- Persyaratan kedua memiliki elemen kuantitatif ("kurang dari 0,0001"), tetapi pengujian intrusi akan sulit dan memakan waktu.
- Apakah tingkat keamanan ini bahkan dijamin untuk aplikasi?
- Dapatkah persyaratan pelengkap lainnya yang terkait dengan keamanan (misalnya, perlindungan kata sandi, handshaking khusus) menggantikan persyaratan kuantitatif yang dicatat?

## 7.1.6 Manajemen Kebutuhan

- ❑ Kebutuhan untuk sistem berbasis komputer berubah, dan keinginan untuk mengubah kebutuhan tetap ada sepanjang umur sistem.
- ❑ Manajemen kebutuhan adalah serangkaian kegiatan yang membantu tim proyek mengidentifikasi, mengontrol, dan melacak kebutuhan dan perubahan kebutuhan setiap saat saat proyek berlangsung.



- 1 Apakah kebutuhan dinyatakan dengan jelas? Bisakah kebutuhan disalahartikan?
- 2 Apakah sumber (misalnya, seseorang, peraturan, dokumen) dari kebutuhan diidentifikasi?  
Apakah pernyataan akhir dari kebutuhan telah diperiksa oleh atau terhadap sumber aslinya?
- 3 Apakah kebutuhan dibatasi dalam istilah kuantitatif?
- 4 Kebutuhan lain apa yang terkait dengan kebutuhan ini?  
Apakah kebutuhan dicatat dengan jelas melalui matriks referensi silang atau mekanisme lain?
- 5 Apakah kebutuhan melanggar batasan domain sistem apa pun?
- 6 Apakah kebutuhan dapat diuji?  
Jika demikian, dapatkah kita menentukan pengujian (terkadang disebut kriteria validasi) untuk menjalankan kebutuhan?
- 7 Apakah kebutuhan dapat dilacak ke model sistem apa pun yang telah dibuat?
- 8 Apakah kebutuhan dapat dilacak ke keseluruhan sistem dan tujuan produk?
- 9 Apakah spesifikasi terstruktur dengan cara yang mengarah pada pemahaman yang mudah, referensi yang mudah, dan terjemahan yang mudah ke dalam produk kerja yang lebih teknis?
- 10 Apakah indeks untuk spesifikasi telah dibuat?
- 11 Apakah kebutuhan yang terkait dengan kinerja, perilaku, dan karakteristik operasional telah dinyatakan dengan jelas?  
Kebutuhan apa yang tampak implisit?

## 7.2 Establishing the Groundwork (Membangun Landasan)

- 7.2.1 Identifikasi Stakeholder
- 7.2.2 Mengenali Banyak Sudut Pandang
- 7.2.3 Bekerja menuju ke Kolaborasi
- 7.2.4 Mengajukan Pertanyaan Pertama
- 7.2.5 Kebutuhan Nonfungsional
- 7.2.6 Ketertelusuran (Traceability)

## 7.2 Establishing the Groundwork

Langkah-langkah yang diperlukan untuk membangun dasar untuk memahami kebutuhan PL—untuk memulai proyek dengan cara yang akan membuatnya terus bergerak maju menuju solusi yang sukses

- 7.2.1 Identifikasi Stakeholder
- 7.2.2 Mengenali Banyak Sudut Pandang
- 7.2.3 Bekerja menuju ke Kolaborasi
- 7.2.4 Mengajukan Pertanyaan Pertama
- 7.2.5 Kebutuhan Nonfungsional
- 7.2.6 Ketertelusuran (Traceability)

- ◆ Pada pengaturan yang ideal, stakeholder dan software engineer bekerja sama dalam satu tim
- ◆ Dalam kasus seperti itu, rekayasa kebutuhan hanyalah masalah melakukan percakapan yang bermakna dengan rekan kerja yang merupakan anggota tim yang dikenal baik.
- ◆ Tetapi kenyataan seringkali sangat berbeda.
  - ◆ Pelanggan atau pengguna akhir mungkin:
    - ◆ tinggal di kota atau negara yang berbeda,
    - ◆ hanya memiliki gagasan yang kabur tentang apa yang diperlukan,
    - ◆ memiliki pendapat yang bertentangan tentang sistem yang akan dibangun,
    - ◆ memiliki pengetahuan teknis yang terbatas, dan
    - ◆ memiliki waktu yang terbatas untuk berinteraksi dengan perekayasa kebutuhan
  - ◆ Tidak satu pun dari hal-hal ini yang diinginkan, tetapi semuanya umum, dan Anda sering dipaksa untuk bekerja dalam batasan yang dipaksakan oleh situasi ini.

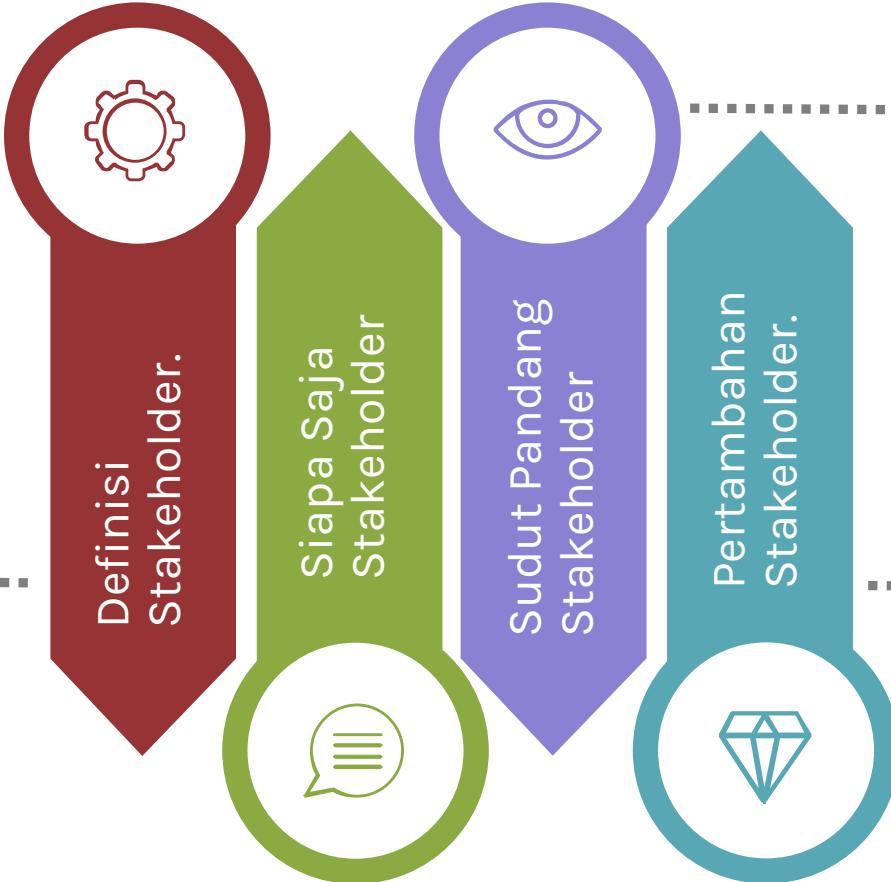
## 7.2.1 Identifikasi Stakeholder

### Definisi

Stakeholder adalah “setiap orang yang memperoleh manfaat baik secara langsung maupun tidak langsung dari sistem yang sedang dikembangkan”.

### Siapa Saja:

Manajer Operasi Bisnis, Manajer Produk, Orang Pemasaran, Pelanggan Internal dan Eksternal, Pengguna Akhir, Konsultan, Insinyur Produk, Software Engineer, Insinyur Dukungan dan Pemeliharaan, dan lain-lain



### Sudut Pandang

Setiap stakeholder memiliki pandangan yang berbeda tentang sistem, mencapai manfaat yang berbeda ketika sistem berhasil dikembangkan, dan terbuka terhadap risiko yang berbeda jika upaya pengembangan gagal

### Pertambahan Stakeholder

Daftar awal akan bertambah ketika para stakeholder dihubungi karena setiap stakeholder akan ditanyai: “Menurut Anda dengan siapa lagi saya harus berbicara?”.

## 7.2.2 Mengenali Banyak Sudut Pandang



### Banyak Stakeholder Berbeda Kebutuhan

- Grup pemasaran tertarik pada fitur yang akan menggairahkan pasar potensial, membuat sistem baru mudah dijual
- Manajer bisnis tertarik pada serangkaian fitur yang dapat dibangun sesuai anggaran dan yang akan siap untuk memenuhi market window yang ditentukan
- Pengguna akhir menginginkan fitur yang familiar dan mudah dipelajari serta digunakan
- Software engineer memperhatikan fungsi yang tidak terlihat oleh stakeholder nonteknis tetapi memungkinkan infrastruktur yang mendukung lebih banyak fungsi dan fitur yang dapat dipasarkan



### Kontribusi Informasi Stakeholder

- Masing-masing konstituen (dan lainnya) akan menyumbangkan informasi untuk proses rekayasa kebutuhan.
- Karena informasi dari berbagai sudut pandang dikumpulkan, kebutuhan yang muncul mungkin tidak konsisten atau mungkin bertentangan satu sama lain.
- Anda harus mengkategorikan semua informasi stakeholder termasuk kebutuhan yang tidak konsisten dan bertentangan dengan cara yang memungkinkan pengambil keputusan untuk memilih serangkaian kebutuhan yang konsisten secara internal untuk sistem.



### Tujuan Rekayasa Kebutuhan

- Beberapa hal dapat menyulitkan untuk memperoleh persyaratan perangkat lunak yang memuaskan penggunanya:
  - tujuan proyek tidak jelas,
- prioritas pemangku kepentingan berbeda,
  - orang memiliki asumsi yang tidak diucapkan,
  - pemangku kepentingan menafsirkan makna secara berbeda, dan
- persyaratan dinyatakan dengan cara yang membuat mereka sulit untuk diverifikasi.
  - Tujuan dari rekayasa kebutuhan yang efektif adalah untuk menghilangkan atau setidaknya mengurangi masalah ini

## 7.2.3 Bekerja Menuju ke Kolaborasi



### PLANNING POKER

- Gunakan skema “voting” berdasarkan poin prioritas
- Semua stakeholder diberikan sejumlah poin prioritas yang dapat “dibelanjakan” pada sejumlah kebutuhan
- Daftar kebutuhan disajikan, dan setiap stakeholder menunjukkan kepentingan relatif dari masing-masing (dari sudut pandangnya) dengan menggunakan satu atau lebih poin prioritas untuk itu.
- Poin yang dihabiskan tidak dapat digunakan kembali.
- Setelah poin prioritas stakeholder habis, tidak ada tindakan lebih lanjut tentang kebutuhan yang dapat diambil oleh orang tersebut.
- Poin keseluruhan yang dihabiskan untuk setiap kebutuhan oleh semua stakeholder memberikan indikasi pentingnya keseluruhan setiap kebutuhan



- Ketika 5 stakeholder terlibat dalam proyek PL, maka Anda mungkin memiliki lima (atau lebih) pendapat berbeda tentang kumpulan kebutuhan yang tepat
- Pelanggan (dan stakeholder lainnya) harus berkolaborasi di antara mereka sendiri (menghindari pertempuran kecil-kecilan) dan dengan praktisi RPL jika ingin menghasilkan sistem yang sukses.
- Tapi bagaimana kolaborasi ini dicapai?

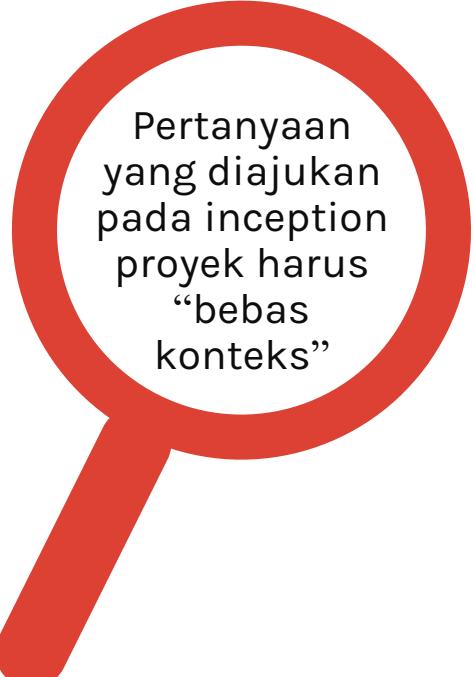


- Pekerjaan seorang perekaya kebutuhan adalah untuk mengidentifikasi area kesamaan (yaitu, kebutuhan yang disetujui semua stakeholder) dan
- Area konflik atau inkonsistensi (yaitu, kebutuhan yang diinginkan oleh satu stakeholder tetapi bertentangan dengan kebutuhan stakeholder lain).
- Tentu saja, kategori terakhir inilah yang menghadirkan tantangan.



- Kolaborasi tidak berarti bahwa kebutuhan “ditentukan oleh komite.”
- Dalam banyak kasus, stakeholder berkolaborasi dengan memberikan pandangan mereka tentang kebutuhan, tetapi “project champion” yang kuat (mis., Manajer bisnis atau teknolog senior) dapat membuat keputusan akhir tentang kebutuhan mana yang tepat.

## 7.2.4 Mengajukan Pertanyaan Pertama



Pertanyaan yang diajukan pada inception proyek harus “bebas konteks”



- Rangkaian pertanyaan bebas konteks pertama berfokus pada pelanggan dan stakeholder lainnya serta tujuan dan manfaat proyek secara keseluruhan:
  - Siapa di balik permintaan pekerjaan ini?
  - Siapa yang akan menggunakan solusinya?
  - Apa manfaat ekonomi dari solusi yang berhasil?
  - Apakah ada sumber lain untuk solusi yang Anda butuhkan?
- Pertanyaan-pertanyaan ini membantu mengidentifikasi semua stakeholder yang akan tertarik dengan PL yang akan dibangun.
- Selain itu, pertanyaan mengidentifikasi manfaat terukur dari implementasi yang sukses dan kemungkinan alternatif untuk pengembangan PL khusus

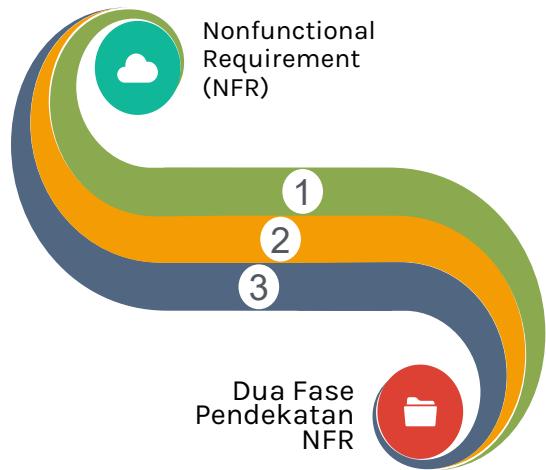


- Kumpulan pertanyaan berikutnya memungkinkan Anda untuk mendapatkan pemahaman yang lebih baik tentang masalah dan memungkinkan pelanggan untuk menyuarakan persepsiannya tentang solusi:
  - Bagaimana Anda mencirikan keluaran "baik" yang akan dihasilkan oleh solusi yang sukses?
  - Problem apa yang akan ditangani oleh solusi ini?
  - Dapatkah Anda menunjukkan kepada saya (atau menjelaskan) lingkungan bisnis di mana solusi akan digunakan?
  - Apakah problem atau kendala kinerja khusus akan mempengaruhi cara pendekatan solusi?



- Rangkaian pertanyaan terakhir berfokus pada efektivitas aktivitas komunikasi itu sendiri.
- Gause dan Weinberg (1989) menyebut ini "meta-questions" dan mengusulkan daftar (disingkat) berikut:
  - Apakah Anda orang yang tepat untuk menjawab pertanyaan-pertanyaan ini?
  - Apakah jawaban Anda "resmi"?
  - Apakah pertanyaan saya relevan dengan masalah yang Anda miliki?
  - Apakah saya terlalu banyak bertanya?
  - Adakah yang bisa memberikan informasi tambahan?
  - Haruskah saya menanyakan hal lain pada Anda?

## 7.2.5 Kebutuhan Nonfungsional (Nonfunctional Requirements – NFR)



### Definisi Kebutuhan Nonfungsional (NFR)

- NFR adalah atribut kualitas, atribut kinerja, atribut keamanan, atau batasan umum pada suatu sistem.
- Ini seringkali tidak mudah bagi para stakeholder untuk mengartikulasikannya.
- Chung [Chu09] menunjukkan bahwa ada penekanan yang berat sebelah pada fungsionalitas PL, namun PL mungkin tidak berguna atau dapat digunakan tanpa karakteristik nonfungsional yang diperlukan.



### Fase Pertama identifikasi NFR

- Seperangkat pedoman RPL ditetapkan untuk sistem yang akan dibangun.
- Termasuk pedoman untuk praktik terbaik, membahas gaya arsitektural (Bab 10) dan penggunaan pola desain/design pattern (Bab 14).
- Daftar NFR (misalnya, kebutuhan yang membahas kegunaan, kemampuan pengujian, keamanan, atau pemeliharaan) kemudian dikembangkan.
- Tabel sederhana mencantumkan NFR sebagai label kolom dan pedoman RPL sebagai label baris.
- Matriks hubungan membandingkan setiap pedoman dengan pedoman lainnya, membantu tim menilai apakah setiap pasangan pedoman saling melengkapi, tumpang tindih, bertentangan, atau independen.



### Fase Kedua identifikasi NFR

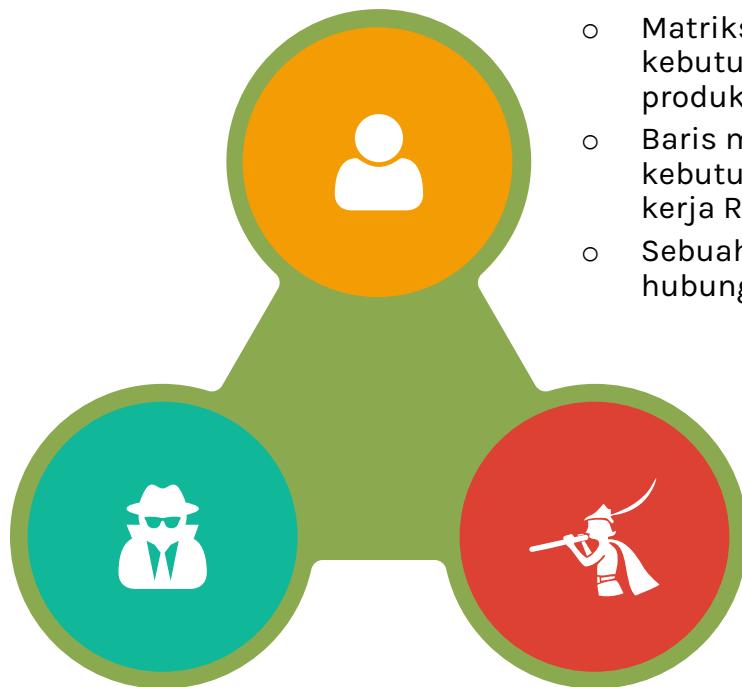
- Tim memprioritaskan setiap kebutuhan nonfungsional dengan membuat seperangkat NFR yang homogen menggunakan seperangkat aturan keputusan yang menetapkan pedoman mana yang harus diterapkan dan mana yang ditolak.

## 7.2.6 Ketertelusuran (Traceability)

Ketertelusuran adalah istilah RPL yang mengacu pada tautan terdokumentasi antara produk kerja (work product) RPL (misalnya, kebutuhan dan kasus uji).

### Kebutuhan dan Work Product Berkembang

- Seiring bertambahnya jumlah kebutuhan dan jumlah produk kerja, maka semakin sulit untuk menjaga agar matriks ketertelusuran tetap mutakhir.
- Meskipun demikian, penting untuk membuat beberapa cara untuk melacak dampak dan evolusi kebutuhan produk



### Matriks Ketertelusuran

- Matriks ketertelusuran memungkinkan seorang perekayasa kebutuhan untuk mewakili hubungan antara kebutuhan dan produk kerja RPL lainnya.
- Baris matriks ketertelusuran diberi label menggunakan nama kebutuhan, dan kolom dapat diberi label dengan nama produk kerja RPL (misalnya, elemen desain atau kasus uji).
- Sebuah sel matriks ditandai untuk menunjukkan adanya hubungan antara keduanya.

### Manfaat Matriks Ketertelusuran

- Matriks ketertelusuran dapat mendukung berbagai kegiatan pengembangan rekayasa.
- Matriks ini dapat memberikan kesinambungan bagi pengembang saat proyek bergerak dari satu fase proyek ke fase lainnya, terlepas dari model proses yang digunakan.
- Matriks ketertelusuran sering dapat digunakan untuk memastikan produk kerja rekayasa telah mempertimbangkan semua persyaratan

# 7.3 Requirements Gathering

7.3.1 Collaborative Requirements Gathering

7.3.2 Usage Scenarios

7.3.3 Elicitation Work Products

## 7.3 Requirements Gathering (Pengumpulan Kebutuhan)

- 7.3.1 Collaborative Requirements Gathering
- 7.3.2 Usage Scenarios
- 7.3.3 Elicitation Work Products



- ◆ Pengumpulan persyaratan menggabungkan elemen pemecahan masalah, elaborasi, negosiasi, dan spesifikasi.
- ◆ Untuk mendorong pendekatan kolaboratif, berorientasi tim untuk pengumpulan persyaratan, pemangku kepentingan bekerja sama untuk mengidentifikasi masalah, mengusulkan elemen solusi, menegosiasikan pendekatan yang berbeda, dan menentukan serangkaian persyaratan solusi awal

## 7.3.1 Collaboration Requirements Gathering (Pengumpulan Kebutuhan Kolaboratif)

- Banyak pendekatan yang berbeda untuk pengumpulan kebutuhan kolaboratif telah diusulkan.
  - Masing-masing menggunakan skenario yang sedikit berbeda, tetapi semuanya menerapkan beberapa variasi pada pedoman dasar berikut:
    - ✓ Rapat/meeting (face-to-face atau virtual) dilakukan dan dihadiri oleh software engineer stakeholder lainnya.
    - ✓ Aturan untuk persiapan dan partisipasi ditetapkan
    - ✓ Sebuah agenda disarankan yang cukup formal untuk mencakup semua poin penting tetapi cukup informal untuk mendorong aliran ide yang bebas.
    - ✓ Seorang "fasilitator" (dapat berupa pelanggan, pengembang, atau orang luar) mengontrol rapat.
    - ✓ Sebuah "mekanisme definisi" (bisa berupa lembar kerja, flip chart, atau stiker dinding atau papan buletin elektronik, ruang obrolan, atau forum virtual) digunakan.
  - Tujuannya adalah untuk mengidentifikasi problem, mengusulkan elemen solusi, menegosiasikan pendekatan yang berbeda, dan menentukan serangkaian kebutuhan solusi awal
-  Satu atau dua halaman "Kebutuhan Produk" dibuat saat inception (Bagian 7.2)
    - Tempat, waktu, dan tanggal pertemuan dipilih;
    - Fasilitator dipilih; dan
    - Peserta dari tim perangkat lunak dan organisasi pemangku kepentingan lainnya diundang untuk berpartisipasi
  -  Jika suatu sistem atau produk akan melayani banyak pengguna, pastikan bahwa kebutuhan diperoleh dari perwakilan pengguna.
  -  Jika hanya satu pengguna yang mendefinisikan semua kebutuhan, maka risiko penerimaan (acceptance) tinggi (artinya mungkin ada beberapa stakeholder lain yang tidak akan menerima produk).
  -  Permintaan produk didistribusikan ke semua peserta sebelum tanggal meeting.

### 7.3.1 Collaboration Requirements Gathering (Pengumpulan Kebutuhan Kolaboratif)

- Contoh narasi permintaan produk home security function dari personel pemasaran yang terlibat dalam proyek SafeHome:
  - Penelitian kami menunjukkan bahwa pasar untuk sistem manajemen rumah tumbuh pada tingkat 40 persen per tahun. Fungsi SafeHome pertama yang kami bawa ke pasar adalah fungsi keamanan rumah. Kebanyakan orang akrab dengan "sistem alarm", jadi ini akan menjadi penjualan yang mudah. Kami mungkin juga mempertimbangkan untuk menggunakan kontrol suara sistem menggunakan beberapa teknologi seperti Alexa.
  - Fungsi keamanan rumah akan melindungi dan/atau mengenali berbagai "situasi" yang tidak diinginkan seperti masuk secara ilegal, kebakaran, banjir, kadar karbon monoksida, dan lain-lain. Ini akan menggunakan sensor nirkabel kami untuk mendeteksi setiap situasi, dapat diprogram oleh pemilik rumah, dan secara otomatis akan menghubungi agen pemantau dan ponsel pemilik ketika situasi terdeteksi.



Pada kenyataannya, orang lain akan berkontribusi pada narasi tersebut selama pertemuan pengumpulan kebutuhan dan lebih banyak informasi akan tersedia.



Tetapi bahkan dengan informasi tambahan, terdapat ambiguitas, kelalaian mungkin ada, dan kesalahan mungkin terjadi. Untuk saat ini, "deskripsi fungsional" sebelumnya sudah cukup.



Saat meninjau permintaan produk di hari-hari sebelum pertemuan, setiap peserta diminta untuk membuat:

- Daftar objek yang merupakan bagian dari lingkungan yang mengelilingi sistem,
- Objek lain yang akan diproduksi oleh sistem, dan
- Objek yang digunakan oleh sistem untuk menjalankan fungsinya.
- Daftar layanan lain (proses atau fungsi) yang memanipulasi atau berinteraksi dengan objek.
- Daftar kendala (misalnya, biaya, ukuran, aturan bisnis) dan
- Kriteria kinerja (misalnya, kecepatan, akurasi, keamanan) juga dikembangkan.



Para peserta diberitahu bahwa daftar tersebut tidak diharapkan lengkap tetapi diharapkan mencerminkan persepsi setiap orang tentang sistem.

### 7.3.1 Collaboration Requirements Gathering (Pengumpulan Kebutuhan Kolaboratif)



#### Object pada SafeHome:

- Control Panel
  - Smoke Detector
- Sensor jendela dan pintu
  - Motion detector
    - Alarm
- An event (suatu sensor telah diaktifasi)
  - Display
  - Tablet
- Nomor telepon
- Telephone calls
  - Dan lain lain

#### Performance Criteria pada SafeHome

- dikembangkan oleh setiap peserta meeting. Misalnya:
- Sensor event harus dikenali dalam 1 detik
- Skema prioritas event harus diimplementasikan



#### Service pada SafeHome

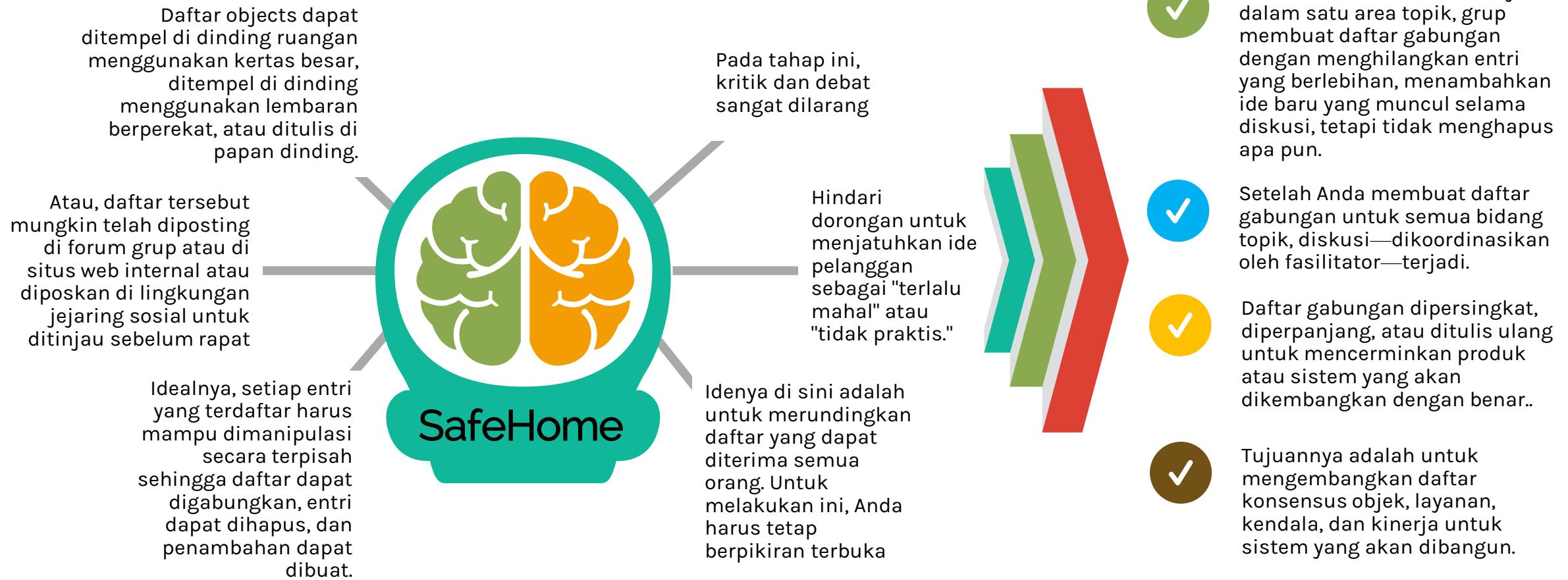
(perhatikan bahwa layanan bekerja pada objek):

- Mengonfigurasi sistem,
- Menyetel alarm,
- Memantau sensor,
- Memanggil telepon menggunakan router nirkabel,
- Memprogram panel kontrol,
- Membaca tampilan.

#### Constraints pada SafeHome

- dikembangkan oleh setiap peserta meeting. Misalnya:
  - Sistem harus mengenali ketika sensor tidak beroperasi,
  - Harus ramah pengguna
  - Harus berinteraksi langsung ke saluran telepon standar

### 7.3.1 Collaboration Requirements Gathering (Pengumpulan Kebutuhan Kolaboratif)



### 7.3.1 Collaboration Requirements Gathering (Pengumpulan Kebutuhan Kolaboratif)



Dalam banyak kasus, objek atau service yang dijelaskan dalam daftar akan memerlukan penjelasan lebih lanjut. Untuk mencapai ini, stakeholder mengembangkan spesifikasi mini (**Mini Specification**) untuk entri pada daftar atau dengan membuat kasus penggunaan yang melibatkan objek atau service

#### Mini Specification untuk object Control Panel SafeHome

- Control Panel adalah unit yang dipasang di dinding dengan ukuran kira-kira  $230 \times 130$  mm.
- Control Panel memiliki konektivitas nirkabel ke sensor dan tablet.
- Interaksi pengguna terjadi melalui keypad yang berisi 12 tombol.
- Layar warna OLED  $75 \times 75$  mm memberikan umpan balik pengguna.
- Perangkat lunak menyediakan petunjuk interaktif, gema, dan fungsi serupa

#### Dipresentasikan dan Dibahas dalam Meeting

- Spesifikasi mini dipresentasikan kepada semua pemangku kepentingan untuk didiskusikan.
- Penambahan, penghapusan, dan penjabaran lebih lanjut dilakukan.
- Dalam beberapa kasus, pengembangan spesifikasi mini akan mengungkap objek, layanan, kendala, atau persyaratan kinerja baru yang akan ditambahkan ke daftar asli.
- Selama semua diskusi, tim dapat mengangkat masalah yang tidak dapat diselesaikan selama rapat.
- Daftar masalah dipertahankan sehingga ide-ide ini akan ditindaklanjuti nanti.



Banyak kekhawatiran pemangku kepentingan (misalnya, akurasi, aksesibilitas data, keamanan) adalah dasar untuk kebutuhan sistem yang tidak berfungsi (Bagian 7.2).

- Saat pemangku kepentingan menyatakan keprihatinan ini, software engineer harus mempertimbangkannya dalam konteks sistem yang akan dibangun.
- Pertanyaan yang harus dijawab adalah:
  - Bisakah kita membangun sistem?
  - Akankah proses pengembangan ini memungkinkan kita untuk mengalihkan pesaing kita ke pasar?
  - Apakah tersedia sumber daya yang memadai untuk membangun dan memelihara sistem yang diusulkan?
  - Akankah kinerja sistem memenuhi kebutuhan pelanggan kami?

## 7.3.2 Usage Scenario

Saat persyaratan dikumpulkan, visi keseluruhan fungsi dan fitur sistem mulai terwujud.

Beralih ke Aktivitas lebih Teknis

Namun, sulit untuk beralih ke aktivitas RPL yang lebih teknis sampai Anda memahami bagaimana fitur akan digunakan oleh kelas pengguna akhir yang berbeda.

Skenario vs Use Case

Skenario, sering disebut use case, memberikan deskripsi tentang bagaimana sistem akan digunakan.

Skenario Penggunaan Sistem

Untuk mencapai ini, pengembang dan pengguna dapat membuat serangkaian skenario yang mengidentifikasi rangkaian penggunaan untuk sistem yang akan dibangun.

Penggunaan Use Case di bagian 7.4

Kasus penggunaan dibahas secara lebih rinci di Bagian 7.4.



## SAFEHOME



### *Developing a Preliminary User Scenario*

**The scene:** A meeting room, continuing the first requirements gathering meeting.

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

#### **The conversation:**

**Facilitator:** We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

**Jamie:** How?

**Facilitator:** We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

**Marketing person:** Um . . . well, this is the kind of thing I'd do if I was away from home and I

had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

**Facilitator (smiling):** That's the reason you'd do it . . . tell me how you'd actually do this.

**Marketing person:** Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user ID and . . .

**Vinod (interrupting):** The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

**Facilitator (interrupting):** That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

**Vinod:** No problem.

**Marketing person:** So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

**Jamie:** What if I forget my password?

**Facilitator (interrupting):** Good point, Jamie, but let's not address that now. We'll make a note of that and call it an exception. I'm sure there'll be others.

**Marketing person:** After I enter the passwords, a screen representing all *SafeHome* functions will appear. I'd select the home security function. The system might request that I verify who I am, say, by asking for my address or phone number or something. It would then display a picture of the security system control

panel along with a list of functions that I can perform—arm the system, disarm the system, disarm one or more sensors. I suppose it might also allow me to reconfigure security zones and other things like that, but I'm not sure.

(As the marketing person continues talking, Doug takes copious notes; these form the basis for the first informal usage scenario. Alternatively, the marketing person could have been asked to write the scenario, but this would be done outside the meeting.)

### 7.3.3 Elisitasi Work Products

- ❖ Produk kerja (work products) yang dihasilkan selama elisitasi kebutuhan akan bervariasi tergantung pada ukuran sistem atau produk yang akan dibangun.
- ❖ Masing-masing produk kerja ini ditinjau oleh semua orang yang telah berpartisipasi dalam elisitasi kebutuhan



## 7.4 Developing Use Cases

## 7.4 Developing Use Cases

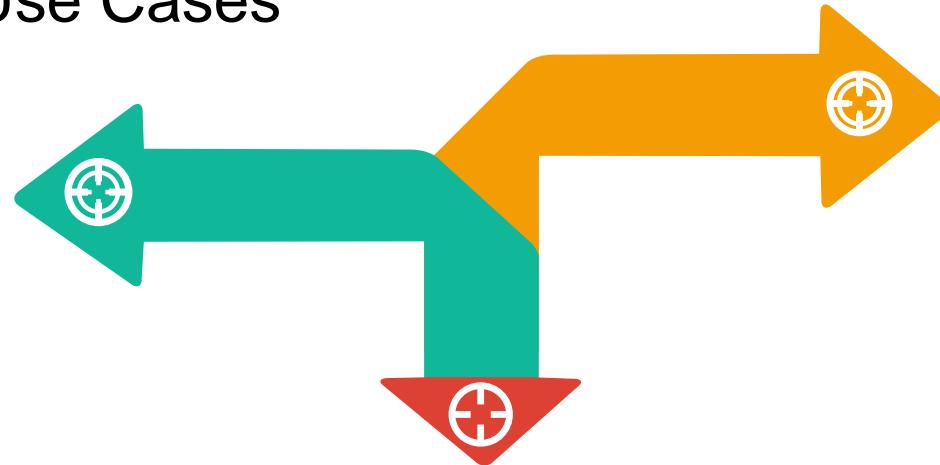
- **Use Case** menceritakan sebuah cerita bergaya tentang bagaimana pengguna akhir (memainkan salah satu dari beberapa kemungkinan peran) berinteraksi dengan sistem pada keadaan tertentu.
  - Teks naratif (cerita pengguna),
  - Garis besar tugas atau interaksi,
  - Deskripsi berbasis template, atau
  - Representasi diagram.
- Terlepas dari bentuknya, use case menggambarkan PL atau sistem dari sudut pandang pengguna akhir.



- ◆ Langkah pertama dalam menulis use case adalah mendefinisikan set "aktor" yang akan terlibat dalam cerita.
  - ◆ Didefinisikan secara lebih formal, aktor adalah segala sesuatu yang berkomunikasi dengan sistem atau produk dan yang berada di luar sistem itu sendiri.
  - ◆ Aktor adalah orang yang berbeda (atau perangkat) yang menggunakan sistem atau produk dalam konteks fungsi dan perilaku yang akan dijelaskan.
  - ◆ Aktor akan mewakili peran yang dimainkan orang (atau perangkat) saat sistem beroperasi.
  - ◆ Setiap aktor memiliki **satu atau lebih tujuan** saat menggunakan sistem.,
- ◆ Penting untuk dicatat bahwa **aktor** dan **pengguna akhir** tidak selalu sama.
  - ◆ Seorang pengguna biasa dapat memainkan beberapa peran berbeda saat menggunakan sistem,
  - ◆ Sedangkan aktor mewakili kelas entitas eksternal (seringkali, tetapi tidak selalu, orang) yang hanya memainkan satu peran dalam konteks use case.

## 7.4 Developing Use Cases

- Misalkan **pengguna** berinteraksi dengan program yang memungkinkan berekspresi dengan **konfigurasi sensor alarm** di **gedung virtual**.
- Setelah meninjau kebutuhan dengan cermat, PL untuk **komputer kontrol** memerlukan **empat mode (peran)** yang berbeda untuk interaksi:
  - mode placement,
  - mode testing,
  - mode monitoring, dan
  - mode pemecahan masalah (troubleshooting).
- Jadi, diperlukan 4 aktor:
  - Editor
  - Tester
  - Monitor
  - Troubleshooter
- Dalam beberapa kasus, pengguna dapat memainkan semua peran.
- Di tempat lain, orang yang berbeda mungkin memainkan peran masing-masing aktor



- Karena elisitasi kebutuhan adalah aktivitas evolusioner, tidak semua aktor diidentifikasi selama iterasi pertama.
- Kemungkinan identifikasi aktor primer pada iterasi pertama dan aktor sekunder setelah sistem dipelajari lebih banyak.
  - Aktor primer berinteraksi untuk mencapai fungsi sistem yang diperlukan dan memperoleh manfaat yang diinginkan dari sistem.
  - Aktor primer bekerja secara langsung dan sering dengan perangkat lunak.
  - Aktor sekunder mendukung sistem sehingga aktor primer dapat melakukan pekerjaannya

- Setelah para aktor diidentifikasi, maka langkah selanjutnya ialah membuat Use Case
- pertanyaan yang harus dijawab oleh use case:
  - 1) Siapa aktor utama, aktor sekunder?
  - 2) Apa tujuan aktor?
  - 3) Prasyarat apa yang harus ada sebelum cerita/story dimulai?
  - 4) Apa tugas atau fungsi utama yang dilakukan oleh aktor?
  - 5) Pengecualian apa yang mungkin dipertimbangkan saat story dijelaskan?
  - 6) Variasi apa dalam interaksi aktor yang mungkin terjadi?
  - 7) Informasi sistem apa yang akan diperoleh, diproduksi, atau diubah oleh aktor?
  - 8) Apakah aktor harus menginformasikan sistem tentang perubahan lingkungan eksternal?
  - 9) Informasi apa yang diinginkan aktor dari sistem?
  - 10) Apakah aktor ingin diberitahu tentang perubahan yang tidak terduga?

## 7.4 Developing Use Cases

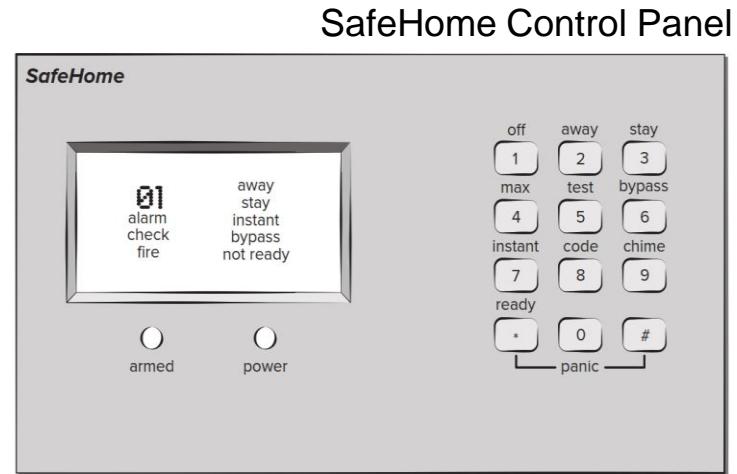
### ○ Empat Aktor SafeHome

- Homeowner – seorang pengguna
- Setup Manager – orang yang sama dengan homeowner tetapi pada peran berbeda
- Sensor – perangkat (device) yang melekat pada sistem
- Monitoring & Response Subsystem – central station yang memonitor fungsi SafeHome Home Security



### ○ Aktor Homeowner

- Aktor Homeowner berinteraksi dengan **fungsi home security** dalam cara berbeda, yaitu **alarm control panel**, **tablet**, atau **ponsel**.
- Homeowner akan:
  - 1) Enter password agar bisa melakukan semua interaksi
  - 2) Menanyakan tentang status zona keamanan (security zone)
  - 3) Menanyakan tentang status sensor
  - 4) Menekan tombol panik (panic button) saat darurat (emergency)
  - 5) Aktivasi dan deaktivasi security system



**SafeHome Control Panel**

### ○ Use case dasar untuk aktivasi sistem pada Panel Kontrol:

- 1) Aktor Homeowner mencermati Panel Kontrol SafeHome untuk menentukan apakah sistem siap menerima input. Jika sistem tidak siap, maka muncul pesan **Not Ready** di layar display LCD.
- 2) Homeowner menggunakan **keypad** untuk mengetik **4-digit password**. Password dibandingkan dengan password valid pada sistem. Jika **password tidak benar** maka Kontrol Panel akan berbunyi **beep** satu kali dan **reset** sendiri untuk meminta input lanjutan. Jika input **password benar**, maka Kontrol Panel akan **menunggu aksi** berikutnya.
- 3) Homeowner memilih dan menekan tombol ‘stay’ atau ‘away’ untuk mengaktifasi sistem. **Stay** hanya mengaktifasi sensor perimeter (sensor pendekripsi gerakan di dalam dinonaktifkan). **Away** mengaktifasi seluruh sensor.
- 4) Ketika aktivasi terjadi, lampu merah alarm bisa dilihat oleh Homeowner

# Use Case Detail

**Use case** : Inisiasi Monitoring

**Primary Actor** : Homeowner

**Goal in context**: Untuk mengatur sistem untuk memonitor sensor ketika Homeowner meninggalkan rumah atau tetap berada di dalam rumah

**Preconditions** : Sistem telah diprogram untuk password dan untuk mengenali berbagai sensor

**Trigger** : Homeowner memutuskan untuk “set” sistem, yaitu menyalakan fungsi-fungsi alarm

**Scenario** :

1. Homeowner mengamati Panel Kontrol
2. Homeowner enter password
3. Homeowner menilih “stay” atau “away”
4. Homeowner mengamati lampu alarm merah untuk menunjukkan bahwa SafeHome telah dipersenjatai (armed)

## Exceptions:

1. Panel Kontrol not ready: Homeowner cek seluruh sensor untuk menentukan mana yang terbuka dan kemudian menutupnya
2. Password tidak benar (Panel Kontrol berbunyi sekali): Homeowner enter ulang password yang benar
3. Password tidak dikenali: Monitoring & Response Subsystem harus dikontak untuk program ulang password
4. Stay dipilih: Panel Kontrol beep dua kali, dan lampu stay menyala; sensor perimeter diaktifasi
5. Away dipilih: Panel Kontrol beep tiga kali, dan lampu away menyala; seluruh sensor diaktifasi

**Priority** : Esensial, harus diimplementasikan

**When available** : Increment pertama

**Frequency of Use** : Per hari seringkali

**Channel to Actor** : Via interface Kontrol Panel

**Secondary Actors** : Support technician, sensors

## Channel to Secondary Actors:

Support technician: phone line

Sensor: interface hardwire dan frekuensi radio

## Open Issues:

1. Haruskah ada cara untuk mengaktifkan sistem tanpa menggunakan password atau dengan password yang disingkat?
2. Haruskah Panel Kontrol menampilkan pesan teks tambahan?
3. Berapa lama Homeowner harus memasukkan password sejak tombol pertama ditekan?
4. Apakah ada cara untuk menonaktifkan sistem sebelum benar-benar diaktifkan?

# 7.5 Building the Analysis Model

- 7.5.1 Elemen Model Analisis
- 7.5.2 Analysis Pattern (Pola Analisis)

# 7.5 Building the Analysis Model

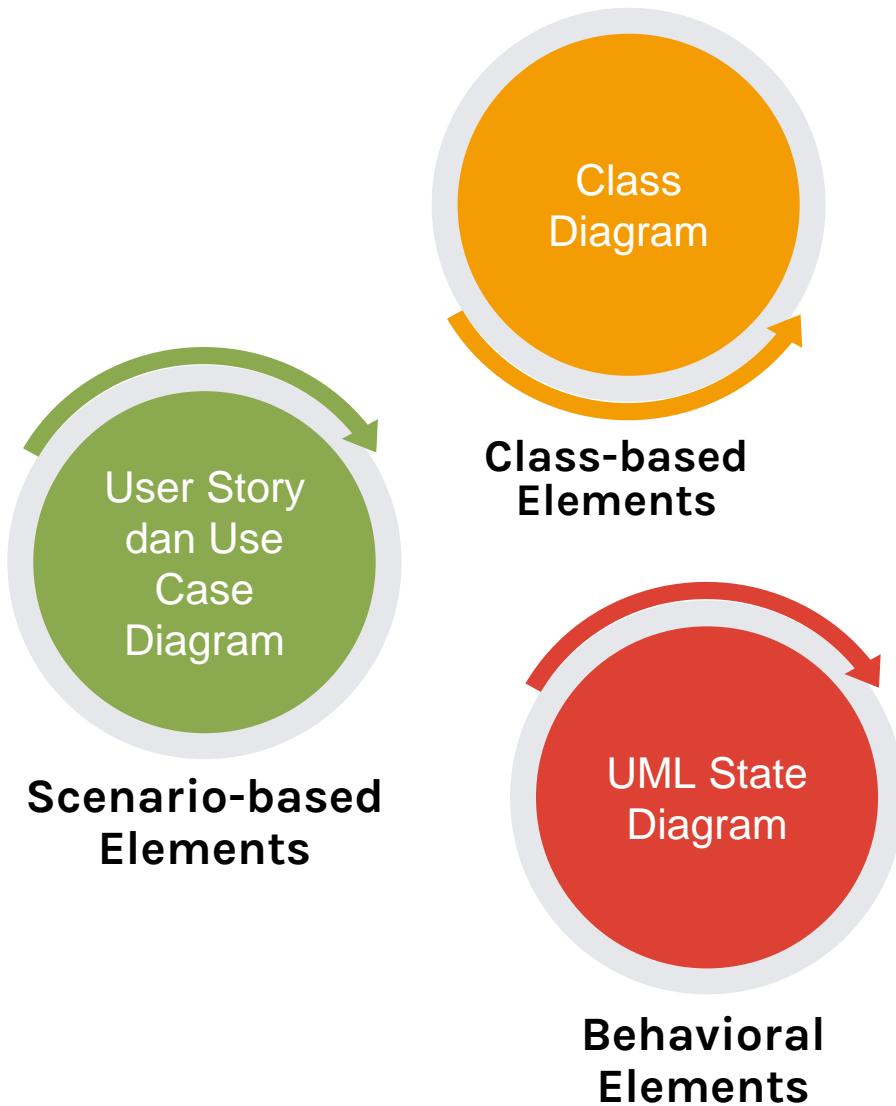
## 7.5.1 Elemen Model Analisis

## 7.5.2 Analysis Pattern (Pola Analysis)

- ◆ Maksud dari model analisis adalah untuk memberikan deskripsi tentang domain informasi, fungsional, dan perilaku yang diperlukan untuk sistem berbasis komputer.
- ◆ Model berubah secara dinamis saat Anda mempelajari lebih lanjut tentang sistem yang akan dibangun, dan saat stakeholder lebih memahami apa yang sebenarnya mereka butuhkan.
- ◆ Oleh karena itu, model analisis adalah gambaran kebutuhan pada waktu tertentu. Anda harus mengharapkannya untuk berubah.
- ◆ Saat model analisis berkembang, elemen tertentu akan menjadi relatif stabil, memberikan dasar yang kuat untuk tugas desain berikutnya.
- ◆ Namun, elemen model lainnya mungkin lebih fluktuatif, menunjukkan bahwa stakeholder belum sepenuhnya memahami persyaratan sistem.
- ◆ Jika tim Anda tidak menggunakan elemen tertentu dari model analisis saat proyek bergerak ke desain dan konstruksi, maka elemen tersebut tidak boleh dibuat di masa mendatang dan tidak boleh dipertahankan karena kebutuhan berubah dalam proyek saat ini.
- ◆ Model analisis dan metode yang digunakan untuk membangunnya disajikan secara rinci pada Bab 8.

## 7.5.1 Elemen Model Analisis

- Beberapa orang berpendapat bahwa yang terbaik adalah **memilih satu mode representasi** (misalnya, **use case**) dan menerapkannya dengan mengesampingkan semua mode lainnya.
- Praktisi lain percaya bahwa ada gunanya menggunakan **beberapa mode representasi** yang berbeda untuk menggambarkan model analisis.
- Menggunakan **mode representasi yang berbeda** memaksa Anda untuk mempertimbangkan kebutuhan dari sudut pandang yang berbeda — sebuah pendekatan yang **memiliki kemungkinan lebih tinggi untuk mengungkap kelalaian, inkonsistensi, dan ambiguitas**.
- Hal ini selalu merupakan ide yang baik untuk **melibatkan stakeholder**. Salah satu cara terbaik untuk melakukannya adalah **meminta setiap stakeholder menulis use case** yang menjelaskan bagaimana PL akan digunakan.

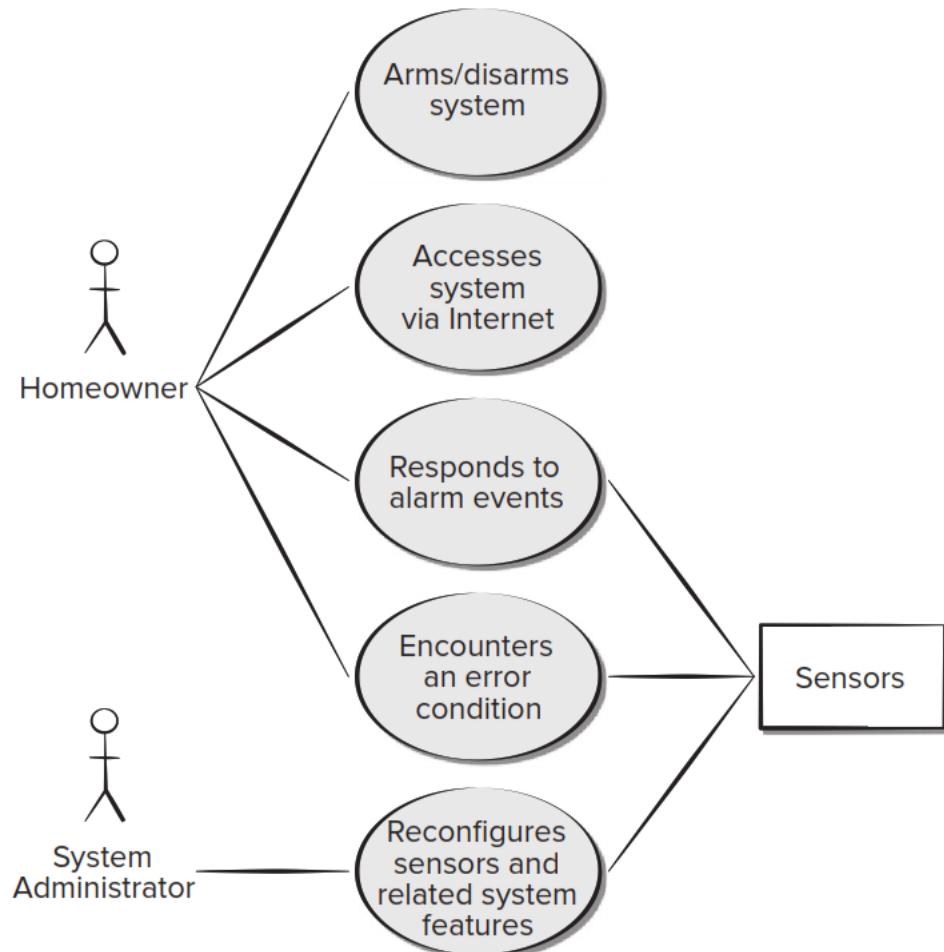


### 7.5.1 Elemen Model Analisis



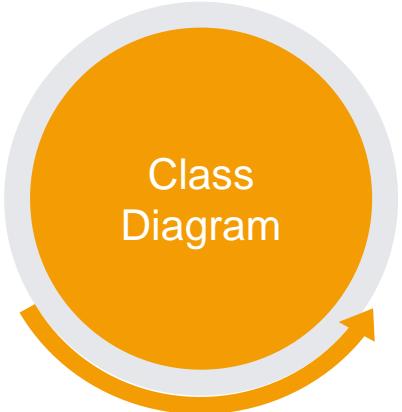
#### Scenario-based Elements

- Elemen berbasis skenario pada model kebutuhan merupakan bagian dari model yang **pertama kali dikembangkan**.
- Elemen berbasis skenario **menjelaskan sistem dari sudut pandang pengguna**
- User story dan use case diagram yang sesuai dapat **dielaborasi menjadi use case berbasis templat**
- Elemen berbasis skenario berfungsi **sebagai input untuk pembuatan elemen pemodelan lainnya**



UML use case diagram untuk  
SafeHome home security function

## 7.5.1 Elemen Model Analisis



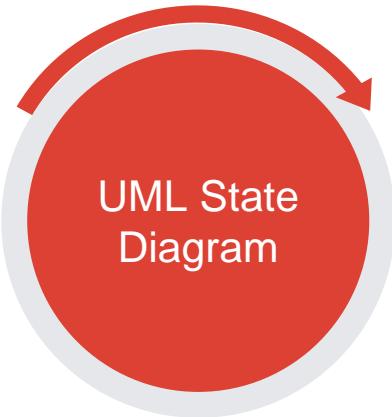
### Class-based Elements

- Setiap usage scenario menyiratkan satu set objek yang dimanipulasi sebagai aktor berinteraksi dengan sistem
- Objek-objek ini dikategorikan ke dalam **class** (misalnya dengan diagram kelas UML) — kumpulan **things** yang memiliki atribut dan perilaku umum yang serupa
- Diagram mencantumkan daftar atribut dari sensor (nama, tipe) dan operations (*identify*, *enable*) yang diterapkan untuk memodifikasi atribut.
- **Class** berkolaborasi satu sama lain dan hubungan serta interaksi antar **class**
- Salah satu cara untuk mengisolasi **class** adalah dengan mencari **kata benda** deskriptif dalam skrip use case
- **Kata kerja** yang ditemukan dalam skrip kasus penggunaan dapat dianggap sebagai kandidat **method** untuk class ini

Sensor
Name
Type
Location
Area
Characteristics
Identify()
Enable()
Disable()
Reconfigure()

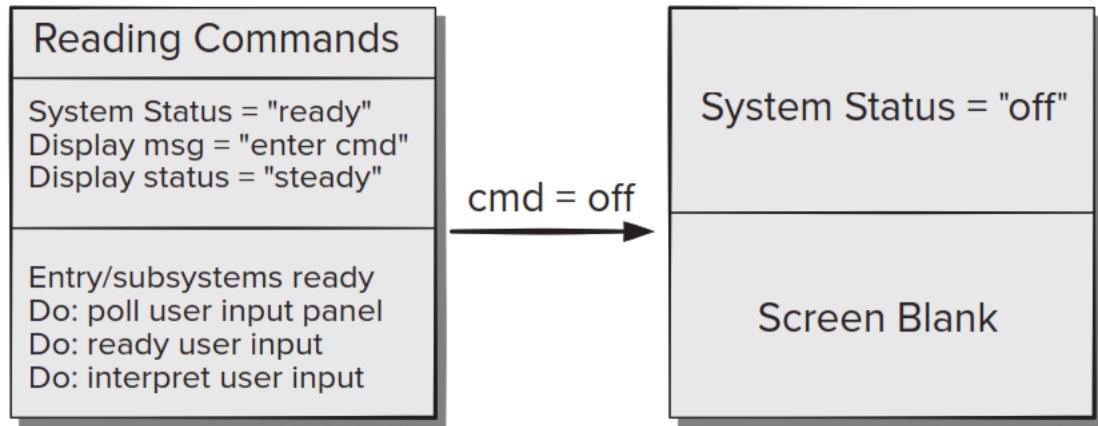
Class Diagram untuk Sensor

## 7.5.1 Elemen Model Analisis



### Behavioral Elements

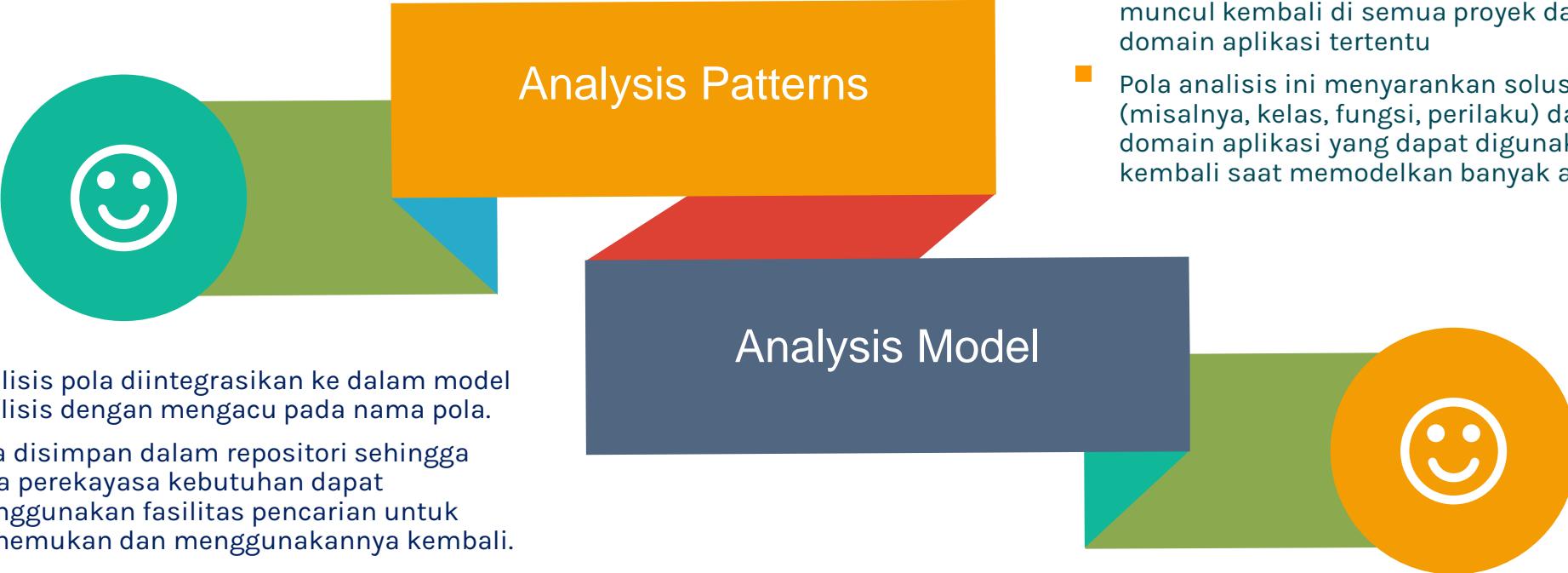
- Model kebutuhan harus menyediakan elemen pemodelan yang menggambarkan perilaku
- **State diagram** adalah salah satu **metode** untuk **merekpresentasikan perilaku** suatu sistem dengan menggambarkan keadaannya (**state**) dan kejadian (**event**) yang menyebabkan sistem berubah state
- State adalah setiap mode perilaku yang dapat diamati secara eksternal.
- State diagram menunjukkan **action** apa (misalnya, aktivasi proses) yang diambil **ketika event terjadi**.
- Stimulus eksternal (event) menyebabkan transisi antar state.



UML State Diagram Notation

Ilustrasi penggunaan state diagram untuk PL yang tertanam (embedded software) di dalam panel kontrol SafeHome yang bertanggung jawab untuk membaca input pengguna

## 7.5.2 Analysis Pattern (Analisis Pola)

- 
- Analisis pola diintegrasikan ke dalam model analisis dengan mengacu pada nama pola.
  - Pola disimpan dalam repositori sehingga para perekayasa kebutuhan dapat menggunakan fasilitas pencarian untuk menemukan dan menggunakannya kembali.
  - Contoh analisis pola dan diskusi lebih lanjut tentang topik ini disajikan dalam Bab 8
- Siapa pun yang telah melakukan rekayasa kebutuhan pada lebih dari beberapa proyek PL memperhatikan bahwa problem tertentu muncul kembali di semua proyek dalam domain aplikasi tertentu
  - Pola analisis ini menyarankan solusi (misalnya, kelas, fungsi, perilaku) dalam domain aplikasi yang dapat digunakan kembali saat memodelkan banyak aplikasi

## 7.6 Negotiating Requirements

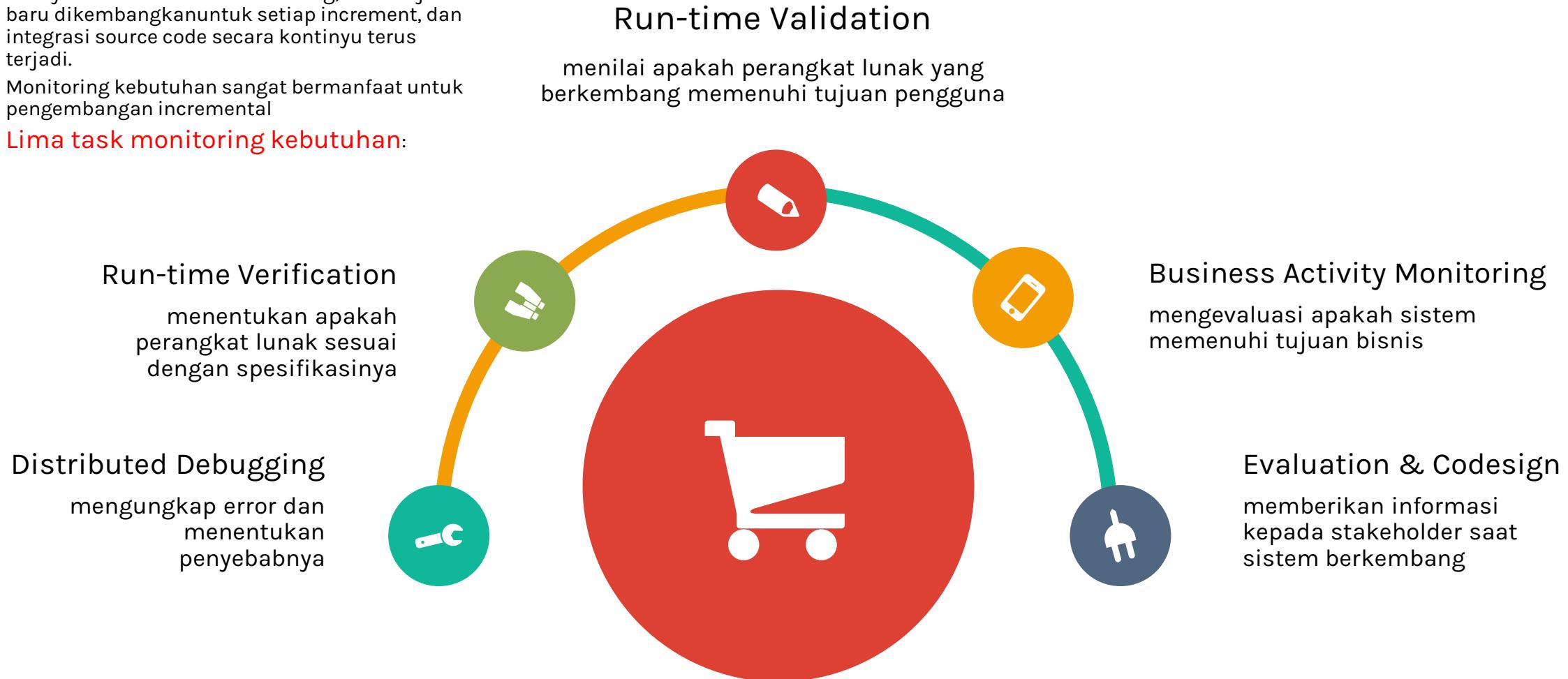
## 7.6 Negotiating Requirements (Negosiasi Kebutuhan)

- ◆ Task rekayasa kebutuhan (insepsi, elisitasim dan elaborasi) dilakukan untuk memperoleh detail kebutuhan pelanggan. Namun sayangnya sulit terpenuhi, sehingga diperlukan negosiasi.
- ◆ Negosiasi dilakukan terhadap 1 atau lebih stakeholder
  - ◆ Stakeholder diminta diminta untuk menyeimbangkan fungsionalitas, kinerja, dan karakteristik produk atau sistem lainnya terhadap biaya dan waktu pengiriman.
- ◆ Maksud dari negosiasi ini adalah untuk mengembangkan rencana proyek yang memenuhi kebutuhan stakeholder sementara pada saat yang sama mencerminkan kendala dunia nyata (misalnya, waktu, orang, anggaran) yang terdapat pada tim PL
- ◆ Negosiasi yang baik memberikan hasil “win-win”
  - ◆ Stakeholder “win” dengan mendapatkan sistem atau produk yang paling memenuhi kebutuhan mereka,
  - ◆ Pengembang “win” dengan bekerja untuk anggaran dan tenggat waktu yang realistik dan dapat dicapai
- ◆ ‘Handshaking’ merupakan salah satu cara untuk mendapatkan hasil “win-win”
  - ◆ tim perangkat lunak mengusulkan solusi untuk kebutuhan, menjelaskan dampaknya, dan mengomunikasikan niat mereka kepada perwakilan pelanggan
  - ◆ Perwakilan pelanggan meninjau solusi yang diusulkan, dengan fokus pada fitur yang hilang dan mencari klarifikasi kebutuhan baru
- ◆ Persyaratan ditentukan cukup baik jika pelanggan menerima solusi yang diusulkan.
- ◆ ‘Handshaking’ cenderung meningkatkan identifikasi, analisis, dan pemilihan varian dan mendorong negosiasi ‘win-win’

## 7.7 Requirements Monitoring

## 7.7 Requirements Monitoring (Monitoring Kebutuhan)

- Pengembangan incremental umum dilakukan. Artinya use case akan berkembang, kasus uji baru dikembangkan untuk setiap increment, dan integrasi source code secara kontinyu terus terjadi.
- Monitoring kebutuhan sangat bermanfaat untuk pengembangan incremental
- **Lima task monitoring kebutuhan:**



## 7.8 Validating Requirements

## 7.8 Validating Requirements (Validasi Kebutuhan)

- ◆ Ketika setiap elemen dari model kebutuhan dibuat, dilakukan pemeriksaan ketidakkonsistenan, kelalaian, dan ambiguitas.
- ◆ Pemeriksaan dilakukan juga untuk model proses agile di mana kebutuhan cenderung dituliskan sebagai user story dan/atau kasus uji.
- ◆ Kebutuhan yang diwakili oleh model diprioritaskan oleh stakeholder dan dikelompokkan dalam paket kebutuhan yang akan diimplementasikan sebagai increment perangkat lunak.

Review model kebutuhan menjawab pertanyaan-pertanyaan berikut:

- 1) Apakah setiap kebutuhan konsisten dengan tujuan keseluruhan untuk sistem atau produk?
- 2) Apakah semua kebutuhan telah ditentukan pada tingkat abstraksi yang tepat? Artinya, apakah beberapa kebutuhan memberikan tingkat detail teknis yang tidak sesuai pada tahap ini?
- 3) Apakah kebutuhan benar-benar diperlukan atau apakah itu mewakili fitur tambahan yang mungkin tidak penting untuk tujuan sistem?
- 4) Apakah setiap kebutuhan dibatasi dan tidak ambigu?
- 5) Apakah setiap kebutuhan memiliki atribusi? Artinya, apakah sumber (umumnya, individu tertentu) dicatat untuk setiap kebutuhan?

- 6) Apakah ada kebutuhan yang bertengangan dengan kebutuhan lain?
- 7) Apakah setiap kebutuhan dapat dicapai dalam lingkungan teknis yang akan menampung sistem atau produk?
- 8) Apakah setiap kebutuhan dapat diuji, setelah diterapkan?
- 9) Apakah model kebutuhan benar mencerminkan informasi, fungsi, dan perilaku sistem yang akan dibangun?
- 10) Apakah model kebutuhan telah "dipartisi" dengan cara yang secara progresif memperlihatkan informasi yang lebih rinci tentang sistem?
- 11) Apakah pola kebutuhan telah digunakan untuk menyederhanakan model kebutuhan? Apakah semua pola telah divalidasi dengan benar? Apakah semua pola konsisten dengan kebutuhan pelanggan?

## 7.9 Rangkuman

# 7.9 Rangkuman

## TASK REKAYASA KEBUTUHAN

- ◆ Dilakukan agar terbentuk fondasi kokoh untuk desain dan konstruksi
- ◆ Dilakukan saat aktivitas komunikasi dan aktivitas pemodelan yang telah ditetapkan untuk proses PL generic
- ◆ Tujuh aktivitas rekayasa kebutuhan—inception, elisitasi, elaborasi, negosiasi, spesifikasi, validasi, dan manajemen—dilakukan oleh anggota tim PL dan stakeholder produk



## INSEPSI & ELISITASI

- ◆ Stakeholder menetapkan kebutuhan masalah dasar, menentukan kendala proyek utama, dan mengatasi fitur dan fungsi utama yang harus ada agar sistem dapat memenuhi tujuannya.
- ◆ Informasi ini disempurnakan dan diperluas selama elisitasi—aktivitas pengumpulan persyaratan yang memanfaatkan pertemuan yang difasilitasi dan pengembangan skenario penggunaan (cerita pengguna)

# Rangkuman...

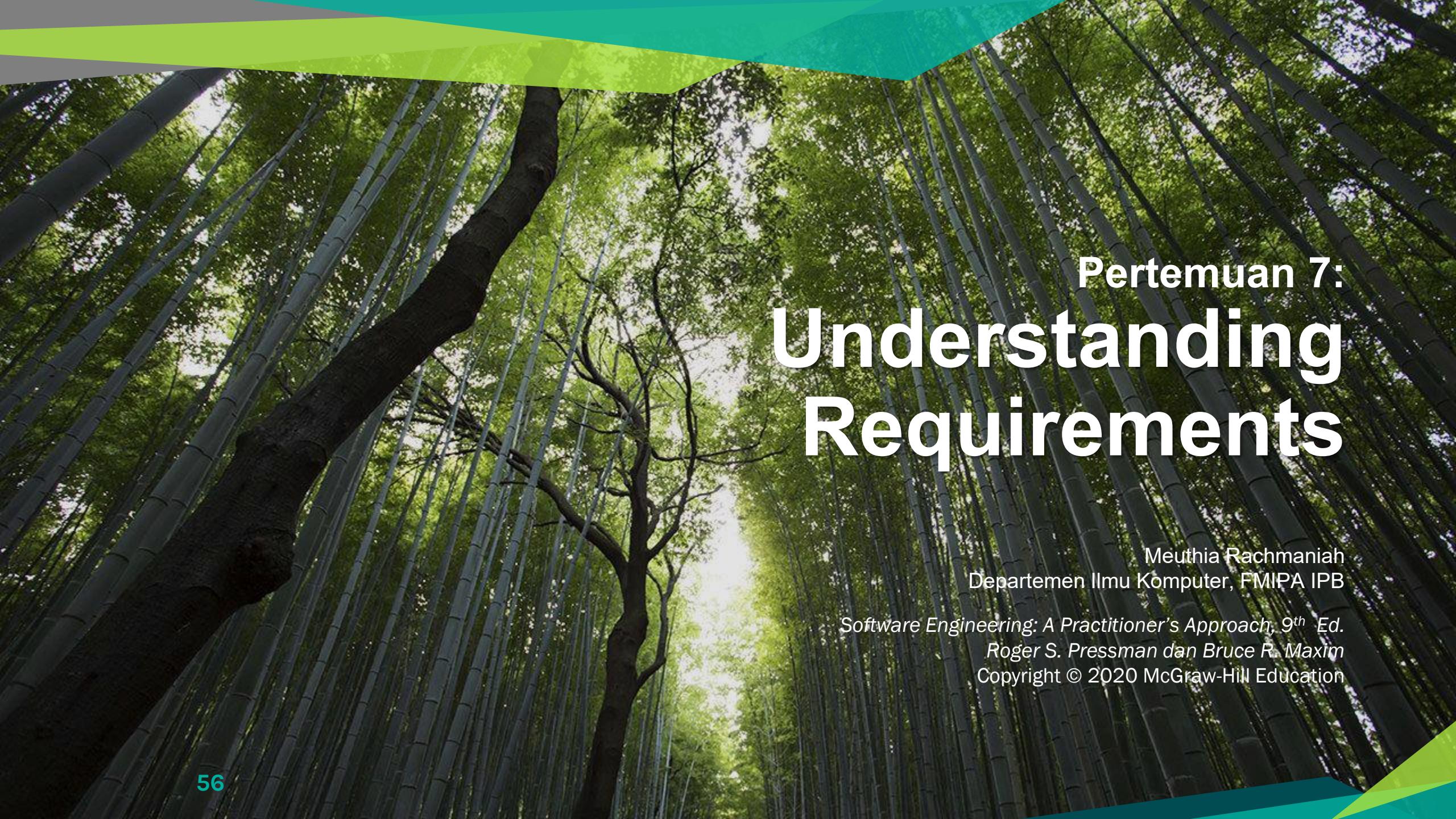
## ELABORASI

- ◆ Elaborasi memperluas kebutuhan dalam model kumpulan elemen berbasis skenario, berbasis aktivitas, berbasis class, dan perilaku.
- ◆ Model dapat mereferensikan pola analisis, karakteristik domain problem yang telah terlihat muncul kembali di berbagai aplikasi.



## NEGOSIASI & VALIDASI

- ◆ Saat kebutuhan telah diidentifikasi dan model persyaratan dibuat, tim PL dan stakeholder proyek lainnya menegosiasikan prioritas, ketersediaan, dan biaya relatif dari setiap kebutuhan.
- ◆ Maksud dari negosiasi ini adalah untuk mengembangkan rencana proyek yang realistik.
- ◆ Setiap kebutuhan perlu divalidasi terhadap kebutuhan pelanggan untuk memastikan bahwa sistem yang tepat akan dibangun



# Pertemuan 7: Understanding Requirements

Meuthia Rachmaniah  
Departemen Ilmu Komputer, FMIPA IPB

*Software Engineering: A Practitioner's Approach, 9<sup>th</sup> Ed.*  
Roger S. Pressman dan Bruce R. Maxim  
Copyright © 2020 McGraw-Hill Education