Rekayasa Perangkat Lunak

Silabus dan Aturan Perkuliahan

Priyanto
E-mail : priyanto@uny.ac.id
Mobile: 0811282609

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik Universitas Negeri Yogyakarta
2014

Produk Fisik

Produk Fisik

Produk Lojik

Rekayasa Perangkat Lunak

Tujuan Mata Kuliah:

Mahasiswa memahami teori dasar dan tahapan rekayasa perangkat lunak, dan menerapkan prinsip-prinsip teori dasar ini pada proyek pengembangan perangkat lunak.

Silabus (1-5)

Pendahuluan	• Silabus dan Peraturan perkuliahan • Meluruskan salah kaprah RPL • Pengantar RPL
Software Engineering	The nature of software, the unique nature of WebApps, the software process (communication, planning, modeling; construction, deployment)
Process Model	The waterfall model, incremental process, RAD model, evolutionary process models

Silabus (2-5)

Software Engineering Practice	• Core principles. • Principles: communication, planning, modeling; construction, deployment
Requirements Modeling	• Requirements analysis, data modeling, class-based modeling • Flow-oriented modeling, behavior modeling

Silabus (3-5)

Design Concept	• Design concept: Abstraction, modularity, information hiding, fuctional independence (coupling and cohesion) • The design model: data design elements, architectural design elements, interface design elements, component-level design elements, deployment-level design elements.
Architectural Design Concept	• Architectural style • Architectural mapping using data flow: transform flow, transaction flow, transform mapping, transaction mapping.

Silabus (4-5)

Component-Level Design	• Component: an object-oriented view, the traditional view • Designing class-based components • Designing traditional components
User Interface Design	The golden rules, interface design steps

Silabus (5-5)

Software Testing Strategies	• A strategic approach to software testing • Test strategies: unit testing, Integration testing • Validation testing: Alpha and Beta testing • System testing: recovery testing, security testing, stress testing, performance testing, deployment testing.
Testing Conventional Application	• Software testing fundamentals • Whitebox testing: basis path testing; control structure testing, blackbox testing • Blackbox testing

Buku Acuan

Pressman, Roger S (2010). *Software Engineering, A Practitioner'a Approach*. SeventhEdition. Singapore: McGraw-Hill Education.

Buku Acuan

Booch, Grady (1991). *Object Oriented Design*. California: The Benjamin/Cummings Publishing Co.

Yourdon, Edward (1989). *Modern Structured Analysis*. New Jersey: Prentice-Hall, Inc.

Lainnya:

- Artikel Software Engineering
- Studi kasus diambilkan dari buku lain atau membuat sendiri

Hybrid Learning

Hybrid Learning

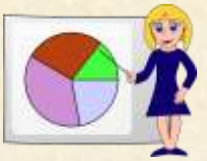
- Tatap muka (sebagian besar)
- *E-learning* (sebagian kecil).

- QUIZ
  - Quiz di kelas
  - Quiz di E-learning, akan dibuka pada waktu tertentu
- UTS dan UAS tetap dilakukan secara *offline* (di kelas) sesuai jadwal.
- Nilai akhir adalah gabungan antara Quiz (kelas & *e-learning*), UTS, Tugas, dan UAS.

Tugas

Tulisan

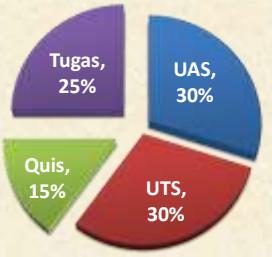
Presentasi Kelompok



E-Learning

- Semua mahasiswa Kelas E dan F wajib mendaftarkan diri di *e-learning*
- Identitas:
  - Nama depan: *Nama Panggilan*
  - Nama belakang: NIM
- Enrolment Key sesuai kelas
  - rple → Kelas E
  - rplf → Kelas F

Evaluasi




Terima Kasih

Perangkat Lunak & Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak

Priyanto
E-mail : priyanto@uny.ac.id
Mobile: 0811282609

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Rekayasa Perangkat Lunak

Tujuan Mata Kuliah:

Mahasiswa memahami teori dasar dan tahapan rekayasa perangkat lunak, dan menerapkan prinsip-prinsip teori dasar ini pada proyek pengembangan perangkat lunak.

Meluruskan RPL

Pergeseran Makna RPL

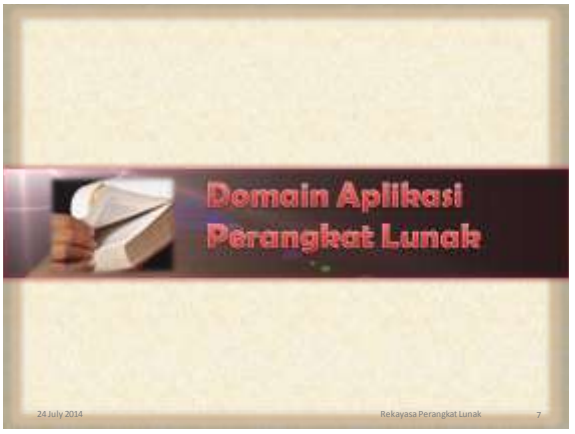
Rekayasa Perangkat Lunak (Software Engineering), sedikit mengalami pergeseran makna di realita dunia pendidikan maupun kurikulum Teknologi Informasi (TI) di Indonesia.

RPL di SMK

- SMK di Indonesia membuka jurusan Program Keahlian Rekayasa Perangkat Lunak
- Secara kurikulum hanya mengajari bahasa C atau Java (mungkin lebih pas disebut jurusan pemrograman komputer)
- Pertemuan ini berusaha meluruskan salah kaprah yang terjadi tentang RPL berdasarkan kesepakatan, acuan, dan standar yang ada di dunia internasional.

Sejarah RPL

- Sejarah munculnya Rekayasa Perangkat Lunak sebenarnya dilatarbelakangi oleh adanya krisis perangkat lunak (software crisis) di era tahun 1960-an.
- Krisis perangkat lunak merupakan akibat langsung dari lahirnya komputer generasi ke 3 yang canggih, ditandai dengan penggunaan Integrated Circuit (IC) untuk komputer.
- Performansi hardware yang meningkat, membuat adanya kebutuhan untuk memproduksi perangkat lunak yang lebih baik.



System Software

System Software. Adalah kumpulan program yang ditulis untuk melayani program-program lain.

- Compilers, editors, dan file management utilities.
- Operating system components, driver, networking SW

Karakteristik: berinteraksi kuat dengan hardware

Application Software

Program *stand-alone* untuk menyelesaikan kebutuhan bisnis yang spesifik.

Contoh: point-of-sale, transaction processing, real-time manufacturing process control.

Embedded Software

SW yang menempel di dalam produk atau sistem yang digunakan untuk mengontrol sistem.

- Terdapat pada produk-produk cerdas: photo copy, microwave oven, mesin cuci, otomotif (fuel control, dashboard display, etc).
- Software memiliki tugas yang khusus dan terbatas, tersimpan di dalam ROM yang digunakan untuk mengontrol perangkat keras

Product-line Software

Dirancang untuk keperluan khusus, untuk pasar terbatas atau masal.

- Untuk pasar terbatas: inventory control
- Untuk pasar masal: Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, aplikasi finansial personal dan bisnis, dsb.

Web–applications Software

"Web apps" adalah *network centric software*, dikembangkan dalam lingkungan komputasi yang canggih tidak hanya menyediakan fitur standalone, fungsi komputasi, tetapi juga terintegrasi dengan database perusahaan dan aplikasi lain.

- Contoh: SIAKAD & SIKEU di UNY, webmail, dll.

Artificial Intelligent Software

Software menggunakan algoritma nonnumerik untuk menyelesaikan permasalahan yang kompleks yang tidak mengikuti komputasi atau algoritma yang *straighrforward*.

- Contoh: robotik, sistem pakar, pengenalan pola (gambar dan suara), game

Sifat Unik WebAPps

- Network intensiveness → Intranet dan/atau Internet
- Concurrency → Jumlah pengguna yang banyak
- Unpredictabeload → jumlah pengguna sulit ditebak
- Performance → menunggu terlalu lama dalam proses
- Availability → 24/7/365

Sifat Unik WebAPps

- Content sensitive → kualitas dan sifat dasar estetis konten menjadi determinan kualitas.
- Continuous evolution → konten diperbaharui menit demi menit.
- Security → sangat diperlukan karena akses tersedia melalui jaringan
- Aesthetics → estitika menjadi kesuksesan desain teknis

Software Process

- Proses adalah kumpulan aktivitas, aksi, dan tugas yang dilakukan ketika produk pekerjaan diiciptakan
- Aktivitas : komunikasi dengan pemangku kepentingan
- Aksi: desain arsitektur
- Tugas: pengujian yang menghasilkan hasil nyata

Process Framework

- Communication
  - Komunikasi dan kolaborasi dengan Customer dan pemangu kepentingan lain)
  - Requirements gathering
- Planning: Estimating, scheduling, tracking
- Modeling: Analysis, Design
- Construction: Code generation & test
- Deployment: delivery, support, feedback

Studi Kasus

Media Pembelajaran Penggunaan Multi Tester

- Communication
- Planning
- Modeling
- Construction
- Deployment

Studi Kasus

Media Pembelajaran Hukum Ohm

- Communication
- Planning
- Modeling
- Construction
- Deployment

Model-Model Proses

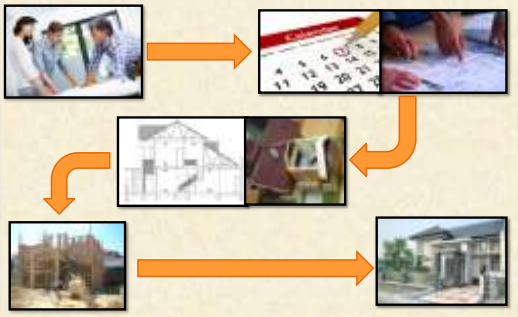
Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id
Mobile: 0811282609

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Komunikasi (?)

Diskusi

Diskusikan Proses Ini



Genenric Process Model

Generic Process Framework

- **Communication**
- **Planning**
- **Modeling**
- **Construction**
- **Deployment**

GPF→ Communication

Melibatkan komunikasi dan kolaborasi yang berat dengan kustomer (dan stkeholders lain) dan mencakup pengumpulan kebutuhan dan aktivitas yang terkait.



GPF → Communication

Project Initiation

- Menetapkan kebutuhan elemen seluruh sistem → menghimpun kebutuhan sistem secara global dengan disertai sedikit analisis dan rancangan secara umum.
- SW selalu merupakan bagian dari sistem yang besar.
- SW akan berinteraksi dengan perangkat keras, manusia, dan basis data.

GPF→ Communication

Requirement Gathering

- Tahap ini melakukan analisis kebutuhan untuk PL yang akan dibuat, hasilnya adalah spesifikasi PL
- Agar menghasilkan spesifikasi yang benar, maka seorang Analis (software engineer) harus memahami secara rinci fungsi, kinerja, dan antar muka yang diperlukan
- Spesifikasi ini dibahas antara analis dan pemakai

GPF → Planning

Menetapkan rencana kerja RPL. Menjabarkan tugas teknis yang akan dilakukan, resiko, sumber daya yang diperlukan, hasil kerja, dan jadwal kerja

GPF → Modeling

Membuat model sehingga antara **pengembang** dan **kustomer** memperoleh pemahaman yang lebih baik pada kebutuhan SW dan desain yang memenuhi kebutuhan tersebut

GPF→ Modeling

Software Requirement Analysis

- melakukan analisis kebutuhan untuk PL yang akan dibuat, hasilnya adalah spesifikasi PL
- Agar menghasilkan spesifikasi yang benar, maka seorang Analis (software engineer) harus memahami secara rinci fungsi, kinerja, dan antar muka yang diperlukan
- Spesifikasi ini dibahas antara analis dan pemakai.

GPF → Modeling

Design

- difokuskan pada tiga bagian utama SW, yaitu: Struktur Data, Arsitektur SW, dan Lojik program.
- Proses perancangan dilakukan berdasar pada spesifikasi tahapan sebelumnya.

GPF → Modeling

- BAGAIMANA merancang struktur data
- BAGAIMANA mengimplementasikan fungsi sebagai arsitektur software
- BAGAIMANA detail prosedur diimplementasikan
- BAGAIMANA disain akan diterjemahkan ke bahasa pemrograman
- BAGAIMANA testing dilaksanakan

GPF → Construction

Aktivitas ini mengkombinasikan:

- **Pengkodean** program (manual atau otomatis) dan
- **Pengujian** yang diperlukan untuk menemukan kesalahan-kesalahan di dalam program.

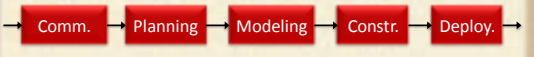
Generic Process Framework → Construction

- **Code:** Proses menterjemahkan rancangan PL menjadi program komputer.
- **Test:** pengujian lojik program, untuk
  - meyakinkan bahwa seluruh statemen sudah benar dan
  - meyakinkan bahwa masukan tertentu akan menghasilkan keluaran tertentu.

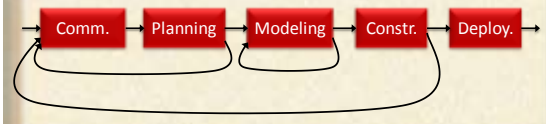
GPF → Deployment

SW sebagai entitas komplet atau sebagai tahapan komplet parsial dikirim kepada kustomer yang mengevaluasi produk dan memberikan feedback berdasar pada evaluasi

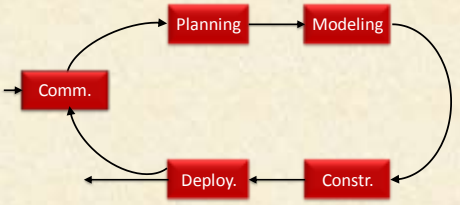
Linear Process Flow



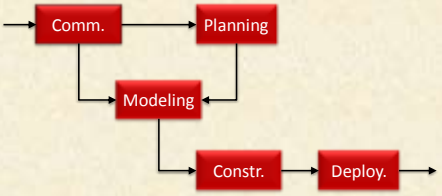
Iterative Process Flow



Evolutionary Process Flow



Parallel Process Flow



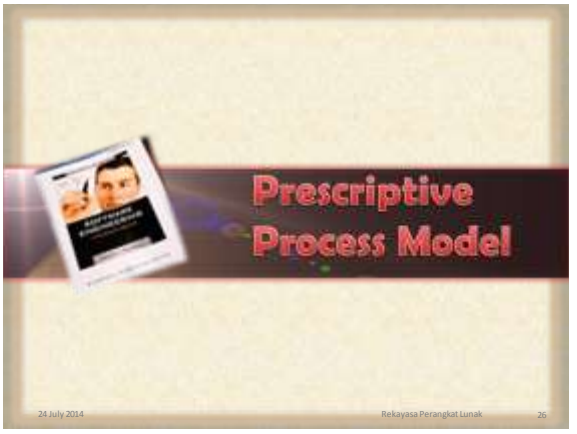
Pengukuran Proses

Proses harus diukur untuk menjamin bahwa proses memenuhi seperangkat kriteria proses dasar untuk kesuksesan RPL

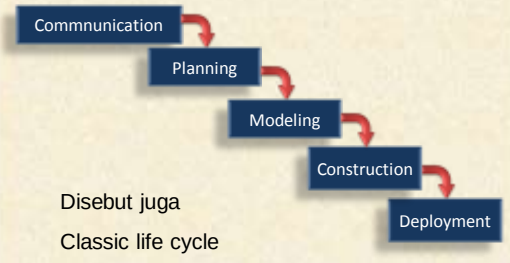
Pengukuran Proses

Pendekatan Pengukuran Proses

- Standard CMMI Assessment Method for Process Improvement (SCAMPI)
- CMM-Based Appraisal for Internal Process Impprovement (CBA IPI)
- SPICE (ISO/IECI 15504)
- ISO 9001:2000 for Software → paling banyak dipakai



Waterfall Model



Disebut juga

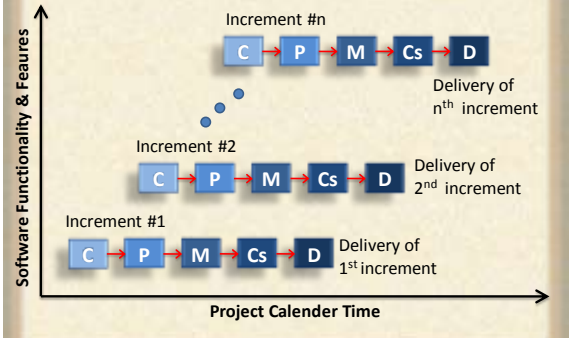
Classic life cycle

Waterfall Model

Kelemahan:

- Terdapat banyak problem, apabila selama pengembangan sering terjadi penambahan
- Pada tahap awal pengembangan PL, sangat sukar bagi para pemakai untuk menjabarkan kebutuhannya secara rinci. Padahal kebutuhan dari pemakai merupakan landasan untuk tahapan berikutnya.
- Pemakai harus sabar untuk dapat melihat produk awal dari program. Kesalahan yang besar baru tampak saat produk awal program dihasilkan, dan apabila hal ini terjadi maka proses pengembangan PL harus dilakukan dari awal.

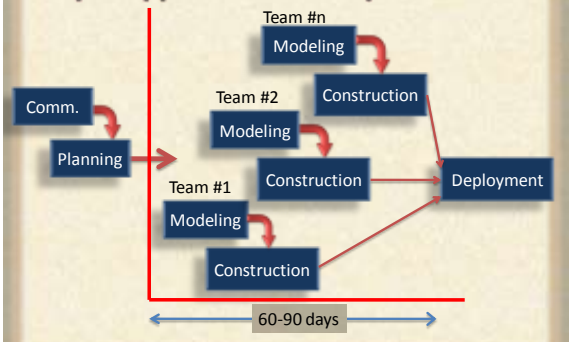
Incremental Model



Incremental Model

- Mengkombinasikan elemen model waterfall yang diterapkan dengan cara iteratif
- Berfokus pada delivery dari produk operasional setiap increment
- Delivery dari first increment digunakan untuk perencanaan second increment, dst.

Rapid Application Development Model



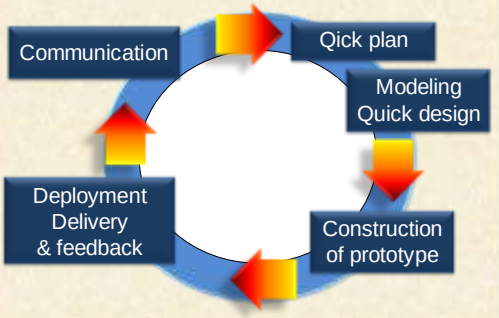
RAD Model

- RAD model adalah adaptasi high speed dari linear sequential model (LSM).
- LSM yang menekankan pada siklus pengembangan yang sangat pendek (60-90 hari)
- Menggunakan *component based construction*, komponen program yang *reusable*.
- Planning sangat penting karena melibatkan banyak tim

RAD Model

Tidak tepat untuk sistem yang memiliki resiko terlalu tinggi: aplikasi baru menggunakan teknologi baru atau software baru yang memerlukan interoperabilitas tinggi dengan program yang sudah ada.

Prototyping Model → Evolutionary



Prototyping

Pembuat software membuat MODEL dari SW yang akan dibuat. Model dapat berbentuk:

- Prototipe kertas atau model berbasis komputer yang menjelaskan bagaimana interaksi antara pemakai dan komputer.
- Prototipe yang mengimplementasikan beberapa bagian fungsi dari PL yang sesungguhnya. Dengan cara ini pemakai akan lebih mendapatkan gambaran tentang program yang akan dihasilkan, sehingga dapat menjabarkan lebih rinci kebutuhannya.
- Menggunakan SWyang sudah ada. Seringkali pembuat PL memiliki beberapa program yang sebagian dari program tersebut mirip dengan program yang akan dibuat.

Prototyping

Cocok untuk kondisi seperti apa?

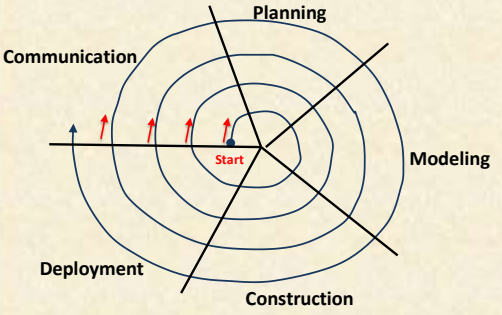
- Sering kali pemakai dapat mendefinisikan secara rinci tujuan dan penggunan software yang dibutuhkan, tetapi tidak dapat mendefinisikan secara rinci kebutuhan masukan, pengolahan, dan keluarannya.
- Di sisi lain, pembuat software tidak tidak memiliki kepastian akan hal tersebut.

Prototyping

Beberapa Permasalahan:

- PL yang dibuat merupakan pengembangan dari model, sehingga kualitasnya rendah (program terlalu besar, sulit dimodifikasi, tidak terdokumentasi dengan jelas). Untuk itu pembuat harus menulis ulang program yang dihasilkan agar berkualitas tinggi.
- Untuk mempercepat pembuatan prototipe, terkadang menggunakan operating system, bahasa pemrograman, dan algoritma yang kurang tepat, dengan pertimbangan pembuat program sudah memahami. Setelah proses pengembangan pembuat program lupa mengapa memilih algoritma yang kurang tepat tersebut.

Spiral Model → Evolutionary



Spiral Model

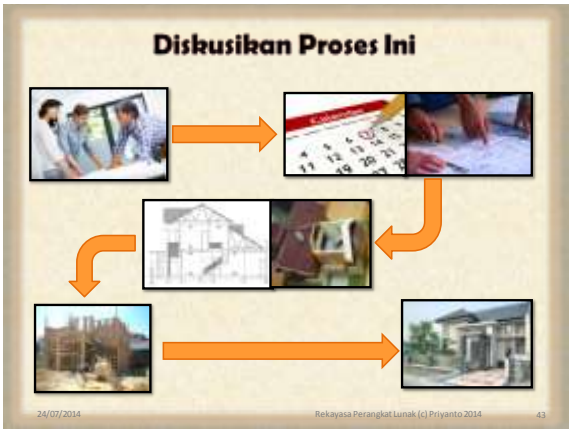
- Proses pengembangan SW yang evolusioner, mengkombinasikan sifat iteratif dan aspek sistematis waterfall
- Dimensi radial menunjukkan makin lama makin lengkap program yang dibangun.
- Dimensi angular menunjukkan kemajuan dalam menyelesaikan siklus spiral.
- Setiap siklus berisi urutan yang sama
- Cocok untuk membangun sistem yang besar

No Silver Bullet

Model mana yang paling baik? TIDAK ADA

Tergantung sistem yang dikembangkan

Sangat dimungkiankan mengunakan kombinasi model untuk memperoleh efisiensi waktu dan hasil yang optimal


Diskusikan Proses Ini
24/07/2014
Rekayasa Perangkat Lunak (c) Priyanto 2014
43


Terima Kasih

Software Engineering Practice

Rekayasa Perangkat Lunak

Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Prinsip yang Memandu Proses

- Cerdas
- Fokus pada kualitas untuk setiap langkah
- Siap untuk beradaptasi
- Bangun tim yang efektif
- Menetapkan mekanisme untuk komunikasi dan koordinasi
- Mengelola perubahan
- Ciptakan produk kerja yang memberikan nilai bagi orang lain

Prinsip yang Memandu Praktik

- Divide and conquer
- Paham dan menggunakan abstraksi
- Berusaha konsisten → dari awal sampai akhir
- Fokus pada transfer informasi
- Membangun SW yang menunjukkan mudulatitas
- Carilah pola → menangkap pengetahuan arsitektur yang baik yang kita pahami yang memenuhi kebutuhan pengguna
- Ingat bahwa seseorang akan memelihara SW.

Communication Principles

Communication

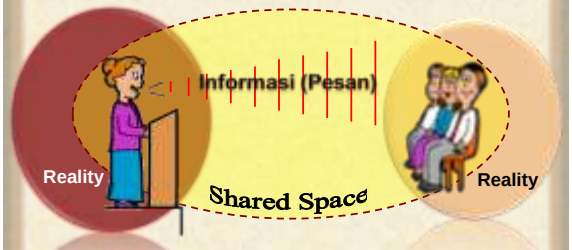
Commnunication → Planning → Modeling

Komunikasi Efektif:

- antar sesama teknis,
- dengan Customer dan stakeholders lain,
- dengan manajer proyek

adalah aktivitas yang PALING MENANTANG dihadapan software engineer

Komunikasi



Komunikasi Efektif apabila:
Pesan Pengirim = Pesan Penerima

? Di jalan, tanpa bicara
Apakah terjadi komunikasi?

? Di Layanan Photocopy, mereka berbicara.
Apakah terjadi komunikasi?

? Ibu dan bayi, mereka tidak berbicara.
Apakah terjadi komunikasi yang efektif?

Communication Principles (1-2)

- Dengarkan → dengarkan dengan baik dan SABAR
- Siapkan sebelum berkomunikasi → Luangkan waktu untuk memahami masalah
- Harus ada yang memfasilitasi komunikasi: Leader, perantara bila ada konflik, menjamin satu prinsip yang diikuti

Communication Principles (2-2)

- Komunikasi tatap muka paling baik
- Buat catatan dan dokumentasi keputusan
- Berisaha untuk berkolaborasi
- Tetap fokus dan modular dalam diskusi
- Jika tidak jelas, buat gambar
- Negosisasi bukan kontes atau permainan. Yang paling baik semua menang

Planning Principles

Planning Practice Principles (1-2)

- Pahami lingkup proyek
- Libatkan kustomer pada aktivitas perencanaan
- Perencanaan adalah itetatif
- Pertimbangkan resiko saat membuat planning
- Tetap realistis → lelah dan salah, menjadi pertimbangan

Planning Practice Principles (2-2)

- Aturlah granularitas saat membuat rencana
- Tentukan bagaimana menjamin kualitas
- Uraikan bagaimana mengakomodasi perubahan
- Sering lakukan pelacakan pada perencanaan, atur kembali bila diperlukan

Modeling Principles

- Tujuan utama Tim SW adalah membangun SW, bukan menciptakan model.
- Buatlah model yang up-to-date; model harus lebih mudah dan lebih cepat untuk membangun SW
- Usahakan membuat model yang paling sederhana

Requirements Modeling Principles

- Domain informasi dari permasalahan harus ditunjukkan dan dipahami → data flow (dari end user, sistem lain, atau divais eksternal)
- Tentukan fungsi yang dilakukan SW → keuntungan langsung pada end user.
- Perilaku software terhadap *event* eksternal
- Model yang menggambarkan informasi, fungsi, dan perilaku, harus dipartisi secara berlapis
- Pekerjaan analisis harus bergerak dari informasi esensial menuju detil implementasi

Design Modeling Principles (1-2)

- Desain harus bisa dilacak ke model analisis
- Selalu mempertimbangkan arsitektur sistem
- Desain data sama penting dengan desain fungsi proses
- Antarmuka (antar komponen) harus dirancang secara cermat
- Desain antarmuka user harus sejalan dengan kebutuhan end user

Design Modeling Principles (2-2)

- Desain level komponen harus *functionally independent*
- Komponen harus *loosely coupled* dengan yang lain dan lingkungan eksternal
- Model harus mudah dipahami
- Desain harus dikembangkan secara iteratif. Untuk setiap iterasi, desainer harus berusaha untuk kesederhanaan

Coding Principles

- Batasi algoritma dengan mengikuti praktik pemrograman terstruktur
- Pilih struktur data yang memenuhi keinginan desain
- Pahami arsitektur SW, buatlah antarmuka yang konsisten
- Peliharalah logika kondisional sesedehana mungkin
- Pilih nama variabel yang memiliki arti
- Buat layout visual (indentasi, baris kosong) untuk membantu pemahaman

Testing

- Testing adalah proses eksekusi program yang bertujuan untuk menemukan error.
- Test case yang baik yang memiliki probabilitas tinggi untuk menemukan Error yang belum ditemukan
- Test yang sukses, yang menemukan error belum ditemukan

Testing Principles

- Semua test harus mengacu pada customer requirements
- Test harus direncanakan jauh sebelum testing dimulai
- The **Pareto principle** (**aturan 80-20)** : 80% error disebabkan oleh 20% dari program
- Testing harus dimulai “in the small” menuju testing “in the large”
- Testing mendalam tidak mungkin → tetapi dalam pengujian, setiap *path* harus dilalui.

Deployment Principles

- Ekspektasi customer terhadap SW harus dikelola
- Paket SW (executable SW, file data, dokumen pendukung) harus dirakit dan diuji
- Rezim pendukung harus harus ditetapkan sebelum SW disampaikan
- Menyiapkan bahan instruksional → bahan pelatihan, petunjuk troubleshoting
- Penyampaian SW dengan banyak *bug* dengan janji akan diperbaiki pada rilis berikutnya, adalah KESALAHAN.

Pengembangan SW Pembelajaran

- Customer: Kemenkominfo & JICA
- Konsultan (Konsursium 3 perusahaan)
- Developer: PT Compnet, Jakarta
- End-users: Guru SD dan SMP

Requirement Modeling

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Memahami kebutuhan suatu masalah adalah salah satu tugas yang paling sulit yang dihadapi seorang software engineer

Understanding Requirements

Menggali Requirements

- Pertemuan dilakukan dan dihadiri oleh software engineer dan customers
- Aturan untuk persiapan dan partisipasi ditetapkan
- Fasilitator (customer, pengembang, atau orang luar) untuk mengontrol pertemuan
- "Mekanisme definisi" (lembar kerja, flip chart, atau stiker dinding, chat room, atau forum virtual) digunakan
- Tujuannya adalah:
  - untuk mengidentifikasi masalah
  - mengusulkan elemen dari solusi
  - menegosiasikan pendekatan yang berbeda, dan
  - menentukan satu set awal persyaratan solusi

Validating Requirements - 1

- Apakah setiap kebutuhan konsisten dengan tujuan keseluruhan sistem /produk?
- Apakah semua persyaratan telah ditetapkan pada tingkat abstraksi yang tepat? Artinya, lakukan beberapa persyaratan menyediakan tingkat detail teknis yang tidak tepat pada level ini?
- Apakah kebutuhan benar-benar diperlukan ataukah merupakan fitur add-on yang mungkin tidak penting untuk tujuan sistem?
- Apakah setiap kebutuhan dibatasi dan tidak ambigu?
- Apakah setiap persyaratan memiliki atribusi? Artinya, apakah sumber (umumnya, individu tertentu) mencatat untuk kebutuhan masing-masing?
- Apakah ada konflik dengan persyaratan persyaratan lainnya?

Validating Requirements - 2

- Apakah setiap kebutuhan dapat dicapai dalam lingkungan teknik yang menjadi rumah sistem atau produk?
- Apakah setiap kebutuhan dapat diuji, setelah diimplementasi?
- Apakah model kebutuhan mencerminkan informasi, fungsi dan perilaku sistem yang akan dibangun?
- Apakah model requirements telah "dipartisi" dengan cara semakin lebih rinci tentang sistem?
- Apakah pola kebutuhan telah digunakan untuk menyederhanakan model requirements.
- Apakah semua pola telah divalidasi dengan benar?
- Apakah semua pola konsisten dengan requirements pelanggan?

Requirement Analysis

Requirement modeling menghasilkan satu atau lebih dari jenis model berikut:

- Scenario based model → requirements dari dari titik pandang berbagai "aktor" sistem
- Data models → menggambarkan permasalahan dari domain informasi
- Class-oriented models → menggambarkan OO classes (atribut & operasi)
- Flow-oriented models → menggambarkan elemen fungsi sistem dan bagaimana mentransformasi data dalam sistem
- Behavioral model → menggambarkan perilaku software terhadap "event" eksternal.

Requirement Modeling: Data Model

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik, Universitas Negeri Yogyakarta
2014

1

Dunia Mini

Database merepresentasikan beberapa aspek dunia nyata, sering kali disebut **dunia mini**. Perubahan pada dunia mini direfleksikan dalam database.

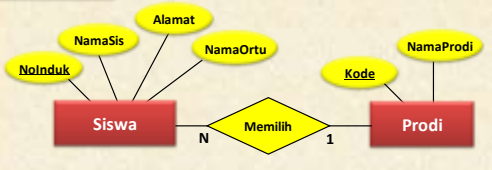
Abstraksi Data

Tiga Level Abstraksi Data

View Level	**Model data konseptual** (*high-level*)
Logical Level	**Model data implementasi** (antara konseptual dan fisik)
Physical Level	**Model data fisik** (*low-level*)

Model Data Konseptual

View Level
Logical Level
Physical Level



Model data yang menyerupai bagaimana pemakai menerima/melihat data

Model Data Implementasi

View Level
Logical Level
Physical Level

Primary Key (PK) Foreign Key (FK)

NoInduk	NamaSis	Alamat	NamaOrtu	KodeProdi

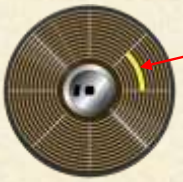
KodeProdi	NamaProdi

Primary Key (PK)

Model data yang dapat diketahui oleh end user tetapi tidak terlalu jauh dengan bagaimana data disimpan di dalam komputer

View Level
Logical Level
Physical Level

Model Data Fisik



NIM : char[10]
Nama : Char[20]
Alamat : Char[50]
NamaOrtu : Char[20]

Model data yang mendeskripsikan bagaimana data disimpan di dalam media penyimpan data.

Pemodelan Data menggunakan Diagram Entity Relationship

Konsep Model Entity-Relationship (ER)

- **Entitas**
  - sesuatu dalam dunia nyata dengan keberadaan yang independen
  - dapat diidentifikasi secara unik.
- **Tipe Entitas**→ sekelompok entitas yang memiliki atribut sama.
- **Atribut** → properti yang mendeskripsikan tipe entitas

Entitas

Evi
12000111
Jl. Adisucipto, Yogyakarta

Edi
12000112
Jl. Pahlawan, Purworejo

Emi
12000113
Jl. Kaliurang, Sleman

Siapa Mereka?

Entitas, Tipe Entitas, Atribut

Entitas

Evi
12000111
Jl. Adisucipto, Yogyakarta

Edi
12000112
Jl. Pahlawan, Purworejo

Emi
12000113
Jl. Kaliurang, Sleman

Tipe Entitas

Mahasiswa

Atribut

Nama
NIM
Alamat

Simbol ER

Tipe Entitas (entity type)

Tipe Relasi (relationship type)

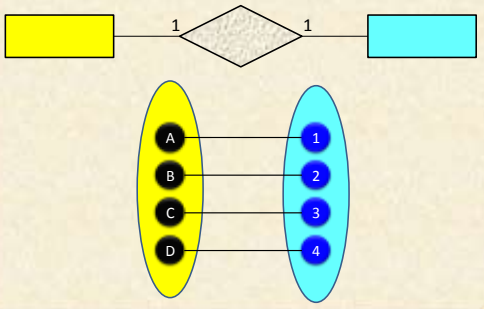
Tipe Entitas Lemah
(weak entity type)

Atribut

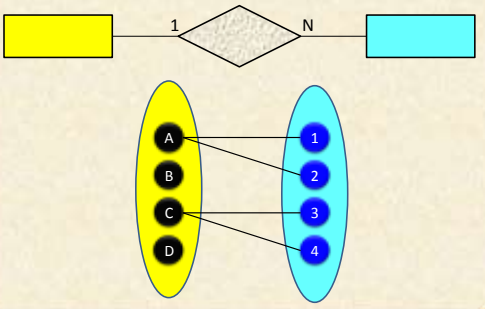
Atribut kunci

Atribut turunan

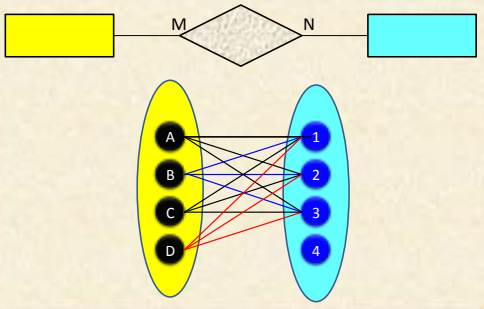
Relasi 1:1



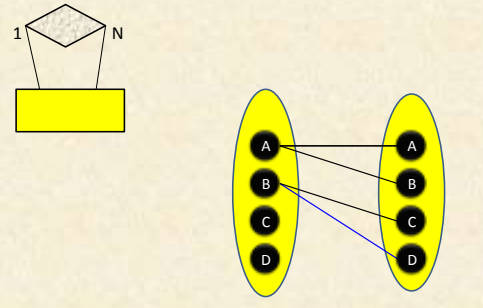
Relasi 1:N



Relasi M:N



Relasi Rekursif



Mapping ER ke Relational

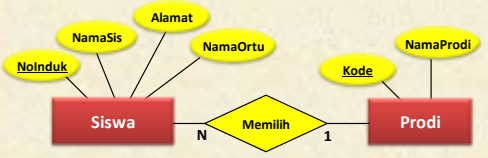
- Setiap entity type dibuat relational (tabel), pilih Key sebagai primary key (pk). Masukkan semua atribut kecuali multivalue.
- Setiap weak entity type dibuat relational. Masukkan semua atrribut kecuali multivalue, tambahkan primary key relasi Strong Entity Owner sebagai atribut. Key = primary key + partial key.
- Untuk binary relationship type 1:1 yang memiliki atribut, masukkan atribut ke entity type dentan total participation constraint.
  Bisa juga dibuat satu tabel baru dengan memasukkan semuak key dari kedua entity type.

Mapping ER ke Relational

- Untuk binary relationship type 1:N (non weak entity type), masukkan key entiity sisi 1 ke sisi N sebagai foreign key (fk).
- Untuk binary relationship type M:N buat tabel baru dengan pk dari kedua pk entity type-nya, masukkan semua atribut relationship tersebut ke tabel.
- Untuk setiap multivalue attribute buat tabel baru, dimana key-nya merupakan gabungan dari atribut tersebut dengan pk entity type → bisa diperlakukan sebagai relationship type M:N
- Untuk *n-ary* relationship, buat tabel baru dengan key merupakan gabungan dari pk entity type tersebut. Masukkan atribut ke tabel.

Contoh & Studi Kasus

Model Data Konseptual



Model Data Implementasi

Primary Key (PK) Foreign Key (FK)

NoInduk	NamaSis	Alamat	NamaOrtu	KodeProdi

KodeProdi	NamaProdi

Primary Key (PK)

Requirement Modeling & Design Modeling: Functional Model

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik, Universitas Negeri Yogyakarta
2014

1

Pemodelan Fungsional & Aliran Informasi

- Pemodelan fungsional adalah bagian dari *Structured Analysis* (SA).
- SA dipolulerkan oleh De Marco tahun 1979
- SA dikembangkan untuk sistem real-time oleh ward dan Melor tahun 1985.
- SA diawali dengan teknik pemodelan informasi
- Sistem berbasis komputer direpresentasikan dengan transformasi informasi

DFD adalah Model Fungsional

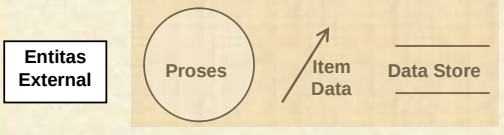
Data Flow Diagram (DFD), melayani dua fungsi:

- Memberikan petunjuk bagaimana data ditransformasikan dalam sistem
- Menggambarkan fungsi (dan sub fungsi) yang mentransformasikan aliran data (*data flow*)
- Deskripsi setiap fungsi pada DFD ada pada spesifikasi proses (*process specification*)

DFD

- DFD adalah grafik yang menggambarkan aliran informasi dan transformasi yang ditunjukkan dengan perpindahan data dari input ke output.
- DFD digunakan untuk merepresentasikan sistem atau software pada beberapa level abstraksi.
- DFD dapat dipartisi menjadi level yang menunjukkan penambahan aliran informasi dan detail fungsi.

Notasi DFD



Entitas Eksternal:

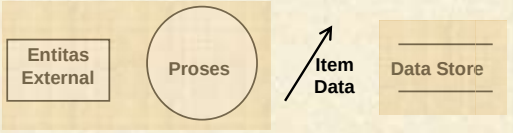
hardware, manusia, program, dsb. Yang memberikan atau menerima informasi

Notasi DFD

Proses = Transformasi:

- terdiri dari perbandingan logika tunggal sampai algoritma numerik yang kompleks

Notasi DFD



Item data:

- data yang masuk ke dan keluar dari proses

Notasi DFD

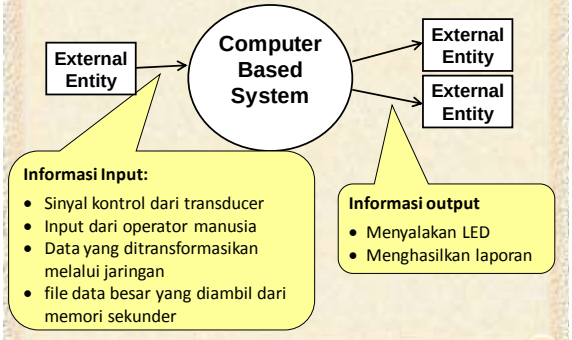
Data Store:

tempat menyimpan informasi yang dapat dipakai oleh banyak proses (buffer atau relational database)

Context Model (Context Diagram)

DFD level 0 disebut *fundamental system model* atau *context model*: menggambarkan seluruh elemen software sebagai **bulatan tunggal** dengan input dan output yang ditunjukkan tanda panah.

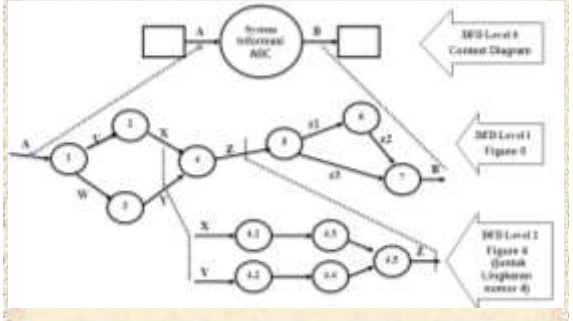
Context Model (Context Diagram)



Levelisasi DFD

- DFD dipartisi menjadi beberapa level
- Pemartisian berakhir apabila setiap bulatan sudah bisa diimplementasikan menjadi **Function** atau **Precedure**.

Levelisasi DFD



Sifat DFD

- Trasform flow
- Transaction flow

Contoh & Studi Kasus

Contoh Sangat Sederhana

Menghitung Tegangan pada Hukum Ohm
V = I * R

Tujuan:
Memberi gambaran utuh DFD sampai Program

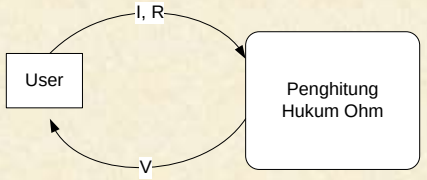
DFD: Transform Flow

Analysis Modeling

Functional Model

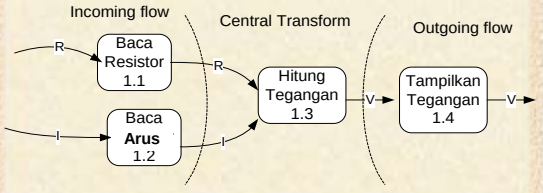
- Akan menghasilkan DFD (*Data Flow Diagram*)
  - DFD terdiri dari DFD level 0, level 1, sampai level n
  - Pemecahan DFD sampai level berapa (n), sampai mudah diimplementasikan dalam program.

DFD Level 0



DFD Level 1

DFD Level 2



Design Modeling

Design Modeling

- Tahap Desain Modeling akan menghasilkan DFD yang lebih halus dan Structure Chart
  - Apabila DFD sudah cukup, langsung dibuat structure chart-nya
  - Pada tahap ini dibuat **spesifikasi proses** yang mendeskripsikan setiap **bulatan proses** pada DFD
  - Structure Chart menggambarkan struktur pemanggilan Fungsi dan Prosedur dalam suatu program.

Spesifikasi Proses

Disusun dalam bahasa ”pascal like”

Proses 1.1. Baca Resistor

Begin
Baca nilia resistor;
End;

Proses 1.2. Baca Arus

Begin
Baca nilai arus;
End;

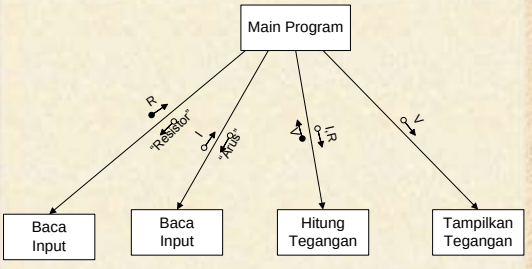
Proses 1.3. Hitung Tegangan

Begin
*V:= I*R;*
End;

Proses 1.4. Tampilkan Tegangan

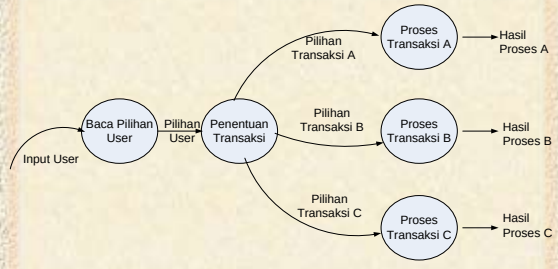
Begin
Tampilkan tegangan;
End;

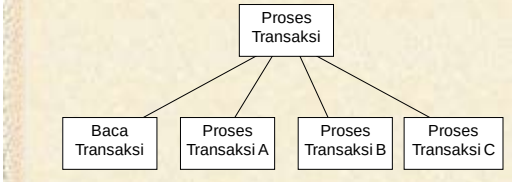
Pemetaan DFD → Structure Chart



DFD: Transaction Flow

DFD Transaction Flow





Design Modeling:
Components Level Design

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

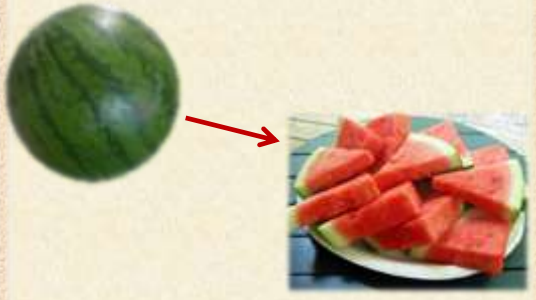
Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Permasalahan

Partisi

- Masalah Besar → membagi/memecah permasalahan tersebut menjadi beberapa bagian yang lebih kecil.
- Menyelesaikan secara bertahap bagian demi bagian, baik secara sendiri maupun berkelompok, sehingga diperoleh solusi dari permasalahan yang besar tersebut.

Partisi



Partisi

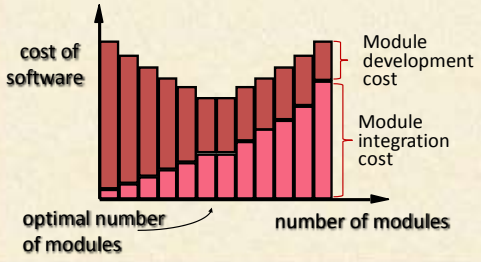
Sebagai gambaran, kita diminta untuk "menyelesaikan" (baca: memakan) buah semangka sampai habis.

- Langkah awal yang dilakukan adalah "mempartisi" (memotong) buah semangka menjadi beberapa bagian
- Memakannya satu persatu, sendiri atau bersama-sama.

Seberapa Banyak Memartisinya?

Modularitas & Permasalahan

Berapakah jumlah modul yang pas untuk desain software tertentu?



Partisi

Program yang besar harus dipartisi menjadi beberapa modul yang mudah diselesaikan

Modul program dapat berupa

- **Procedure** dan/atau
- **Function**.

Dua Paradigma Partisi

- *Structured Development* (SD)
  - SP mempartisi program berdasarkan **kata kerja** (fungsi sistem) atau *behavior*
  - struktur data dan *behavior* terpisah
- *Object Oriented Development* (OOD).
  - mempartisi program berdasarkan **kata benda** (objek diskrit).
  - mengenkapsulasi struktur data dan *behavior* dalam satu objek.

Information Hiding

- Setiap modul tersembunyi dengan yang lain
- Modul harus dirancang agar informasi (prosedur dan data) yang berada di dalam modul tidak dapat diakses oleh oleh modul lain yang tidak memerlukan informasi tersebut
- Kesempurnaan *information hiding* diperoleh pada OOP

Reusable

- ***Reusable*** adalah kunci pokok dalam pengembangan perangkat lunak, tema inilah yang mengilhami perancangan modul program dan perkembangan paradigma pengembangan perangkat lunak secara umum.
- ***Reusable*** bisa diperoleh bila menerapkan *information hiding*.

Functional Independence

- *Functional Independence* merupakan kunci perancangan yang baik dan kunci kualitas program.
- Merupakan hasil pertumbuhan langsung konsep abstraksi dan information hiding.
- Memiliki subfungsi yang spesifik dan antarmuka yang sederhana apabila dipandang dari bagian lain dalam struktur program.

Keuntungan Modul yang Efektif

- Mengurangi kompleksitas
- Mempermudah perubahan
- Lebih mudah diimplementasikan dan dapat dikerjakan secara paralel (Tim)
- mudah membagi dalam tim,
- perambatan kesalahan berkurang, dan
- *reusable* bertambah.

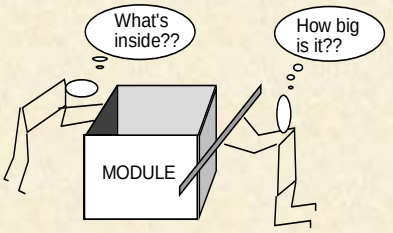
Interface

Interface Modul Program

Interface

```
void buat_garis (int x, char y)
{
   int i;
    for (i = 0; i <= x; x++)
       printf (y);
   printf ("\n");;
}
```


Mengukur Modul: Dua Sudut Pandang

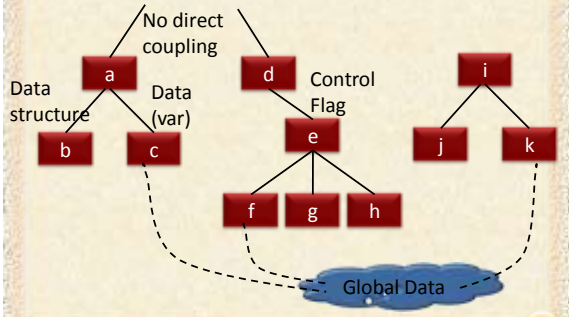


Diukur dengan dengan dua kriteria kualitatif, yaitu: *Cohesion* dan *Coupling*.

Coupling (1-5)

- Coupling is measure of interconnection among modules in a program structure.
- ***Low coupling***: modul memiliki kopling antar modul yang lemah atau sebebas mungkin dengan modul yang lain (independen). Kopling tergantung pada kompleksitas antarmuka modul.

Coupling (2-5)



Coupling (3-5)

- Content coupling (high)
  - satu modul menggunakan informasi yang dikelola oleh modul lain.
  - Satu modul bercabang ke tengah-tengah modul lain.
- Common coupling → beberapa modul mengakses data global.

Coupling (4-5)

- External coupling → bila dua modul memakai bersama antarmuka alat eksternal
- Control coupling (moderate coupling) → sangat umum dalam desain SW. Salah satu modul ditentukan oleh "control flag" yang diset oleh modul lain.

Coupling (5-5)

- Stamp coupling (Data-structured coupling) → antar modul melewatkan struktur data
- Data coupling (LOW) → melewatkan data antar modul
- No coupling Modules do not communicate at all with one another.

Bandingkan

```
void buat_garis (int x, char y)
{
   for (i = 0; i <= x; x++)
      printf (y);
   printf ("\n");;
}
```

```
void buat_garis (int x, char y)
{
   int i;
   for (i = 0; i <= x; x++)
      printf (y);
   printf ("\n");;
}
```


Cohesion

- ***High cohesion*** (*functional cohesion*): modul hanya melakukan **satu tugas** dan memerlukan sedikit interaksi dengan modul lain dalam satu program. Singkatnya: ***do just one thing***.
- Modul yang baik (modul yang kohesif atau *functional cohesion*) → *high cohesion*

Bandingkan

```
Void  pangkat_tiga (int x)
{
   int   z;
   z := x * x * x;
   printf ("Hasil 1 =  \n", z);
   buat_garis (10,'=');
}
```

```
Int  pangkat_tiga (int x)
{
   Int  z;
    z := x * x * x;
   return z;
}
```

Tugas

Baca tentang Cohesion

Software Testing 01:
Sostware Testing Strategies

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

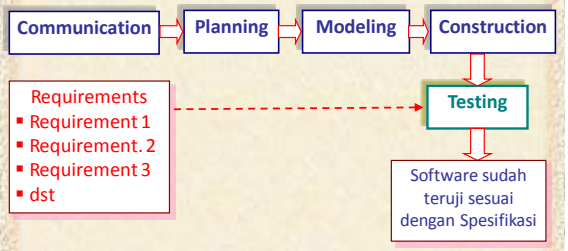
Software Testing

- Testing adalah proses eksekusi program yang bertujuan untuk menemukan kesalahan
- *Testcase* yang baik dapat menemukan error yang belum ditemui sebelumnya.

Prinsip-prinsip Testing

- Seluruh test harus sesuai dengan *customer requirement*
- Dipersiapkan jauh sebelum mulai
- Testing harus dimulai “in the small” dan maju menuju testing “in the large”.
- Dilakukan oleh pihak ketiga

Testing & Siklus Pengembangan Software



Pendekatan Stratejik Software Testing

Verification & Validation

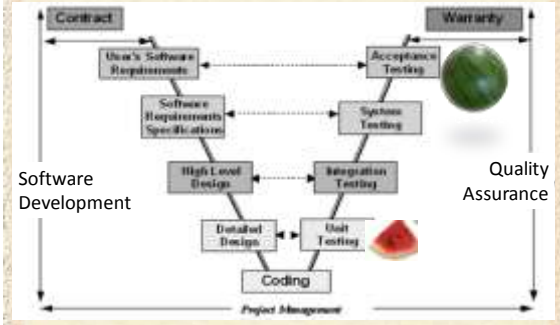
- **Verification** → Seperangkat aktivitas yang menjamin bahwa software mengimplementasikan fungsi spesifik secara benar.

  Verification: Are we building the product right?

- **Validation** → Seperangkat aktivitas yang menjamin bahwa software yang sudah dibangun dapat dilacak ke kebutuhan user.

  Validation: Are we building the right product?

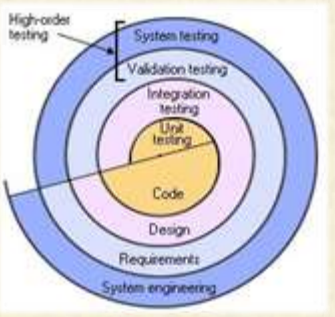
V-Model



Software Development

Quality Assurance

Testing Strategy

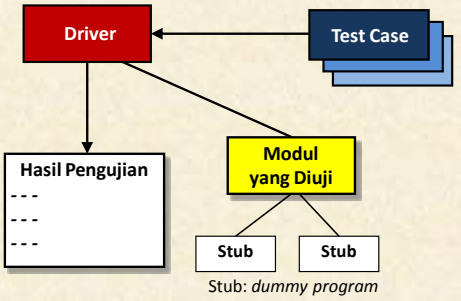


Software Testing Strategies

- Unit Testing
- Integration Testing
- Validation Testing
- System Testing

Test Strategies for Conventional Software

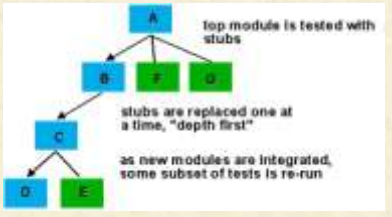
Unit Testing



Stub: *dummy program*

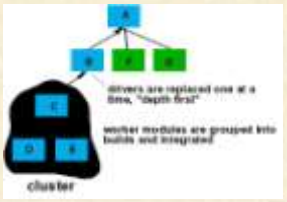
Integration Testing → Top-Down

- Superordinat sebagai driver
- Stub sebagai pengganti subordinat



Integration Testing → Bottom-Up

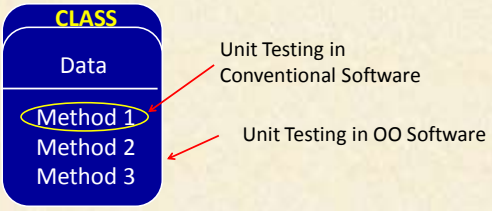
- Komponen level bawah dikombinasikan menjadi clusters (build) → membentuk subfungsi tertentu
- Driver adalah program kendali untuk testing
- Jika cluster sudah ditest, driver dibuang



Test Strategies for Object-Oriented Software

Test Strategies for OO Software

Unit Testing: Conventional vs OO



Integration Testing in OO Context

OO tidak memiliki struktur hirarki (top-down dan bottom-up) seperti konvensional.

Ada 2 strategi untuk integration testing sistem OO

- Thread-based testing → mengintegrasikan seperangkat Class yang diperlukan untuk merespon terhadap satu input atau event untuk sistem
- Used-based testing → memulai konstruksi sistem dengan menguji classes (independent) yang menggunakan server class. Lapis berikutnya adalah menguji dependent class yang menggunakan independent class

Driver dan Stub pada OO Software

- Juga berlaku pada OO software
- Driver dapat digunakan untuk menguji operasi class level terrendah atau kelompok class
- Stub dapat digunakan dalam situasi dimana class yang diperlukan belum diimplementasikan

Test Strategies for WebApps

Test Strategies for WebApps

- Model konten untuk WebApp ditinjau untuk mengungkap kesalahan.
- The model interface ditinjau untuk memastikan bahwa semua kasus penggunaan dapat diakomodasi.
- Model desain untuk WebApp ditinjau untuk mengungkap kesalahan navigasi.
- User interface diuji untuk mengungkap kesalahan dalam presentasi dan/atau mekanisme navigasi.
- Setiap komponen fungsional adalah unit yang diuji.
- Navigasi seluruh arsitektur diuji.

Test Strategies for WebApps

- WebApp diimplementasikan dalam berbagai konfigurasi lingkungan yang berbeda dan diuji untuk kompatibilitas dengan setiap konfigurasi.
- Tes keamanan yang dilakukan dalam upaya untuk mengeksploitasi kerentanan dalam webapp atau dalam lingkungannya.
- Tes kinerja yang dilakukan.
- WebApp diuji oleh populasi dikendalikan dan dipantau pengguna akhir. Hasil interaksi mereka dengan sistem dievaluasi untuk kesalahankonten dan navigasi, masalah kegunaan, masalah kompatibilitas, dan keandalan dan kinerja.

Validation Testing

Validation Testing

- **Alpha Testing**
  → Developer's Site by Customer
  (seperti test drive di pabrik mobil)
  - Developer's Site by Customer
  - Pada lingkungan yang terkendali
- **Beta testing** (Customer acceptance testing)
  → Customer's sites by end user
  - Customer's sites by end user
  - "live" application pada lingkungan yang tidak bisa dikendalikan developer
  - User melaporkan hasil ke developer

System Testing

System Testing

- Recovery testing
- Security testing
- Stress testing
- Performance testing
- Deployment testing

Recovery Testing

- Sistem berbasis komputer harus bisa merecover dari kesalahan dan mengulangi proses dalam waktu yang telah ditetapkan
- Dalam banyak kasus, sistem harus *fault tolerant*, kesalahan proses tidak menyebabkan seluruh sistem berhenti
- Recovery testing → memaksa SW untuk rusak dengan berbagai cara dan membuktikan bahwa recovery dilakukan secara tepat

Security Testing

- Membuktikan bahwa mekanisme proteksi telah memproteksi dari penetrasi yang tidak tepat
- Selama security testing, penguji berperan sebagai individu yang ingin memasuki sistem

Stress testing

- Menghadapkan program pada situasi yang tidak normal
- Penguji bertanya: seberapa tinggi ketidak normalan sebelum rusak
- **Stress testing** → mengeksekusi sistem dalam keadaan permintaan sumber daya (kuantitas, frekuensi, atau volume) yang tidak normal
  - Memberi interupsi 10x/detik dari batas normal 1-2x/detik
  - Input data rate ditinggikan
  - Test case memerlukan eksekusi memori makksimum
- Prinsip: penguji berusaha menenggelampan program

Performace Testing

- Berhubungan dengan Stress testing
- Dirancang untuk menguji *run-time performance* SW dalam konteks sistem yang terintegrasi

Deployment Testing

- Software harus dieksekusi pada berbagai macam platform dan lebih dari satu operating system.
- Menguji seluruh prosedur instalasi
- Disebut juga *configuration testing*.

Software Testing 02:
Testing Conventional Applications

Rekayasa Perangkat Lunak
Priyanto
E-mail : priyanto@uny.ac.id

Program Studi Pendidikan Teknik Informatika
Jurusan Pendidikan Teknik Elektronika
Fakultas Teknik , Universitas Negeri Yogyakarta
2014

Internal & External View of Testing

Internal & External View of Testing

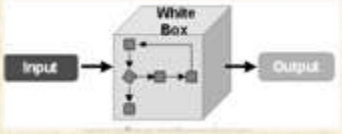
White-box Testing
(Glass-box Testing)
- Basis path testing
- Control structure testing

Black-box Testing

White-Box Testing

White-box Testing

Metode pengujian yang menggunakan struktur kontrol desain prosedural untuk menghasilkan *testcase*.



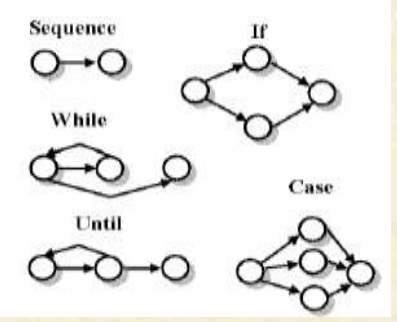
White-box Testing

- Menjamin seluruh *independent path* di dalam modul diuji minimal satu kali
- Menguji seluruh keputusan logika (True & False)
- Mengeksekusi seluruh *loop* sesuai dengan batasan operasinya
- Menguji struktur data internal untuk menjamin validitas

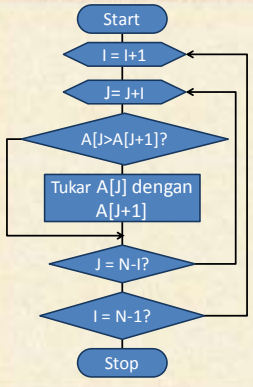
Basis Path Testing

- Adalah WB testing yang pertama kali diusulkan oleh Tom McCabe (1976)
- Metode *basis path* memungkinkan *testcase designer* memperoleh ukuran kompleksitas lojik dari desain prosedural dan menggunakan ukuran ini sebagai pemandu untuk menentukan *basis set execution path*

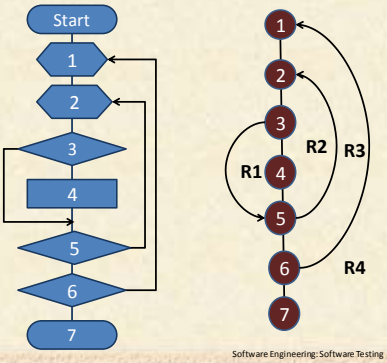
Notasi Flow Graph



Flowchart Bubble Sort



Flowchart vs Flow Graph

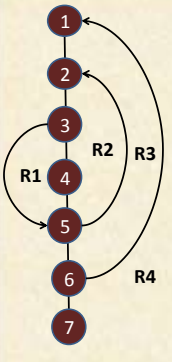


Cyclomatic Complexity

Cyclomatic Complexity berdasar pada teori graf, dihitung dengan salah satu dari dua cara:

- Cyclomatic Complexity = Jumlah region flowgraph
- Cyclomatic Complexity, V(G) = E – N + 2
  E = jumlah edge
  N = jumlah node

Cyclomatic Complexity



Independent program path

- *Path 1: 1 2 3 4 5 2*
- Path 2: 1 2 3 4 5 6 1
- Path 3: 1 2 3 5 6 7
- Path 4: 1 2 3 4 5 6 7

- Region = 4
- Node= 7
- Edge = 9

Cyclomatic Complexity:

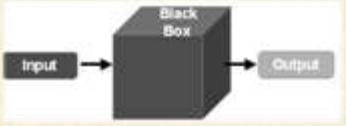
- V(G) = Region = 4
- V(G) = E - N + 2 = 9 -7 +2 = 4

Control Structure Testing

- Condition testing
- Data flow testing
- Loop testing
  - Simple loop
  - Nested loop
  - Concatenated loop

Black-box Testing

- Memfokuskan pada kebutuhan **fungsional** program
- **Bukan alternatif** dari white-box testing, tetapi **complementer**



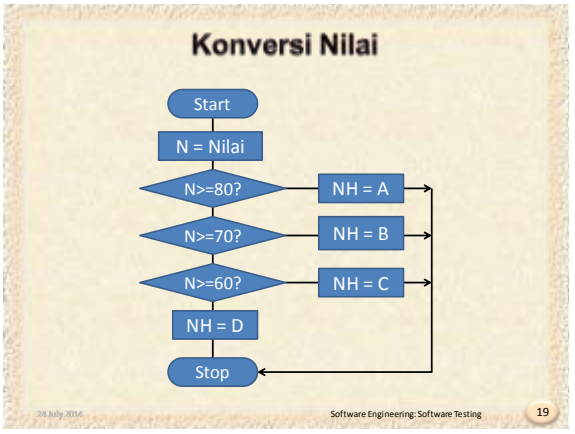
Black-box Testing

Menemukan kesalahan dengan kategori:

- Fungsi-fungsi yang tidak benar
- Error pada struktur data atau akses database eksternal
- Error Antarmuka, performa
- Error inisialisasi dan terminasi

Black-box Testing

- Menguji beberapa aspek sistem dengan sedikit memperhatikan struktur lojik internal program
- Digunakan untuk menunjukkan fungsi program beroperasi:
  Input diterima → Output benar


Konversi Nilai
Start
N = Nilai
N>=80?
NH = A
N>=70?
NH = B
N>=60?
NH = C
NH = D
Stop
24 July 2014
Software Engineering: Software Testing
19


Terima Kasih