TIF08 - Rekayasa Perangkat Lunak

DESAIN dan IMPLEMENTASI

Pertemuan ke: 9-10

Sub-CPMK

- Mahasiswa mampu merancang dan menunjukkan isu-isu implementasi perangkat lunak (C3,A3)

Materi:

1. Rancangan Berorientasi Object
2. Pola-Pola Rancangan
3. Masalah Implementasi
4. Pengembangan Open Source

1. Rancangan Berorientasi Objek

Rancangan Berorientasi Objek

- Desain dan implementasi perangkat lunak adalah tahap dalam proses rekayasa perangkat lunak di mana sistem perangkat lunak yang dapat dijalankan dikembangkan.
- Untuk beberapa sistem sederhana, rekayasa perangkat lunak berarti desain dan implementasi perangkat lunak dan semua aktivitas rekayasa perangkat lunak lainnya digabungkan dengan proses desain & implementasi.

Rancangan Berorientasi Objek (Lanj..)

- Namun, untuk sistem besar, desain dan implementasi perangkat lunak hanyalah salah satu dari sejumlah proses rekayasa perangkat lunak (rekayasa persyaratan, verifikasi dan validasi, dll.).
- Desain perangkat lunak dan kegiatan implementasi selalu sisipi.

Rancangan Berorientasi Objek (Lanj..)

- Desain perangkat lunak adalah aktivitas kreatif di mana kita mengidentifikasi komponen perangkat lunak dan hubungannya, berdasarkan kebutuhan pelanggan.
- Implementasi adalah proses mewujudkan desain sebagai program.
- Terkadang ada tahap desain terpisah, dan desain ini dimodelkan dan didokumentasikan.

Rancangan Berorientasi Objek (Lanj..)

- Di lain waktu, desain ada di kepala programmer atau dibuat sketsa kasar di papan tulis atau lembaran kertas.
- Desain adalah tentang bagaimana menyelesaikan suatu masalah, sehingga selalu ada proses desain.
- Namun, tidak selalu diperlukan atau tepat untuk mendeskripsikan desain secara detail menggunakan UML atau bahasa deskripsi desain lainnya .

Rancangan Berorientasi Objek (Lanj..)

- Desain dan implementasi terkait erat, dan kita biasanya harus mempertimbangkan masalah implementasi saat mengembangkan desain.
- Misalnya, menggunakan UML untuk mendokumentasikan desain mungkin hal yang benar untuk dilakukan jika kita memprogram dalam bahasa berorientasi objek seperti Java atau C #.

Rancangan Berorientasi Objek (Lanj..)

- Tidak ada gunanya menggunakan UML jika kita mengimplementasikan sistem kita dengan mengkonfigurasi paket off-the-shelf.
- Seperti yang dibahas pada metode agile biasanya bekerja dari sketsa informal desain dan menyerahkan keputusan desain kepada pemrogram.

Rancangan Berorientasi Objek (Lanj..)

- Salah satu keputusan implementasi terpenting yang harus dibuat pada tahap awal proyek perangkat lunak adalah apakah akan membangun atau membeli perangkat lunak aplikasi.
- Untuk berbagai jenis aplikasi, sekarang dimungkinkan untuk membeli sistem aplikasi off-the-shelf yang dapat diadaptasi dan disesuaikan dengan kebutuhan pengguna.

Rancangan Berorientasi Objek (Lanj..)

- Misalnya, jika ingin menerapkan sistem rekam medis, kita bisa membeli paket yang sudah digunakan di rumah sakit.
- Biasanya lebih murah dan lebih cepat untuk menggunakan pendekatan ini daripada mengembangkan sistem baru dalam bahasa pemrograman konvensional.

Rancangan Berorientasi Objek (Lanj..)

- Saat kita mengembangkan sistem aplikasi dengan menggunakan kembali produk (*reuse*) off-the-shelf, proses desain berfokus pada cara mengonfigurasi produk sistem untuk memenuhi persyaratan aplikasi.
- Kita tidak mengembangkan model desain sistem, seperti model objek sistem dan interaksinya.

Rancangan Berorientasi Objek (Lanj..)

- Sistem berorientasi objek terdiri dari objek yang berinteraksi yang mempertahankan state lokalnya sendiri dan menyediakan operasi pada state tersebut.
- Representasi state bersifat pribadi dan tidak dapat diakses langsung dari luar objek.

Rancangan Berorientasi Objek (Lanj..)

- Proses desain berorientasi objek melibatkan perancangan kelas objek dan hubungan antara kelas-kelas ini.
- Kelas-kelas ini mendefinisikan objek dalam sistem dan interaksinya. Ketika desain direalisasikan sebagai program, objek dibuat secara dinamis dari definisi kelas ini.

Rancangan Berorientasi Objek (Lanj..)

- Objek mencakup data dan operasi untuk memanipulasi data itu. Oleh karena itu, mereka dapat dipahami dan dimodifikasi sebagai entitas yang berdiri sendiri.
- Mengubah implementasi objek atau menambahkan layanan seharusnya tidak memengaruhi objek sistem lainnya.

Rancangan Berorientasi Objek (Lanj..)

- Karena objek dikaitkan dengan berbagai hal, sering kali ada pemetaan yang jelas antara entitas dunia nyata (seperti komponen perangkat keras) dan objek pengontrolnya dalam sistem.

Rancangan Berorientasi Objek (Lanj..)

- Untuk mengembangkan desain sistem dari konsep menjadi desain mendetail dan berorientasi objek, kita perlu:
  1. Memahami dan mendefinisikan konteks dan interaksi eksternal dengan sistem.
  2. Merancang arsitektur sistem.
  3. Identifikasi objek utama dalam sistem.
  4. Kembangkan model desain.
  5. Tentukan antarmuka.

Rancangan Berorientasi Objek (Lanj..)

- Seperti semua aktivitas kreatif (*creative activities*), desain bukanlah proses sekuensial yang jelas. Kita mengembangkan desain dengan mendapatkan ide, mengusulkan solusi, dan menyempurnakan solusi ini saat informasi tersedia.
- Kita pasti harus mundur dan mencoba lagi ketika masalah muncul.

Rancangan Berorientasi Objek (Lanj..)

- Terkadang kita menelusuri opsi secara detail untuk melihat apakah opsi tersebut berfungsi; di lain waktu kita mengabaikan detail hingga prosesnya terlambat.
- Terkadang kita menggunakan notasi, seperti UML, untuk memperjelas aspek desain dengan tepat;

Rancangan Berorientasi Objek (Lanj..)

- Pada kasus ini menjelaskan desain perangkat lunak berorientasi objek dengan mengembangkan desain untuk bagian perangkat lunak yang disematkan untuk stasiun cuaca di alam terbuka.
- Stasiun cuaca alam digunakan di daerah terpencil. Setiap stasiun cuaca merekam informasi cuaca lokal dan secara berkala mentransfernya ke sistem informasi cuaca, menggunakan tautan satelit.

1.1. System Context & Interactions

- Tahap pertama dalam proses desain perangkat lunak adalah: pemahaman hubungan antara perangkat lunak yang sedang dirancang dan lingkungan eksternalnya.

1.1. System Context & Interactions (Lanj.)

- Ini penting untuk menyediakan fungsionalitas sistem yang diperlukan dan bagaimana menyusun sistem untuk berkomunikasi dengan lingkungannya.
- Memahami konteks untuk menetapkan batasan sistem.

1.1. System Context & Interactions (Lanj.)

- Menetapkan batas sistem membantu memutuskan fitur apa yang diimplementasikan dalam sistem yang sedang dirancang dan fitur apa yang ada di sistem terkait lainnya.

1.1. System Context & Interactions (Lanj.)

- Dalam hal ini, perlu memutuskan bagaimana fungsionalitas didistribusikan antara sistem kontrol untuk semua stasiun cuaca dan perangkat lunak yang disematkan di stasiun cuaca itu sendiri.

1.1. System Context & Interactions (Lanj.)

- Model konteks sistem dan model interaksi menyajikan pandangan yang saling melengkapi dari hubungan antara sistem dan lingkungannya:
  1. Model konteks sistem adalah model struktural yang mendemonstrasikan sistem lain di lingkungan sistem yang sedang dikembangkan.

1.1. System Context & Interactions (Lanj.)

2. Model interaksi adalah model dinamis yang menunjukkan bagaimana sistem berinteraksi dengan lingkungannya saat digunakan.

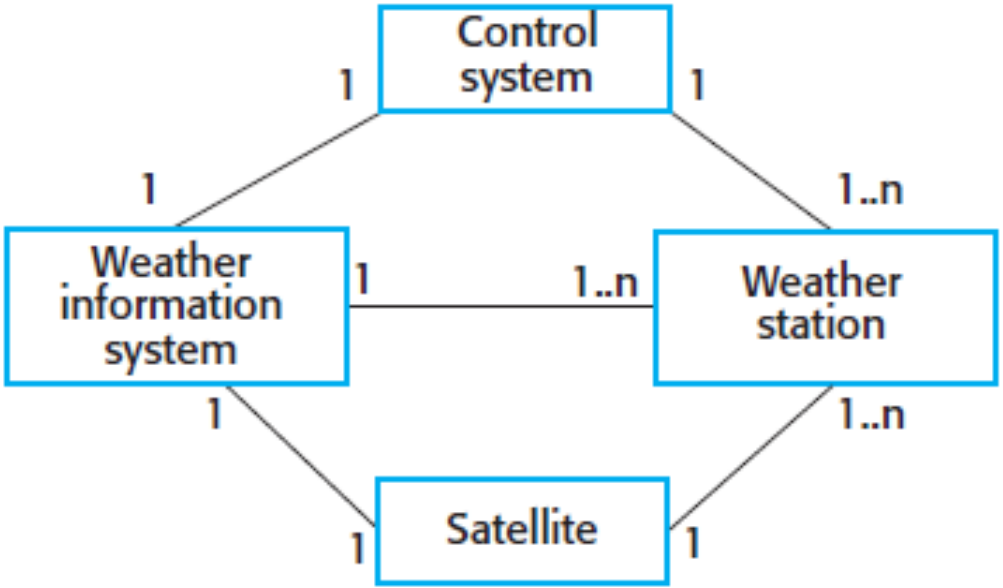
1.1. System Context & Interactions (Lanj.)

- Model konteks suatu sistem dapat direpresentasikan menggunakan asosiasi.
- Asosiasi hanya menunjukkan bahwa ada beberapa hubungan antara entitas yang terlibat dalam asosiasi.

1.1. System Context & Interactions (Lanj.)

- Gambar 6.1 menunjukkan bahwa sistem di lingkungan setiap stasiun cuaca adalah sistem informasi cuaca, sistem satelit onboard, dan sistem kendali.
- Informasi kardinalitas pada link menunjukkan bahwa ada sistem kontrol tunggal tetapi beberapa stasiun cuaca, satu satelit, dan satu sistem informasi cuaca umum.

1.1. System Context & Interactions (Lanj.)



Gambar 6.1: System context for the weather station

1.1. System Context & Interactions (Lanj.)

- Saat memodelkan interaksi sistem dengan lingkungannya, harus menggunakan pendekatan abstrak yang tidak menyertakan terlalu banyak detail.
- Salah satu cara untuk melakukannya adalah dengan menggunakan model use case.

1.1. System Context & Interactions (Lanj.)

- Setiap kasus use case mewakili interaksi dengan sistem.
- Setiap kemungkinan interaksi diberi nama dalam elips, dan entitas eksternal yang terlibat dalam interaksi tersebut diwakili oleh figur actor.

1.1. System Context & Interactions (Lanj.)

- Model use case untuk stasiun cuaca ditunjukkan pada Gambar 6.2. Hal ini menunjukkan bahwa stasiun cuaca berinteraksi dengan sistem informasi cuaca untuk melaporkan data cuaca dan status perangkat keras stasiun cuaca tersebut.

1.1. System Context & Interactions (Lanj.)

- Interaksi lainnya adalah dengan sistem kontrol yang dapat mengeluarkan perintah kontrol stasiun cuaca tertentu. Actor digunakan dalam UML untuk mewakili sistem lain serta pengguna manusia.

1.1. System Context & Interactions (Lanj.)

- Masing-masing use case ini harus dijelaskan dalam bahasa alami yang terstruktur.
- Ini membantu perancang mengidentifikasi objek dalam sistem dan memberi mereka pemahaman tentang apa yang akan dilakukan sistem.

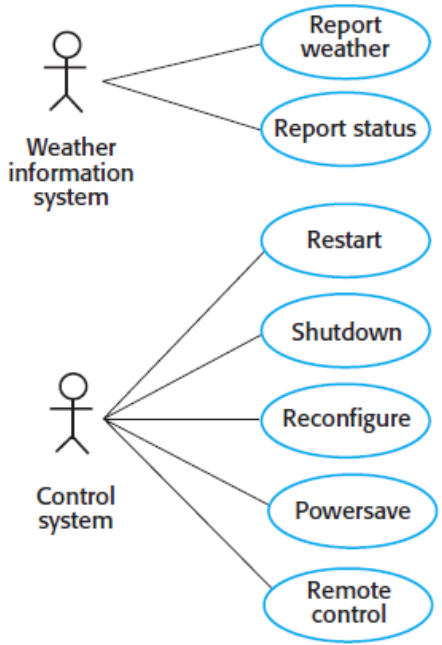
1.1. System Context & Interactions (Lanj.)

- Gunakan format standar untuk uraian ini yang dengan jelas mengidentifikasi informasi apa yang dipertukarkan, bagaimana interaksi dimulai, dan sebagainya.
- Sistem tertanam sering dimodelkan dengan menggambarkan bagaimana mereka menanggapi stimuli internal atau eksternal.

1.1. System Context & Interactions (Lanj.)

- Oleh karena itu, stimuli dan respons terkait harus dicantumkan dalam deskripsi. Gambar 6.3 menunjukkan deskripsi use case Laporan Cuaca dari Gambar 6.2 yang didasarkan pada pendekatan ini.

1.1. System Context & Interactions (Lanj.)



Gambar 6.2: Weather station use cases

1.1. System Context & Interactions (Lanj.)

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Data	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum and average wind speeds; the total rainfall; and the wind direction as sampled at 5-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour, but this frequency may differ from one station to another and may be modified in future.

Gambar 6.3: Use case description—Report weather

1.2. Architectural Design

- Setelah interaksi antara sistem perangkat lunak dan lingkungan sistem telah ditentukan, kita menggunakan informasi ini sebagai dasar untuk merancang arsitektur sistem.
- Tentu saja, kita perlu menggabungkan pengetahuan ini dengan pengetahuan umum kita tentang prinsip-prinsip desain arsitektur dan dengan pengetahuan domain yang lebih detail.

1.2. Architectural Design (Lanj.)

- Kita mengidentifikasi komponen utama yang membentuk sistem dan interaksinya.
- Kita kemudian dapat merancang organisasi sistem menggunakan pola arsitektur seperti model berlapis atau klien-server.

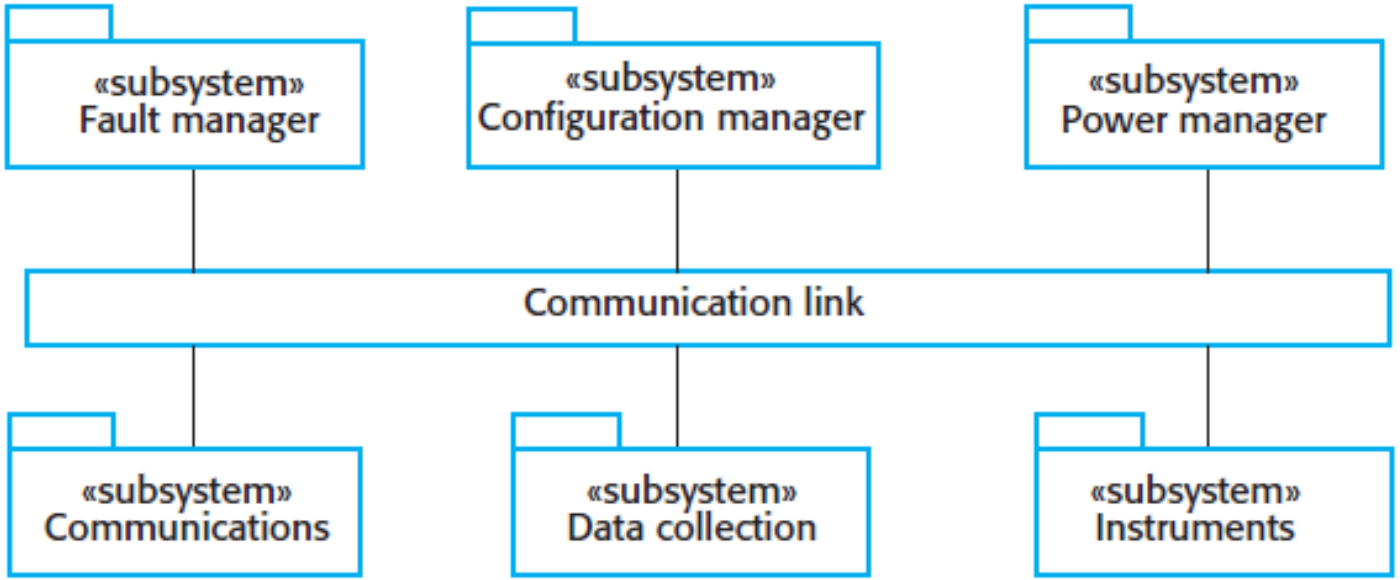
1.2. Architectural Design (Lanj.)

- Desain arsitektur tingkat tinggi untuk perangkat lunak stasiun cuaca ditunjukkan pada Gambar 6.4.
- Stasiun cuaca terdiri dari subsistem independen yang berkomunikasi dengan menyiarkan pesan pada infrastruktur umum, ditunjukkan sebagai Link komunikasi pada Gambar 6.4.

1.2. Architectural Design (Lanj.)

- Setiap subsistem mendengarkan pesan pada infrastruktur itu dan mengambil pesan yang ditujukan untuknya.
- "Model pendengar" ini adalah style arsitektur yang umum digunakan untuk sistem terdistribusi.

1.2. Architectural Design (Lanj.)



Gambar 6.4: High-level architecture of weather station

1.2. Architectural Design (Lanj.)

- Ketika subsistem komunikasi menerima perintah kontrol, seperti shutdown, perintah tersebut diambil oleh subsistem lainnya, yang kemudian ditutup dengan cara yang benar.

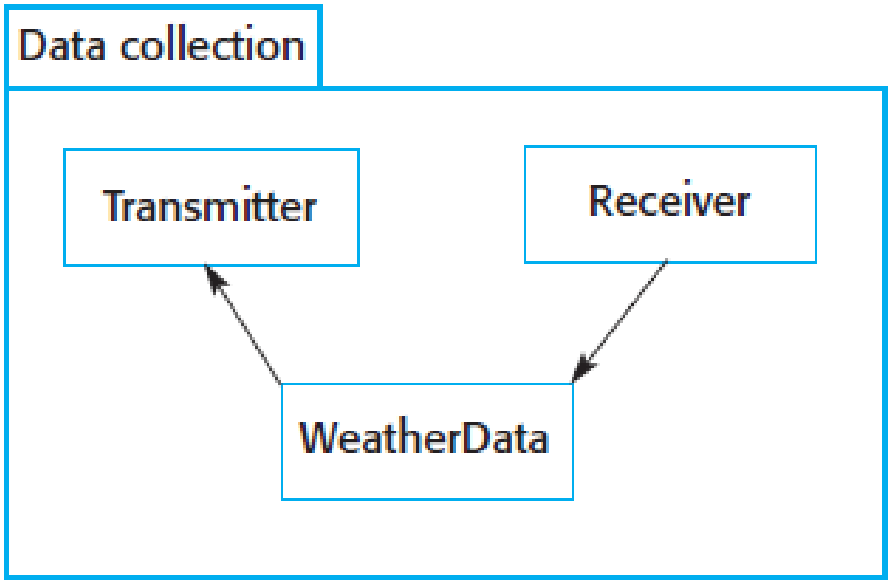
1.2. Architectural Design (Lanj.)

- Manfaat utama dari arsitektur ini adalah mudah untuk mendukung konfigurasi subsistem yang berbeda karena pengirim pesan tidak perlu mengalamatkan pesan ke subsistem tertentu.

1.2. Architectural Design (Lanj.)

- Gambar 6.5 menunjukkan arsitektur subsistem pengumpulan data, yang termasuk dalam Gambar 6.4.
- Transmitter dan Receiver objek yang berkaitan dengan mengelola komunikasi, dan WeatherData objek merangkum informasi yang dikumpulkan dari instrumen dan ditransmisikan ke sistem informasi cuaca.

1.2. Architectural Design (Lanj.)



Gambar 6.5: Architecture of data collection system

1.3. Object Class Identification

- Pada tahap proses desain, harus memiliki beberapa ide tentang objek penting dalam sistem yang didesain.
- Saat pemahaman tentang desain berkembang, kits menyempurnakan ide-ide ini tentang objek sistem.
- Deskripsi use case membantu mengidentifikasi objek dan operasi dalam sistem.

1.3. Object Class Identification (Lanj.)

- Dari use case penggunaan laporan cuaca, jelas bahwa kita perlu mengimplementasikan objek yang mewakili instrumen yang mengumpulkan data cuaca dan objek yang mewakili ringkasan data cuaca.
- Kita juga biasanya memerlukan objek sistem tingkat tinggi atau objek yang merangkum interaksi sistem yang ditentukan dalam kasus penggunaan.
- Dengan mempertimbangkan objek ini, kita dapat mulai mengidentifikasi kelas objek umum dalam sistem.

1.3. Object Class Identification (Lanj.)

- Dengan mempertimbangkan objek ini, kita dapat mulai mengidentifikasi kelas objek umum dalam sistem.
- Karena desain berorientasi objek berkembang pada 1980-an, berbagai cara untuk mengidentifikasi kelas objek dalam sistem berorientasi objek disarankan:

1.3. Object Class Identification (Lanj.)

1. Gunakan analisis gramatikal dari deskripsi bahasa alami dari sistem yang akan dibangun. Objek dan atribut adalah kata benda; operasi (method) atau layanan adalah kata kerja.
2. Menggunakan entitas berwujud (benda) dalam domain aplikasi seperti pesawat terbang, peran seperti manajer, acara seperti permintaan, interaksi seperti rapat, lokasi seperti kantor, unit organisasi seperti perusahaan, dan sebagainya.

1.3. Object Class Identification (Lanj.)

3. Gunakan analisis berbasis skenario di mana berbagai skenario penggunaan sistem diidentifikasi dan dianalisis secara bergantian. Saat setiap skenario dianalisis, tim yang bertanggung jawab untuk analisis harus mengidentifikasi objek, atribut, dan operasi yang diperlukan.

1.3. Object Class Identification (Lanj.)

- Dalam praktiknya, kita harus menggunakan beberapa sumber pengetahuan untuk menemukan kelas objek. Kelas objek, atribut, dan operasi (method) yang awalnya diidentifikasi dari deskripsi sistem informal dapat menjadi titik awal untuk desain.

1.3. Object Class Identification (Lanj.)

- Informasi dari pengetahuan domain aplikasi atau analisis skenario kemudian dapat digunakan untuk memperbaiki dan memperluas objek awal.
- Informasi ini dapat dikumpulkan dari dokumen persyaratan, diskusi dengan pengguna, atau analisis sistem yang ada.

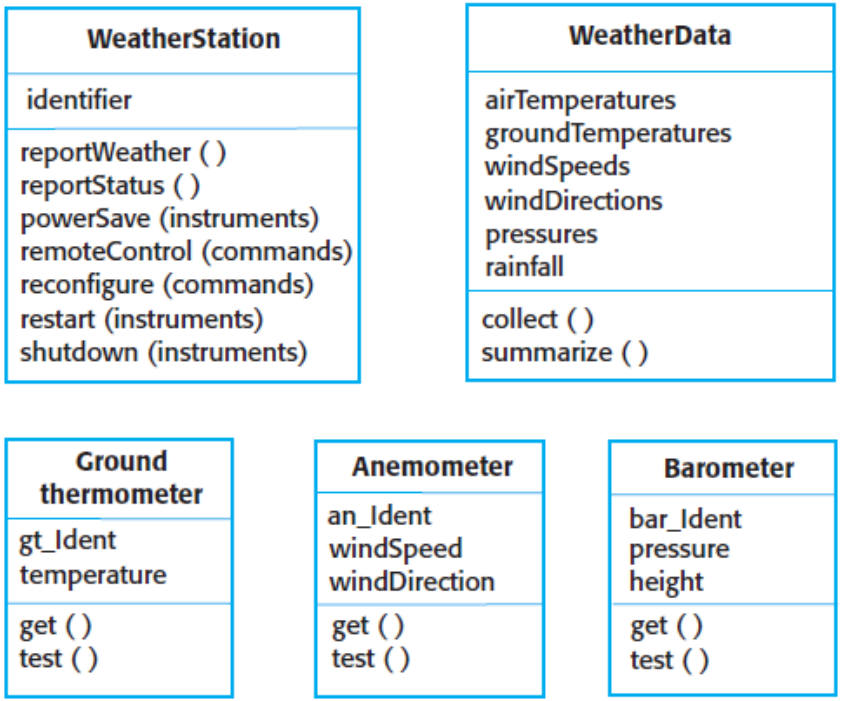
1.3. Object Class Identification (Lanj.)

- Selain objek yang mewakili entitas di luar sistem, kits mungkin juga harus mendesain "objek implementasi" yang digunakan untuk menyediakan layanan umum seperti penelusuran dan pemeriksaan validitas.

1.3. Object Class Identification (Lanj.)

- Di stasiun cuaca hutan belantara, identifikasi objek didasarkan pada perangkat keras berwujud dalam sistem.
- Karena tidak memiliki ruang untuk memasukkan semua objek sistem di sini, tetapi telah menunjukkan lima kelas objek pada Gambar 6.6.

1.3. Object Class Identification (Lanj.)



Gambar 6.6: Weather station objects

1.3. Object Class Identification (Lanj.)

- Tanah termometer, Anemometer, dan Barometer objek adalah aplikasi objek domain, dan Weatherstation dan WeatherData benda telah diidentifikasi dari sistem deskripsi dan skenario (use case) deskripsi:

1.3. Object Class Identification (Lanj.)

1. Kelas objek WeatherStation menyediakan antarmuka dasar stasiun cuaca dengan lingkungannya. Operasinya didasarkan pada interaksi yang ditunjukkan pada Gambar 6.3. disini menggunakan satu kelas objek, dan itu mencakup semua interaksi ini. Alternatifnya, kita dapat merancang antarmuka sistem sebagai beberapa kelas berbeda, dengan satu kelas per interaksi.
2. Kelas objek WeatherData bertanggung jawab untuk memproses perintah laporan cuaca. Ini mengirimkan data yang diringkas dari instrumen stasiun cuaca ke sistem informasi cuaca.

1.3. Object Class Identification (Lanj.)

3. Kelas objek Termometer Tanah , Anemometer, dan Barometer berhubungan langsung dengan instrumen dalam sistem. Mereka mencerminkan entitas perangkat keras yang nyata dalam sistem dan operasi berkaitan dengan pengendalian perangkat keras itu. Objek ini beroperasi secara mandiri untuk mengumpulkan data pada frekuensi yang ditentukan dan menyimpan data yang dikumpulkan secara lokal. Data ini dikirim ke objek WeatherData berdasarkan permintaan.

1.3. Object Class Identification (Lanj.)

- Anda menggunakan pengetahuan tentang domain aplikasi untuk mengidentifikasi objek lain, atribut. dan layanan:

1. Stasiun cuaca sering berada di tempat terpencil dan menyertakan berbagai instrumen yang terkadang salah. Kegagalan instrumen harus dilaporkan secara otomatis. Ini menyiratkan bahwa Anda memerlukan atribut dan operasi untuk memeriksa fungsi instrumen yang benar.

1.3. Object Class Identification (Lanj.)

1. Ada banyak stasiun cuaca terpencil, jadi setiap stasiun cuaca harus memiliki pengenalnya sendiri sehingga dapat diidentifikasi secara unik dalam komunikasi.
2. Karena stasiun cuaca dipasang pada waktu yang berbeda, jenis instrumen mungkin berbeda. Oleh karena itu, setiap instrumen juga harus diidentifikasi secara unik, dan database informasi instrumen harus dipelihara.

1.3. Object Class Identification (Lanj.)

- Pada tahap proses desain ini, kita harus fokus pada objek itu sendiri, tanpa memikirkan bagaimana objek tersebut dapat diimplementasikan.
- Setelah kita mengidentifikasi objek, kita kemudian menyempurnakan desain objek.

1.3. Object Class Identification (Lanj.)

- Kita mencari fitur umum lalu mendesain hierarki pewarisan untuk sistem. Misalnya, kita dapat mengidentifikasi superclass Instrumen, yang mendefinisikan fitur umum dari semua instrumen, seperti pengenal, dan operasi get dan uji.

1.3. Object Class Identification (Lanj.)

- Kita juga dapat menambahkan atribut dan operasi baru ke superclass, seperti atribut yang mencatat seberapa sering data harus dikumpulkan.

1.4. Design Models

- Model desain atau sistem, seperti yang dibahas pertemuan 5, menunjukkan objek atau kelas objek dalam suatu sistem.
- Juga menunjukkan asosiasi dan hubungan antara entitas ini. Model-model ini adalah jembatan antara kebutuhan sistem dan implementasi sistem.

1.4. Design Models (Lanj.)

- Mereka harus abstrak sehingga detail yang tidak perlu tidak menyembunyikan hubungan antara mereka dan persyaratan sistem.
- Namun, mereka juga harus menyertakan detail yang cukup bagi pemrogram untuk membuat keputusan implementasi.

1.4. Design Models (Lanj.)

- Tingkat detail yang butuhkan dalam model desain bergantung pada proses desain yang digunakan.
- Model detail, yang berasal dari model abstrak tingkat tinggi, digunakan agar semua anggota tim memiliki pemahaman yang sama tentang desain.

1.4. Design Models (Lanj.)

- Langkah penting dalam proses desain adalah memutuskan model desain yang dibutuhkan dan tingkat detail yang diperlukan dalam model ini.
- Ini tergantung pada jenis sistem yang dikembangkan. Sistem pemrosesan data sekuensial sangat berbeda dari sistem waktu nyata.

1.4. Design Models (Lanj.)

- UML mendukung 13 jenis model yang berbeda, namun tidak harus menggunakan semua jenis model di UML.
- Meminimalkan jumlah model yang diproduksi mengurangi biaya desain dan waktu yang dibutuhkan untuk menyelesaikan proses desain.

1.4. Design Models (Lanj.)

Saat menggunakan UML untuk mengembangkan desain, harus mengembangkan dua jenis model desain, yaitu:

1. ***Model struktural,*** yang menggambarkan struktur statis dari sistem menggunakan kelas-kelas objek dan hubungannya. Hubungan penting yang mungkin didokumentasikan pada tahap ini adalah hubungan generalisasi (pewarisan), uses/used-by relationships, and composition relationships.

1.4. Design Models (Lanj.)

2. ***Model dinamis,*** yang mendeskripsikan struktur dinamis sistem dan menunjukkan interaksi waktu proses yang diharapkan antara objek sistem. Interaksi yang dapat didokumentasikan termasuk sequence diagram, state diagram.

1.4. Design Models (Lanj.)

- Tiga jenis model UML sangat berguna untuk menambahkan detail ke use case dan model arsitektur:
  1. Model subsistem, yang menunjukkan pengelompokan objek secara logis ke dalam subsistem yang koheren. Ini direpresentasikan menggunakan bentuk diagram kelas dengan setiap subsistem ditampilkan sebagai paket dengan objek tertutup. Model subsistem adalah model struktural.

1.4. Design Models (Lanj.)

2. Model sekuens, menunjukkan urutan interaksi objek. Ini direpresentasikan menggunakan sekuens UML atau diagram kolaborasi. Model sekuens adalah model dinamis.
3. Model state mesin, yang menunjukkan bagaimana objek mengubah statusnya sebagai respons terhadap event. Ini direpresentasikan dalam UML menggunakan diagram state. Model mesin state adalah model dinamis.

1.4. Design Models (Lanj.)

- Model subsistem adalah model statis untuk menunjukkan bagaimana desain diatur ke dalam kelompok objek yang terkait secara logis, seperti pada Gambar 6.4 untuk menampilkan subsistem dalam sistem pemetaan cuaca.

1.4. Design Models (Lanj.)

- Selain model subsistem, juga dapat merancang model objek terperinci, yang menunjukkan objek dalam sistem dan asosiasinya (pewarisan, generalisasi, agregasi, dll.).

1.4. Design Models (Lanj.)

- Namun, ada bahayanya melakukan terlalu banyak pemodelan, kita tidak boleh membuat keputusan mendetail tentang penerapan yang paling baik dibiarkan sampai sistem diterapkan.

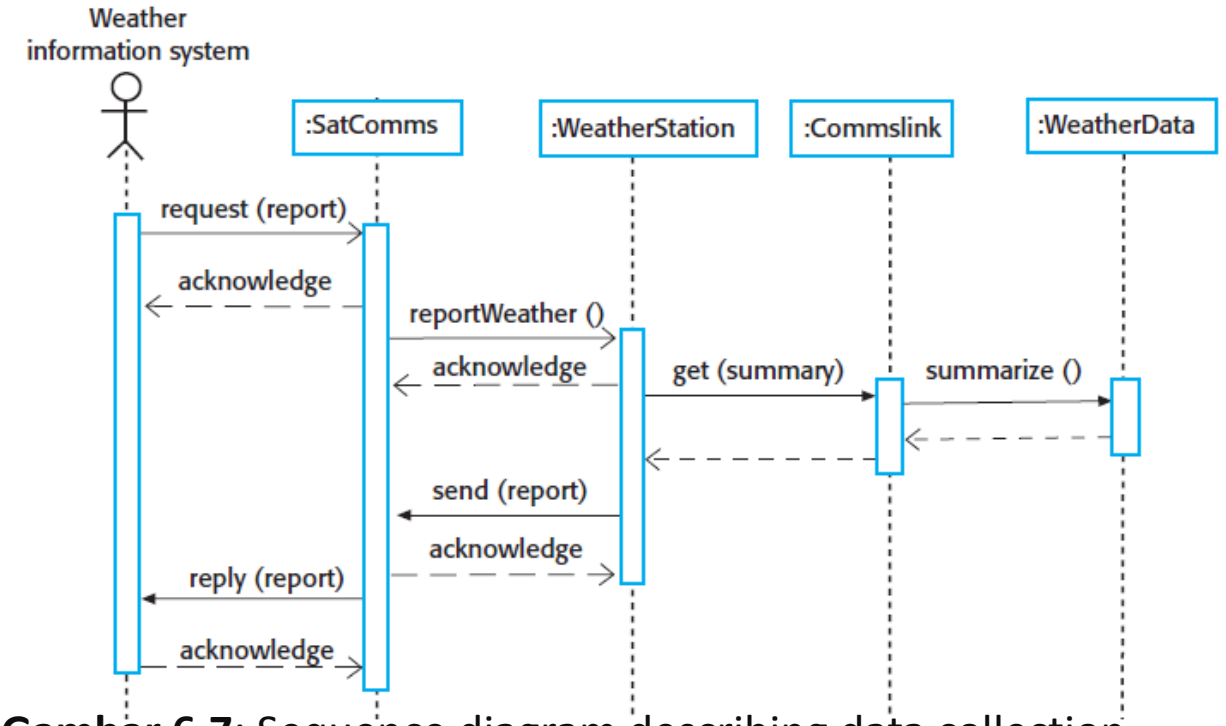
1.4. Design Models (Lanj.)

- Model sequence adalah model dinamis yang menggambarkan setiap interaksi, urutan interaksi objek yang terjadi.
- Saat mendokumentasikan desain, kita harus menghasilkan model sequence untuk setiap interaksi yang signifikan.

1.4. Design Models (Lanj.)

- Jika telah mengembangkan model use case, maka harus ada model sequence untuk setiap use case yang telah kita identifikasi.
- Gambar 6.7 adalah contoh model sekuens, diagram ini menunjukkan urutan interaksi yang terjadi ketika sistem eksternal meminta data ringkasan dari stasiun cuaca. kita membaca diagram urutan dari atas ke bawah:

1.4. Design Models (Lanj.)



Gambar 6.7: Sequence diagram describing data collection

1.4. Design Models (Lanj.)

1. Objek SatComms menerima permintaan dari sistem informasi cuaca untuk mengumpulkan laporan cuaca dari stasiun cuaca. Itu menyatakan telah menerima permintaan ini. Panah aktor pada pesan terkirim menunjukkan bahwa sistem eksternal tidak menunggu balasan tetapi dapat melanjutkan dengan pemrosesan lainnya.
2. SatComms mengirim pesan ke WeatherStation, melalui tautan satelit, untuk membuat ringkasan data cuaca yang dikumpulkan. Sekali lagi, panah pada aktor menunjukkan bahwa SatComms tidak menangguhkan dirinya sendiri menunggu jawaban.

1.4. Design Models (Lanj.)

3. WeatherStation mengirim pesan ke objek Commslink untuk meringkas data cuaca. Dalam kasus ini, gaya kepala panah persegi menunjukkan bahwa instance kelas objek WeatherStation menunggu balasan.
4. Commslink memanggil metode ringkasan dalam objek WeatherData dan menunggu balasan.
5. Ringkasan data cuaca dihitung dan dikembalikan ke WeatherStation melalui objek Commslink .
6. WeatherStation kemudian memanggil objek SatComms untuk mengirimkan data ringkasan ke sistem informasi cuaca, melalui sistem komunikasi satelit.

1.4. Design Models (Lanj.)

- SatComms dan Weatherstation objek dapat diimplementasikan bersamaan proses, yang eksekusi bisa ditunda dan dilanjutkan.
- SatComms objek misalnya mendengarkan pesan dari sistem eksternal, decode pesan-pesan ini, dan inisiasi cuaca operasi stasiun.

1.4. Design Models (Lanj.)

- Diagram sekuen digunakan untuk memodelkan perilaku gabungan dari sekelompok objek, tetapi kita mungkin juga ingin meringkas perilaku suatu objek atau subsistem dalam menanggapi pesan dan peristiwa (event).

1.4. Design Models (Lanj.)

- Untuk melakukan ini, kita dapat menggunakan model satste mesin (satechart diagam) yang menunjukkan bagaimana instance objek mengubah status bergantung pada pesan yang diterimanya.

1.4. Design Models (Lanj.)

- Gambar 6.8 adalah state diagram untuk sistem stasiun cuaca yang menunjukkan bagaimana sistem tersebut merespon permintaan untuk berbagai layanan. Kita dapat membaca diagram ini sebagai berikut:

1. Jika status sistem adalah Shutdown, maka sistem dapat merespons pesan restart(), konfigurasi ulang() atau powerSave(). Panah tanpa label dengan blob hitam menunjukkan bahwa status Shutdown adalah status awal. Pesan restart() menyebabkan transisi ke operasi normal. Baik pesan powerSave() dan reconfigure() menyebabkan transisi ke keadaan di mana sistem mengkonfigurasi ulang sendiri. State Diagram menunjukkan bahwa konfigurasi ulang hanya diperbolehkan jika sistem telah dimatikan.

1.4. Design Models (Lanj.)

2. Dalam status Running, sistem mengharapkan pesan lebih lanjut. Jika shutdown() pesan yang diterima, objek kembali ke negara shutdown.
3. Jika pesan reportWeather() diterima, sistem berpindah ke status Meringkas. Ketika selesai, sistem berpindah ke status Transmitting di mana informasi dikirim ke sistem jarak jauh. Kemudian kembali ke status Running.

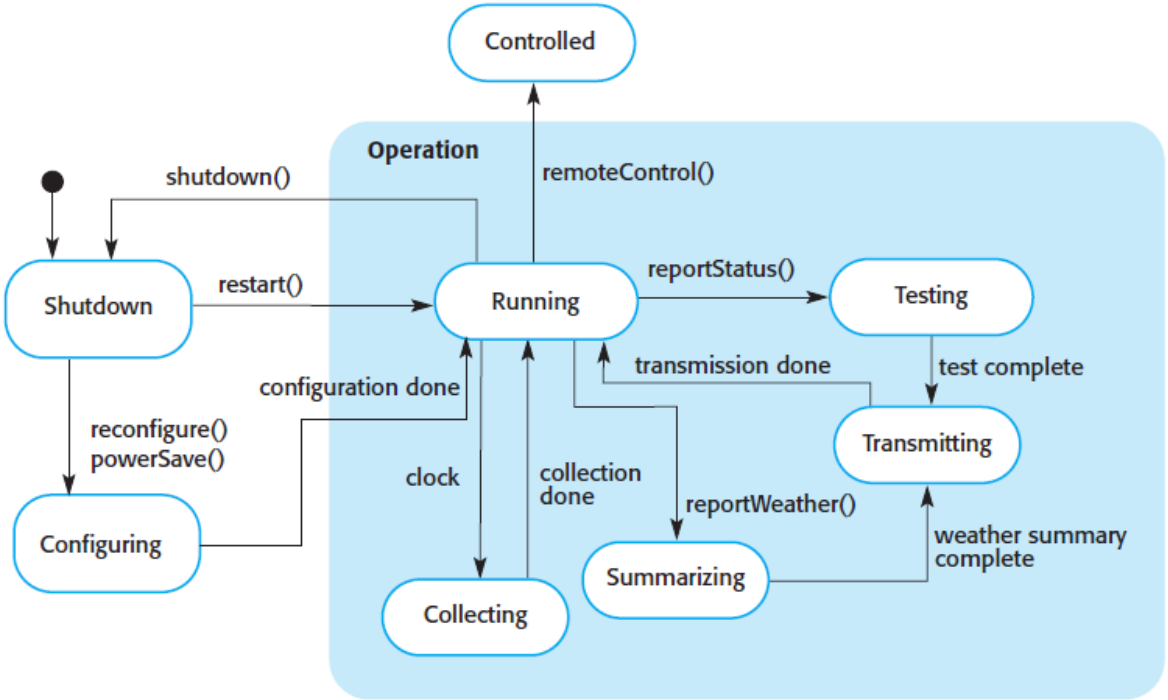
1.4. Design Models (Lanj.)

5. Jika sinyal dari jam diterima, sistem berpindah ke status Mengumpulkan, di mana ia mengumpulkan data dari instrumen. Setiap instrumen diinstruksikan secara bergiliran untuk mengumpulkan datanya dari sensor terkait.
6. Jika pesan remoteControl() diterima, sistem berpindah ke status terkontrol di mana ia menanggapi sekumpulan pesan berbeda dari ruang remote control. Ini tidak ditampilkan pada diagram ini.

1.4. Design Models (Lanj.)

- Diagram status adalah model tingkat tinggi yang berguna dari suatu sistem atau operasi objek. Namun, Anda tidak memerlukan diagram status untuk semua objek di sistem. Banyak objek sistem dalam sistem yang sederhana, dan operasinya dapat dengan mudah dijelaskan tanpa model status.

1.4. Design Models (Lanj.)



Gambar 6.8: Weather station state diagram

1.5 Interface Specification

- Bagian penting dari setiap proses desain adalah spesifikasi antarmuka antar komponen dalam desain. Kita perlu menentukan antarmuka sehingga objek dan subsistem dapat dirancang secara paralel. Setelah antarmuka ditentukan, pengembang objek lain mungkin berasumsi bahwa antarmuka akan diimplementasikan.

1.5 Interface Specification (Lanj.)

- Desain antarmuka berkaitan dengan menentukan detail antarmuka ke suatu objek atau sekelompok objek.
- Ini berarti mendefinisikan tanda tangan dan semantik dari layanan yang disediakan oleh objek atau sekelompok objek.

1.5 Interface Specification (Lanj.)

- Antarmuka dapat ditentukan dalam UML menggunakan notasi yang sama dengan diagram kelas. Namun, tidak ada bagian atribut, dan stereotip UML «antarmuka» harus disertakan di bagian nama.
- Semantik antarmuka dapat didefinisikan menggunakan bahasa kendala objek (OCL).

1.5 Interface Specification (Lanj.)

- Kita tidak boleh menyertakan detail representasi data dalam desain antarmuka, karena atribut tidak ditentukan dalam spesifikasi antarmuka.
- Namun, kita harus menyertakan operasi untuk mengakses dan memperbarui data. Karena representasi data disembunyikan, itu dapat dengan mudah diubah tanpa mempengaruhi objek yang menggunakan data tersebut.

1.5 Interface Specification (Lanj.)

- Ini mengarah pada desain yang secara inheren lebih mudah dirawat. Misalnya, representasi larik dari tumpukan dapat diubah menjadi representasi daftar tanpa memengaruhi objek lain yang menggunakan tumpukan.
- Sebaliknya, kita biasanya harus mengekspos atribut dalam model objek, karena ini adalah cara paling jelas untuk mendeskripsikan karakteristik esensial objek.

1.5 Interface Specification (Lanj.)

- Tidak ada hubungan 1:1 yang sederhana antara objek dan antarmuka.
- Objek yang sama mungkin memiliki beberapa antarmuka, yang masing-masing merupakan sudut pandang tentang metode yang disediakannya.

1.5 Interface Specification (Lanj.)

- Ini didukung langsung di Java, di mana antarmuka dideklarasikan secara terpisah dari objek dan objek "mengimplementasikan" antarmuka.
- Sama halnya, sekelompok objek semuanya dapat diakses melalui satu antarmuka.

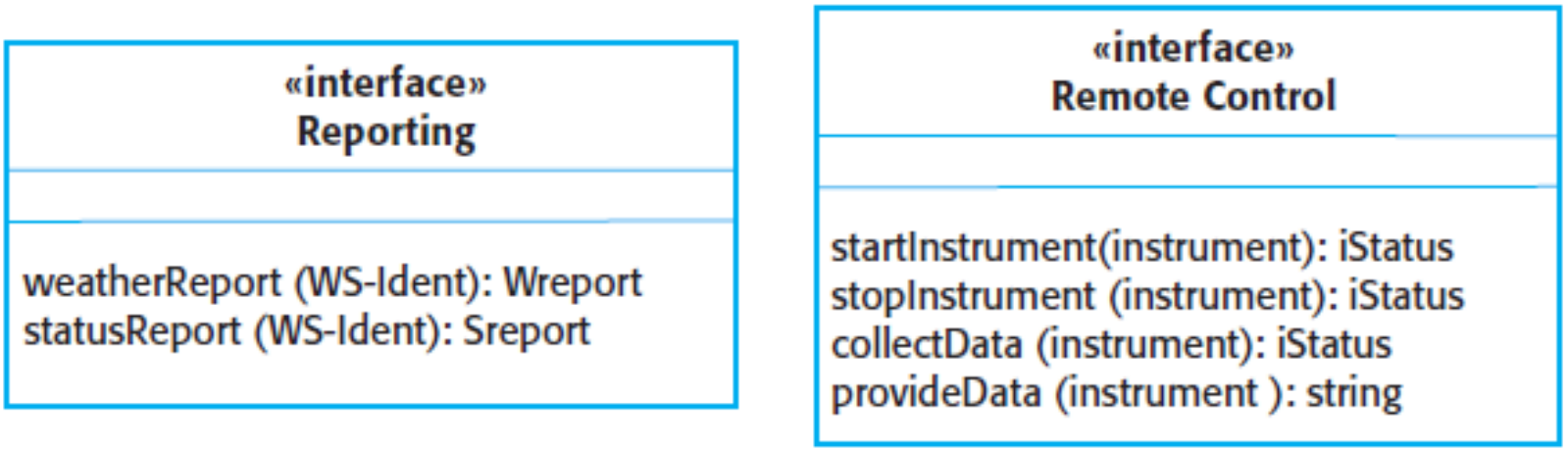
1.5 Interface Specification (Lanj.)

- Gambar 6.9 menunjukkan dua antarmuka yang dapat didefinisikan untuk stasiun cuaca.
- Antarmuka kiri adalah antarmuka pelaporan yang mendefinisikan nama operasi yang digunakan untuk menghasilkan laporan cuaca dan status.

1.5 Interface Specification (Lanj.)

- Ini memetakan langsung ke operasi di objek WeatherStation.
- Antarmuka remote control menyediakan empat operasi, yang memetakan ke satu metode di objek WeatherStation. Dalam kasus ini, operasi individu dikodekan dalam string perintah yang terkait dengan metode remoteControl, ditunjukkan pada Gambar 6.6.

1.5 Interface Specification (Lanj.)



Gambar 6.9: Weather station interfaces

2. Pola-Pola Rancangan

Pola-Pola Rancangan

Gambar 6.10: The Observer pattern

Pattern name: Observer

Description: Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.

Problem description: In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated.

This pattern may be used in situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

Solution description: This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. The abstract objects include general operations that are applicable in all situations. The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed.

The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer that allows these copies to be kept in step. The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.

The UML model of the pattern is shown in Figure 7.12.

Consequences: The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

Pola-Pola Rancangan (Lanj..)

- Pola adalah cara menggunakan kembali pengetahuan dan pengalaman desainer lain. Pola desain biasanya dikaitkan dengan desain berorientasi objek. Pola yang diterbitkan sering kali mengandalkan karakteristik objek seperti pewarisan dan polimorfisme untuk memberikan keumuman. Namun, prinsip umum merangkum pengalaman dalam suatu pola adalah prinsip yang dapat diterapkan secara setara untuk semua jenis desain perangkat lunak. Misalnya, kita bisa:

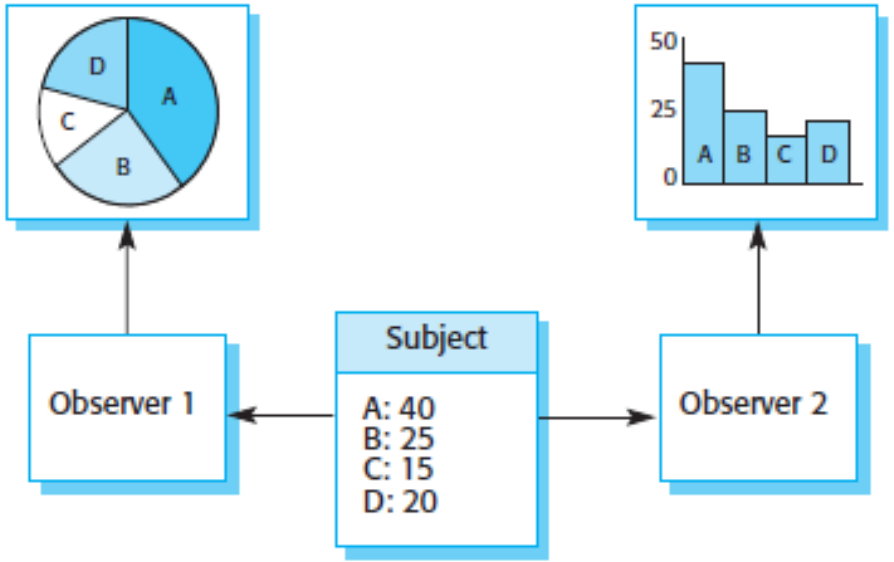
Pola-Pola Rancangan (Lanj..)

1. Nama yang merujuk pada pola tersebut.
2. Deskripsi area masalah yang menjelaskan kapan pola dapat diterapkan.
3. Deskripsi solusi dari bagian-bagian solusi desain, hubungan dan tanggung jawabnya. Ini bukan deskripsi desain yang konkret. Ini adalah template untuk solusi desain yang dapat digunakan dengan berbagai cara. Ini sering diekspresikan secara grafis dan menunjukkan hubungan antara objek dan kelas objek dalam solusi.

Pola-Pola Rancangan (Lanj..)

4. Pernyataan tentang konsekuensi — hasil dan trade-off — dari penerapan pola. Ini dapat membantu desainer memahami apakah suatu pola dapat digunakan dalam situasi tertentu atau tidak.

Pola-Pola Rancangan (Lanj..)



Gambar 6.11: Multiple displays

Pola-Pola Rancangan (Lanj..)

- Untuk mengilustrasikan deskripsi pola, menggunakan pola Observer, diambil dari buku pola Gang of Four. Ini ditunjukkan pada Gambar 6.10. Dalam uraian menggunakan empat elemen uraian penting dan juga menyertakan pernyataan singkat tentang apa yang dapat dilakukan oleh pola tersebut.

Pola-Pola Rancangan (Lanj..)

- Pola ini dapat digunakan dalam situasi di mana diperlukan presentasi yang berbeda dari suatu keadaan objek. Ini memisahkan objek yang harus ditampilkan dari berbagai bentuk presentasi. Ini diilustrasikan pada Gambar 6.11, yang menunjukkan dua presentasi grafis berbeda dari dataset yang sama.

Pola-Pola Rancangan (Lanj..)

- Representasi grafis biasanya digunakan untuk mengilustrasikan kelas objek dalam pola dan hubungannya. Ini melengkapi deskripsi pola dan menambahkan detail ke deskripsi solusi. Gambar 6.12 adalah representasi dalam UML dari pola Observer.

Pola-Pola Rancangan (Lanj..)

- Untuk menggunakan pola dalam desain perlu menyadari bahwa masalah desain apa pun yang dihadapi mungkin memiliki pola terkait yang dapat diterapkan. Contoh masalah seperti itu, yang didokumentasikan dalam buku pola asli Gang of Four, meliputi:

1. Beri tahu beberapa objek bahwa status beberapa objek lain telah berubah (pola Observer).

Pola-Pola Rancangan (Lanj..)

2. Rapikan antarmuka ke sejumlah objek terkait yang sering dikembangkan - Gambar 6.12 UML dioperasikan secara bertahap (pola Façade).
3. Sediakan cara standar untuk mengakses elemen dalam koleksi, terlepas dari bagaimana koleksi tersebut diimplementasikan (pola Iterator).
4. Memungkinkan kemungkinan untuk memperluas fungsionalitas kelas yang ada saat runtime (pola Dekorator).

Pola-Pola Rancangan (Lanj..)

- Pola mendukung penggunaan ulang konsep tingkat tinggi. Saat mencoba menggunakan kembali komponen yang dapat dijalankan pasti akan dibatasi oleh keputusan desain terperinci yang telah dibuat oleh pelaksana komponen ini.
- Menggunakan pola berarti menggunakan kembali gagasan tetapi dapat menyesuaikan penerapannya agar sesuai dengan sistem yang kembangkan.

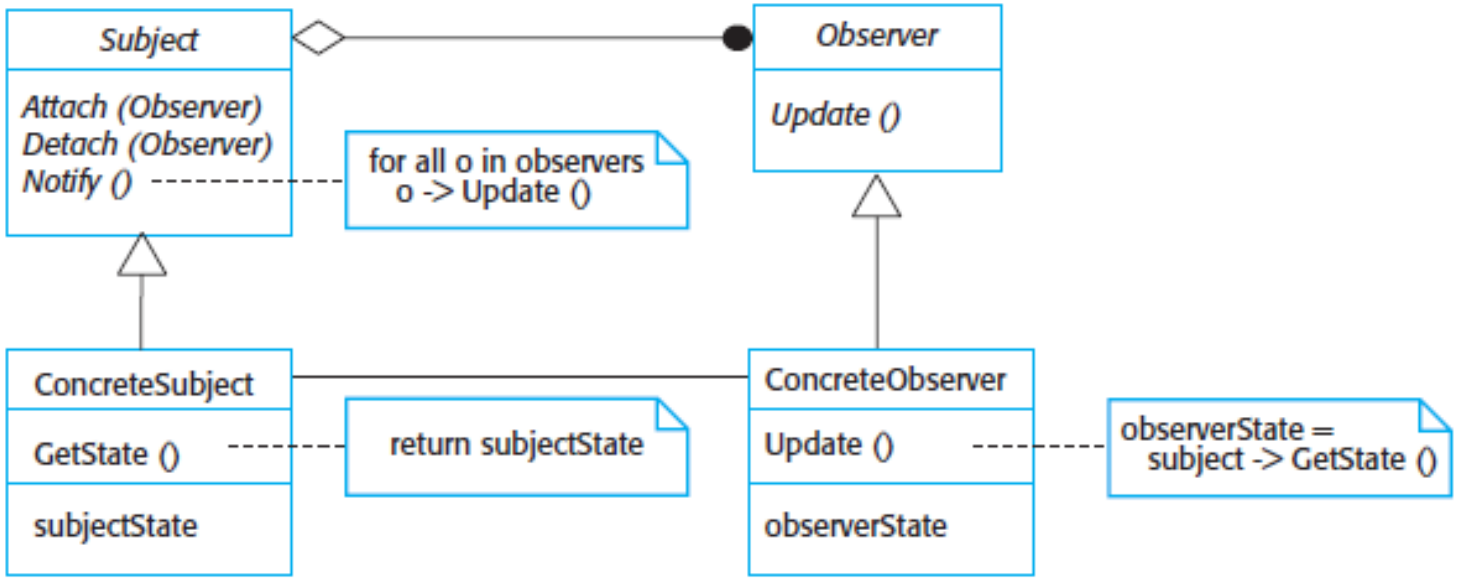
Pola-Pola Rancangan (Lanj..)

- Saat mulai mendesain sistem, mungkin sulit untuk mengetahui sebelumnya, apakah memerlukan pola tertentu.
- Oleh karena itu, menggunakan pola dalam proses desain sering kali melibatkan pengembangan desain, mengalami masalah, dan kemudian mengenali bahwa pola dapat digunakan.

Pola-Pola Rancangan (Lanj..)

- Pola adalah ide bagus, tetapi memerlukan pengalaman desain perangkat lunak untuk menggunakannya secara efektif. Harus mengenali situasi di mana suatu pola dapat diterapkan.

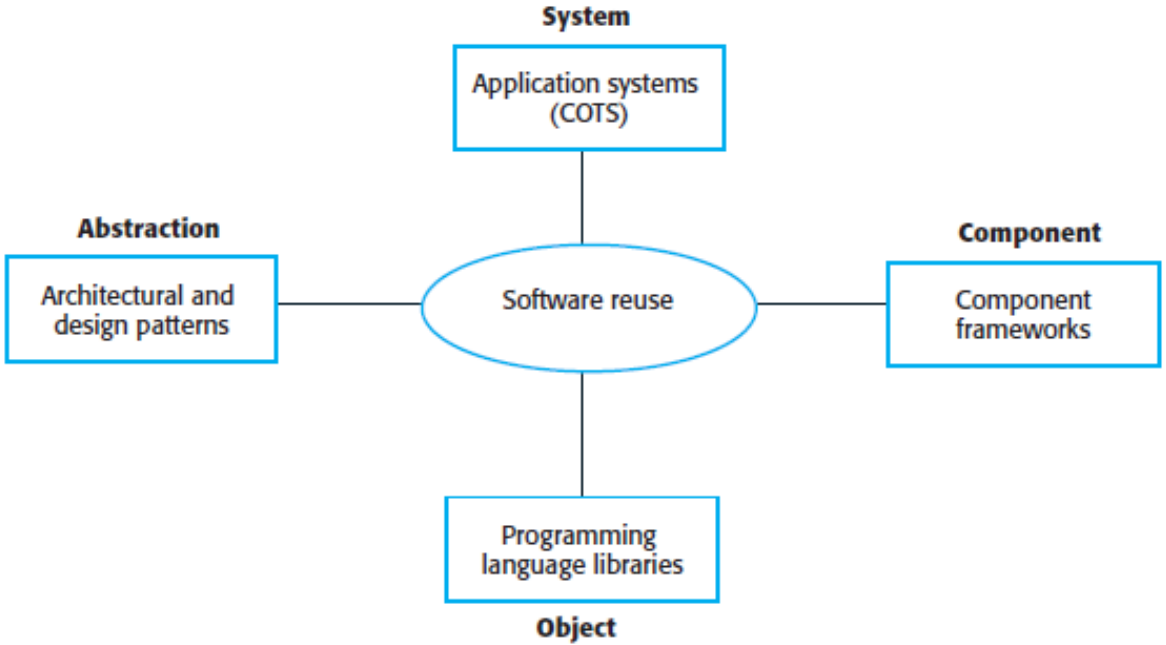
Pola-Pola Rancangan (Lanj..)



Gambar 6.12: A UML model of the Observer pattern

3. Masalah Implementasi

Masalah Implementasi



Gambar 6.13: Software Reuse

Masalah Implementasi (Lanj..)

- Rekayasa perangkat lunak mencakup semua aktivitas yang terlibat dalam pengembangan perangkat lunak dari persyaratan awal sistem hingga pemeliharaan dan pengelolaan sistem yang diterapkan.

Masalah Implementasi (Lanj..)

- Sebuah tahap kritis dari proses ini adalah, membuat versi executable dari perangkat lunak. Implementasi dapat melibatkan pengembangan program dalam bahasa pemrograman tingkat tinggi atau rendah atau menyesuaikan dan mengadaptasi sistem umum yang tersedia untuk memenuhi persyaratan khusus organisasi.

Masalah Implementasi (Lanj..)

- Materi ini dimaksudkan untuk pendekatan yang tidak bergantung pada bahasa pemrograman.
- Beberapa aspek implementasi yang sangat penting untuk rekayasa perangkat lunak adalah:

Masalah Implementasi (Lanj..)

1. *Penggunaan Kembali* Sebagian besar perangkat lunak modern dibangun dengan menggunakan kembali komponen atau sistem yang ada.
2. *Manajemen konfigurasi* Selama proses pengembangan, banyak versi berbeda dari setiap komponen perangkat lunak dibuat.
3. *Pengembangan target host* Perangkat lunak produksi biasanya tidak dijalankan pada komputer yang sama dengan lingkungan pengembangan perangkat lunak.

3.1 Reuse

- Dari tahun 1960-an hingga 1990-an, sebagian besar perangkat lunak baru dikembangkan dari awal, dengan menulis semua kode dalam bahasa pemrograman tingkat tinggi.
- Satu-satunya penggunaan kembali atau perangkat lunak yang signifikan adalah penggunaan kembali fungsi dan objek di library bahasa pemrograman.

3.1 Reuse (Lanj.)

- Namun, biaya dan tekanan jadwal membuat pendekatan ini menjadi semakin tidak dapat dijalankan, terutama untuk sistem komersial dan berbasis Internet.
- Akibatnya, pendekatan pengembangan berdasarkan penggunaan kembali perangkat lunak yang ada sekarang menjadi norma untuk banyak jenis pengembangan sistem.

3.1 Reuse (Lanj.)

- Pendekatan berbasis penggunaan kembali sekarang banyak digunakan untuk semua jenis sistem berbasis web, perangkat lunak ilmiah, dan, semakin meningkat, dalam rekayasa sistem tertanam.

3.1 Reuse (Lanj.)

- Penggunaan kembali perangkat lunak dimungkinkan pada sejumlah level yang berbeda, seperti yang ditunjukkan pada Gambar 6.13.

3.1 Reuse (Lanj.)

- Dengan menggunakan kembali perangkat lunak yang ada, dapat mengembangkan sistem baru lebih cepat, dengan risiko pengembangan yang lebih sedikit dan biaya yang lebih rendah. Karena perangkat lunak yang digunakan kembali telah diuji di aplikasi lain, perangkat lunak tersebut seharusnya lebih andal daripada perangkat lunak baru. Namun, ada biaya yang terkait dengan penggunaan kembali:

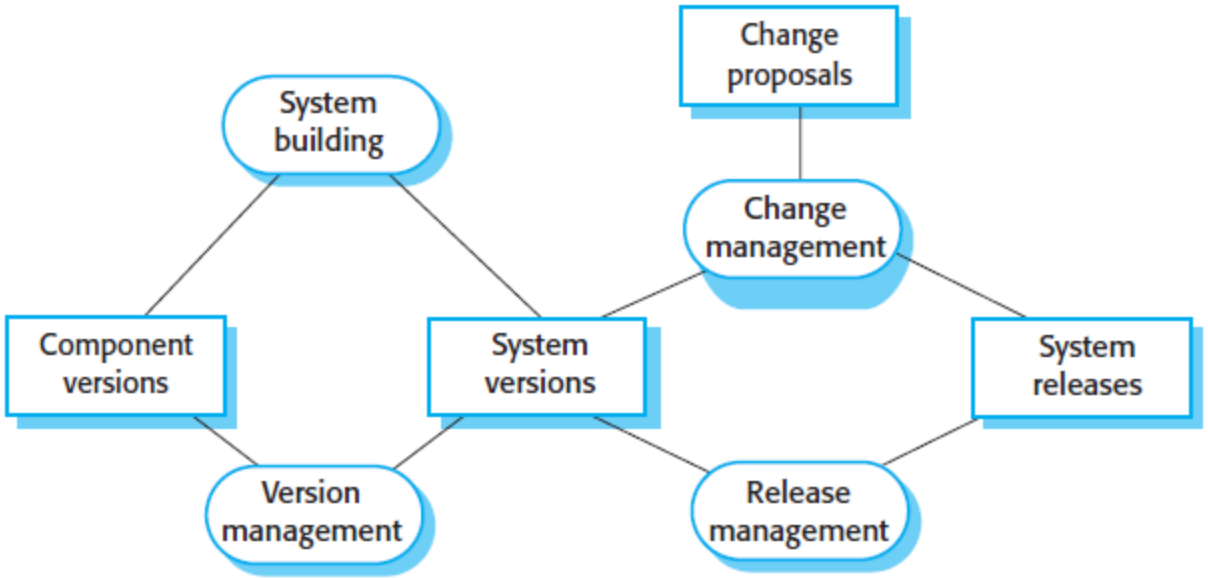
3.1 Reuse (Lanj.)

1. Biaya waktu yang dihabiskan untuk mencari perangkat lunak untuk digunakan kembali dan menilai apakah itu memenuhi kebutuhan Anda atau tidak. Anda mungkin harus menguji perangkat lunak untuk memastikan bahwa itu akan berfungsi di lingkungan Anda, terutama jika ini berbeda dari lingkungan pengembangannya.
2. Jika memungkinkan, biaya pembelian perangkat lunak yang dapat digunakan kembali. Untuk besar off-the - rak sistem, biaya ini bisa sangat tinggi.

3.1 Reuse (Lanj.)

3. Biaya adaptasi dan konfigurasi komponen perangkat lunak atau sistem yang dapat digunakan kembali untuk mencerminkan persyaratan sistem yang sedang Anda kembangkan.
4. Biaya integrasi elemen perangkat lunak yang dapat digunakan kembali satu sama lain (jika Anda menggunakan perangkat lunak dari sumber yang berbeda) dan dengan kode baru yang telah Anda kembangkan. Mengintegrasikan perangkat lunak dapat digunakan kembali dari penyedia yang berbeda dapat diffi - kultus dan mahal karena penyedia dapat membuat bertentangan asumsi tentang bagaimana perangkat lunak masing-masing akan digunakan kembali.

3.1 Reuse (Lanj.)

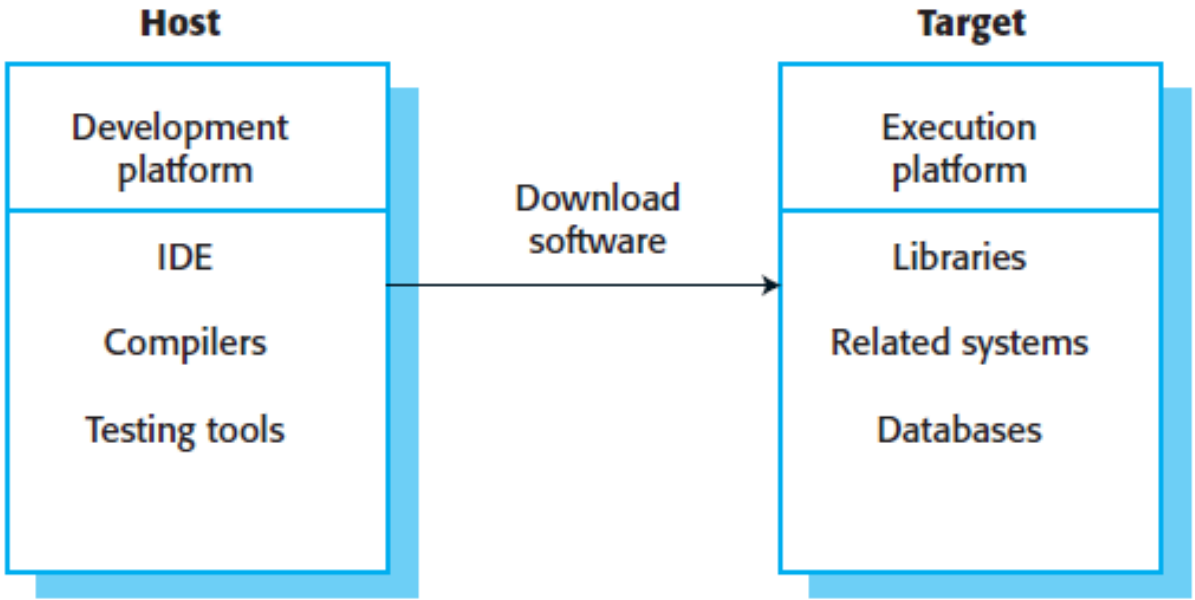


Gambar 6.14: Configuration management

3.2 Configuration Management

- Dalam pengembangan perangkat lunak, perubahan terjadi setiap saat, jadi manajemen perubahan sangat penting.
- Seperti yang ditunjukkan pada Gambar 6.14, ada empat aktivitas manajemen konfigurasi dasar:

3.2 Configuration Management (lanj.)



Gambar 6.15: Host-target development

3.3 Host-Target Development

- Sebagian besar pengembangan perangkat lunak profesional didasarkan pada model target-host (Gambar 6.15).
- Perangkat lunak dikembangkan pada satu komputer (host) tetapi berjalan pada mesin terpisah (target).

3.3 Host-Target Development (Lanj.)

- Secara lebih umum, kita dapat berbicara tentang platform pengembangan (host) dan platform eksekusi (target).
- Sebuah platform lebih dari sekedar perangkat keras. Ini mencakup sistem operasi yang terpasang ditambah perangkat lunak pendukung lainnya seperti database mengelola - sistem ment atau, untuk platform pengembangan, sebuah lingkungan pengembangan interaktif.

3.3 Host-Target Development (Lanj.)

- Jika sistem target telah menginstal middleware atau perangkat lunak lain yang perlu gunakan, maka harus dapat menguji sistem menggunakan perangkat lunak itu.
- Sebuah platform pengembangan perangkat lunak harus menyediakan berbagai alat untuk dukungan perangkat lunak proses rekayasa. Ini mungkin termasuk:

3.3 Host-Target Development (Lanj.)

1. Kompilator terintegrasi dan sistem pengeditan yang diarahkan sintaks yang memungkinkan membuat, mengedit, dan menyusun kode.
2. Sistem debugging bahasa.
3. Alat pengeditan grafis, seperti alat untuk mengedit model UML.

3.3 Host-Target Development (Lanj.)

4. Alat pengujian, seperti JUnit , yang dapat secara otomatis menjalankan serangkaian pengujian pada versi baru program.
5. Alat untuk mendukung refactoring dan visualisasi program.
6. Alat manajemen konfigurasi untuk mengelola versi kode sumber dan untuk mengintegrasikan dan membangun sistem.

3.3 Host-Target Development (Lanj.)

- Sebagai bagian dari proses pengembangan, perlu membuat keputusan tentang bagaimana perangkat lunak yang dikembangkan akan diterapkan pada platform target.
- Ini mudah untuk sistem tertanam, di mana targetnya biasanya satu komputer.
- Masalah yang harus dipertimbangkan dalam mengambil keputusan ini adalah:

3.3 Host-Target Development (Lanj.)

1. Persyaratan perangkat keras dan perangkat lunak.
2. Persyaratan ketersediaan sistem.
3. Komunikasi komponen.

3.3 Host-Target Development (Lanj.)

- Kita dapat mendokumentasikan keputusan tentang penyebaran perangkat keras dan perangkat lunak menggunakan diagram penyebaran UML, yang menunjukkan bagaimana komponen perangkat lunak didistribusikan di seluruh platform perangkat keras.

4. Pengembangan Open Source

Pengembangan Open Source

- Pengembangan open-source adalah sebuah pendekatan untuk pengembangan perangkat lunak di mana kode sumber dari perangkat lunak sistem diberikan dan pengguna diundang untuk berpartisipasi.
- Ada asumsi bahwa kode tersebut akan dikontrol dan dikembangkan oleh kelompok inti kecil, bukan pengguna kode.

Pengembangan Open Source (Lanj..)

- Perangkat lunak sumber terbuka memperluas gagasan ini dengan menggunakan Internet untuk merekrut populasi pengembang sukarela yang jauh lebih besar.
- Banyak dari mereka juga merupakan pengguna kode.

Pengembangan Open Source (Lanj..)

- Pada prinsipnya, setiap kontributor proyek open-source dapat melaporkan dan memperbaiki bug serta mengusulkan fitur dan fungsionalitas baru.
- Namun, dalam praktiknya open-source masih bergantung pada sekelompok pengembang inti yang mengontrol perubahan pada perangkat lunak.

Pengembangan Open Source (Lanj..)

- Perangkat lunak open-source adalah tulang punggung Internet dan rekayasa perangkat lunak.
- Sistem operasi Linux adalah sistem server yang paling banyak digunakan, seperti juga server web Apache sumber open-source.

Pengembangan Open Source (Lanj..)

- Produk open-source umum lainnya adalah Java, Eclipse IDE, dan sistem manajemen basis data mySQL.
- Sistem operasi Android diinstal pada jutaan perangkat seluler. IBM dan Oracle, mendukung gerakan open-source dan mendasarkan perangkat lunak mereka pada produk open source.

Pengembangan Open Source (Lanj..)

- Biasanya murah atau bahkan gratis untuk memperoleh perangkat lunak open source.
- Manfaat utama lainnya dari penggunaan produk open source adalah bahwa sistem open source yang banyak digunakan sangat andal.

Pengembangan Open Source (Lanj..)

- Mereka memiliki pengguna yang besar yang bersedia memperbaiki masalah sendiri daripada melaporkan masalah ini ke pengembang dan menunggu rilis baru dari sistem.
- Bug ditemukan dan diperbaiki lebih cepat daripada biasanya dibandingkan dengan perangkat lunak berbayar (license).

Pengembangan Open Source (Lanj..)

- Untuk perusahaan pengembangan perangkat lunak, ada dua masalah open source yang harus dipertimbangkan:
  1. Haruskah produk yang sedang dikembangkan menggunakan komponen open source?
  2. Haruskah pendekatan open source digunakan untuk pengembangan perangkat lunaknya sendiri?

Pengembangan Open Source (Lanj..)

- Jawaban atas pertanyaan-pertanyaan ini tergantung pada jenis perangkat lunak yang sedang developed dan latar belakang dan pengalaman dari tim pengembangan.

4.1 Open-Source Licensing

- Meskipun prinsip dasar pengembangan open source adalah bahwa kode sumber harus tersedia secara bebas, ini tidak berarti bahwa siapa pun dapat melakukan apa yang mereka inginkan dengan kode itu.

4.1 Open-Source Licensing (Lanj.)

- Secara hukum, pengembang kode (baik perusahaan atau individu) memiliki kode. Mereka dapat membatasi cara penggunaannya dengan memasukkan ketentuan yang mengikat secara hukum dalam lisensi perangkat lunak sumber terbuka.

4.1 Open-Source Licensing (Lanj.)

- Beberapa pengembang open source percaya bahwa jika komponen open source digunakan untuk mengembangkan sistem baru, maka sistem tersebut juga harus open source.

4.1 Open-Source Licensing (Lanj.)

- Orang lain bersedia mengizinkan kode mereka digunakan tanpa batasan ini. Sistem yang dikembangkan mungkin merupakan hak milik dan dijual sebagai sistem sumber tertutup.

4.1 Open-Source Licensing (Lanj.)

- Sebagian besar lisensi open source (Chapman 2010) merupakan varian dari salah satu dari tiga model umum:
  1. Lisensi Publik Umum GNU (GPL). Ini adalah apa yang disebut lisensi timbal balik yang secara sederhana berarti bahwa jika Anda menggunakan perangkat lunak open source yang dilisensikan di bawah lisensi GPL, maka Anda harus menjadikan perangkat lunak itu sebagai open source.

4.1 Open-Source Licensing (Lanj.)

2. Lisensi Publik Umum Lesser GNU (LGPL). Ini adalah varian dari lisensi GPL tempat Anda dapat menulis komponen yang ditautkan ke kode open source tanpa harus menerbitkan sumber komponen ini. Namun, jika Anda mengubah komponen berlisensi, maka Anda harus menerbitkannya sebagai open source.
3. Lisensi Berkley Standard Distribution (BSD). Ini adalah lisensi nonreciprocal, yang berarti Anda tidak berkewajiban untuk mempublikasikan ulang setiap perubahan atau modifikasi yang dibuat

4.1 Open-Source Licensing (Lanj.)

- Kode open source Anda dapat memasukkan kode dalam sistem berpemilik yang dijual. Jika Anda menggunakan komponen open source, Anda harus mengetahui pembuat asli kode tersebut. Lisensi MIT adalah varian dari lisensi BSD dengan ketentuan yang serupa.

4.1 Open-Source Licensing (Lanj.)

- Masalah lisensi penting karena jika Anda menggunakan perangkat lunak open source sebagai bagian dari produk perangkat lunak, maka Anda mungkin diwajibkan oleh persyaratan lisensi untuk membuat produk Anda sendiri open source.

4.1 Open-Source Licensing (Lanj.)

- Jika Anda membangun perangkat lunak yang berjalan pada platform open source tetapi tidak menggunakan kembali komponen open source, maka lisensi tidak menjadi masalah.
- Namun, jika Anda menanamkan software open-source dalam perangkat lunak Anda, Anda perlu proses dan basis data untuk melacak apa yang telah digunakan dan kondisi lisensi mereka.

4.1 Open-Source Licensing (Lanj.)

- Bayersdorfer menyarankan perusahaan yang mengelola proyek menggunakan open source harus:
  1. Buat sistem untuk memelihara informasi tentang komponen sumber terbuka yang diunduh dan digunakan. Anda harus menyimpan salinan lisensi untuk setiap komponen yang valid pada saat komponen itu digunakan. Lisensi dapat berubah, jadi Anda perlu mengetahui ketentuan yang telah Anda setujui.

4.1 Open-Source Licensing (Lanj.)

2. Ketahuilah berbagai jenis lisensi dan pahami bagaimana komponen dilisensikan sebelum digunakan. Anda dapat memutuskan untuk menggunakan komponen di satu sistem tetapi tidak di sistem lain karena Anda berencana menggunakan sistem ini dengan cara yang berbeda.
3. Waspadai jalur evolusi untuk komponen. Anda perlu tahu sedikit tentang proyek sumber terbuka tempat komponen dikembangkan untuk memahami bagaimana mereka dapat berubah di masa depan.

4.1 Open-Source Licensing (Lanj.)

4. Mendidik orang tentang open source. Tidaklah cukup hanya memiliki prosedur untuk memastikan kepatuhan dengan ketentuan lisensi. Anda juga perlu mendidik devel - opers tentang open source dan lisensi open source.
5. Memiliki sistem audit. Pengembang, dengan tenggat waktu yang ketat, mungkin tergoda untuk melanggar persyaratan lisensi. Jika memungkinkan, Anda harus memiliki perangkat lunak untuk mendeteksi dan menghentikan ini.

4.1 Open-Source Licensing (Lanj.)

6. Berpartisipasilah dalam komunitas sumber terbuka. Jika Anda mengandalkan produk open-source, Anda harus berpartisipasi dalam komunitas dan membantu mendukung perkembangannya.

4.1 Open-Source Licensing (Lanj.)

- Pendekatan open-source adalah salah satu dari beberapa model bisnis untuk perangkat lunak.
- Dalam model ini, perusahaan merilis sumber perangkat lunak mereka dan menjual layanan tambahan serta saran terkait hal ini.

4.1 Open-Source Licensing (Lanj.)

- Mereka mungkin juga menjual software berbasis cloud, sebuah pilihan yang menarik bagi pengguna yang tidak memiliki keahlian untuk mengelola sistem open-source mereka sendiri.

Ringkasan

- Desain dan implementasi perangkat lunak adalah aktivitas yang saling terkait. Tingkat detail dalam desain bergantung pada jenis sistem yang dikembangkan dan apakah Anda menggunakan pendekatan yang digerakkan oleh rencana atau agile.

Ringkasan (Lanj.)

- Proses desain berorientasi objek meliputi aktivitas merancang arsitektur sistem, mengidentifikasi objek dalam sistem, mendeskripsikan desain menggunakan model objek yang berbeda, dan mendokumentasikan antarmuka komponen.

Ringkasan (Lanj.)

- Berbagai model yang berbeda dapat diproduksi selama proses desain berorientasi objek. Ini termasuk model statis (model kelas, model generalisasi, model asosiasi) dan model dinamis (sequential model, state model).

Ringkasan (Lanj.)

- Antarmuka komponen harus didefinisikan secara tepat agar objek lain dapat menggunakannya. Stereotipe antarmuka UML dapat digunakan untuk mendefinisikan antarmuka.

Ringkasan (Lanj.)

- Ketika mengembangkan perangkat lunak, Anda harus selalu mempertimbangkan kemungkinan menggunakan kembali perangkat lunak yang ada, baik sebagai komponen, atau sistem lengkap.

Ringkasan (Lanj.)

- Manajemen konfigurasi adalah proses mengelola perubahan pada sistem perangkat lunak yang berkembang. Ini penting ketika tim bekerja sama untuk mengembangkan perangkat lunak.

Ringkasan (Lanj.)

- Kebanyakan pengembangan perangkat lunak adalah pengembangan target-host. Anda menggunakan IDE pada mesin host untuk mengembangkan perangkat lunak, yang ditransfer ke mesin target untuk dieksekusi.

Ringkasan (Lanj.)

- Pengembangan open source melibatkan pembuatan kode sumber dari sistem yang tersedia untuk umum. Artinya, banyak orang yang dapat mengusulkan perubahan dan peningkatan perangkat lunak.

Studi Kasus

- Silahkan lanjutkan membreakdown studi kasus pada materi 5 dengan prototype model.

Referensi

- Presman, Maxim, Software Engineering, USA.
- Sommerville Ian, 2016, Software Engineering, USA, PEARSON.
- Timoty C, Robert, Object Oriented Software Engineering, USA, Ms Graw Hill.
- Rosa A.S, Rekayasa Perangkat Lunak, Bandung, Penerbit Informatka.

TERIMA KASIH