**Pengertian DSM**

Pemodelan Domain-Special terutama bertujuan untuk melakukan dua hal. Pertama, tingkatkan abstraksi di luar pemrograman dengan menentukan solusinya dalam bahasa yang secara langsung menggunakan konsep dan aturan dari domain masalah yang spesifik. Kedua, menghasilkan produk akhir dalam bahasa pemrograman yang dipilih atau bentuk lain dari spesifikasi tingkat tinggi ini. Biasanya pembuatan kode lebih lanjut didukung oleh kode kerangka yang menyediakan implementasi atom umum untuk aplikasi dalam domain. Otomatisasi pengembangan aplikasi yang lebih luas dimungkinkan karena bahasa pemodelan, kode generator, dan kode kerangka kerja memerlukan persyaratan dari domain aplikasi yang sempit. Dengan kata lain, mereka adalah domainspeci fi c dan sepenuhnya berada di bawah kendali pengguna mereka.

**Prinsip Dasar DSM Yang Berisi Definisi**

Pemodelan Domain-Special secara substansial meningkatkan tingkat abstraksi sementara pada saat bersamaan mempersempit ruang perancangan, seringkali pada satu rangkaian produk untuk satu perusahaan. Dengan bahasa DSM, masalahnya dipecahkan hanya sekali dengan memodelkan secara visual solusi dengan hanya menggunakan konsep domain yang familiar. Produk akhir kemudian dihasilkan secara otomatis dari spesifikasi tingkat tinggi ini dengan generator kode domain-spesifik. Dengan DSM, tidak ada lagi kebutuhan untuk membuat pemetaan yang rawan kesalahan dari konsep domain hingga konsep desain dan konsep bahasa pemrograman. Dalam pengertian ini, DSM mengikuti resep yang sama yang membuat bahasa pemrograman berhasil di masa lalu: menawarkan tingkat abstraksi yang lebih tinggi dan membuat pemetaan otomatis dari konsep tingkat yang lebih tinggi ke konsep tingkat rendah yang diketahui dan digunakan sebelumnya. Saat ini, DSM menyediakan cara untuk terus meningkatkan deskripsi perangkat lunak ke tingkat yang lebih abstrak. Abstraksi yang lebih tinggi ini tidak didasarkan pada konsep pengkodean saat ini atau pada konsep tujuan umum namun pada konsep yang spesifik untuk setiap domain aplikasi.

**Perbedaan UML dan DSM**

* DSM tampil lebih baik dibandingkan dengan UML atau bahasa umum lainnya. Misalnya, waktu untuk mengembangkan aplikasi menggunakan DSM adalah sebagian kecil dari waktu untuk membuat diagram UML dan menulis kodenya secara manual.
* DSM juga mencegah kesalahan sejak awal karena bahasa tersebut tidak mengizinkan desain ilegal atau 14 desain PENDAHULUAN yang tidak mengikuti aturan arsitektur yang mendasarinya.
* Kode yang dihasilkan juga bebas dari kebanyakan jenis kesalahan, sintaks, dan kesalahan logika yang ceroboh karena pengembang yang lebih berpengalaman telah menentukan kode generator. Kami memeriksa pengaruh pengembangan DSM terhadap produktivitas, kualitas produk, dan keahlian pengembang yang dibutuhkan secara lebih rinci pada bab berikutnya. Salah satu contohnya jelas tidak bisa mencakup spektrum yang luas dari situasi pengembangan dan domain aplikasi yang mungkin terjadi. Contoh mobile adalah satu dari ribuan, jika tidak ratusan ribu, dari berbagai domain aplikasi. Pada prinsipnya, solusi DSM yang berbeda dapat digunakan dalam semuanya karena setiap proyek pembangunan perlu menemukan solusi dalam masalah domain dan memetakannya ke dalam implementasi di domain solusi. Semua proyek pembangunan menghadapi tantangan yang sama. Pada Bagian III buku ini, kami memeriksa kasus DSM yang berbeda dan bertujuan untuk menjangkau lebih banyak domain masalah yang menghasilkan berbagai jenis kode, mulai dari Assembler hingga 3GL dan berorientasi objek hingga bahasa scripting dan XML.

**Keuntungan DSM:**

* Tingkat abstraksi yang rendah mengarah pada model yang menggambarkan kode. Hal ini terbukti juga pada model yang menggambarkan perilaku: contoh tipikal adalah menunjukkan nama metode langsung, parameter, variabel, dan faktor-faktor lain dalam produk. Tingkatan ini tidak banyak diminati karena model hanya memetakan beberapa domain yang sudah mendekati persyaratan pengkodean teknis. Juga kebanyakan pengembang yang bekerja di tingkat kode sering menemukannya lebih baik untuk menulis struktur yang sama secara langsung dalam bahasa pemrograman daripada menggunakan UML dan bahasa tekstual tambahan.
* DSM pada dasarnya menggabungkan langkah satu dan dua dan mengotomatisasi langkah ketiga sepenuhnya. Tingkat abstraksi yang meningkat ditambah dengan kode yang dihasilkan secara otomatis adalah tujuan Pemodelan Khusus Domain. DSM tidak berharap bahwa semua kode dapat dihasilkan dari model, namun apa pun yang dimodelkan dari perspektif pemodel, menghasilkan kode akhir yang lengkap. Kelengkapan transformasi ini telah menjadi tonggak otomasi dan meningkatkan abstraksi di masa lalu. Dalam DSM, kode yang dihasilkan bersifat fungsional, mudah dibaca, dan efisien - idealnya terlihat seperti kode tulisan tangan oleh pengembang berpengalaman yang menentukan generator.

Manfaat DSM tersedia saat solusi DSM - bahasa dan generator - tersedia. Biasanya bukan itu yang kita butuhkan untuk mengembangkan solusi DSM terlebih dahulu. Pada Bagian IV kami menunjukkan bagaimana mendefinisikan bahasa pemodelan dan generator kode untuk domain aplikasi. Panduan ini didasarkan pada pengalaman kami membangun pengembangan berbasis model dengan DSM di berbagai domain yang berbeda menghasilkan kode target yang berbeda untuk lingkungan target yang berbeda.

**Apa Kelemahan Dasar DSM**

1. Biaya Membangun Solusi DSM

Biaya Membangun Solusi DSM Manfaat DSM tidak gratis; seseorang harus membuat solusi DSM. Ini adalah investasi awal. Biaya penggunaan DSM lebih besar pada awalnya daripada saat melanjutkan Praktik saat ini, yang tidak memerlukan investasi tambahan. Biaya awal pembangunan DSM biasanya terdiri dari hampir seluruhnya dari SDM, yaitu pakar domain dan pengembang berpengalaman.

1. Biaya Pemeliharaan dan Pengembangan

Setelah solusi DSM sedang digunakan, biaya tambahan terjadi untuk perawatannya. Biaya ini relatif terhadap perubahan dibuat dalam larutan DSM. Jika domainnya statis dan perusahaan terus menggunakan solusi DSM asli tidak berubah-yang biasanya jarang-biaya perawatannya minimal.

1. Biaya Menunggu

Kita perlu mempertimbangkan juga biayanya menunggu: manfaatnya hilang karena solusi DSM belum tersedia.

1. Ketiadaan control pada message-passing dasar.Lengkapi parameter pada rutin DSM untuk dapat mengontrol message-passing

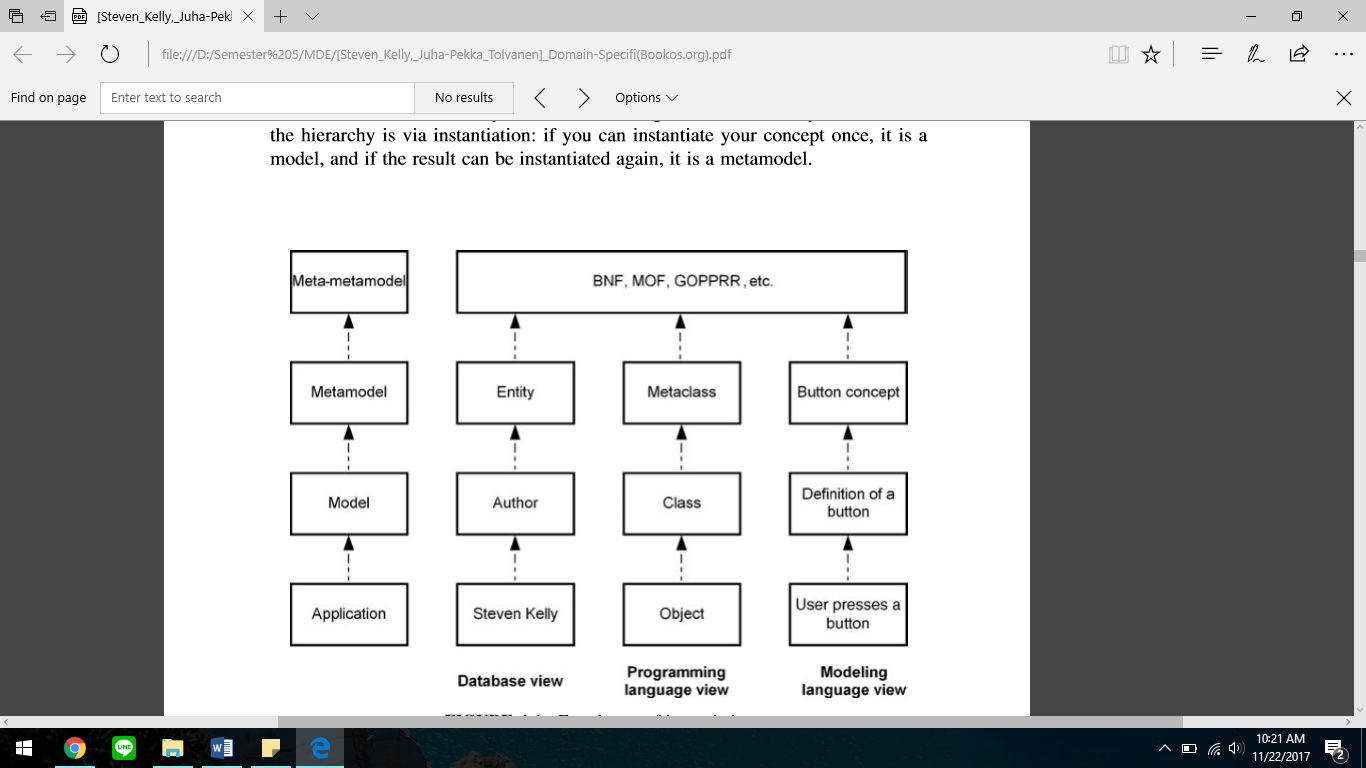
**Hal yang harus di perhatikan :**

Sebagian besar konsep dalam bahasa pemodelan datang dengan aturan. Aturan ini harus dikenali secara alami oleh bahasa juga. Jenis aturan yang dapat Anda tentukan bergantung pada alat DSM dan bahasa metamodeling yang diterapkan. Kami membahas peraturan di sini secara independen dari banyak pelajaran, namun sebagian besar pemikiran tentang bahasa ibu yang sesuai dengan yang berikut:

* Konvensi penamaan, misalnya, nilai harus dimulai dengan huruf besar atau tidak boleh menyertakan karakter tertentu.
* Keunikan, misalnya, tidak ada unsur lain dengan nilai properti yang sama.
* Wajib sifat memaksa elemen untuk memiliki nilai.
* Nilai default
* Kejadian: sebuah konsep hanya dapat memiliki sejumlah contoh dalam sebuah model.
* Aturan mengikat yang mengatur unsur mana yang bisa dihubungkan bersama.
* Konektivitas aturan yang menyatakan berapa kali sebuah objek mungkin memiliki jenis koneksi tertentu.
* Aturan penggunaan kembali yang menyatakan bahwa seorang modeler dapat memilih nilai tertentu atau mengacu pada elemen model yang lain.
* Aturan hubungan N-ary menyebutkan berapa banyak benda yang bisa dihubungkan oleh satu hubungan.
* Mengintegrasikan model, seperti berbagi nilai yang sama dengan elemen lain, mungkin di model lain, dan mungkin dibuat dengan bahasa lain.
* Aturan penataan model, seperti hierarki atau referensi ke perpustakaan

Anda dapat mengidentifikasi peraturan ini dari sumber yang sama dengan Anda mengidentifikasi konsep domain. Namun, peraturan bahasa yang rinci seringkali tidak dapat dideteksi dari materi yang ada, dan cara tercepat adalah dengan memeriksanya langsung dengan pakar domain. Anda mungkin juga melihat jenis aturan yang digunakan model perhitungan yang dipilih.

**Metamodel:**

Dalam DSM, bahasa pemodelan harus didefinisikan secara formal dan didukung oleh beberapa alat. Jika tidak, tidak mungkin membuat model dan menghasilkan kode dari mereka. Spesifikasi bahasa biasanya disebut metamodel. **Kata "meta" digunakan karena spesifikasi bahasa satu tingkat lebih tinggi dari model biasa. Dalam bentuknya yang paling sederhana, kita dapat mengatakan bahwa metamodel adalah model konseptual pemodelan** Ini menggambarkan konsep bahasa, sifat mereka, hubungan hukum antara peraturan bahasa, struktur modelisasi, dan aturan ketidakmampuan. Dalam semua kasus paling sederhana, dukungan untuk penggunaan kembali dan pendekatan integrasi model yang berbeda juga penting. Desain dan definisi bahasa juga mencakup

Tugas metamodeling: memetakan konsep domain ke berbagai elemen bahasa, seperti objek, sifat mereka, dan hubungannya, yang ditentukan sebagai hubungan dan peran yang dimainkan objek. Kata kunci "meta" berarti pemodelan (meta) di bagian bawah abstraksi dan logika lebih tinggi dari pemodelan biasa dalam pengembangan perangkat lunak. Sebuah catatan yang tidak dapat dipahami tentangmetamodeling dapat ditemukan diJarkeetal. (1998).

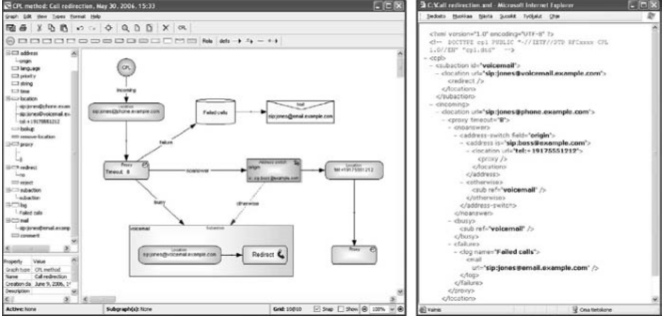
Alat berbasis metamodel mengikuti arsitektur tiga level. Tingkat terendah, tingkat model, sama dengan alat CASE. Ini termasuk desain sistem sebagai model. Tingkat menengah berisi model bahasa, yaitu metamodel. **Metamodel mencakup konsep, aturan, dan notasi diagram dari bahasa tertentu.** Misalnya, metamodel dapat menentukan konsep seperti "kasus penggunaan" dan "aktor", bagaimana kaitannya, dan bagaimana hal itu diwakili. Namun, alih-alih dimasukkan ke dalam kode dalam alat, seperti pada alat CASE yang diperbaiki, bahasa pemodelan disimpan sebagai data dalam alat.

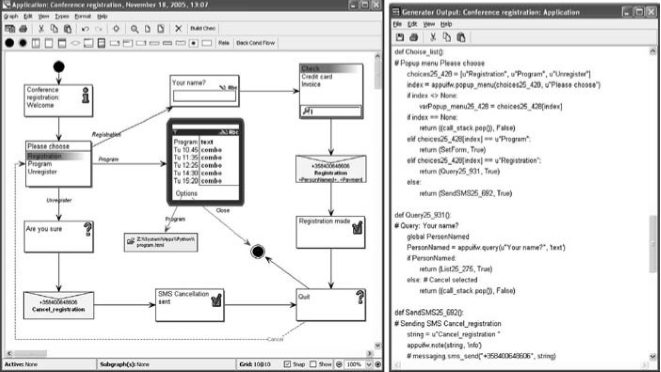
Tidak seperti UML atau CASE tool, metamodel mengizinkan pengguna untuk mengakses danmemodifikasi spesifikasi bahasa. Hal ini dicapai dengan memiliki tingkat ketiga yang lebih tinggi yang mencakup bahasa metamodeling untuk menentukan bahasa pemodelan. Tingkat ini biasanya merupakan bagian "hard-coded" dari alat metamodeling. Ketiga tingkatan itu saling terkait: model didasarkan pada metamodel, yang pada gilirannya didasarkan pada bahasa metamodeling. Jelas, tidak ada pemodelan yang mungkin dilakukan tanpa semacam metamodel. Struktur ketergantungan ini serupa dengan objek, kelas, dan metaclass pada beberapa bahasa pemrograman berorientasi objek.

**Model :**

Di DSM, model adalah sumber utama pengembang membuat, mengedit, dan menghapus spesifikasi sistem. Meskipun perubahan pada kerangka bahasa, generator, dan domain dimungkinkan, kebanyakan pengembang biasanya berfokus pada pemodelan. Model spesifik domain kemudian digunakan untuk menghasilkan kode secara langsung. Di DSM, kita harus menghindari situasi di mana model diterjemahkan ke model lain untuk pengeditan di masa depan. Pendekatan ini tidak bekerja di masa lalu dengan kode dan tidak akan bekerja dengan model. Gunakan internal ke alat sebagai produk antara adalah cerita yang berbeda. Tapi sebaiknya jangan memodifikasi hasil transformasi atau proses pembangkitan. Everymodel didasarkan pada beberapa bahasa tersirat secara implisit atau eksplisit. Secara umum, model yang diungkapkan dalam DSM memiliki karakteristik yang serupa dengan model lainnya. Ada beberapa perbedaan penting. Di DSM, pemodel bekerja dengan menggunakan konsep domain. Ini diberikan oleh bahasa yang benar. Model DSM juga formal, berdasarkan abstraksi yang lebih tinggi daripada coding, mengikuti aturan domain, dan didasarkan pada konsep yang familiar bagi pengembang yang bekerja di domain target bahasa.

**Contoh DSM beserta gambarnya:**

Contoh 1: IP telephony and call processing

Contoh 2 : Mobile phone applications using a Python framework

**Arsitektur Dasar DSM**

1. Bahasa

Bahasa menyediakan abstraksi untuk pengembangan dan karena itu adalah bagian yang paling terlihat bagi pengembang. Di DSM, digunakan untuk membuat spesifikasi yang diperlakukan programmer manual sebagai kode sumber.

1. Model

Di DSM, model adalah sumber utama pengembang membuat, mengedit, dan menghapus spesifikasi sebuah sistem. Meski perubahan bahasa, generator, dan Kerangka domain dimungkinkan, sebagian besar pengembang biasanya fokus pada pemodelan.

1. Generator Kode

Di DSM, generator kode mengubah model menjadi kode untuk interpretasi atau kompilasi menjadi executable. Dengan memberikan otomasi, mereka berkontribusi terhadap produktivitas dan peningkatan kualitas pendekatan DSM. Kode yang dihasilkan biasanya lengkap dari perspektif pemodel.

1. Kerangka Kerja Domain Dan Lingkungan Target

Kerangka domain menyediakan antarmuka antara kode yang dihasilkan dan lingkungan sasaran yang mendasarinya.

**Syntax dan Simantik**

Untuk bahasa domain-spesifik, definisi yang sama berlaku berlaku untuk bahasa secara umum. Memodelkan secara umum sebagian dari konsentrastaksian inti. Di sisi sintaks, kita dapat membedakan antara sintaks abstrak dan konkret. Yang pertama menunjukkan struktur dan aturan gramatikal sebuah bahasa. Yang terakhir ini berhubungan dengan simbol notasional dan bentuk representasional yang digunakan bahasa. Untuk meningkatkan abstraksi desain dan menghasilkan morecompletecode, biasanya Anda memerlukan sintaks dan semantik.

1. **Syntax :**

Sintaks menentukan struktur konseptual suatu bahasa: konstruksi bahasa pemodelan, sifat dan koneksi mereka satu sama lain. Di DSM, pemodelan konstruksi idealnya langsung berasal dari masalah domain. Sintaks abstrak dari bahasa pemodelan biasanya ditentukan dalam metamodel (lihat Bagian 4.122.4 untuk rinciannya). Sintaks dari bahasa pemodelan berarti lebih dari sekedar kata-kata yang dicadangkan. Hal ini biasa dilihat juga mencakup peraturan gramatikal yang perlu diikuti saat menentukan model. Dalam DSM, peraturan ini berasal dari domain dan didefinisikan dalam bahasa sehubungan dengan konsep pemodelan. Aturan diperlukan untuk menghindari pembuatan kode dari model yang memiliki kesalahan. Memiliki aturan di tempat selama pemodelan juga membuat penerapan generator menjadi lebih mudah karena generator tidak perlu memulai dengan terlebih dahulu memeriksa apakah model benar. Dalam DSM, aturan diperiksa, jika mungkin, sedini mungkin karena ini memungkinkan mendeteksi dan mencegah kesalahan saat mereka termurah untuk memperbaiki. Pertimbangkan di sini alternatif: menemukan kesalahan selama pembuatan kode atau dari kode yang dihasilkan. Menempatkan aturan dalam bahasa, bukan di generator, lebih masuk akal jika ada beberapa generator: selalu lebih baik untuk memeriksa model satu kali daripada melakukannya untuk setiap generator. Aturan bahasa biasanya membatasi bagaimana model dapat dibuat: mereka menentukan nilai hukum, hubungan antara konsep, dan bagaimana konsep tertentu harus digunakan. Aturannya dapat bervariasi dari aturan ketepatan model yang ketat dan pengecekan konsistensi terhadap peraturan yang membimbing daripada menerapkan cara pemodelan tertentu. Setelah aturan didefinisikan, bahasa pemodelan-yang diundangkan oleh perangkat pendukung - menjamin bahwa semua pengembang mengikuti peraturan domain yang sama. Aturannya kembali secara signifikan mengurangi kemungkinan ruang desain - jenis aplikasi yang dapat ditulis dengan bahasa ini - dan membantu memastikan perancang hanya membuat aplikasi yang sesuai. Jika berbagai aplikasi perlu diperluas, bahasa pemodelan tentu saja dapat diperluas kemudian.

1. **Simantik :**

Setiap konsep pemodelan memiliki beberapa makna, semantik. Bila kita menambahkan elemen menjadi model atau menghubungkan elemen bersama kita menciptakan makna. Dalam DSM, semantik bahasa pemodelan sampai batas tertentu langsung dari domain masalah. Contohnya membantu di sini: jika kita mengembangkan sistem infotainment untuk mobil, konsep pemodelan, seperti "tombol", "menu", dan "acara", sudah memiliki makna yang jelas dalam domain aplikasi. Ini tidak seperti bahasa pemodelan tujuan umum di mana semantik tidak memetakan ke masalah domain tertentu, namun diserahkan kepada pengembang untuk memetakan semantik dan konsep bahasa ke domain masalah tertentu. Penelitian penggunaan bahasa pemodelan (mis., Wijers, 1991) menunjukkan bahwa setiap pengembang membuat pemetaan ini berbeda. Tidak mengherankan jika pemodel menggunakan bahasa tujuan umum menciptakan berbagai jenis model untuk masalah yang sama. DSM berbeda karena bertujuan mengandalkan konsep dan semantik yang datang langsung dari masalah domain. Penggunaan semantik domain dalam bahasa tidak terbatas hanya pada konsep tapi juga mencakup hubungan antara konstruksi pemodelan serta aturan terkait. Mengikuti contoh infotainment mobil, menu dalam sistem infotainment biasanya dapat memicu tindakan atau membuka submenu. Dengan demikian, dalam bahasa spesifik domain, sebuah menu hanya dapat dihubungkan ke tindakan atau ke menu lain. Yang pertama dapat didefinisikan dengan hubungan "transisi" dari sebuah menu ke sebuah tindakan dan yang terakhir dengan hubungan "submenu" dari menu ke contoh menu yang lain. Untuk mengikuti semantik dengan ketat, bahasa pemodelan juga mencakup batasan yang memungkinkan hanya satu hubungan, baik "transisi" atau "submenu", yang akan ditentukan dari setiap menu.

Semantik domain masalah bukan satu-satunya sumber semantik DSM. Seperti semua bahasa pemodelan yang menargetkan pembuatan kode, kita harus mengenali sisi semantik dari sisi implementasi: bagaimana konstruksi pemodelan dipetakan ke domain solusi yang diberikan. Pemetaan ini dibuat bukan untuk masalah domain tapi ke bahasa lain, disini ke bahasa pemrograman. Ini biasanya disebut operational semantics. Penting untuk disadari bahwa semantik operasional tidak bisa menjadi satu-satunya sumber semantik. Jika ya, maka bahasa pemodelan akan benar-benar memetakan oneto-satu ke bahasa pemrograman yang dihasilkan. Abstraksi akan sama dan manfaat pembuatan kode minimal. Contoh klasik di sini adalah pemetaan kelas dalam diagram ke kelas dalam sebuah kode. Pengembang yang membuat model kelas demikian sudah berfikir dengan konsep dan semantik kode. Jika kita ingin meningkatkan abstraksi dan meningkatkan produktivitas, semantik masalah masalah domain lebih banyak daripada semantik domain solusi.

https://books.google.co.id/books?id=Rfa5DcATAEQC&pg=PA328&lpg=PA328&dq=KELEMAHAN+DSM&source=bl&ots=BzksXamFA0&sig=lxtxQtdmljYGFsiiFyrbpQz1uJ0&hl=id&sa=X&ved=0ahUKEwjz4q\_dltHXAhUPOrwKHfS7BDsQ6AEIUTAH#v=onepage&q=KELEMAHAN%20DSM&f=false