Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.201851610
Vol. 5, No. 1, Maret 2018, hlm. 77-86 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 51/E/KPT/2017 e-ISSN: 2528-6579

PEMODELAN USE CASE (UML): EVALUASI TERHADAP BEBERAPA KESALAHAN DALAM PRAKTIK

Tri A. Kurniawan

Software Engineering Research Grup (SERG), Fakultas Ilmu Komputer, Universitas Brawijaya
Email: triak@ub.ac.id

(Naskah masuk: 15 Januari 2018, diterima untuk diterbitkan: 15 Februari 2018)

Abstrak

UML sudah menjadi bahasa pemodelan baku dalam pengembangan sistem perangkat lunak. Pemodelan yang penting dalam UML, untuk menjelaskan aspek fungsionalitas sistem, adalah pemodelan *use case*. *Use case* dideskripsikan secara tekstual dalam bentuk *use case scenario* untuk menjelaskan interaksi yang terjadi antara aktor dengan sistem. Selanjutnya, *use case* diilustrasikan secara visual dalam bentuk *use case diagram* untuk menggambarkan konteks dari sistem yang dikembangkan. Dalam praktiknya, kedua model tersebut tidak sulit untuk dibuat meskipun oleh orang yang belum berpengalaman. Namun demikian, pemodelan *use case* yang dihasilkan, baik dalam konteks pembelajaran konsep pengembangan perangkat lunak di kampus maupun dalam konteks implementasi di industri perangkat lunak, tidak sedikit yang mengandung kesalahan baik secara sintaksis maupun semantik. Artikel ini bertujuan untuk melakukan evaluasi terhadap beberapa kesalahan tersebut sehingga bisa dijadikan acuan dalam membangun model yang baik dan benar sehingga mampu menjelaskan sistem yang dikembangkan secara tepat. Evaluasi dilakukan dengan mengidentifikasi dan mengklasifikasi kesalahan, serta merekomendasi perbaikan yang diperlukan berdasarkan kajian teori dan spesifikasi UML. Artikel ini telah membahas secara detil 11 jenis kesalahan dalam pembuatan *use case scenario* dan 7 jenis kesalahan dalam penggambaran *use case diagram*, masing-masing disertai dengan contoh kasus yang relevan.

Kata kunci: *unified modeling language, use case diagram, use case scenario, kesalahan, orientasi objek*

USE CASE (UML) MODELING: EVALUATION ON SOME PITFALLS IN PRACTICES

Abstract

UML has become a modeling language standard in the software system development. An important modeling in UML, which is used to explain the functionalities of a system, is the use case modeling. A use case is textually described in a use case scenario in order to outline the interactions that occur between the actor(s) and the system. Further, the use cases of a system are visually illustrated in a use case diagram. In practice, both models are not difficult to create, even by an inexperienced person. However, such produced models, either in the context of learning the concept of software development on campuses or in the context of implementation in software industries, contain some errors in both syntax and semantics. This article aims to evaluate such common pitfalls as a guidance in creating good and correct use cases, which appropriately describe the system being developed. It was performed by identifying and classifying the pitfalls, as well as recommending the required corrections based on theories and the specification of UML. This article discussed in detail 11 pitfall types in developing use case scenario and 7 pitfall types in creating use case diagram, equipped with some relevant examples respectively.

Keywords: *unified modeling language, use case diagram, use case scenario, pitfalls, object oriented*

1. PENDAHULUAN

Bahasa pemodelan perangkat lunak *unified modeling language* (UML), sejak pertama kali diperkenalkan pada tahun 1997, saat ini telah berkembang menjadi sebuah bahasa pemodelan yang baku (*de facto*) di dalam sebuah pengembangan perangkat lunak (Engels, et al., 2000) (Larman, 2005) (Lange, et al., 2006). UML digunakan dalam pengembangan sistem perangkat lunak yang menggunakan pendekatan berorientasi objek. Intensitas penggunaan UML yang tinggi ini didukung dengan semakin matangnya konsep pemodelan yang dirumuskan dalam setiap rilis spesifikasi UML yang dikembangkan oleh Object Management Group

(OMG)[1]. Sampai tahun 2017, OMG telah merilis 11 versi spesifikasi UML, yang terakhir adalah versi 2.5.1 yang termasuk dalam revisi UML 2.0[2]. Di sisi lain, pengembangan alat bantu untuk pemodelan dengan UML berkembang cukup pesat dan sebagiannya tergolong sebagai *free software* sehingga tersedia banyak pilihan bagi pengembang perangkat lunak untuk menggunakannya, antara lain: StarUML[3], ArgoUML[4], UML Designer[5].

UML menyediakan banyak sekali diagram yang diperlukan untuk menjelaskan sistem yang sedang dikembangkan, baik dari aspek statis maupun dinamisnya (OMG, 2017). Salah satu diagram penting yang digunakan untuk mengilustrasikan kebutuhan (*requirements*) dari sistem adalah *use case (UC) diagram*, yang menjelaskan secara visual konteks dari interaksi antara aktor dengan sistem. Setiap *use case* menyatakan spesifikasi perilaku (fungsionalitas) dari sistem yang sedang dijelaskan yang memang dibutuhkan oleh aktor untuk memenuhi tujuannya. Namun demikian, penjelasan detil dari interaksi yang terjadi antara aktor dan sistem, berkaitan dengan sebuah use case tertentu, harus dijelaskan secara deskriptif dalam sebuah *use case (UC) scenario*. Oleh karena itu, *UC scenario* dan *UC diagram*, yang dibutuhkan dalam pemodelan UC dari sebuah sistem, harus mampu menjelaskan fungsionalitas sistem secara lengkap dan valid.

Dalam praktiknya, pembuatan *UC scenario* dan *UC diagram* bisa dilakukan dengan cukup mudah, meskipun oleh pengembang sistem yang belum berpengalaman. Namun demikian, pembuatan sebuah penjelasan *use case* yang baik dan bermanfaat ternyata membutuhkan keahlian dan pengalaman yang cukup (Cockburn, 2000) (Adolph, et al., 2002). Sebagai akibat dari minimnya pengetahuan dan pengalaman dalam pemodelan UC maka *UC scenario* dan *UC diagram* yang dihasilkan ternyata masih sering mengandung kesalahan sehingga model gagal merepresentasikan fungsionalitas dari sistem dengan tepat. Terlebih lagi, tidak sedikit pengembang yang masih memiliki paradigma yang salah yang memandang bahwa program yang bisa dieksekusi (*executable codes*) adalah satu-satunya produk yang penting dari sebuah pengembangan perangkat lunak (Pressman, 2010), sehingga aspek pemodelan dan dokumentasi sistem menjadi terabaikan.

Selanjutnya, berdasarkan pengalaman penulis dalam mengampu mata kuliah Rekayasa Perangkat Lunak dan/atau Analisis dan Perancangan Sistem dan/atau Pemodelan Perangkat Lunak di tingkat sarjana dan/atau pascasarjana selama hampir 15 tahun, tidak sedikit mahasiswa-mahasiswi yang menghasilkan *UC scenario* dan *UC diagram* yang salah, baik secara sintaksis maupun semantik. Disamping itu, laporan skripsi yang mengandung penjelasan sistem dengan menggunakan *UC scenario* dan *UC diagram* juga tidak sedikit yang salah, meskipun sudah melewati proses pembimbingan dan pengujian. Lebih dari itu, ternyata tidak sedikit artikel jurnal ilmiah yang juga mengandung penjelasan *UC scenario* dan *UC diagram* yang tidak tepat. Hal yang sama juga terjadi pada dunia industri perangkat lunak (Sinnig, et al., 2005) (Kjeoy & Stalheim, 2007). Lebih dari itu, pengalaman penulis di dunia praktis industri perangkat lunak selama hampir 7 tahun juga mengonfirmasikan fenomena tersebut. Kesalahan-kesalahan yang umum dilakukan dalam beberapa kasus tersebut, antara lain penjelasan yang umum dan hanya interaksi dari sisi aktor saja pada pembuatan *UC scenario*, dan penggambaran UC yang mengandung urutan sebagaimana yang ada dalam konsep *data flow diagram* (DFD) pada pembuatan *UC diagram*. Kesalahan dalam pemodelan UC, sehingga tidak sesuai dengan sistem sebenarnya yang dikembangkan, akan membuat proses pengembangan perangkat lunak menjadi lebih sulit untuk dikelola akibat adanya inkonsistensi (Bennet, et al., 2010). Kesalahan ini juga akan berdampak pada kemudahan proses pemeliharaan (*maintenance*) yang dilakukan setelah perangkat lunak selesai dibuat (Chen & Huang, 2009). Untuk menghindari terjadinya inkonsistensi tersebut, sudah ada beberapa penelitian yang mencoba melakukan pengecekan kesalahan yang ada pada *UC* scenario sehingga bisa dilakukan antisipasi perbaikan sedini mungkin (Sinha, et al., 2010)(Liu, et al., 2014). Namun demikian, potensi permasalahan yang mungkin terjadi pada pembuatan *UC scenario* dan *UC diagram* akan lebih efektif untuk diantisipasi jika kita mampu mengetahui kesalahan-kesalahan yang harus dihindari.

Artikel ini mencoba mengidentifikasi dan menjelaskan berbagai bentuk kesalahan yang sering terjadi pada pembuatan *UC scenario* dan *UC diagram*. Selanjutnya, rekomendasi pembetulan yang dibutuhkan sesuai dengan kaidah yang benar juga akan dijelaskan secara detil dengan menggunakan beberapa contoh kasus yang relevan. Dengan demikian, kesalahan-kesalahan yang sama bisa dihindari dalam proses pengembangan perangkat lunak sehingga bisa dihasilkan produk perangkat lunak yang berkualitas dan konsisten.

Pembahasan dalam artikel ini terbagi dalam beberapa bagian. Bagian 2 menjelaskan konsep pemodelan *UC scenario* dan *UC diagram* dalam UML 2.0. Bagian 3 membahas secara detil evaluasi beberapa kesalahan yang sering terjadi berikut rekomendasi perbaikan yang sesuai yang diperlukan dalam pemodelan. Terakhir, Bagian 4 menjelaskan kesimpulan dari pembahasan yang telah dilakukan.

2. PEMODELAN UC DALAM UML 2.0

[1] Website: http://www.omg.org
[2] Website: http://www.omg.org/spec/UML/2.5.1
[3] Website: http://staruml.io/
[4] Website: http://argouml.tigris.org/
[5] Website: http://www.umldesigner.org/

UML dibuat untuk menyediakan perangkat yang dibutuhkan oleh para pengembang perangkat lunak dalam melakukan analisis, perancangan dan implementasi dari sistem berbasis perangkat lunak (OMG, 2017). Spesifikasi UML versi 2.5.1 merupakan rilis terbaru dari UML revisi 2 (UML 2.0) sebagai pengembangan konsep pemodelan sistem yang ada pada revisi 1 (UML 1.0). Dari kedua revisi tersebut, sejauh ini OMG sudah merilis 11 versi sejak rilis versi 1.1 pada bulan Desember 1997[6].

UML 2.0 memiliki 14 diagram yang disediakan untuk menjelaskan sistem dari 2 aspek, yaitu aspek perilaku (*behavioral aspect*) yang bersifat dinamis dan aspek struktural (*structural aspect*) yang bersifat statis (OMG, 2017). Aspek perilaku menjelaskan perubahan yang terjadi pada sistem sejalan dengan perubahan waktu. Sedangkan, aspek struktural mendeskripsikan struktur dari elemen-elemen pembentuk sistem yang tidak ada kaitannya dengan waktu, yaitu konsep dari sebuah aplikasi. *UC diagram* adalah salah satu diagram yang diklasifikasikan ke dalam aspek perilaku. Deskripsi perilaku dari setiap UC dijelaskan secara detil dan terpisah dengan menggunakan dokumen secara tekstual, yaitu *UC scenario* atau *UC specification* atau *UC description*. Namun demikian, pemodelan *UC* yang utama adalah penjelasan secara tekstual dalam bentuk *UC scenario*, sedangkan *UC diagram* adalah sebagai pelengkap (Larman, 2005).

Setiap *UC* menyatakan perilaku yang harus dijalankan oleh sistem dalam kaitannya dengan satu atau lebih aktor (OMG, 2017). Oleh karena itu, UC merupakan abstraksi dari interaksi yang terjadi antara aktor dengan sistem sehingga tujuan dari aktor bisa tercapai. Berdasarkan perspektif ini maka sebuah UC semestinya dipandang dari sisi aktor dan bukan dari sisi sistem, sehingga penamaan UC juga didasarkan atas tujuan yang ingin dicapai oleh aktor.

2.1. *UC scenario*

UC scenario merupakan penjelasan secara tekstual dari sekumpulan skenario interaksi. Setiap skenario mendeskripsikan urutan aksi/langkah yang dilakukan aktor ketika berinteraksi dengan sistem, baik yang berhasil maupun gagal.

UC scenario dijelaskan secara tekstual dalam beberapa format tergantung kebutuhannya, yaitu singkat (*brief*), informal (*casual*), atau lengkap (*fully dressed*) (Larman, 2005), yang bisa dijelaskan dalam bentuk tabel dengan 1 kolom atau 2 kolom (Cockburn, 2000). Pada format singkat, penjelasan diberikan cukup 1 paragraf yang mengacu hanya pada skenario yang berhasil. Pada format informal, penjelasan diberikan dalam beberapa paragraf yang mencakup semua skenario, baik yang berhasil maupun gagal. Sedangkan, pada format lengkap, penjelasan dibuat secara detil disertai dengan bagian-bagian pendukung yang penting. Format terakhir ini yang banyak digunakan di dalam praktik. Bagian-bagian penting tersebut adalah (Larman, 2005):

- **aktor primer** (*primary actor*), yaitu aktor yang menginisiasi layanan sistem untuk mencapai tujuan dari aktor tersebut. Jumlah aktor primer dimungkinkan lebih dari 1.
- **prakondisi** (*preconditions*), yaitu kondisi spesifik yang harus terpenuhi sebelum sebuah UC bisa diinisiasi atau dieksekusi oleh aktor primer. Jumlah prakondisi bisa lebih dari 1 keadaan.
- **alur utama** (*main or basic flow*), yaitu jalur interaksi yang mengarahkan pada skenario yang berhasil sehingga tujuan aktor bisa terpenuhi. Jalur ini hanya terdiri dari 1 jalur saja.
- **alur alternatif** (*alternative flows*), yaitu jalur alternatif dari interaksi yang terjadi antar aktor dengan sistem yang mencakup pencabangan (pilihan) maupun skenario yang gagal sehingga tujuan aktor tidak terpenuhi. Jalur ini bisa terdiri dari lebih dari 1 jalur kemungkinan.
- **kondisi akhir** (*postconditions*), yaitu kondisi spesifik yang harus terjadi ketika *UC* berhasil dijalankan atau dieksekusi secara lengkap, sebagai representasi dari tujuan yang ingin dicapai oleh aktor primer. Jumlah kondisi akhir bisa lebih dari 1 keadaan.

2.2. *UC diagram*

Sebuah *UC diagram* menyatakan visualisasi interaksi yang terjadi antara pengguna (aktor) dengan sistem. Diagram ini bisa menjadi gambaran yang bagus untuk menjelaskan konteks dari sebuah sistem sehingga terlihat jelas batasan dari sistem (Larman, 2005). Ada 2 elemen penting yang harus digambarkan, yaitu aktor dan UC. Aktor adalah segala sesuatu yang berinteraksi langsung dengan sistem, bisa merupakan orang (yang ditunjukkan dengan perannya dan bukan namanya/personilnya) atau sistem komputer yang lain. Aktor dinotasikan dengan simbol gambar orang-orangan (*stick-man*) dengan nama kata benda di bagian bawah yang menyatakan peran/sistem. Aktor bisa bersifat primer, yaitu yang menginisiasi berjalannya sebuah UC, atau sekunder, yaitu yang membantu berjalannya sebuah UC. UC dinotasikan dengan simbol elips dengan nama kata kerja aktif di bagian dalam yang menyatakan aktivitas dari perspektif aktor. Setiap aktor dimungkinkan untuk berinteraksi dengan sistem dalam banyak UC. Sebaliknya, setiap UC bisa dijalankan oleh lebih dari satu aktor.

Antar aktor maupun antar UC bisa memiliki relasi, masing-masing dengan spesifikasi yang berbeda. Sebuah UC, disebut dengan *base UC*, bisa memiliki relasi dengan 1 atau lebih UC yang lain, disebut dengan *supplier UC*, dalam bentuk **extend** dan/atau **include**. Relasi **extend** menyatakan bahwa fungsionalitas dari *base UC* bisa diperluas oleh *supplier UC*, jika dibutuhkan, di dalam aksekusi alur

[6] Website: http://www.omg.org/spec/UML

alternatif yang ada pada *UC scenario* dari *base UC*. Sedangkan, relasi **include** menyatakan bahwa fungsionalitas dari *base UC* selalu hanya bisa dipenuhi dengan bantuan dari *supplier UC* di dalam eksekusi alur utama yang ada pada *UC scenario* dari *base UC*. Dalam hal ini, relasi **include** dan **extend** tidak menjelaskan urutan eksekusi apapun antara *base UC* dan *supplier UC*, baik dalam alur utama maupun alternatif yang dijelaskan dalam *UC scenario* dari *base UC*. Selanjutnya, sebuah aktor, disebut aktor induk, bisa memiliki relasi **inheritance** dengan aktor yang lain, disebut aktor turunan, yang menyatakan bahwa sebuah aktor merupakan turunan dari aktor yang lain. Aktor turunan akan memiliki hak akses terhadap fungsionalitas sistem yang lebih luas dibandingkan dengan aktor induk.

Gambar 1 mengilustrasikan sebuah *UC diagram* sederhana yang terdiri dari 2 buah aktor primer, yaitu Pengguna dan Teller, serta 1 aktor sekunder, yaitu Sistem Antar Bank. Sistem tersebut memiliki fungsionalitas yang direpresentasikan dalam 5 UC, yaitu Login, Cetak Tabungan, Simpan Dana, Transfer Dana, dan Hitung Saldo. Aktor Teller merupakan turunan dari aktor Pengguna, ketika Pengguna dinyatakan valid saat melakukan *login*. Siapapun bisa melakukan *login*, itulah yang disebut dengan Pengguna. Teller, saat mentransfer dana, **bisa** melibatkan proses perhitungan saldo jika transfer yang dilakukan menggunakan rekening nasabah (ditunjukkan dengan relasi **extend**). Di sisi lain, perhitungan saldo **harus** selalu dilakukan saat Teller melakukan penyimpanan dana di rekening nasabah (ditunjukkan dengan relasi **include**). Untuk mengakhiri akses, Teller melakukan *logout*.

3. EVALUASI KESALAHAN DAN REKOMENDASI PERBAIKAN

Pada pembahasan berikut akan dijelaskan beberapa jenis kesalahan yang sering terjadi dalam pemodelan UC, baik pada *scenario* maupun *diagram*. Pada setiap kesalahan yang teridentifikasi, rekomendasi pembetulan juga dijelaskan secara detil untuk menghindari terjadinya kesalahan yang sama.

3.1. *UC scenario*

Pada bagian ini, beberapa kesalahan di dalam penulisan *UC scenario* akan dijelaskan secara singkat berdasarkan klasifikasinya.

3.1.1. Fokus pada diagram

Penjelasan secara tekstual dalam *UC scenario* adalah hal utama dalam pemodelan UC, dan bukan pada *UC diagram* (Larman, 2005). Kesalahan yang sering dilakukan adalah lebih banyak menghabiskan waktu untuk membuat *UC diagram* dibandingkan menuliskan *UC scenario*. Rekomendasi perbaikannya adalah kembali fokus pada penulisan *UC scenario* dengan detil dan akurat dalam porsi yang lebih dibandingkan pembuatan *UC diagram*.

3.1.2. Penamaan UC yang salah

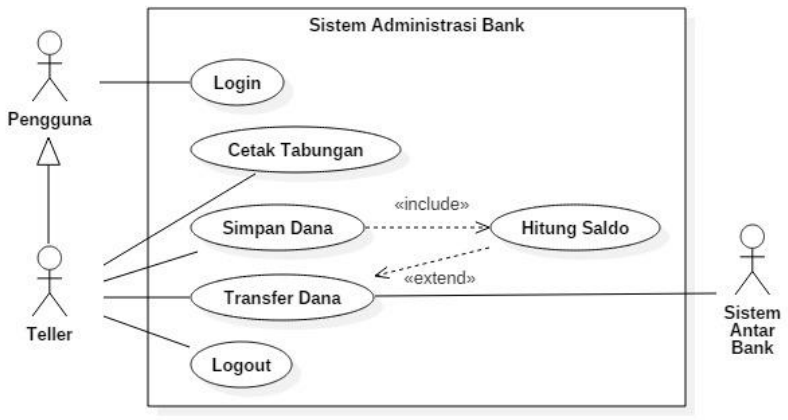
Penamaan UC diawali tidak dengan menggunakan kata kerja dan lebih merepresentasikan sesuatu berdasarkan sudut pandang sistem dan bukan aktor. Sebagai contoh, nama UC `Pemrosesan Transfer Dana` adalah contoh yang salah karena diawali dengan kata benda (yaitu pemrosesan) dan merepresentasikan apa yang dilakukan oleh sistem (yaitu memproses). Rekomendasi perbaikannya adalah menggunakan nama yang diawali dengan kata kerja (Larman, 2005) dan merepresentasikan tujuan yang ingin dicapai oleh aktor (Lilly, 1999), sehingga menjadi `Transfer Dana`.

3.1.3. Penjelasan yang terlalu umum

Dalam menjelaskan interaksi yang terjadi antara aktor dengan sistem pada alur utama dan/atau alur alternatif, informasi yang diberikan bersifat umum sebagaimana diilustrasikan dalam contoh potongan interaksi berikut.

a. `Sistem menampilkan form isian data pribadi.`
b. `Aktor mengisi form data pribadi.`

Pada alur tersebut, informasi tentang **data pribadi** yang harus ditampilkan oleh sistem dan diisi oleh aktor tidak dijelaskan secara spesifik, terdiri dari elemen data apa saja. Deskripsi yang dituliskan dalam



Gambar 1. Sebuah *UC diagram* sederhana dari Sistem Administrasi Bank (parsial)

UC scenario seperti ini akan menyulitkan pada tahapan pemodelan berikutnya, karena penjelasan ini akan menjadi salah satu bahan dalam mengidentifikasi objek-objek yang akan terlibat dalam pendefinisian diagram urutan (*sequence diagram*) maupun diagram klas (*class diagram*). Rekomendasi perbaikannya adalah menuliskan ulang dalam bentuk yang lebih spesifik sebagaimana contoh potongan interaksi berikut.

a. Sistem menampilkan form isian data pribadi yang terdiri dari nama, alamat, tempat lahir, dan tanggal lahir.
b. Aktor mengisi form data pribadi secara lengkap.

3.1.4. Identifikasi aktor yang salah

Kesalahan terjadi karena identifikasi aktor yang tidak cermat. Hal ini bisa berkaitan dengan kondisi akhir yang tidak konsisten (misalnya untuk kasus *login*) atau konteks sistem yang tidak jelas (pada level model aplikasi atau model bisnis).

Untuk kasus *login*, kondisi akhir yang harus terpenuhi adalah aktor sudah bisa dikenali sistem sebagai aktor yang valid, sehingga siapapun aktor (baik yang nantinya valid atau tidak) bisa menginisiasi *login*. Jadi, bukan aktor yang valid saja yang bertindak sebagai aktor primer untuk kasus *login*. Sebagai contoh, aktor **Teller** tidak tepat sebagai aktor primer untuk UC Login karena **Teller** adalah aktor yang baru teridentifikasi secara valid pada kondisi akhir setelah UC Login dijalankan secara lengkap. Sehingga rekomendasinya, aktor primer yang diidentifikasi adalah **Pengguna**, sebagai aktor umum yang belum dikenali oleh sistem.

Sedangkan untuk konteks sistem yang tidak jelas, kesalahan terjadi karena level abstraksinya yang tidak sama, apakah aktor pada level model aplikasi atau model bisnis. Jika pada level model aplikasi, maka peran yang berinteraksi langsung dengan sistem yang harus menjadi aktor primer. Rekomendasinya adalah perlu mengidentifikasi aktor secara tepat sesuai dengan konteksnya.

3.1.5. Identifikasi prakondisi dan kondisi akhir yang tidak tepat

Prakondisi dan kondisi akhir yang harus dipenuhi tidak dijelaskan secara tepat dan spesifik. Keduanya harus menyatakan sebuah keadaan objek dari domain masalahnya sistem yang bisa diobservasi, bukan menunjukkan sebuah proses atau aksi, baik sebagai syarat sebuah UC bisa dieksekusi (prakondisi) atau sebagai keadaan yang harus terpenuhi ketika UC telah dieksekusi secara lengkap (kondisi akhir). Sebagai contoh, untuk kasus *login* maka prakondisinya adalah halaman Login sudah terbuka atau tersedia, dan **bukan** data Teller sudah ada di basis data. Sedangkan kondisi akhirnya adalah aktor sudah teridentifikasi sebagai Teller. Kedua kondisi sebaiknya dijelaskan dalam bentuk kalimat pasif (Larman, 2005). Rekomendasi yang perlu dilakukan adalah mendefinisikan prakondisi dan kondisi akhir untuk setiap UC secara spesifik dengan menggunakan kalimat pasif.

3.1.6. Tidak ada penjelasan aliran utama dan/atau aliran alternatif

Aliran utama dan/atau aliran alternatif tidak dijelaskan sama sekali di dalam *UC scenario*, sehingga interaksi yang terjadi antara aktor dengan sistem pada sebuah UC tidak bisa tergambarkan secara baik. Penjelasan UC hanya diberikan untuk hal-hal yang berkaitan dengan aktor primer, deskripsi UC, prakondisi, dan kondisi akhir. Rekomendasi perbaikannya adalah kedua aliran harus diberikan penjelasan yang baik, paling tidak aliran utama harus ada dalam sebuah UC karena hal itu menggambarkan bagaimana interaksi dasar yang harus terjadi antara aktor dengan sistem sehingga kondisi akhir terpenuhi. Aliran alternatif, baik yang merupakan pencabangan atau skenario gagal, dalam sebuah UC mungkin tidak ada sama sekali.

3.1.7. Interaksi hanya dari sisi aktor

UC scenario dijelaskan hanya berdasarkan interaksi yang diinisiasi oleh aktor primer saja. Respon yang diberikan oleh sistem tidak pernah dijelaskan. Penjelasan yang seperti ini tidak lengkap karena tidak mampu menggambarkan bagaimana sistem merespon terhadap apa yang dilakukan oleh aktor, sebagaimana contoh potongan interaksi berikut.

a. Sistem menampilkan form isian data pribadi yang terdiri dari nama, alamat, tempat lahir, dan tanggal lahir.
b. Sistem menyimpan informasi data pribadi yang telah diisi oleh aktor.

Rekomendasi perbaikannya adalah dengan menjelaskan secara lengkap gambaran interaksi antara aktor dengan sistem yang dilihat dari dua sisi, sebagaimana contoh potongan interaksi berikut.

a. Sistem menampilkan form isian data pribadi yang terdiri dari nama, alamat, tempat dan tanggal lahir.
b. Aktor mengisi form secara lengkap kemudian memerintahkan sistem untuk memproses.
c. Sistem melakukan validasi dan menyimpan data.

3.1.8. Aliran utama dan/atau aliran alternatif tidak jelas

Kesalahan ini terjadi ketika penjelasan interaksi pada aliran utama dicampuradukkan dengan penjelasan untuk aliran alternatif, sebagaimana contoh potongan interaksi berikut.

a. Sistem melakukan validasi dan menyimpan data.
    i. Jika validasi salah, maka sistem menampilkan kembali form isian data pribadi berikut pesan kesalahan "Data tidak valid".

ii. Jika data tidak berhasil disimpan karena permasalahan di basis data, maka sistem menampilkan pesan "Data gagal disimpan".

iii. Jika data berhasil disimpan maka sistem menampilkan pesan "Data berhasil disimpan".

Rekomendasi perbaikannya adalah penjelasan aliran utama harus terpisah dengan aliran alternatif. Pada aliran utama tidak boleh ada penjelasan yang mengandung pilihan dan/atau kegagalan, sebagaimana contoh potongan interaksi berikut.

a. Sistem melakukan validasi dan menyimpan data.
b. Sistem menampilkan pesan "Data berhasil disimpan".

Pilihan dan/atau kegagalan harus dijelaskan pada bagian aliran alternatif secara terstruktur dengan mengacu pada urutan interaksi yang berpotensi terjadinya kegagalan dan/atau pilihan dalam aliran utama, sebagaimana contoh berikut.

a. Kegagalan validasi dan penyimpanan:
   i. Jika validasi salah, maka sistem menampilkan kembali form isian data pribadi berikut pesan kesalahan "Data tidak valid".
   ii. Jika data tidak berhasil disimpan karena permasalahan di basis data, maka sistem menampilkan pesan "Data gagal disimpan".

3.1.9. Penjelasan yang bersifat white-box

Kesalahan ini terjadi jika penjelasan dalam aliran utama dan/atau aliran alternatif diberikan terlalu teknis yang berkaitan dengan bagaimana sistem bekerja secara internal, sebagaimana contoh potongan interaksi berikut.

a. Sistem melakukan validasi dan menyimpan data ke basis data pada Tabel DataPribadi dengan menggunakan *stored procedure* sp_DataPribadi.

Rekomendasi perbaikannya adalah dengan memberikan penjelasan yang bersifat *black-box* (Larman, 2005), yaitu menjelaskan apa (*responsibilities*) yang harus dilakukan oleh sistem dan bukan bagaimana sistem melakukannya, sebagaimana contoh potongan interaksi berikut.

a. Sistem melakukan validasi dan menyimpan data.

3.1.10. Hanya menjelaskan aliran utama/normal

Kesalahan ini terjadi jika dalam *UC scenario* hanya menjelaskan aliran utama/normal saja tanpa aliran alternatif, baik yang berupa pencabangan (kondisi/opsi/pilihan) maupun yang berupa kegagalan. Rekomendasi perbaikannya adalah setiap *UC scenario* harus mengandung berbagai skenario kegagalan dan/atau skenario yang mungkin sebagai pilihan yang harus diputuskan oleh aktor.

3.1.11. Penyebutan aktor yang inkonsisten dengan yang ada di UC diagram

Kesalahan ini terjadi jika aktor (primer/sekunder) yang disebutkan dalam *UC scenario* tidak sesuai dengan sebutan aktor yang digunakan dalam *UC diagram*. Sebagai contoh, aktor yang disebutkan dalam *UC scenario* adalah Pengguna, tetapi yang digambarkan dalam *UC diagram* adalah aktor User. Rekomendasi perbaikannya adalah dengan menyesuaikan penyebutan semua nama aktor yang ada di *UC scenario* dan *UC diagram*.

3.2. *UC diagram*

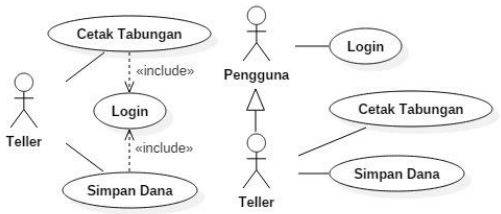
Pada bagian ini, pembahasan akan dilakukan berdasarkan klasifikasi kesalahan yang terjadi dalam pembuatan *UC diagram* yang disertai dengan fragmen diagram yang relevan. Kasus yang digunakan adalah Sistem Administrasi Bank yang dijelaskan secara sederhana dan bersifat parsial (tidak menggambarkan sistem secara lengkap).

3.2.1. UC yang berurut

Kesalahan ini terjadi ketika kita ingin menggambarkan bahwa UC pertama memerlukan fungsionalitas dari UC kedua sebagai syarat agar UC pertama bisa dijalankan, dengan menggunakan relasi **include**. Sehingga, *UC diagram* yang dihasilkan terkesan ada penjelasan tentang urutan eksekusi UC, sebagaimana yang terjadi dalam pemodelan proses dengan *data flow diagram* (DFD). Dalam praktiknya, kesalahan ini biasa terjadi dalam menggambarkan UC Login yang memiliki relasi dengan UC yang lain, dimana UC Login harus dilakukan sebelum UC yang lain bisa diinisiasi. Gambar 2a menjelaskan bahwa untuk bisa menjalankan UC Cetak Tabungan dan Simpan Dana maka aktor Teller harus melakukan login dengan menginisiasi UC Login. Diagram seperti ini tidak tepat, karena:

- UC diagram tidak menggambarkan urutan eksekusi UC oleh aktor, tapi diagram ini menjelaskan urutan.
- Logika yang dijelaskan dalam diagram dengan relasi **include** menyatakan bahwa dalam alur utama UC Cetak Tabungan dan Simpan Dana terdapat aktivitas untuk melakukan login melalui inisiasi UC Login. Hal ini tidak sesuai dengan deskripsi yang ada pada *UC scenario* untuk masing-masing UC tersebut dimana: (i) prakondisi dari UC Cetak Tabungan dan Simpan Dana adalah aktor sudah valid login sebagai Teller; dan (ii) alur utama masing-masing UC tersebut juga tidak menjelaskan aktivitas untuk melakukan login.

- Dalam realisasi sistemnya, aktivitas login hanya dilakukan sekali di awal untuk validasi aktor dan **bukan** setiap kali menginisiasi UC `Cetak Tabungan` dan `Simpan Dana` sebagaimana dijelaskan dalam diagram dengan relasi **include**.



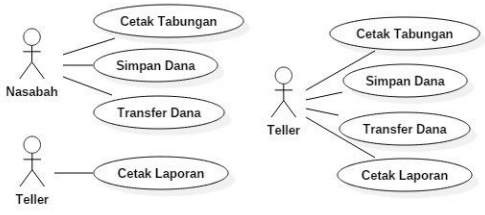
(a) (b)

Gambar 2. UC yang berurut: (a) diagram yang salah; (b) rekomendasi perbaikan

Gambar 2b menjelaskan diagram lain untuk sistem yang sama, yang merupakan rekomendasi perbaikan atas diagram yang dijelaskan pada Gambar 2a. Ada tambahan aktor baru, yaitu Pengguna, yang merupakan abstraksi dari pemakai sistem secara umum yang belum tervalidasi melalui proses login sebagai Teller. Diagram ini direkomendasikan dengan pertimbangan:

- Diagram yang baru sama sekali tidak menggambarkan urutan eksekusi UC.
- Logika yang ada pada diagram sesuai dengan deskripsi yang ada pada *UC scenario*, baik prakondisi, alur utama maupun kondisi akhir dari setiap UC.
- Pada realisasi sistemnya, login hanya sekali dilakukan untuk validasi aktor umum, yaitu Pengguna. Begitu aktor sudah tervalidasi sebagai Teller, maka aktor sudah bisa mengakses UC yang memang disediakan untuk aktor Teller.

3.2.2. Batasan sistem yang inkonsisten



(a) (b)

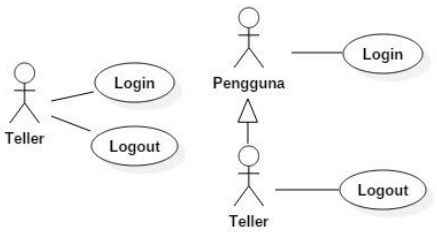
Gambar 3. Batasan sistem yang inkonsisten: (a) diagram yang salah; (b) rekomendasi perbaikan

Kesalahan ini terjadi karena kita tidak konsisten dalam menentukan batasan sistem yang akan dijelaskan dalam diagram, apakah pada level model aplikasi, atau level model bisnis (Lilly, 1999). Sehingga, kedua level model tersebut akan tergambar pada diagram yang sama sebagaimana diilustrasikan pada Gambar 3a. Diagram ini tidak tepat karena aktor Nasabah hanya ada pada level model bisnis bank. Pada level ini, Nasabah memang berhak untuk mencetak buku tabungannya. Namun demikian, Nasabah tidak ada pada level model aplikasi karena Nasabah tidak melakukan akses langsung pada aplikasi bank, yang berbeda dengan Teller.

Rekomendasi yang diilustrasikan pada Gambar 3b menjelaskan bahwa aktor Teller adalah yang tepat untuk dimodelkan pada level aplikasi. Dalam kenyataannya, sistem yang dibangun nantinya adalah sistem yang diakses langsung oleh Teller untuk melayani Nasabah dalam rangka mencetak buku tabungan, menyimpan dana dan mentransfer dana. Nasabah tidak memiliki akses langsung ke sistem meskipun aktivitas itu merupakan kebutuhannya.

3.2.3. Identifikasi aktor yang tidak tepat

Kesalahan ini terjadi ketika kita menentukan aktor dari sebuah UC yang tidak sesuai dengan penjelasan yang ada pada *UC scenario*, berkaitan dengan prakondisi dan kondisi akhir. Dalam praktiknya, kesalahan ini sering terjadi dalam menentukan aktor untuk UC `Login`. Gambar 4a menggambarkan kesalahan tersebut, dengan pengertian umum yang banyak dipahami: yang melakukan login adalah aktor Teller. Diagram seperti ini tidak tepat, karena UC `Login` memiliki tujuan untuk melakukan validasi apakah aktor yang mengakses sistem memiliki otoritas yang valid atau tidak. Sehingga, prakondisinya adalah aktor belum dikenali sebagai aktor yang valid dan kondisi akhir yang harus terpenuhi jika alur utama berhasil dijalankan adalah aktor teridentifikasi sebagai Teller. Dari logika ini maka semestinya aktor yang menginisiasi UC `Login` adalah aktor yang belum dikenali sebagai Teller, tetapi aktor yang bersifat umum. Berbeda dengan UC `Logout` yang bertujuan untuk keluar dari sistem dimana aktor yang menginisiasinya adalah aktor yang sudah dikenali oleh sistem, yaitu Teller.



(a) (b)

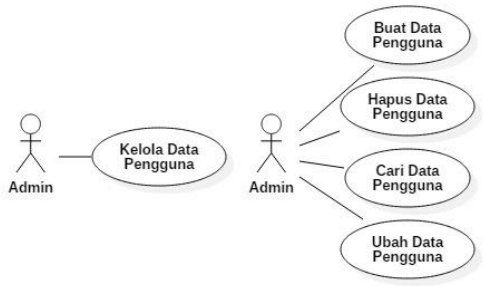
Gambar 4. Identifikasi aktor yang salah: (a) diagram yang salah; (b) rekomendasi perbaikan

Gambar 4b mengilustrasikan rekomendasi untuk memperbaiki diagram yang salah tersebut. Aktor Pengguna adalah aktor yang bersifat umum yang belum dikenali oleh sistem yang menginisiasi UC `Login`. Siapapun bisa menginisiasi UC `Login`, tetapi kondisi akhirnya yang akan membedakan apakah skenario berhasil atau gagal yang akan terjadi.

Jika berhasil, maka aktor Pengguna tadi akan dikenali sebagai aktor Teller (direpresentasikan dengan relasi **inheritance**) yang memiliki hak untuk menginisiasi semua UC yang relevan dengan aktor Teller.

3.2.4. Abstrak UC

Kesalahan ini terjadi karena kita ingin melakukan penyederhanaan diagram dengan melakukan komposisi UC sehingga tidak terlalu banyak UC yang ditampilkan. Kondisi ini sedikit banyak terpengaruh oleh konsep yang ada pada pemodelan dengan DFD, dimana kita bisa membuat sebuah proses yang merupakan gabungan/komposisi dari beberapa proses yang ada pada DFD level di bawahnya. Selain itu, pemodelan seperti ini bisa juga terpengaruh oleh struktur menu aplikasi yang akan dibuat. Padahal, *UC diagram* semestinya tidak ada hubungannya sama sekali dengan struktur menu maupun desain *graphical user interface* (GUI) dari aplikasi (Lilly, 1999). Gambar 5a mengilustrasikan kesalahan tersebut yang sering terjadi pada aktivitas CRUD (*create*, *read*, *update*, dan *delete*) untuk sebuah data/objek tertentu, misalnya data pengguna. Kondisi ini kemudian diwakili oleh sebuah UC yang bersifat umum, misalnya `Kelola Data Pengguna`. Semestinya, setiap UC harus merepresentasikan setiap tujuan spesifik yang ingin dicapai oleh setiap aktor (Larman, 2005) (OMG, 2017). UC `Kelola Data Pengguna` tidak memiliki tujuan yang spesifik sebagaimana yang semestinya direpresentasikan pada kondisi akhir yang harus dicapai, apakah data terhapus atau data terbentuk atau yang lain. UC yang seperti ini dikenal sebagai *fat use case* dan harus dihindari karena tidak bisa menjelaskan secara benar fungsionalitas sistem yang harus terpenuhi (Lilly, 1999), meskipun dikecualikan dalam (Larman, 2005).



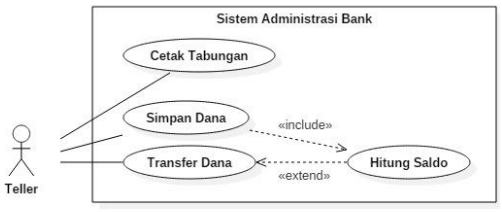
(a) (b)

Gambar 5. UC abstrak yang salah: (a) diagram yang salah; (b) rekomendasi perbaikan

Gambar 5b merupakan salah satu alternatif solusi untuk menjelaskan bahwa setiap UC memiliki kondisi akhir yang spesifik, sebagai representasi tujuan yang ingin dicapai oleh aktor yang spesifik pula (OMG, 2017). Sebagai contoh, prakondisi dan kondisi akhir yang harus terpenuhi pada UC `Buat Data Pengguna` akan berbeda sama sekali dengan UC `Hapus Data Pengguna`, atau yang lain.

3.2.5. UC tidak lengkap tergambarkan

Kesalahan ini terjadi karena kita menganggap bahwa *UC diagram* yang dibuat tidak perlu mengilustrasikan seluruh UC yang ada pada sistem, cukup UC yang penting-penting saja yang sesuai dengan fungsionalitas utama sistem. Semua UC yang (dianggap) semestinya sudah ada tidak perlu digambarkan, yang dalam praktiknya UC `Login` dan UC `Logout` sering diabaikan. Gambar 6 mengilustrasikan sebuah *UC diagram* dari Sistem Administrasi Bank (secara parsial) yang tidak mengilustrasikan bagaimana sistem bisa mengenali aktor Teller, karena tidak ada UC `Login` dalam diagram tetapi pasti ada dalam realisasinya nanti.



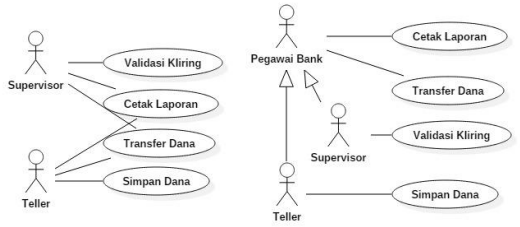
Gambar 6. Sistem tidak tergambarkan secara utuh dalam model: bagaimana mekanisme sistem mengenali aktor Teller?

Rekomendasi perbaikan untuk kasus ini adalah diagram yang dijelaskan pada Gambar 1, dengan menyertakan UC `Login` dan UC `Logout`. Dengan ilustrasi seperti itu maka konteks dari sistem bisa tergambarkan secara lengkap, karena UC *diagram* pada dasarnya juga merupakan diagram yang tepat untuk menjelaskan batasan atau konteks dari sistem yang dikembangkan (Larman, 2005) (OMG, 2017). Dalam konteks ini, apa yang harus berada di dalam atau luar sistem bisa tergambarkan secara jelas. Selanjutnya, setiap UC yang diilustrasikan pada diagram memiliki deskripsi yang detil terkait skenario interaksinya di dalam UC *scenario*. Dengan demikian, interaksi yang terjadi ketika aktor melakukan login bisa dijelaskan secara spesifik dalam UC *scenario*, karena dimungkinkan adanya perbedaan pola interaksi login sesuai kebutuhan.

3.2.6. Generalisasi aktor yang tidak tepat

Kesalahan ini terjadi karena kita terpengaruh dengan konsep penuruan klas (*class inheritance*) di dalam pendekatan berorientasi objek, dimana beberapa klas yang memiliki kesamaan pada sebagian karakteristiknya maka bisa dibuat generalisasinya menjadi klas baru sebagai klas induk. Sekali lagi, hal ini terjadi dalam rangka penyederhanaan diagram untuk menghindari saling silang penggambaran garis relasi aktor dengan UC. Tetapi, penyederhanaan ini berdampak pada makna yang tidak tepat dari sistem yang sedang dijelaskan, sebagaimana diilustrasikan

pada Gambar 7a dan 7b. Diagram awal yang menjelaskan sistem digambarkan pada Gambar 7a, aktor Supervisor dan Teller berbagi UC yang sama yaitu `Cetak Laporan` dan `Transfer Dana`, selain UC yang berbeda di masing-masing aktor. Kedua aktor tersebut memang merepresentasikan peran yang nyata ada di dalam praktik perbankan. Untuk meningkatkan *readability* diagram, perubahan dilakukan dengan melakukan generalisasi aktor menjadi Pegawai Bank untuk mengakomodasi kesamaan pada sebagian UC di antara kedua aktor sebelumnya, sebagaimana diilustrasikan pada Gambar 7b. Generalisasi ini bermasalah karena aktor Pegawai Bank tidak merujuk pada sebuah peran spesifik yang memiliki tujuan tertentu yang harus dicapai di dalam sistem (Larman, 2005). Sebagai rekomendasi, diagram yang dijelaskan pada Gambar 7a adalah ilustrasi yang sudah benar yang menjelaskan secara spesifik aktor (primer) dan UC yang saling berhubungan.

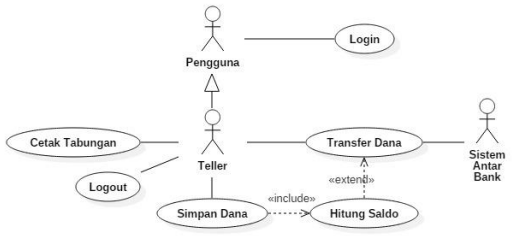


(a) (b)

Gambar 7. Generalisasi aktor yang tidak tepat: (a) diagram awal; (b) generalisasi aktor

3.2.7. Format yang tidak terstruktur

Kesalahan ini terjadi ketika UC *diagram* digambarkan dengan tidak menggunakan format yang terstruktur, sebagaimana diilustrasikan pada Gambar 8. Dari diagram ini, kita tidak bisa membedakan bagian yang mana yang harus berada di dalam atau di luar sistem. Aktor mana yang bersifat primer atau sekunder juga tidak bisa dijelaskan secara akurat. Kedua permasalahan tersebut berdampak pada ketidakjelasan konteks dari sistem.



Gambar 8. Sistem tidak tergambarkan secara terstruktur

Rekomendasi perbaikan yang diberikan mengacu pada Gambar 1. Pada diagram ini, konteks sistem dijelaskan secara terstruktur dimana bagian yang berada di dalam kotak adalah sistem yang sedang dijelaskan dan akan dikembangkan (OMG, 2017). Sedangkan, yang di luar kotak adalah entitas yang berinteraksi langsung dengan sistem yang berupa aktor, di sebelah kiri adalah aktor primer sedangkan di sebelah kanan adalah aktor sekunder.

4. KESIMPULAN

Identifikasi terhadap berbagai kesalahan yang mungkin terjadi dalam pemodelan UC, baik dalam pembuatan *UC scenario* maupu *UC diagram*, telah dibahas dengan menggunakan contoh dan ilustrasi yang relevan. Masing-masing terdapat 11 jenis kesalahan untuk *UC scenario* dan 7 kesalahan untuk *UC diagram*. Dengan mengenal klasifikasi kesalahan yang ada pada pemodelan UC diharapkan permasalahan dalam pengembangan sistem perangkat lunak bisa diantisipasi lebih awal sehingga bisa meningkatkan kualitas perangkat lunak yang dihasilkan dan kemudahan pemeliharaan perangkat lunak di kemudian hari.

DAFTAR PUSTAKA

ADOLPH, S., COCKBURN, A. & BRAMBLE, P., 2002. *Patterns for Effective Use Cases.* Boston, USA: Addison-Wesley Longman Publishing Co., Inc..

BENNET, S., MCROBB, S. & FARMER, R., 2010. *Object-Oriented Systems Analysis and Design Using UML*. 4th Edition. McGraw-Hill Education.

CHEN, J.-C. & HUANG, S.-J., 2009. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6), pp. 981-992.

COCKBURN, A., 2000. *Writing Effective Use Cases.* 1st Edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

ENGELS, G., HECKEL, R. & SAUER, S., 2000. UML --- A Universal Modeling Language?. In: M. Nielsen & D. Simpson (Editors.). *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000 Aarhus, Denmark, June 26--30, 2000 Proceedings.* Springer Berlin Heidelberg, pp. 24-38.

KJEOY, M. A. & STALHEIM, G. M., 2007. *Use Cases in Practice:: A Study in the Norwegian Software Industry*, Master Thesis: Norwegian University of Science and Technology.

LANGE, C. F. J., CHAUDRON, M. R. V. & MUSKENS, J., 2006. In Practice: UML Software Architecture and Design Description. *IEEE Software,* 23(2), pp. 40-46.

LARMAN, C., 2005. *Applying UML and Patterns.* 3rd Edition. NJ: Prentice Hall.

LILLY, S., 1999. *Use case pitfalls: top 10 problems from real projects using use cases.* Santa Barbara, CA, USA, Proceedings of Technology of Object-Oriented Languages and Systems, 1999 - TOOLS 30.

LIU, S. ET AL., 2014. Automatic Early Defects Detection in Use Case Documents. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated software engineering.* ACM, pp. 785-790.

OMG, 2017. *OMG Unified Modeling Language (OMG UML) Ver. 2.5.1.* Object Management Group.

PRESSMAN, R. S., 2010. *Software Engineering: A Practitioner's Approach.* 7th Edition. McGraw-Hill Higher Education.

SINHA, A., SUTTON JR, S. M. & PARADKAR, A., 2010. Text2Test: Automated inspection of natural language use cases. In: *2010 Third International Conference on Software Testing, Verification and Validation (ICST).* IEEE, pp. 155-164.

SINNIG, D., RIOUX, F. & CHALIN, P., 2005. *Use cases in practice: a survey.* Proc. of CUSEC, 2005.