UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO GILBERTO HEITOR GAIEWSKI

Comparison of Techniques for Traceability in DOORS

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Trabalho realizado na Technische Universität Berlin dentro do acordo de dupla diplomação UFRGS - TU Berlin.

Orientador: Prof. Dr. Raul Fernando Weber
Orientador alemão: Prof. Dr. Eng. Stefan Jähnichen
Co-orientador alemão: Thomas Noack

Porto Alegre
2015

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Heitor Gaiewski, João Gilberto

Comparison of Techniques for Traceability in DOORS / João Gilberto Heitor Gaiewski. – Porto Alegre: CIC da UFRGS, 2015.

15 f.: il.

Trabalho de conclusão (graduação) – Universidade Federal do Rio Grande do Sul. Curso de Ciência da Computação, Porto Alegre, BR–RS, 2015. Orientador: Raul Fernando Weber.

1. Traceability. 2. Rastreabilidade. 3. Requirements management software. 4. DOORS. 5. DXL. I. Weber, Raul Fernando. II. Título.

RESUMO

Engenharia de Requisitos é um tema com importância cada vez maior. Gerenciar um número cada vez maior de requisitos não é uma tarefa trivial. Software de Gerenciamento de Requisitos (Requirements Management Software - RMS) exerce um papel fundamental, ajudando a cumprir tal tarefa. Alguns RMS, em particular, fornecem suporte para rastreabilidade de requisitos e especificações. Há, contudo, diferentes técnicas para alcançar o mesmo objetivo. Este trabalho analisa e compara diferentes técnicas de rastreabilidade. Essas técnicas possuem um propósito de aplicação para um contexto específico: rastrear requisitos em um RMS chamado DOORS. Tal contexto emergiu da dissertação de Noack, onde um método para associar requisitos de sistema e casos de teste, de maneira reusável, foi apresentado.

Palavras-chave: Traceability. rastreabilidade. requirements management software. DOORS. DXL.

RESUMO ESTENDIDO

Este é um resumo estendido em português para a Universidade Federal do Rio Grande do Sul do original que segue. O trabalho de conclusão original, em inglês, foi apresentado na Technische Universität Berlin através do programa de dupla diplomação UNIBRAL II entre as duas universidades. O documento do trabalho original segue anexado ao final deste resumo.

1 INTRODUÇÃO

Este trabalho — cujo título, em português, é **Comparação de Técnicas de Rastreabilidade na ferramenta DOORS** — emergiu a apartir da dissertação de doutorado de Noack [13][1], portanto está fortemente relacionado à mesma. O grupo, onde Noack desenvolveu sua tese, trabalha em parceria com a indústria automotiva alemã, em especial com a questão de gerência de requisitos de sistema. Engenharia de requisitos é um tema importantíssimo na indústria. Em 2013 mais de 87 milhões de veículos foram produzidos no mundo (5,7 milhões na Alemanha e 3,7 milhões no Brasil) [1]. Para cada um desses veículos, para cada um de seus componentes, há uma documentação de seus requisitos. Toda vez que novos componentes ou novos veículos são planejados, mais especificações e requisitos são gerados. Todas essas informações são processadas através de um Software de Gerência de Requisitos (Requirements Management Software - RMS).

Noack, em sua dissertação, desenvolveu uma técnica para ser utilizada durante a geração de novos requisitos de sistema, com base em requisitos ou especificações já existentes. Por exemplo, quando uma nova versão de um veículo é projetada com base em sua versão anterior, os requisitos que não serão modificados podem ser reutilizados. Essa associação entre os requisitos novos e aqueles que foram reutilizados é realizada através de uma técnica que aplica o conceito de rastreabilidade.

Esse é o ponto de inserção deste trabalho, onde técnicas alternativas para rastreabilidade são analisadas e comparadas. Nele, a ferramenta de RMS específica utilizada é a ferramenta DOORS. DOORS é uma espécie de banco de dados para requisitos, possui uma representação de requisitos individuais e também para especificações inteiras, como será explicado em mais detalhes na seção de fundamentos. O foco deste trabalho é na rastreabilidade dentro da ferramenta DOORS e as diferentes técnicas para alcançá-la. Dado isto, a pergunta principal deste trabalho é: *como alcançar rastreabilidade no DOORS?*

Sobre o conteúdo deste resumo, primeiramente faz-se necessária uma apresentação dos conceitos básicos, afim de obter uma compreensão mais clara do restante do trabalho; depois haverá um detalhamento do contexto e do trabalho relacionado; então, o problema será apresentado, seguido pelas técnicas propostas para resolvê-lo; e, finalmente, a comparação entre as diferentes técnicas será mostrada.

[1] As referências citadas neste resumo são encontradas ao final da versão original deste trabalho. A numeração apresentada entre colchetes é exatamente a mesma do trabalho original. Portanto, para encontrar a fonte da tese de doutorado citada, basta verificar a referência de número 13 na bilbiografia do trabalho original, anexada ao fim deste resumo.

2 FUNDAMENTOS

Neste capítulo serão mostrados alguns dos conceitos mais básicos para a compreensão deste trabalho. É importante observar que a traduçao das definições apresentadas a seguir é uma tentativa inteiramente livre de prover ao leitor um bom entendimento das mesmas. Para maior precisão, recomenda-se a leitura do trabalho original, onde os conceitos aparecem na língua original em que foram escritos.

2.1 Rastreabilidade (Traceability)

O potencial para estabelecer e usar rastros.[8]

2.1.1 Rastro (Trace)

Um trio de elementos: um artefato origem, um artefato destino e um elo associando ambos artefatos.[8]

2.1.2 Rastrear (to trace)

O ato de seguir um elo partindo de um artefato origem para um artefato destino ou vice-versa.[8]

2.1.3 Rastrear na direção primária

Quando um elo é rastreado partindo do artefato origem em direção ao artefato destino.[8]

2.1.4 Rastrear na direção reversa

Quando um elo é rastreado partindo do artefato destino em direção ao artefato origem.[8]

2.2 DOORS

DOORS significa *Dynamic Object-Oriented Requirements System*, ou algo semelhante a *Sistema Dinâmico de Requisitos Orientado a Objetos*. De forma sucinta, DOORS é um Sistema de Gerência de Requisitos desenvolvido pela IBM, que funciona como uma espécie de banco-de-dados para requisitos. Seus componentes de principal interesse neste trabalho são *módulos*, *objetos* e *elos (links)*, os quais serão descritos a seguir.

2.2.1 Módulos

Um módulo é basicamente um conjunto de *objetos*. Ele é similar a uma tabela em um banco-de-dados. DOORS permite a criação de múltiplos módulos. Quando criado, cada módulo recebe um identificador único, além de um nome descritivo e colunas, as quais representam os atributos dos objetos do módulo. Um módulo pode ser usado, por exemplo, para documentar a especificação de um produto inteiro.

2.2.2 Objetos

Um objeto é um elemento de um módulo, onde dados são, de fato, armazenados. Ele é similar a uma linha numa tabela em um banco-de-dados. Um objeto possui minimamente dois atributos: um identificador localmente único (dentro do módulo) e uma descrição textual. Um objeto pode ter tantos atributos quantos forem necessários. Um objeto pode ser usado, por exemplo, para representar um requisito individual de um produto. No DOORS, um objeto é um artefato (origem ou destino).

2.2.3 Elos (ou *links*)

No DOORS, existe um componente chamado conjunto de elos, que é basicamente uma relação entre um módulo origem e um módulo destino. Um elo é um elemento dessa relação, representado como um par de objetos: um deles pertence ao módulo origem; o outro pertence ao módulo destino. Quando há um elo entre dois objetos, é possível rastrear qualquer um deles a partir do outro. Neste trabalho, em vez de *elos*, emprega-se o termo *Real Links* para referir-se aos elos. *Real Links* são a técnica de rastreabilidade padrão da ferramenta DOORS.

3 CONTEXTO

A idéia principal deste trabalho surgiu após a elaboração da dissertação de Noack: *Associação Automática de Especificações de Requisitos de Sistema e Especificações de Casos de Teste — Um Método orientado a Reusabilidade* [1].

Em sua dissertação, Noack considera o uso de uma Especificação de Requisitos de Sistema original (já existente) para criar uma nova Especificação de Requisitos de Sistema. Também é feita uma associação automática entre a Especificação de Requisitos de Sistema original e a nova, através da geração de *rastros* entre ambas as especificações. Através disso, por exemplo, um requisito individual da Especificação de Requisitos de Sistema original será automaticamente associado a um requisito individual da nova Especificação de Requisitos de Sistema. Essa associação ocorre por meio de um rastro ou, mais especificamente, do elo desse rastro.

Noack também considera o caso em que a Especificação de Requisitos de Sistema original já estava associada com uma Especificação de Casos de Teste. Neste caso, a nova Especificação de Requisitos de Sistema também será, no momento de sua criação, automaticamente associada com a Especificação de Casos de Teste. Da mesma forma, essa associação ocorre por meio de um rastro ou, mais especificamente, do elo desse rastro. A Especificação de Casos de Teste é, de fato, reusada pela nova Especificação de Requisitos de Sistema, porque a necessidade da geração de uma nova Especificação de Casos de Teste foi completamente eliminada. Em sua tese, uma Especificação de Requisitos de Sistema é geralmente um artefato origem. Já os artefatos destino podem ser tanto uma Especificação de Requisitos de Sistema quanto uma Especificação de Casos de Teste. Entre duas Especificações de Requisitos de Sistema existe um rastro chamado de *reuso*, isto é, uma Especificação *reusa* a outra. Entre uma Especificação de Requisitos de Sistema (origem) e uma Especificação de Requisitos de Teste (destino) existe um rastro de *verificação*, isto é, uma Especificação de Casos de Teste *verifica* uma Especificação de Requisitos de Sistema (se rastreado na direção reversa).

3.1 O Problema

No trabalho realizado por Noack, a técnica de rastreabilidade *Real Links* foi utilizada inicialmente. Na prática, no entanto, esta técnica apresentou desvantagens para os usuários. A principal reclamação dos usuários foi sobre o tempo necessário para concluir a exclusão de um

[1] Tradução livre do título original em alemão: *Automatische Verlinkung von Testfaellen und Anforderungen, Eine wiederverwendungsorientierte Methode*.

objeto dentro de um módulo novo, gerado a partir do reuso de um módulo origem. A ferramenta DOORS não permite a exclusão de objetos, quando os mesmos estão associados (atráves de *Real Links*) a outros objetos. Para remover um objeto em tal condição, faz-se necessária a exclusão do todos os elos que o mesmo possui com outros objetos. Essa tarefa precisava ser executada manualmente pelos usuários e exigia uma sequência longa de ações (e.g. trocas de tela ou de janela, cliques do mouse) para excluir cada objeto, isto impactou e reduziu significativamente a produtividade dos usuários. Na prática essa situação surgia frequentemente. É fácil compreender esse fato, pois quando uma nova Especificação de Requisitos de Sistema reusa outra, haverá novos requisitos, requisitos modificados e também requisitos que foram removidos. Portanto, surgiu uma necessidade para uma nova técnica de rastreabilidade dentro da ferramenta DOORS. Com isso, é possível estabelecer as seguintes perguntas a serem respondidas:

1. *Como alcançar rastreabilidade na ferramenta DOORS?*
2. *Quais técnicas de rastreabilidade podem ser empregadas?*
3. *Como se dá a performance dessas técnicas?*

3.2 Estratégias de Solução

Para resolver esse problema é necesário eliminar completamente o uso da técnica padrão de rastreabilidade da ferramenta DOORS (*Real Links*). A primeira estratégia substitui o conjunto de elos por colunas extra nos módulos origem e destino, essa técnica chama-se **Column Link**. A técnica *Column Link* gera, no entanto, uma nova situação, onde há uma restrição. Faz-se necessária uma nova técnica, que respeita essa restrição, a qual chama-se *Only Target Column*. Por sua vez, a técnica *Only Target Column* apresenta problemas de performance, pois realiza, de forma *ingênua*, operações redundantes. Por essa razão, essa técnica será referenciada como **Only Target Column (Naive)**[2]. Por fim, uma última técnica é apresentada para resolver o problema de performance, removendo as operações redundantes e usando mais memória em vez de tempo de processamento. Todas as técnicas serão descritas, em mais detalhes no capítulo seguinte.

[2] *Naive* significa *ingênuo* em inglês.

4 TÉCNICAS DE RASTREABILIDADE

Neste trabalho, um total de 4 técnicas foram analisadas e comparadas. Todas as técnicas foram implementadas através de algoritmos na linguagem DXL [1], os quais podem ser vistos em formato de pseudo-código no capítulo 4 deste trabalho em sua versão original, em inglês.

4.1 Técnica I: Real Links

Como antes mencionado, esta técnica utiliza os conjuntos de elos da ferramenta DOORS. Esses conjuntos de elos são representados dentro da ferramenta como uma tabela. Essa tabela mostra basicamente o produto cartesiano dos objetos de dois módulos ($A \times B$). Cada célula da tabela possui uma cor mais clara, quando não há um elo entre os objetos, e uma cor mais escura, quando há um elo entre os objetos (formando um *rastro*). Um exemplo de um conjunto de elos pode ser visto na figura 4.2 deste trabalho em sua versão original.
Rastrear um objeto com essa técnica é possível através da interação direta com a ferramenta DOORS ou então através de comandos existentes na linguagem DXL.

4.2 Técnica II: Column Link

Esta técnica abdica completamente o uso de conjuntos de elos. Em vez disso, a tabela do conjunto de elos é substituída por uma coluna extra em cada um dos módulos origem e destino. Então, por exemplo, cada linha no módulo de origem (cada objeto origem) possuirá não somente um identificador e uma descrição textual, mas também um atributo com referências para os identificadores daqueles objetos destino, que estiverem associados ao objeto origem. Por exemplo, se houver um rastro entre um objeto $o1$ no módulo origem A e um objeto $d1$ no módulo destino B, haverá um atributo $o1.ref\_B.id = \{d1.id\}$. Note que o identificador único $B.id$ do módulo destino deve estar presente no nome do atributo de referência, para permitir rastreabilidade entre múltiplos módulos. De forma simétrica, há uma coluna $ref\_A.id$ no módulo destino B e um atributo $d1.ref\_A.id = \{o1.id\}$. Note que o valor do atributo é um conjunto de identificadores, pois nada impede que um mesmo objeto tenha rastros com objetos distintos.
Esta técnica disponibiliza a informação dos rastros de maneira redundante, pois ambos os mó-

[1] DXL = *DOORS eXtension Language*.

dulos origem e destino possuem os dados necessários para rastrear os elos de seus objetos. O algoritmo para rastrear os elos entre um módulo origem e um módulo destino é simétrico e pode ser usado em ambas as direções primária e reversa. Esse algoritmo recebe como entrada dois módulos A, B e retorna uma lista de rastros, que pode ser interpretada como uma lista de pares chave-valor, contendo todos os objetos de B como chaves da lista; e como valor, o conjunto dos objetos de A associados com o respectivo objeto chave. O algoritmo em pseudo-código encontra-se na seção 4.2 deste trabalho em sua versão original (vide pg. 22 e 23).

Problema: esta técnica exige a modificação do módulo origem, adicionando uma coluna extra ao mesmo. Na prática, alguns módulos são reutilizados por diversos outros módulos, portanto exigem a criação de n colunas adicionais. Com isso, surge claramente uma restrição que proíbe a modificação do módulo de origem, ou seja, somente o módulo destino pode conter uma coluna extra.

4.3 Técnica III: Only Target Column (Naive)

O nome desta técnica, em tradução literal, é *Apenas Coluna Destino (ingênuo)*. Este nome é um reflexo direto da restrição apresentada após a técnica *Column Link*. Esta técnica abdica completamente qualquer modificação na estrutura do módulo origem. Há, portanto, uma coluna extra somente no módulo destino.

Felizmente, a mesma função usada para a técnica *Column Link* ainda pode ser aplicada para rastrear na direção reversa, pois toda a informação dos rastros está disponível no módulo destino. O mesmo não é verdade quanto ao módulo origem, pois este não possui qualquer tipo de informação sobre os rastros. Com isso, faz-se necessário um novo algoritmo para rastrear na direção primária.

Esse algoritmo foi aplicado por Noack e baseia-se em uma estratégia bem simples. Ele tenta resolver o problema primeiro de forma local, isto é, toma-se apenas um objeto origem e procura-se todos os seus rastros no módulo destino. Conseqüentemente, para cada objeto origem haverá uma varredura completa do módulo destino. O algoritmo consiste então, de um laço dentro de outro, gerando uma complexidade quadrática, como pode ser visto em pseudo-código da seção 4.3 na versão original deste trabalho (pg. 24 e 25).

Problema: a performance desta técnica para rastrear em direção primária mostrou-se inaceitável. Existe uma certa redundância no seu algoritmo, gerando um nível de complexidade excessivo, por isso esta técnica foi classificada como ingênua.

4.4 Técnica IV: Only Target Column (Cache)

Esta técnica respeita, assim como a técnica anterior, a restrição de não modificar o módulo origem. Há, portanto, uma coluna extra somente no módulo destino. A idéia básica desta técnica é parar de varrer o módulo destino repetidas vezes. Para tanto, é necessário armazenar em memória os resultados encontrados na primeira varredura, como uma espécie de *Cache*. A coluna extra do módulo destino, que possui todas as informações de rastro com o módulo origem, será o alvo da primeira e única varredura do algoritmo. Para cada objeto destino, o conjunto de identificadores origem associados a ele será analisado. A partir disso, será construída uma espécie de *Cache* em formato de lista de pares chave-valor. Quando um identificador origem estiver presente na coluna extra do módulo destino, esse identificador será criado como chave da *Cache* (caso ainda não tenha sido criado anteriormente) e, como valor correspondente, o identificador do objeto destino será acrescentado ao conjunto-valor. Ao final da varredura, a *Cache* será uma lista onde cada elemento tem como chave o identificador origem e como valor um conjunto de todos os identificadores de objetos destino associados ao objeto origem. Esse resultado será armazenado em memória. O pseudo-algoritmo desta técnica encontra-se na seção 4.4 da versão original deste trabalho (pg. 26 e 27).
Nas próximas execuções do algoritmo, nenhuma varredura será executada, mas sim uma consulta à *Cache* construída anteriormente.

5 ANÁLISE E RESULTADOS

Para realizar a comparação entre as diferentes técnicas, foram elaborados 8 cenários distintos. Os cenários analisam cada uma das 4 técnicas em 2 situações diferentes: rastrear na direção primária e rastrear na direção reversa.

Para cada cenário são necessários minimamente:

- Um módulo origem com 2.000 entradas;
- Um módulo destino com 2.000 entradas;
- Ao menos um rastro (elo) para cada objeto origem e destino;
- Cerca de 10% de rastros múltiplos;
- Mesmo ambiente de execução;

A quantidade fixa de entradas (objetos ou linhas) de cada módulo é uma estimativa feita com base na experiência dos usuários e de Noack, onde uma Especificação de Requisitos de Sistema possui em média 2.000 requisitos individuais. Por *rastro múltiplo* entende-se um objeto que participa de, no mínimo, 2 rastros (por exemplo, no atributo da coluna extra existe um conjunto com mais de um elemento). É importante destacar, também, que todos os testes foram executados a partir de um mesmo ambiente de execução.

A idéia básica é medir os tempos de execução de cada técnica em cada cenário, mas sem prerrogativa de qualidade da precisão na medição. O objetivo é verificar se os tempos de execução são aceitáveis, através da comparação entre eles. Todos os dados necessários foram gerados com a ajuda de *scripts* desenvolvidos na linguagem DXL.

5.1 Resultados

Os dados de tempos de execução, em milisegundos, apresentados a seguir foram obtidos através de uma média aritmética de um número de execuções, não superior a 5 vezes, em cada cenário. Se necessário, o tempo de execução em segundos será mostrado entre parênteses.

5.1.1 Técnica I: Real Links

- **Rastrear na direção primária:** 515 ms (0,5 s)
- **Rastrear na direção reversa:** 574 ms (0,6 s)

5.1.2 Técnica II: Column Link

- **Rastrear na direção primária:** 1955 ms (2 s)
- **Rastrear na direção reversa:** 1497 ms (1,5 s)

5.1.3 Técnica III: Only Target Column (Naive)

- **Rastrear na direção primária:** 972.042 ms (972 s ou 16 min 12 s)
- **Rastrear na direção reversa:** 1482 ms (1,5 s)

5.1.4 Técnica IV: Only Target Column (Cache)

- **Rastrear na direção primária:** 663 ms (0,7 s)
- **Rastrear na direção reversa:** 1482 ms (1,5 s)

5.2 Análise dos Resultados

De forma sucinta, pode-se dizer que o único resultado completamente inaceitável apresentado ocorre na técnica III, no cenário *rastrear na direção primária*: mais de 16 **minutos**. Felizmente, o problema de performance apresentado nesse caso foi superado pela técnica IV que, já na sua primeira execução apresenta um resultado muito próximo daquele apresentado pela técnica I (padrão disponbilizada pela ferramenta DOORS): menos de 1 **segundo**. É importante relatar que, assumindo que não hajam alterações nos dados dos módulos, a técnica IV (*Cache*) apresenta, a partir de sua segunda execução, tempos de execução tão pequenos que não podem ser medidos em milisegundos. Sendo assim, quando rastrear na direção primária, a partir da segunda execução, já existe um ganho de performance sobre a técnica I.
Todos os demais resultados são considerados satisfatórios ou aceitáveis.

6 CONCLUSÃO E PERSPECTIVA FUTURA

Após a realização deste trabalho pode-se verificar que as perguntas apresentadas, na seção 3.1 deste resumo, foram devidamente respondidas. As perguntas de número 1 e 2 foram respondidas tanto na seção 3.2 quanto no capítulo 4 deste resumo, onde as técnicas são apresentadas; e a pergunta de número 3 foi respondida no capítulo 5 deste resumo, onde os resultados são apresentados. Também foi mostrado que, mesmo que a ferramenta DOORS já tivesse sua própria técnica de rastreabilidade, é possível implementar alternativas para a mesma.

Para cada problema encontrado em cada uma das técnicas apresentadas, foi buscada e encontrada uma solução. Durante o processo, quatro técnicas diferentes foram criadas e analisadas. O resultado obtido no final foi satisfatório, preenchendo as expectativas.

Um fato que poderia passar despercebido e merece destaque é a genericidade das técnicas apresentadas. Os exemplos sempre trouxeram o contexto de requisitos à tona, mas na verdade não há nenhuma dependência das técnicas quanto a isso. Ou seja, as técnicas aplicam-se aos módulos e objetos, o valor semântico atribuído a eles não exerce nenhuma influência sobre elas. Um objeto, por exemplo, pode representar requisitos, um caso de teste ou qualquer outra informação e, mesmo assim, a rastreabilidade do mesmo continua sendo fornecida pelas técnicas apresentadas.

É importante ressaltar que algumas condições devem ser atingidas, afim de que as técnicas tenham um desempenho aceitável. A técnica denominada *Cache*, por exemplo, possui essa nomenclatura mais como uma referência do que por definição. Na verdade, toda vez que ocorre uma modificação nos dados dos módulos, é necessário recomputar os valores da *Cache*. Ainda assim, é possível obter um desempenho aceitável assumindo que o número de consultas à *Cache* deve ser relativamente maior que o número de modificações nos dados dos módulos. Por outro lado, é possível que essa técnica seja aprimorada. Por exemplo, adicionando-se mecanismos de consistência, para atualizar modificações automaticamente e também implementar persistência dos dados em memória não-volátil. Isso permitiria que a *Cache* fosse armazenada e pudesse ser recarregada, até mesmo com o encerramento da execução do seu processo ou quando ocorresse uma queda de energia, forçando o desligamento do computador rodando o processo.

Por fim, pode-se dizer que este trabalho deixa algumas *portas* abertas para trabalhos futuros.

Comparison of Techniques for Traceability in DOORS

Author:
João Gilberto Heitor Gaiewski

Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
Lehrstuhl Softwaresystemtechnik

Bachelor Thesis

Examiners:
Prof. Dr.-Eng. Stefan Jähnichen
Prof. Dr. Peter Pepper

Supervisor:
Thomas Noack

Berlin 2015

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertig habe. Die selbständige und eigenhändige Anfertigung dieser Arbeit versichere ich an Eides statt.

Berlin, 2015

Unterschrift

Abstract

The subject Requirements Engineering is very important in the industry. To manage crescent amounts of product requirements is not an easy task. Requirements Management Software (RMS) plays a fundamental role, helping to deal with all that information. Some RMS, in particular, support traceability of requirements and specifications. There are, however, different techniques to achieve it. This work compares and evaluates different traceability techniques. These techniques are intended to be applied in a specific context: trace reused system requirements in a RMS called DOORS. This context emerged from Noack's dissertation, where a method to automatically link requirements and test cases, in a reusability-oriented way, is presented.

Zusammenfassung

Requirements Engineering ist ein Thema mit immer größer werdender Relevanz. Die Verwaltung von immer größeren Mengen von Anforderungen für Produkten stellt keine triviale Aufgabe dar. Requirements Management Software (RMS) hat bei dieser Aufgabe einen fundamentalen Stellenwert, um diese wachsende Menge an Daten verarbeiten und verwalten zu können. Einige RMS bieten dazu Anforderungsverfolgbarkeit an. Um diese zu erreichen, werden unterschiedliche Techniken verwedent. Diese Arbeit vergleicht und evaluiert verschiedene Verfolgbarkeitstechniken, die in einem spezifischen Kontext angewandt werden: die Anforderungsverfolgbarkeit des DOORS-RMS. Dieser Kontext wurde erstmalig in der Dissertation von Noack erwähnt, die eine Methode zur automatischen Verlinkung von Testfällen und Anforderungen präsentiert hat.

Contents

List of Acronyms

DOORS Dynamic Object-Oriented Requirements System

DXL DOORS eXtension Language

IBM International Business Machines Corporation

RMS Requirements Management Software

SRS System Requirements Specification

STS Specification of Test Suites

List of Figures

List of Tables

1 Introduction

Requirements Engineering is very important in the industry. To understand why, it is good to start with an example. In 2013, the automotive industry produced more than 87 million 350 thousand vehicles world wide (for mere curiosity: 5.7 million in Germany and 3.7 million in Brasil) [1].

Before one of these vehicles (for example, a car) went to production, it was, of course, modeled and planned. At some point in the planning process, there should be a document with its specification written in natural language, also known as System Requirements Specification (SRS). The SRS describes in detail how does the car work. However, it is not that simple. Every part of the car and every single function of its parts must be specified. The SRS might contain, for example, a section for the front door functionality and another one describing the door lock mechanism, and so on, for every component and function. Considering the amount of components needed to build a car, multiplied by the number of different functions each component has, it is possible to imagine how huge is the resulting set of functions that have to be specified/documented. For each function, one corresponding requirement has to be written. Inside an automotive industry, with lots of employees, a simple text document containing the requirements would obviously not work. To accomplish such a task (writing requirements for the entire set of functions), one can count with the help of specialized computer programs: Requirements Management Software (RMS). Roughly, a RMS can be seen as a requirements-database. With a RMS it is possible, for example, to create and manage the System Requirements Specification (SRS) of a car, simultaneously and among multiple users. For that purpose, a RMS is clearly much more feasible than a simple text document.

If there is a project for a new version of a car, a new specification (SRS) can be created within the RMS. That means, someone would have to write the entire new specification from the beginning, with its complete huge set of requirements. Since it specifies a new version of a car, the SRS contains

some new functions and, maybe, does not have some parts or functions from the previous version. But many functions are very similar or even exactly the same. For example, the new version could be a convertible car, with only two doors instead of four. So, when creating the SRS of this new version, the basic functions are still the same. It would be a waste of time to rewrite them. A RMS has a way to solve this problem: the ability to *trace* requirements. Through *Traceability*, different systems can share some requirements in common, without the need to rewrite them. It saves time. Multiple requirements can be shared across different systems. For example, when the same engine is used in many car models, their specification will share the engine requirements.

If there is a change in the original requirement, which is reused in several other SRS, what happens? For example, suppose that the design of a car trunk led people to risky situations, where they can get hurt when closing the car trunk in a certain way. Then, there is an urgent need to change the car trunk's design, in all car models containing it. The requirements of the car trunk also need to be changed, across all specifications containing it. Ideally, in a RMS, when changing the original requirement, all requirements shared with other SRS shall be automatically "updated". This can be achieved through *Traceability* techniques.

The problem

In this work, the specific RMS used is called DOORS. In DOORS, data can be represented as modules, objects or links. From the motivation example, requirements are implemented as objects and System Requirements Specification (SRS) are implemented as modules. This work focuses on objects traceability across different modules. In fact, there are different ways to achieve it. Given that, the main question is: *how to achieve Traceability in DOORS?* There is a related work, which has shown efficiency problems. In this work, different *Traceability* techniques will be presented and compared.

Contents

First, a definition for traceability will be given, as well as a definition for Requirements Engineering. Second, some basic concepts of DOORS will be introduced to the reader, followed by its initial data configuration (modules and objects). Next, different approaches for traceability in DOORS will be discussed and the techniques implemented will be shown. Then the different scenarios will be introduced. Finally a comparison of the techniques will take place.

2 Fundamentals

In this chapter some basic concepts for the comprehension of this work will be introduced. The first sections present more abstract concepts, describe the major area and the fundamentals of this work. The last section presents technical information about DOORS. In a higher level, the context of this work is within Software Engineering, more specifically Traceability and Requirements Engineering.

2.1 Requirements Engineering

The goal of this section is to bring basic knowledge of Requirements Engineering. For sure, the most meaningful word in this expression is not "Engineering", but "Requirements". The word "requirement" implies something that is needed, compulsory. But what is a requirement in the context of software engineering? Sommerville says a requirement (or system requirement) defines "what the system is required to do and the circumstances under which it is required to operate" [2]. According to Robertson, a requirement may even be a quality the system must have [3]. It is interesting to observe that even IEEE changed its definition of requirement, between the years of 1990 and 1998. The latest is still the current standard definition, as one can verify in the review of 2005 [4].

> **Requirement (IEEE, 1990)** — *"(1) A condition or capability needed by a user to solve a problem or achieve an objective.*
> *(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents."* [5]

> **Requirement (IEEE, from 1998 on)** — *"A statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable,*

> *and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).”* [6]

For the purposes of this work, the latest definition is more suitable, specially because it says a requirement is a statement — that means, text written in natural language, which is a very concrete description — and it contains the fundamental part, referring to needed functionality, characteristics and constraints.

Requirements can be further analyzed and divided into different types: functional requirements, non-functional requirements and constraints[1]. Functional requirements are the thing the system must do [3]. Non-functional requirements may be a quality the system must have, it may be, for example, a security requirement [3]. A non-functional requirement may also be related to performance or usability of the system [2]. Constraints are another type of requirements. They are usually global restrictions to the shape of a project or to its design [3] (for example: size, time or budget constraint). Deeper analysis on types of requirements is out of the scope of this work, since the focus is just to have a basic knowledge of Requirements Engineering.

Thus, *what is Requirements Engineering?* In 1998, Sommerville mentioned it as “a new term invented to cover the activities involved in discovering, documenting and maintaining requirements for a system [...]” [2]. A similar, but more recent, definition is given by Hull:

> **Requirements Engineering** — *“The subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction.”* [7]

She alerts about the emphasis on the term *Engineering*, in order to make clear that Requirements Engineering is a broad classification. It is not restricted to software requirements, but comprises system (or product) requirements in

[1] For more information on types of requirements, it is recommended to take a deeper look into Robertson’s book [3].

general. She also alerts that it is important to distinguish Requirements Engineering from the activities inside it, like requirements analysis, management and development [7].

2.2 Traceability

The goal of this section is to bring basic knowledge on Traceability. The word "traceability" derives from "trace". Thus, in order to understand the first, a gaze into the latter is necessary. The common meaning of trace could be interpreted as an evidence, a vestige, a footprint of an animal or even an object left in the scene of a crime; in the other hand, it could be the action of searching for the origins of these evidences, vestiges or footprints. In the context of Software Engineering, Cleland-Huang et al. constructed relatively analogous definitions, in an abstract way:

> **Trace (as a noun)** — *"A triplet of elements: a source artifact, a target artifact and a trace link associating both artifacts [...]."* [8]
>
> **Trace (as a verb)** — *"The act of following a trace link from a source artifact to a target artifact or vice-versa [...]."* [8]

These definitions are based on terms, such as "source/target artifact" and "trace link", which were not yet introduced. It is possible to *trace back* their meaning, since they were also formally defined:

> **Source artifact** — *"The artifact from which a trace originates."* [8]
>
> **Target artifact** — *"The artifact at the destination of a trace."* [8]

Source and target differ just in the semantics and direction they have. Both source and target artifacts can be generalized as a trace artifact.

> **Trace artifact** — *"A traceable unit of data."* [8]

By unit of data, Cleland-Huang et al. mean it might be at any granularity level. Hence, an atomic requirement or a set of multiple requirements, are possible trace artifacts. A module or class in a programming language, a test

case, a database model, an entity in a database, a prototype for the layout of a web page and a stakeholder (yes, a human being) are another examples of artifacts, each one from a different *type*.

> **Trace artifact type** — *"A label that characterizes those trace artifacts that have the same or a similar structure (syntax) and/or purpose (semantics)."* [8]

In the previous examples, a "set of requirements" is an artifact of type "requirement"; a "prototype for the layout of a web page" is an artifact of type "design"; and a "test case" is an artifact of type "test case".

The last term mentioned in both trace definitions (as a verb and as a noun), which was still not introduced until this point, is *trace link*.

> **Trace link** — *"A specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact."* [8]

Although this definition implicitly describes a direction for the trace link (from source to target, or *primary trace link direction*), it does not necessarily limit the traversal of the trace link. In other words, the link could also be traversed from target to source artifact (also known as *reverse trace link direction*), depending on the implementation. If so, the result would be called a *bidirectional trace link*.

> **Primary trace link direction** — *"When a trace link is traversed from its specified source artifact to its specified target artifact."* [8]
>
> **Reverse trace link direction** — *"When a trace link is traversed from its specified target artifact to its specified source artifact."* [8]
>
> **Bidirectional trace link** — *"A trace link that can be used in both primary and reverse trace link directions [...]."* [8]

Trace links, as well as trace artifacts, can also be of a certain type. The type of a trace link should describe its semantics, or meaning. For example: target

artifact "reuses" source artifact, where the trace link type is "reuses". A set of trace links builds up a *trace relation*.

> **Trace link type** — *"A label that characterizes those trace links that have the same or a similar structure (syntax) and/or purpose (semantics)."* [8]
>
> **Trace relation** — *"All the trace links created between two sets of specified trace artifact types. The trace relation is [...] a collection of traces."* [8]

After an introduction to the meanings of the word *trace*, just a small step is needed to present a definition of traceability, constructed on top of this basic knowledge.

> **Traceability** — *"The potential for traces to be established and used. [...] It is thereby an attribute of an artifact (or of a collection of artifacts). Where there is traceability, tracing can be undertaken and the specified artifacts should be traceable."* [8]

This is a very rich definition, that carries a high level of abstraction. It can be applied for different contexts, because it is very generic and, at the same time, very precise. There are other definitions for traceability within a special context. A classical definition by Gotel and Finkelstein makes an interesting contrast to the one just presented, since it applies to the specific context of requirements traceability.

> **Requirements Traceability** — *"The ability to describe and follow the life of a requirement in both forward and backward directions [...], from its origins through its development and specification, to is subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases."* [9]

In the previous definition, there are concerns about origins, different phases and deployment of a requirement. When put in this order (origins to deployment), comparing different levels of abstraction, traceability can be seen as a vertical line, connecting the points in the life of a requirement. This is

called, according to Cleland-Huang et al., vertical traceability or the ability for *vertical tracing*. When considering traces at the same level of abstraction, the name given by the authors is horizontal traceability or the ability for *horizontal tracing*.

> **Vertical tracing** — *"In software and systems engineering contexts, the term is commonly used when tracing artifacts at differing levels of abstraction so as to accommodate life cycle-wide or end-to-end traceability, such as from requirements to code. Vertical tracing may employ both* forward tracing *and* backward tracing*."* [8]

> **Horizontal tracing** — *"In software and systems engineering contexts, the term is commonly used when tracing artifacts at the same level of abstraction, such as: (i) traces between all the requirements created by "Mary"; (ii) traces between requirements that are concerned with the performance of the system; or (iii) traces between versions of a particular requirement at different moments in time. Horizontal tracing may employ both* forward tracing *and* backward tracing*."* [8]

In the previous definitions, some terms, such as forward and backward were mentioned. There are also special definitions for them:

> **Forward tracing** — *"In software and systems engineering contexts, the term is commonly used* ***when the tracing follows subsequent steps in a developmental path****, which is not necessarily a chronological path,* ***such as forward from requirements through design to code****. Note that the trace links themselves could be used in either a primary or reverse trace link direction, dependent upon the specification of the participating traces."* [8]

> **Backward tracing** — *"In software and systems engineering contexts, the term is commonly used* ***when the tracing follows antecedent steps in a developmental path****, which is not necessarily a chronological path,* ***such as backward from code through design to requirements****. Note that the trace links themselves could*

> *be used in either a primary or reverse trace link direction, dependent upon the specification of the participating traces.”* [8]

By the end of this section, the most important traceability concepts for this work were given. This section should serve as a reference for the reader.

2.3 DOORS

DOORS is a Requirements Management Software commercialized by IBM. DOORS stands for *Dynamic Object-Oriented Requirements System* [10]. In a very rough way, DOORS can be described as a database for requirements. Indeed, its functionality goes far beyond that[2]. The goal of this section is to bring only the needed knowledge about this tool, in order to understand it within the context of this work. The practical tasks of this work were developed using this tool. The main components of interest here are *modules*, *objects* and *links*, which will be examined and explained in the next few paragraphs.

Modules

There are different types of modules. When the word “module” is used, it is usually meant, more precisely, “formal module”. For the sake of simplicity, in this document the variant “module” will be adopted. A module is basically a set of *objects*. It is similar to a table in a database. In DOORS, multiple modules can be created. When created, each module receives one unique identifier. A module has a name, which is merely descriptive and can be changed. A module has also columns, which show attributes of its *objects*. A module can be used, for example, to document product requirements.

Objects

An object is an element of a module, where data can be stored. It is similar to a row inside a database’s table. An object has at least two attributes: a

[2] For more specific information about DOORS, check the online documentation at [10]

locally unique numerical identifier (only unique inside its module) and a textual description. An object may have as many custom attributes as desired. An object can be used, for example, to represent a single requirement of a product. In DOORS, an object is a *trace artifact*.

Links

In DOORS, there is a type of module called *link module*. A link module contains a list of *link sets*. A set of links, or link set, is a relation (R) between a source module (S) and a target module (T)[3]. A link is an element of the link set and is represented as a pair of objects, one belongs to the source module (s) and the other one belongs to the target module (t). In respect to its target module, a link is called an "incoming link"; in respect to its source module, it is called "outgoing link". DOORS links are *bidirectional trace links*, therefore they are obviously *trace links* too. From now on, and only for the scope of this work, DOORS links will be referred simply as *real links*. In DOORS, real links are the standard technique to provide *Traceability*.

$$R_{S \to T} \subseteq S \times T; link \in R_{S \to T} \iff \exists s \in S; \exists t \in T; link = (s, t)$$

DXL

DOORS eXtension Language (DXL) is a C–like scripting language [11]. DXL works similarly to a macro script, in relation to a database management system; or bash scripting, in relation to Unix-based operational systems. It allows one to performs actions, control and give commands to the system. There are several critiques about it though. DXL was not designed following all the best practices, it has indeed some design flaws and workarounds to enable some features (see, for example [12]). It is still a very useful tool, since it avoids a lot of manual work and has well designed features also (see, for example *skip lists*, at pg. 140 of the reference manual [11]). All techniques for traceability proposed in this work were implemented through DXL.

[3] In DOORS, source and target modules will indicate the *primary trace link direction*. Despite of that, links can be navigated in both directions.

3 Problem and Related Work

This chapter aims to examine, in a more detailed way, the problem for which this work proposes a solution: how to achieve traceability in DOORS, comparing different techniques. But first, the origins of the problem must be understood. For this reason, it is important to start with a section about related works and, after that, it will be much easier to introduce and comprehend the problem.

3.1 Related Work

The core idea of this work arose after a dissertation elaborated by Noack: "Automatic Linkage of System Requirements Specifications and Specification of Test Suites — A Reusability-oriented Method"[1].

In his dissertation, Noack has chosen a reusability-oriented approach[2]. The author considers the reuse of one original System Requirements Specification to create a new System Requirements Specification (SRS) — a clear example of *horizontal tracing* — and also establishes a method to automatically link the original SRS with the new SRS [13], through a *trace relation*. In a finer level of granularity, single original product requirements in the reused SRS are automatically linked with the respective single product requirements in the new SRS. This linkage happens through a *trace link*.

Noack also considers the scenario where the reused original SRS was already linked with a Specification of Test Suites (STS). In this *vertical tracing* sce-

[1] Free translation (from German). The original title is available in the bibliography [13].

[2] In this work, no definition for the concept of reusability is presented. Since reusability is a main issue rather in Noack's dissertation, than in this work, it would be out of context. It is recommended to search through the author's bibliography, in order to get a definition for reusability [13].

nario, the new SRS will, by its creation time, not only be automatically linked with the reused original SRS, but it will also be automatically linked with the Specification of Test Suites (STS) from the original SRS [13]. This linkage happens through a *trace link*, just the same way as with the reused SRS. Indeed, the STS are also *reused* by the new SRS, because two different SRS will automatically share the same STS, without the need to create a new one. In other words, the STS will be used again (or *reused*), without the need to be duplicated. In a finer level of granularity, the *trace artifacts* (i.e. SRS, STS) are reduced, for example, to single product requirements (instead of SRS) and single test cases (instead of STS).

In Noack's dissertation, a *source artifact* is usually a SRS (also: a single product requirement); a *target artifact* is either a STS (also: a single test case) or another SRS (also: another single product requirement); the *trace link type* chosen by the author is named either "reuses" — for a *trace link* between two SRS — or "verifies" — for a *trace link* between a SRS and a STS. The author also has decided to use the *reverse trace link direction* to capture traceability semantics [13].

A typical example of semantics is expressed by the following sentence:

> "*Target artifact reuses source artifact*"

Other example (Vertical Tracing):

> "*Test case T1.1 verifies product requirement PR1.1*"

Another example (Horizontal Tracing), a bit more concrete:

> "*System Requirements Specification SRS-new-car-version-2 reuses System Requirements Specification SRS-car-version-1*"

Traceability Techniques

In his DOORS–Plugin "Auto-Linker" [13], Noack implemented *Traceability* firstly through real links. Afterwards, for some specific reasons (which will be discussed later in this document), he had to migrate to a different solution,

without real links. This solution replaced real links by one new column in each of both source and target modules, as a new representation of the trace link. But it also had a problem (discussion further in this document). From now on and only for the context of this work, this solution will be identified as *Column Link*. Its details will be discussed further in this document. Then, Noack came up with a different approach, which had performance problems. This last approach will be identified, within the context of this work, as *Only Target Column*.

Finally, an important observation about Noack's dissertation is that the linkage of trace artifacts enables *Requirements Traceability*.

3.2 The Problem

In this section, the problems encountered by Noack will be described. The reasons for the different approaches and the unsolved problem, which was left open by Noack and is the main task of this work, will be explained in more detail.
In an abstract level, three questions may resume the problem:

1. How to achieve *Traceability* in DOORS?
2. Which different traceability techniques can be used?
3. How do such techniques perform?

To answer the first two questions, one needs to start to talk about links in DOORS (real links). Then, one needs to understand why to try different approaches, if DOORS already provides a standard way to create *Traces*?

Real Link *Traceability* can be achieved through the use of real links, link sets and link modules. It is an integrating part of DOORS. However, there are some very small implementation details about real links, which slow down the time to accomplish trivial tasks. In practice, that turns out to be a problem.

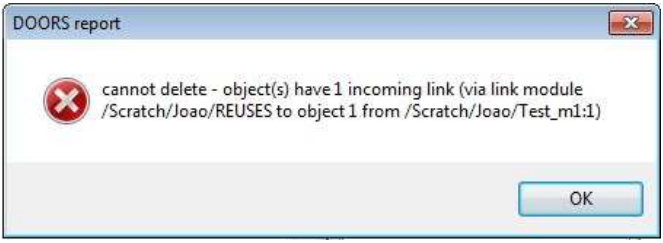

Figure 3.1: Delete object error message

There is one key example, which enables to notice this problem. It remounts the steps needed to delete an object.
To delete an object is usually a trivial task, which takes only one step to be finished (one mouse *click* at the confirmation dialog). This is not true if the object in question is linked (through a real link) with any other object. In this case, the user receives an error message telling that the object has an incoming link and can not be deleted. The error message does not even explicitly tell what to do, in order to *de facto* delete that object (figure 3.1). There is, however, a workaround to delete the linked object, which requires extra effort: first, the link module has to be opened (step 1); then, the wished link set has to be selected in the full list of link sets (step 2); all the specific links, containing the object in question have to be deleted, one-by-one — that requires at least two *clicks* per object, because a confirmation dialog is shown — (step 3); after saving the link set, it is possible to delete the object in question (step 4).
It takes at least 4 steps to be able to delete a single object, when this object is linked through real links (not to mention the number of *clicks*). Hence arose the need for an alternative way to provide *Traceability* in DOORS.

Column Link This was the first alternative tried out by Noack. This solution replaces real links by one new column in each of both source and target modules — a new way to represent the trace link. The problem from this

approach is that it modifies the source module and that is not desirable, thus not allowed. It is not hard to perceive why such thing is not desirable: one single source module might be linked with many other target modules. Therefore, it needs one extra column for each target module it is linked with. Thus, the source module multiplies its size and would be very slow to execute read operations (in other words, the source module would take much more time to be loaded or opened). In addition, the information provided by the numerous column links would prejudice the comprehension of the source module's real content.

Only Target Column This was the last alternative tried out by Noack, which had efficiency problems. It could have been named *Only Target Column Link*, because it is based on the previous approach. The main difference lays on the removal of the link column from the source module, so that the source module is not modified. The very well known Computer Science's trade-off shows up again: *memory usage* $\times$ *time consumption*. By eliminating a column, a lack of information is provided (less memory usage). To fulfill this lack of information, there is a need to compute it through algorithms (more time consumption). The strategy applied by Noack was based on a very simple idea and had efficiency problems. Its main problem was to go through the source module, scanning every single object and, for each one, scanning the whole target module. That means, the algorithm's complexity was $O(n^2)$, or quadratic-complexity [14]. For the context of this work, this approach will be referred as *Only Target Column (Naive)*.

In this work, this efficiency problem is going to be solved, through a technique similar to the last one, and all the different techniques are going to be compared in terms of performance.

3.3 Solution Strategy

The basic idea behind the solution of this efficiency problem lies on reduction of the algorithm's complexity. There are some steps in the *Only Target Column (Naive)* approach, which perform redundant operations. A key strategy for the solution is to build a type of *Cache* to store the important information, instead of searching for it again and again every time.

4 Techniques for Traceability

In this chapter, all the different techniques for *Traceability* in DOORS will be presented. The goal of this chapter is to answer the two first questions presented in the previous chapter, section 3.2 (pg. 17) and solve the problem from the technique *Only Target Column (Naive)*.

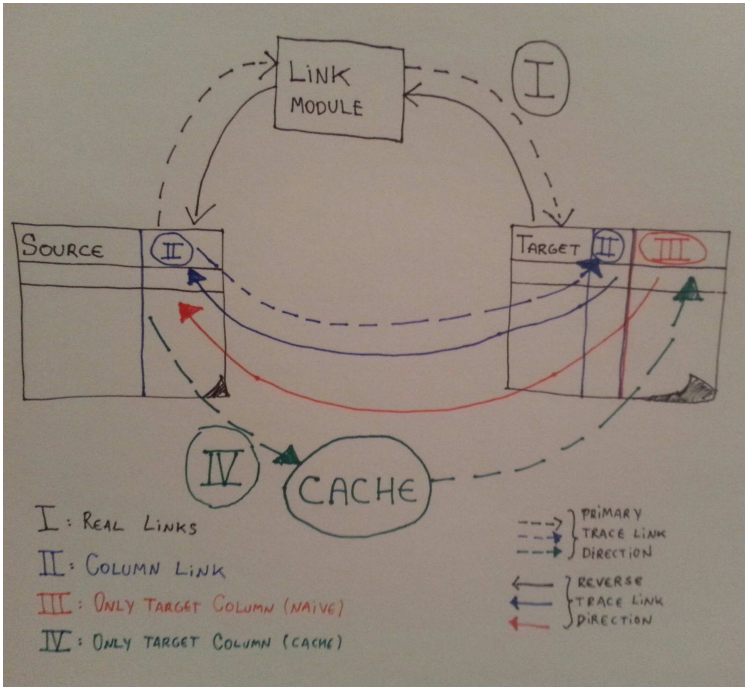

Figure 4.1: Big picture

An overview of the problem and all presented techniques, each one numbered from I to IV according to the progression level of the solution, is shown in

figure 4.1: real links (I) use a link module; column link (II) uses an extra column in both source and target modules and in both primary and reverse trace link directions; only target column (III) uses an extra column only in the target module and for the reverse trace link direction, being "naive" in the primary trace link direction; finally, only target column with cache (IV) does the primary trace link direction using extra memory, very similarly to a caching system.

In order to better understand all the techniques, the same *trace artifacts* (modules A, B and their objects) and the same *trace relation* ($R_{A \to B}$) will be used as example until the end of this chapter. These are defined as follows:

$$A = \{a1, a2, a3\}, B = \{b1, b2, b3\}$$
$$R_{A \to B} = \{(a2, b2)\}$$

4.1 Real Links

Real links or, more precisely, link sets are the base for *Traceability* in DOORS. A link set contains a matrix representing the *Trace Relation* between two arbitrary modules, also known as *Traceability Matrix* [8]. The link set is graphically represented through a *traceability matrix* M, with one cell for each possible element in $A \times B$; i.e. M has one line for each element of A ($|A|$ lines), one column for each element of B ($|B|$ columns) and is represented as $M_{|A| \times |B|}$ (in the example, $M_{3 \times 3}$). In this matrix, a cell is marked with a darker color if and only if $link \in M$; otherwise it is marked with a lighter color. In the example, it is possible to note that the cell corresponding to the pair (a2, b2) is marked with a darker color (figure 4.2), which means "there is a link between artifacts a2 and b2".

When using real links, there are two ways *to trace* objects. One of them is through the user graphic interface, by clicking on an object. The other way is through DXL, using the operators "->" and "<-" (see pg. 377, "Finding Links" at [11]).

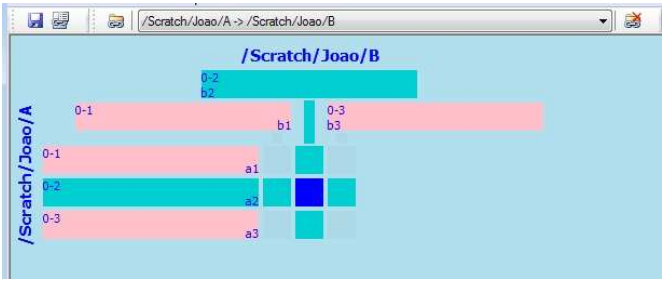

Figure 4.2: DOORS link set

4.2 Column Link

This technique completely abdicates to use real links. The *Column Link* technique consists of creating an extra column in both target and source modules (respectively *target artifact* and *source artifact*), in order to store the references between the traced objects (*trace link*). The column name must indicate the module to which it makes reference. In DOORS, each module is associated to a code, which was automatically generated and uniquely identifies the module. This code is returned by the DXL function *uniqueID* (see pg. 254, 255 at [11]). It is possible to use that code to reference a module inside another module. In practice, this was implemented by appending the unique identifier of the source module to the end of the target module's column name and *vice-versa*. The value stored in this column is a sequence of object identifiers (integer numbers separated by comma). Given that the unique identifiers from modules A, B are respectively "000006f7" and "000006f8", figures 4.3 and 4.4 show how the modules will look like.

ID		TRACE-LINK-COLUMN.000006f8
1	a1	
2	a2	2
3	a3	

Figure 4.3: Module A with extra column

ID		TRACE-LINK-COLUMN.000006f7
1	b1	
2	b2	2
3	b3	

Figure 4.4: Module B with extra column

The *Column Link* technique implements *tracing* of objects based on very simple DXL functions (see functions *getListOfTraces* and *trace* at pseudo-code 4.1). In this section, the alias *f* is equivalent to the function name *getListOfTraces*. The function *f* takes two modules as parameters and returns a *list of traces*. For the sake of comprehension, the first parameter will be called *source module* and the second parameter will be called *target module*. The function *f traces* objects from the target module to the source module. It is important to notice that each element in the return value of *f* is a *trace*, which means it is composed by a *source object identifier* plus a *list of all target object identifiers, which are linked to that source object* (this list is returned by the function *trace*, see at pseudo-code 4.1).

Listing 4.1: Pseudo-code for *Column Link* DXL functions

```
List getListOfTraces(Module source, Module target) {
  List listOfTraces = create
  Object t
  for t in target do {
    List sourceObjectsLinkedWith_t
    sourceObjectsLinkedWith_t = trace(t, source)

    /*** Use Object (t) as index for the List ***/
    put(listOfTraces, t, sourceObjectsLinkedWith_t)
  }
  return listOfTraces
}
```

```
List trace(Object targetObj, Module source) {
  List objectsTraced = create
  string columnLink = getColumnLinkName(source)
  List IDs = split(targetObj.columnLink)

  int id
  for id in IDs do {
    Object traced = object(id, source)
    if(null != traced) {
      put(objectsTraced, uniqueID(traced), traced)
    }
  }

  return objectsTraced
}
```

Fortunately, in the *Column Link* technique, both modules A and B contain the extra column, enabling to use exactly the same function *to trace* in both *primary* and *reverse* trace link directions — in the example, both ways $f(A, B)$ or $f(B, A)$.

4.3 Only Target Column (Naive)

This technique is almost the same as the previous one, except for one thing: the source module is not allowed to be modified or, more explicitly, there is no extra column in the source module (still in the target module, though)[1]. Consequently, the same function from the previous technique can be used in the *reverse trace link direction* (when tracing objects from target module to source module), because, in the target module, there is still one extra column containing the references to the source objects. But a new function is needed

[1] That is the base for this technique's name, since "*only* in the *target* module there is an extra *column*"

to trace objects in the *primary trace link direction* (from source module to target module), because, in the source module, there is no column containing references to the target objects anymore.
This technique's approach is analogue to the previous technique, where a function (f) iterates through the source module and calls a sub-function (f_{sub}) to do the actual task. This sub-function is applied to each object (s) in the source module (S) and searches for objects in the target module (list of target objects T_s), which are linked to the source object.

$$s \in S; T_s \subset T;$$
$$f_{sub} : S \rightarrow T; f_{sub}(s) = T_s$$
$$f : (S, T) \rightarrow list\{(s, T_s)\}$$

In the pseudo-code 4.2, f_{sub} is represented by the function *trace* and f by the function *getListOfTraces*.

Listing 4.2: Pseudo-code for *Only Target Column (Naive)* technique

```
/*** [Naive strategy / main function] ***/
List getListOfTraces(Module source, Module target) {
  List listOfTraces = create
  Object s
  for s in source do {
    List targetObjectsLinkedWith_s
    targetObjectsLinkedWith_s = trace(s, target)
    /*** List Index is of type int ***/
    int sID = uniqueID(s)
    put(listOfTraces, sID, targetObjectsLinkedWith_s)
  }
  return listOfTraces
}
```

```
/*** [Naive strategy / sub-function] ***/
List trace(Object s, Module target) {
  List targetIDs = create
  int sourceID = uniqueID(s)
  Module source = module(s)
  string columnName = getColumnLinkName(source)
  Object t
  for t in target do {
    List IDs = split(t.columnName)
    if(find(IDs, sourceID)) {
      int targetID = uniqueID(t)
      put(targetIDs, targetID, targetID)
    }
  }
  return targetIDs
}
```

This technique is considered "naive", because it has an unnecessary complexity, as mentioned in the end of chapter 3, while scanning the whole target module for each source object, and this is always done again each time f is called.

4.4 Only Target Column (Cache)

This technique evolved from the previous one, thus they are very similar in many aspects but one: tracing in the primary trace link direction.

Core Idea The basic idea behind this technique is to save time by "caching" results and reuse them again when requested. The function f has the same goal as the naive technique, which is to get all target objects linked to each object in the source module. In other words, the goal is to get a list of traces

from the source module to the target module. The difference is that, in this case, f is less complex, because it needs only the target module (the one which contains the extra column) and the extra column's name — which contains the references to the source module — as input parameters, and returns a *list of traces* (see pseudo-code 4.3, function *getCachedResults*).

The first time this technique is executed, it will build up and store the list of traces. This process takes more time in the first execution and almost no time for the next ones. Out of that, the complexity will be approximately *O(n)* over the size of the target module, which is less than in the naive technique.

Listing 4.3: Pseudo-code for the cache technique

```
/*** [Caching strategy] ***/
List getListOfTraces(Module source, Module target) {
  string columnName = getColumnLinkName(source)

  return getCachedResults(target, columnName)
}


/*** CACHE ***/
global List cache_listOfTraces = null


/*** [Build or simply Return cached results] ***/
List getCachedResults(Module target, string columnName) {

  if(cache_listOfTraces == null) {
    /*** CREATE the Cache ***/
    cache_listOfTraces = create
```

```
    int srcID
    Object targetObj

    for targetObj in target do {

      int targetID = uniqueID(targetObj)
      List IDs = split(targetObj.columnName)

      for srcID in IDs do {

        List targetIDs

        /*** Traces srcID through the cache ***/
        find(cache_listOfTraces, srcID, targetIDs)

        /*** targetIDs has the values found ***/
        if(targetIDs == null) {
          targetIDs = create
        }

        put(targetIDs, targetID, targetID)

        /*** UPDATE the Cache ***/
        put(cache_listOfTraces, srcID, targetIDs)
      }
    }
  }

  return cache_listOfTraces
}
```

5 Implementation Details

The goal of this chapter is to provide the necessary information to understand the next chapter (*Results and Comparison*). After developing the theory and the algorithms, it is needed to make an experiment to test the practical side. The main tool used during the experiment was IBM Rational DOORS (version 9.5) and the programming language in which the algorithms were implemented is DXL.

The idea was to prepare different *scenarios* to be able to compare all techniques. For each technique two scenarios were needed: one scenario considering the *primary trace link direction* and another one considering the *reverse trace link direction*. That means, tracing objects in both ways: from source to target and from target to source. In the end, there will be 8 scenarios to measure execution times.

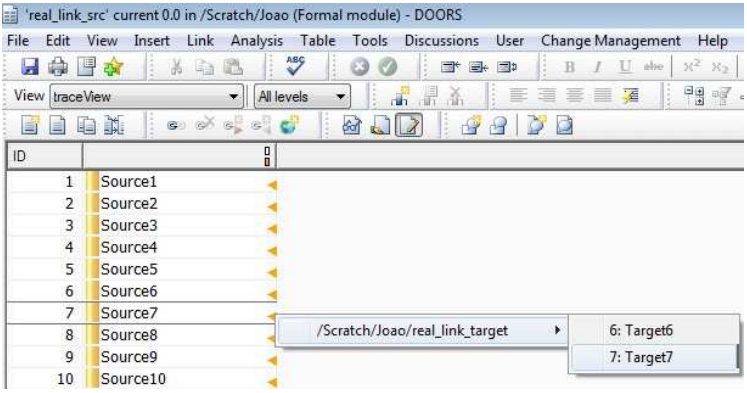

Figure 5.1: Real link source module

Concretely, in DOORS' database, pairs of modules were generated. A total of six modules (3 pairs of modules): *real link* source and target modules; *column link* source and target modules; and *only target column* source and

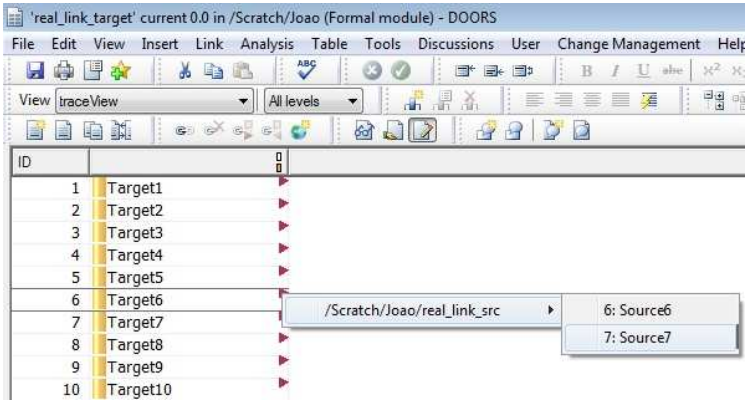

Figure 5.2: Real link target module

target modules. Both scenarios for techniques *Only Target Column (Naive)* and *Only Target Column (Cache)* use the same source and target modules. Additionally, a link module and a link set were created for the technique considering real links (see figures 5.1 and 5.2 to see how objects are linked in this technique).

Each module was populated with two thousand (2000) entries, or rows. Each row has an ID column, a "text" column and, only in some cases, an extra column to store the trace link information (for an example, see figure 5.3). Each object is stored in a row and the IDs were generated sequentially starting by 1 until 2000 (see figures 5.4 and 5.5).

In each scenario, trace links were created between source and target modules, exactly as follows: every source and target objects are part of at least one trace link; from all trace links, there are 10% of *multilinks* — when a source object is linked with more than one target object and *vice-versa* (in figure 5.6, target objects with ID 2, 9 and 20 have multilinks).

All this data was inserted or generated with the help of a DXL script, specially customized for each module to generate all the entries for all modules in the experiment.

ID		WvT.ReuseID.000006ef
4	Source4	4
5	Source5	5
6	Source6	6
7	Source7	7
8	Source8	8
9	Source9	9
10	Source10	10
11	Source11	11
12	Source12	11,12
13	Source13	13
14	Source14	14
15	Source15	14,15
16	Source16	15,16
17	Source17	17
18	Source18	18
19	Source19	19
20	Source20	20
21	Source21	21
22	Source22	22
23	Source23	23
24	Source24	23,24

Figure 5.3: Column link source module

It is important to observe that all the tests were executed from the same machine and same internet connection (approximately 30 Mbps download rate and 10 Mbps upload rate). Also, the server was located in the same geographical area of the client machine, so that there is no significant noise from the network connection in the results obtained, which will be presented in the next chapter.

ID		WvT.ReuseID.000006ee
1979	Target1979	1979,1980
1980	Target1980	1980
1981	Target1981	1981
1982	Target1982	1982
1983	Target1983	1983
1984	Target1984	1984
1985	Target1985	1985
1986	Target1986	1986
1987	Target1987	1987
1988	Target1988	1988
1989	Target1989	1989
1990	Target1990	1990
1991	Target1991	1991
1992	Target1992	1992
1993	Target1993	1993
1994	Target1994	1994
1995	Target1995	1995
1996	Target1996	1996
1997	Target1997	1997
1998	Target1998	1998
1999	Target1999	1999
2000	Target2000	2000

Figure 5.4: Column link target module

ID	
1972	Source1972
1973	Source1973
1974	Source1974
1975	Source1975
1976	Source1976
1977	Source1977
1978	Source1978
1979	Source1979
1980	Source1980
1981	Source1981
1982	Source1982
1983	Source1983
1984	Source1984
1985	Source1985
1986	Source1986
1987	Source1987
1988	Source1988
1989	Source1989
1990	Source1990
1991	Source1991
1992	Source1992
1993	Source1993
1994	Source1994
1995	Source1995
1996	Source1996
1997	Source1997
1998	Source1998
1999	Source1999
2000	Source2000

Figure 5.5: Only target column source module

ID		WvT.ReuseID.000006f0
1	Target1	1
2	Target2	2,3
3	Target3	3
4	Target4	4
5	Target5	5
6	Target6	6
7	Target7	7
8	Target8	8
9	Target9	9,10
10	Target10	10
11	Target11	11
12	Target12	12
13	Target13	13
14	Target14	14
15	Target15	15
16	Target16	16
17	Target17	17
18	Target18	18
19	Target19	19
20	Target20	20,21
21	Target21	21

Figure 5.6: Only target column target module

6 Results and Comparison

The goal of this chapter is to answer the third and last question from section 3.2 (pg. 17). In other words, the goal is to compare the performance of all presented techniques and to analyze which one reaches most efficiency. It is important to note that the precision degree of the measures taken is not the goal of this work, but the measured values are consistent and this is essential to be able to compare the techniques.

The approach used to measure performance is very simple. It is, basically, to simulate scenarios with big amounts of data, so that it is possible to check whether the results are acceptable or not. There will be an execution scenario for each technique. A scenario is composed fundamentally by a source and a target module, with a considerable number of entries and trace links between source and target objects. As explained in the previous chapter, there are 2000 entries for the source module and also 2000 entries for the target module (this number was chosen to approximate a real scenario, for example, the number of requirements from a product). An entry could be, for example, a requirement or any other traceable artifact. Each technique has its own scenario (total of 4 scenarios). Each technique will be examined in two different ways: in the primary trace link direction and in the reverse trace link direction (total of 8 cases). Eventually the primary trace link direction will be the point of interest. The execution times of each case will be plot into a table with 4 rows and 2 columns (table 6.1), each row contains the execution times of a specific technique in both trace link directions; each column contains the execution times of different techniques in a specific trace link direction.

The results presented are the mean time obtained over a couple of executions. Each case was executed about 2 to 4 times, just to verify their consistency. The result presented in table 6.1 shows the mean time of execution for each case. All results are shown in milliseconds and, if needed, the corresponding amount of seconds is shown in parenthesis. A special DXL function, called

"getTickCount_()", was used to measure the execution time in milliseconds.

Technique	**Primary Trace Link Direction**	**Reverse Trace Link Direction**
I: Real Links	515	574
II: Column Link	1955	1497
III: Only Target Column (Naive)	972042 (972 s)	1482
IV: Only Target Column (Cache)	663	1482

Table 6.1: Results in ms (milliseconds)

Analyzing the results in table 6.1, it is possible to see that the most efficient technique, in both primary and reverse trace link directions, is still the one using *Real Links*. Considering *Real Links* in the reverse trace link direction, its results are almost 3 times faster as the results from the other techniques. But, in the primary trace link direction, there are very different results, especially the *Only Target Column (Cache)* technique, which lays on the second place, only 148ms behind the fastest one.

Although they are not the most efficient ones, the results presented by the *Column Link* technique are considered acceptable, in both trace link directions (1497 and 1955 milliseconds — circa 1,5 and 2 seconds). When analyzing the reverse trace link direction, the same can be said about all techniques: they have acceptable results (less than 1,5 seconds).

One result which is completely unacceptable is the performance of the *Only Target Column (Naive)* technique in the primary trace link direction, which takes approximately 16min 12s to be executed (972s). But this performance problem was already expected and for this reason the *Only Target Column (Cache)* technique was developed. The *Only Target Column (Cache)* technique presented outstanding results in the primary trace link direction, about 1466 times better than the *Only Target Column (Naive)* technique. In the reverse trace link direction, both techniques have the same execution time (1482ms), because they use the same algorithm — but only in this direction.

There is one thing about performance which is not explicit in the results table: those measures consider only the first time the technique (or its algorithm) is executed. Of course for techniques I, II and III (see table 6.1) there is no performance improvement by the next time(s) the technique is executed. That means, if one technique has execution time t and will be executed n times, the total amount of execution time will be $n * t$. This is not true to the *Only Target Column (Cache)* technique, because it stores the results in the computer's working memory, thus the first execution takes more time than the next ones. That means, as long as the program or the computer are not closed nor turned off, in n times the *Only Target Column (Cache)* technique will have a total execution time of $t + (n - 1) * t_{cache}$, where t_{cache} is the execution time for retrieving results previously stored in the working memory (in the "cache") and t is the execution time for the very first execution. In practice, the results encountered for t_{cache} are much smaller than one millisecond ($t_{cache} <<< 1\ ms$). That makes t_{cache} absolutely insignificant for the scope of this work. Even the DXL function "getTickCount_()", after several executions, repeatedly returned 0 milliseconds. As a corollary:

$$t_{cache} = 0$$
$$\Downarrow$$
$$t + (n - 1) * t_{cache} = t$$

The last equation presented can be read as follows: for n executions, the *Only Target Column (Cache)* technique has a total execution time of approximately t, or, approximately the same execution time as for only one execution. Back to the results on table 6.1, the *Only Target Column (Cache)* technique makes a performance gain over the *Real Links* technique by the second execution ($n = 2$) in the primary trace link direction (since $2 * 515 > 663$) and by the third execution ($n = 3$) in the reverse trace link direction (because $3 * 574 =$

$1722 > 1482$). Finally, it is needed to say that this performance gain is only possible with the assumption that objects, modules and trace relations do not change at all or, at least, not very often. If there is a high frequency of changes in that data, each change will impact on the performance of the *Only Target Column (Cache)* technique and probably the gain over the *Real Links* technique will be very small and will occur seldom, probably only after a very large number of executions (n proportional to the number of times a change was effected).

7 Future Perspective

After the conclusion of this work, a couple alternative ways to provide traceability within DOORS were analyzed, despite the fact that DOORS already has a standard way to provide traceability. In practice though, in specific situations, some small details from the standard DOORS mechanism were not desirable. For this reason, new ways of providing traceability were needed.

The first approach tried to outline those inconvenient situations, replacing the link module and the link set by an additional column in the source module and another additional column in the target module. Unfortunately, but not surprising, the first approach was imperfect — it also has presented undesirable features. There was a restriction, not allowing a source module to be modified.

Therefore, a new technique was established, based on the idea of eliminating the inconveniences brought by the previous technique. The new approach accepted the limitation imposed and removed the additional column from the source modules, only a target module was able to be modified and, thus, to have an extra column to store the trace link information. The source module ID was part of the column's name and the object IDs were the content, so that it was possible to trace which target object was linked to which object in which source module. The first algorithm tried out with this restriction presented performance problems, when applied to a big amount of input data (e.g. all entries from a module).

A new concept of algorithm was developed, in order to improve the performance. The last technique was established. The technique made an "exchange" between memory and execution time. It used more memory to store precomputed results, instead of every time searching for them through the whole module. That reduced significantly the cost of retrieving such results. This technique was named after the concept of cache memory, because there are several similarities between them.

As can be seen, each problem encountered in a traceability technique was analyzed and solved. And so was it done, again and once again, from technique to technique, until the last one. In the end there were four different techniques, each one generated upon its predecessor. With that, it became possible to make a comparison of all techniques and analyze the advantages and disadvantages of each one.

This work has shown that it is possible to use creativity to develop alternative ways to achieve the same goal inside an existent tool, when the tool does not provide some feature exactly as wanted, but at least provides enough resources for customization (e.g. DXL). This work made clear that it is possible to have *Traceability* in DOORS using a non-standard technique, and proved that its performance is acceptable. The benefit brought by this work will be seen by DOORS users: they will have an easier way of performing some trivial tasks of processing objects inside modules and their trace relations (e.g. to delete an object linked to other objects, will require less "clicks"), consequently they will spend less time in such tasks and increase their productivity.

For the future, there are still some open questions. Specially in what concerns the last presented technique, the *Only Target Column (Cache)* technique. Although its name remarks to cache memory systems, this technique cannot be considered a caching system. This technique was not made to deal with changes in the "cached" results (modifications, deletions, etc). It is possible to start a completely new work about that, to create mechanisms to guarantee the consistency between the real data (in the modules) and the "cached" results. Actually, these results are stored only in working memory and this is another aspect which can be further developed. It is possible, for example, to start to store the "cached" results using a persistent memory system (e.g. save results to a file). With that it would be faster to load the "cached" results from a file, than to compute them again each time DOORS is started. Thus, this work has left the *doors* opened for a future work.

8 Bibliography

[1] OICA. (2013) Production Statistics. International Organization of Motor Vehicle Manufacturers. Accessed: Jan. 2015. [Online]. Available: http://www.oica.net/category/production-statistics/2013-statistics/ Cited on page 1.

[2] I. Sommerville and G. Kotonya, *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, 1998. Cited on pages 5 and 6.

[3] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 2nd ed. Addison-Wesley, 2006. Cited on pages 5 and 6.

[4] IEEE, "IEEE Standard for Application and Management of the Systems Engineering Process," *IEEE Std 1220-2005 (Revision of IEEE Std 1220-1998)*, pp. 1–87, 2005. Cited on page 5.

[5] ——, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990. Cited on page 5.

[6] ——, "IEEE Standard for Application and Management of the Systems Engineering Process," *IEEE Std 1220-1998*, 1998. Cited on page 6.

[7] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, 3rd ed. Springer, 2011. Cited on pages 6 and 7.

[8] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and Systems Traceability*. Springer, 2012, vol. 2, no. 3. Cited on pages 7, 8, 9, 10, 11, and 22.

[9] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of the First International Conference on Requirements Engineering*. IEEE, 1994, pp. 94–101. Cited on page 9.

[10] IBM Knowledge Center. (2015) Rational DOORS 9.5.0. International Business Machines Corporation. Accessed: Feb. 2015. [Online]. Available: http://www-01.ibm.com/support/knowledgecenter/SSYQBZ_9.5.0/com.ibm.doors.homepage.doc/helpindex_doors.html Cited on page

11.

[11] IBM. (2015) DXL Reference Manual. International Business Machines Corporation. Accessed: Feb. 2015. [Online]. Available: http://www-01.ibm.com/support/knowledgecenter/SSYQBZ_9.5.0/com.ibm.doors.requirements.doc/topics/dxl_reference_manual.pdf Cited on pages 12, 13, 22, and 23.

[12] T. Goodman. (2015) Programming in DXL. SmartDXL. Accessed: Mar. 2015. [Online]. Available: http://www.smartdxl.com/content/?page_id=163 Cited on page 12.

[13] T. Noack, "Automatische Verlinkung von Testfaellen und Anforderungen, Eine wiederverwendungsorientierte Methode," Ph.D. dissertation, SWT — Fakultaet IV der Technischen Universitaet Berlin, Berlin, Germany, 2014. Cited on pages 15 and 16.

[14] C. H. Papadimitriou, *Computational Complexity*. John Wiley & Sons, 2003. Cited on page 19.