ROSKILDE UNIVERSITET

SEMESTER PROJECT

Antipatterns: A literature review and study to find reasons behind antipatterns usage

Authors:
Devendra PRASHIN
Filip Andrzej GERMANEK
Gurdeep KAUR
Jens Bo PEDERSEN
Soniya Subba BEGHA

Supervisor:
Anders LASSEN

A report submitted in fulfillment of the requirements for finishing third semester

in the

MSc in Computer Science and Informatics
IMT

December 17, 2019

Contents

1 Introduction **3**
1.1 Problem Formulation . . . 4

2 Background and Related Work **5**
2.1 Informatics . . . 5
2.2 Software Antipatterns . . . 6
2.2.1 Human Behavior and Antipatterns . . . 7
2.3 Software Development Antipatterns . . . 8
2.3.1 Software Performance Antipatterns . . . 8
2.4 Software Architecture Antipatterns . . . 9
2.4.1 Project Management Antipatterns . . . 10

3 Tree Structure **12**
3.1 Problem . . . 12
3.2 Objective . . . 12
3.3 Antipattern . . . 13
3.3.1 Problems With The Antipattern . . . 13
3.4 Informatics . . . 13
3.4.1 Systems Development and Design . . . 13
3.4.2 Functional Viewpoint . . . 14
3.4.3 Information Viewpoint . . . 15
Comment ID As A Foreign Key (Antipattern Solution) . . . 15
Path Enumeration . . . 16
Nested Sets . . . 17
Closure Table . . . 17

4 SQL Injection **19**
4.1 Problem . . . 19
4.1.1 How Does It Work? . . . 19
4.2 Antipattern . . . 19
4.2.1 Problems With The Antipattern . . . 19
4.3 Informatics . . . 20
4.3.1 Business Impact . . . 20
4.3.2 Security Perspective . . . 21
4.4 Solutions To The Antipattern . . . 22

5 Summary **24**

6 Glossary **26**

List of Figures

2.1 Design of patterns and antipatterns[10] 6
2.2 Category of Codesmells [13] 8
2.3 UML of antipatterns[19] 10
2.4 Relationship B/W three different antipatterns [19] 11

3.1 Comment ID as a foreign key solution design 15
3.2 Path Enumeration solution design 16
3.3 Nested Sets solution design 17
3.4 Closure Table solution design 17

4.1 Security perspective [14] 22
4.2 Parameterize dynamic values example 23

Chapter 1

Introduction

This report is not particularly about the merits or demerits of using antipatterns in database or development design – much has been written about this already. This report takes a critical look at reasons that force us to use patterns and antipatterns in particular situations. Antipattern knowledge is a good way of learning about commonly occurring problems and bad practices. This helps us when trying to solve them, prevent from these bad practices or refactor them by studying others negative experiences. Therefore, proper documentation of antipatterns becomes crucial in order to share experiences in this regard. However, available antipattern information in the literature is mostly informal and unstructured[19]. To get a better understanding of antipatterns for researchers and project managers, formal and more organized documentation of antipatterns is a necessity[19].

In 1994, the "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) released the book "Design Patterns: Elements of Reusable Object-Oriented Software". When doing so, they popularised the concept of "Patterns" for use in Computer Science, a term originally coined by Christopher Alexander in "A Pattern Language" (1977) for Architecture[8]. However, the closest Alexander got to a concrete definition of the term "Pattern" was:

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

When human does the translation of business concepts into software applications then there is a chance of running into problems related to usage of antipatterns. These problems are created during human run processes that require shared vision and cooperation to make sure everyone involved understands the system. So, antipatterns can be the result of a manager, developer not knowing any better, not having enough knowledge or experience in solving a problem or having applied a perfectly good pattern in the wrong context[10].

Antipatterns provide a real-world experience in recognizing recurring problems in the software industry and provide a detailed remedy for the most common circumstances. They also describe the measure that can be taken at several levels to improve the development of an application, the design of the software systems and the effective management of software projects.

Several approaches of detecting antipatterns were proposed recently. Manual and automated approaches use different techniques and were developed in order to detect antipatterns at different levels of software life cycle. This study aims to present antipatterns related to Relational Databases[10].

While our original hope was to examine antipatterns in real life as a case study, this project has evolved to instead examine a selection of antipatterns through the

lens of viewpoints and perspectives. The report is divided into few Chapters. Chapter 1 contains the introduction and problem formulation. Chapter 2 explains the background study and related work. Furthermore, Chapter 3 and Chapter 4 contain our first Case Study of 'Tree Structure' and 'SQL injection' respectively . Last chapter will include the Summary and results part of this report.

1.1 Problem Formulation

"Antipatterns provide information on commonly occurring solutions to problems that generate negative consequences"[1]. In an organisation antipatterns could be one of reason behind total or partial failure of a software project.Accordingly to Jim Coplien: "An antipattern is something that looks like a good idea, but which backfires badly when applied"[10]. Even though every project has to face constraints which can lead to its failure such as budget, schedule, external laws etc these constrains are always taken into account and project managers are aware of them. Antipatterns are more difficult to detect but the damage caused by their application can have a negative influence on schedule and budget as well. Therefore,the aim of this project is to study and better understand the antipatterns and reasons that lead to the usage of SQL antipatterns in databases.

We plan to base this study on literature review followed by case studies. We had examined two concrete antipattern cases (Tree Structure and SQL Injection) with regards to how they were implemented, why they were implemented and which negative consequences they had. We tried to establish common presumptive evidence using Tree Structure and SQL Injection examples which helps detecting when an antipattern is being used in your project.

Chapter 2

Background and Related Work

In this chapter, we will focus on the background and related work by starting with the field of Informatics. It will be followed by a discussion resolving around literature related to our own previous work with antipatterns. We describe different areas of antipatterns in the software industry ending with a summary of how our project relates to this past work.

2.1 Informatics

“Informatics uses technology in the collection, classification, storage, retrieval, and dissemination of information to solve problems, improve business processes and facilitate the wants and needs of humans”[7]. It is basically the science of information. The field takes into consideration the interaction between the information systems and the user.

Informatics is a branch of information engineering which can be seen as the process of involvement of information processing and the engineering of information systems[21]. However, when we see it according to the academic perspective it only applies to information science. The field considers interaction between humans and information alongside the construction of interfaces, organisations, technologies and systems. Therefore, the study of Informatics has wide-range of areas and is aimed at more focused field of study within Computer Science. With respect to our project, it falls into few of the major core areas we studied during previous semester which are defined below:

- **Systems Development and Design**
  Account for, identify and analyze challenges related to IT architecture.IT architecture development including concepts of viewpoints and perspectives, and Architecture Skeleton construction. Iterative development phase[14].While working with antipatterns explore how system development and design affect the implementation of antipatterns in particular situation. We have also considered few of the important viewpoints and perspectives like information viewpoint which describes the way system will store, manipulate or manage information.Furthermore, functional viewpoint explain the systems functional elements, responsibilities and primary interactions.

- **IT Security**
  "Security as the set of processes and technologies that allow the owners of resources in the system to reliably control who can access which resources" [14]. The reason that we need security in our systems is that they contain valuable

information and sensitive operations, and we want to be sure these are accessed or executed only by certain people. Items in the system that we are trying to protect are known in security jargon as resources.

- **Human Computer Interaction**
  Human Computer Interaction(HCI) has its roots in the main areas of industrial engineering, human factors and cognitive psychology with the focus on the development of user-friendly IT. Another issue that should be considered in this interaction is the impact that information systems may have on humans and organizations[16]. There is general consensus that the adoption of any IT/IS brings change. Furthermore, IT/IS and organizations have a mutual influence on each other, meaning that technology affects organizations and that organizations necessarily affect for instance the design, the choice and the management of those systems.

2.2 Software Antipatterns

"A design pattern documents a commonly recurring and proven structure of interconnected design elements that solves a general design problem within a particular context"[14]. However, antipatterns are conceptually similar to patterns in that they document recurring solutions to common design problems.

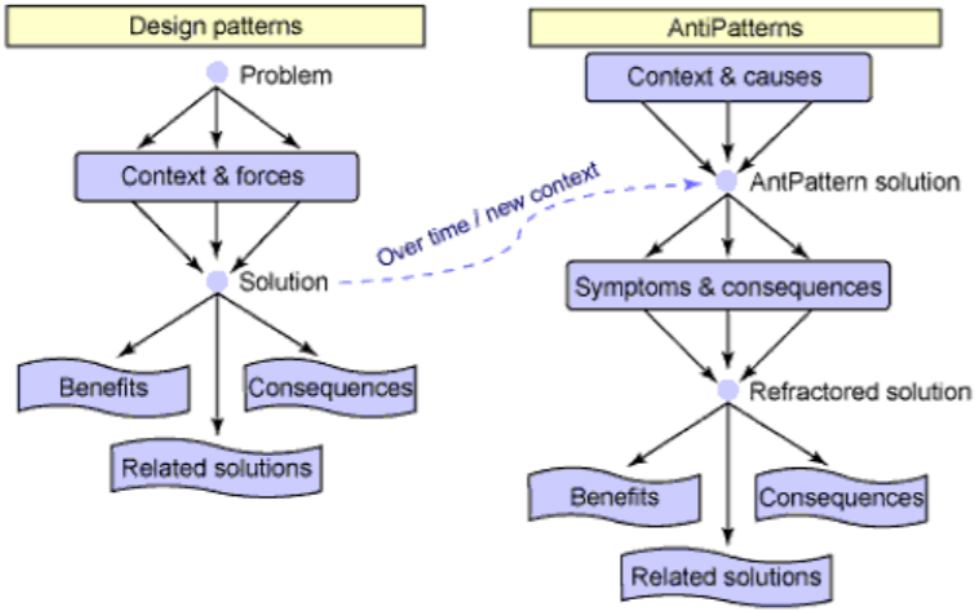

FIGURE 2.1: Design of patterns and antipatterns[10]

The role of patterns is to ensure positive solutions to a commonly occurring problem, whereas antipatterns have two solutions as producing problematic solutions which cause negative consequences and difficulties and refactored solutions which make antipatterns beneficial[10]. Antipatterns are closely related to design patterns since design patterns can evolve into antipatterns as it can be observed from above figure.

It can be seen that design patterns have two essences as a problem and a solution. A problem can be varied according to its context and design forces. Besides, a pattern documents a repeatable solution to the associated problem. A solution generates benefits, consequences and related solutions[17]. On the other hand, an antipattern presents a solution commonly applied to a particular problem which generates negative consequences. An antipattern provides information about the reason of the wrong practice applied to a specific problem and shows how to prevent and correct the solution. Because of the pace of technological advancements, technology is changing rapidly. Therefore, patterns are transforming into antipatterns over time. That's why the number of existing antipatterns are more than patterns today [10], [17]. To clarify the antipatterns concept, it can be divided into two types: A simple antipattern tells the reader how to go from a problem to a poor solution. Simple antipatterns therefore, focus on presenting negative solutions. An amelioration antipattern tells the reader how to go from a problem to a bad solution, but also how to get from that bad solution to a good solution[4]. It defines a migration (or refactoring) from negative solutions to positive solutions. It tells you why the bad solution looks attractive, why it turns out to be bad in conjunction with the desired new outcome or behaviour and what positive patterns are applicable instead [4].

Antipatterns are poor solutions of recurring design problems, which decrease software quality. Numerous antipatterns have been outlined in the literature as violations of various quality rules. Most of these antipatterns have been defined in terms of code quality metrics. However, identifying antipatterns at the design level would improve considerably[6]. Furthermore, Dodani added that developing patterns is a bottom-up process, whereas developing antipatterns is a top-down process. He insisted that we should learn from our 'mistakes' which are antipatterns[6]. It is not difficult to produce antipatterns based on one's own experience. An antipattern can be easily derived by generalizing project cases where bad decisions have been made[1].

Each proposed antipattern template may be represented with a different viewpoint. However, different templates that have been proposed for the documentation of antipatterns may cause confusion to software project managers.

2.2.1 Human Behavior and Antipatterns

It is important to understand the human behavior in any organization before we explore the antipatterns. One of the ways humans solve newly encountered problems is by subconsciously applying a previously successful solution to a similar or related problem[12]. But why is there a need to study human behavior patterns? And why do we bother?

First, it can be used to understand what motivates people. Understanding their motivation makes it easier to look for solutions that are mutually beneficial or avoid them causing problems. Furthermore, understanding individual motives help the manager to organize the teams. That is the reason, many organizations use personality tests in their hiring practices to ensure corporate cultural alignment. Everyone has to deal with some difficult people[12].

According to Brown[10], people who cause trouble are called "corncobs". The term "corncob", means a general pain in the buttocks. Difficult people obstruct and divert the software development process[10]. Therefore, individual agendas should be addressed through various tactical, operational and strategic organizational actions.

2.3 Software Development Antipatterns

Software Development AntiPatterns are also called as Codesmell.Code smells were introduced by Fowler and Back [13]. Codesmell describe a code structure that is likely to cause problems and that can be removed through refactoring.They commonly increase the software's defectiveness and change proneness and increase maintenance effort. A key goal of development antipatterns is to describe useful forms of software refactoring. Software refactoring is a form of code modification, used to improve the software structure in support of subsequent extension and long-term maintenance. In most cases, the goal is to transform code without impacting correctness.Furthermore, codesmells can be classified into below category purposed by Mäntylä and Lassenius[15].

Category name	Code smells
The Bloaters	Blob
	Large Class
	Long Method
	Long Parameter List
	Complex Class
	Swiss Army Knife
The Change Preventers	*Spaghetti Code*
The Dispensables	Lazy Class
	Speculative Generality
	Many Field Attributes But Not Complex
	Duplicated Code
The Encapsulators	*Message Chain*
The Object-Orientation Abusers	Anti-Singleton
	Refused Parent Bequest
	Base Class Knows Derived Class
	Base Class Should Be Abstract
	Class Data Should Be Private
	Tradition Breaker
The Object-Orientation Avoiders	*Functional Decomposition*

FIGURE 2.2: Category of Codesmells [13]

This figure describes the different categories and different types of codesmells in software development antipatterns. The Bloaters are objects that have grown too much and can become hard to manage. This category includes the codesmells Blob, Long Method, Large Class, and Long Parameter List. Similary, The Dispensables are unnecessary code fragments that should be removed. This includes the codesmells Lazy Class, Duplicated Code, and Speculative Generality.

2.3.1 Software Performance Antipatterns

The concept of design patterns has been introduced several decades ago for defining good practices to design software[9]. Along with patterns there is a creation of

many antipatterns. These design patterns and antipatterns are being very powerful instruments of software developer for enhancing the quality of the product. Among many antipatterns Smith and Williams have introduced specific antipatterns as performance antipatterns which are bad design practices that may lead performance to degrade.

The main point of this paper is to reduce the gap between design patterns and performance antipatterns to fulfill the performance requirements which leads to the better quality of software. The authors are concerned about removing the performance antipattern (Empty Semi-Trucks) and introducing design patterns (Session Facade, batching etc.) to improve the quality of the software design. They are working in a fuzzy context where threshold values related to performance antipattern metrics (e.g. the number of connections that a component has with other components in the system is too high) can't be determined, but only their lower and upper bounds do. In this context, the ranking criteria for design pattern is best to use for the removal of antipatterns.

They describe the refactoring process and the ranking criteria to drive the choice of design pattern towards the removal of performance antipatterns. E.g. a design model of the current version of the application code which lacks some performance requirement i.e. a response time of a service must be less than 2 seconds. To verify if this condition is met, there should be performance analysis using specific performance model as a Querying Network (in case of design mode) or by monitoring the running application (in case of the current version of the app code).

2.4 Software Architecture Antipatterns

"The architecture of a system or software is the set of fundamental concepts or properties of the system in its environment, embodied in its elements,relationships, and the principles of its design and evolution"[14]. Architecture antipatterns focus on the system-level and enterprise-level structure of applications and components. Although the engineering discipline of software architecture is relatively immature, what has been determined repeatedly by software research and experience is the overarching importance of architecture in software development. As stated by Rozanski & Woods "Unified Modeling Language (UML) is the most widely understood modeling language around the world. UML model, to help you investigate scenarios without building full prototypes[14]. Below mentioned is the UML representation of one proposed generic antipattern viewpoint [19] though,unclear modeling viewpoint can causes problematic ambiguities in object models.However,this viewpoint considers software architecture and software development antipatterns, aiming at a broader audience and antipattern application scope. The viewpoint models certain relationships between antipatterns: An antipattern may replace, use, refine or require another antipattern.As the old requirements are refined, new requirements are gathered and processed. According to the project managers who answered the questionnaire, the main source of changes was team members' ideas and clients' demands and requirement change because of technical reasons was observed in one instance.

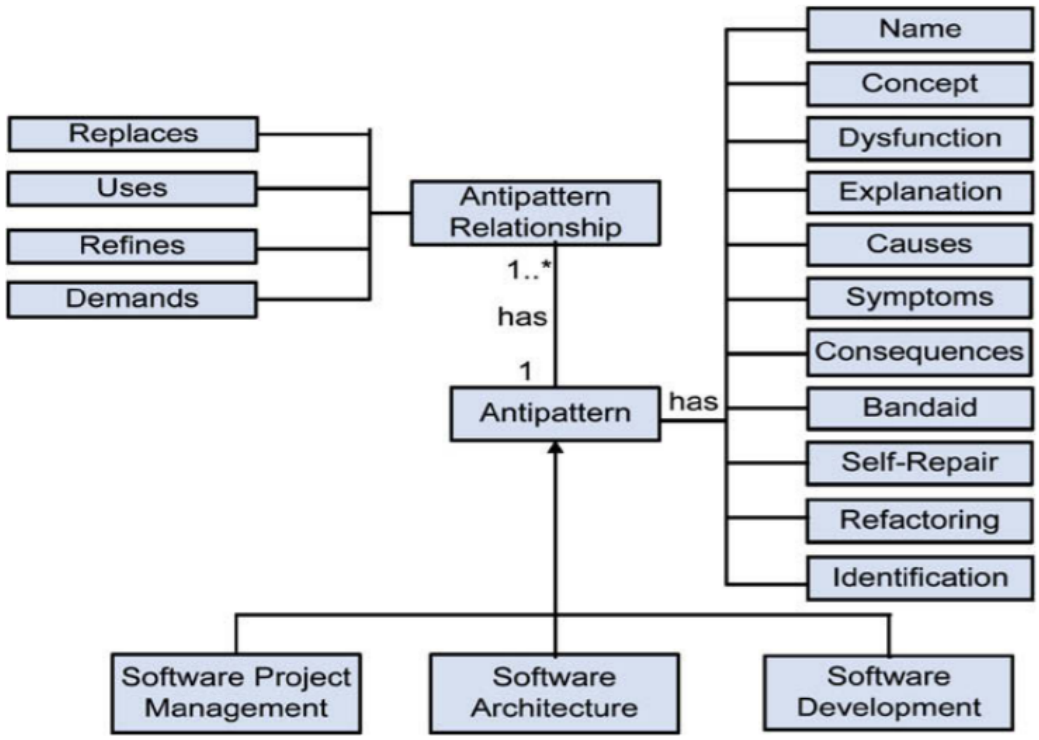

FIGURE 2.3: UML of antipatterns[19]

This model can be assumed as an important step towards the architecture of an intelligent system that aims to assist project managers in detecting and resolving appropriate antipatterns during the execution of a software project. Software architecture is a subset of the overall system architecture, which includes all design and implementation aspects, including hardware and technology selection. Important principles of architecture include the following[13]:

- Architecture provides a view of the whole system. This distinguishes architecture from other analysis and design models that focus on parts of a system.
- An effective way to model whole systems is through multiple viewpoints. The viewpoints correlate to various stakeholders and technical experts in the system-development process.

Architecture antipatterns are mainly related to dependency and interface issues: Ambiguous Interface, Redundant Interface, Overused Interface, Cyclic Dependency[13].

2.4.1 Project Management Antipatterns

There are several reasons which can lead software projects to partial or complete failure. These reasons can be basically gathered under two viewpoints as external(customer expectations, schedule related issues and so on) and internal. Internal constraints affect basic management practices (planning, team building, decision making and so on) in a negative way. Software project management antipatterns are one of the underlying reasons for failure of a project [19]. In project management, antipattern is defined as commonly repeated bad practice [1]. There are various

templates used to document antipatterns. When documenting antipattern, it is important to identify not only what caused given antipattern but also how can it be fixed.

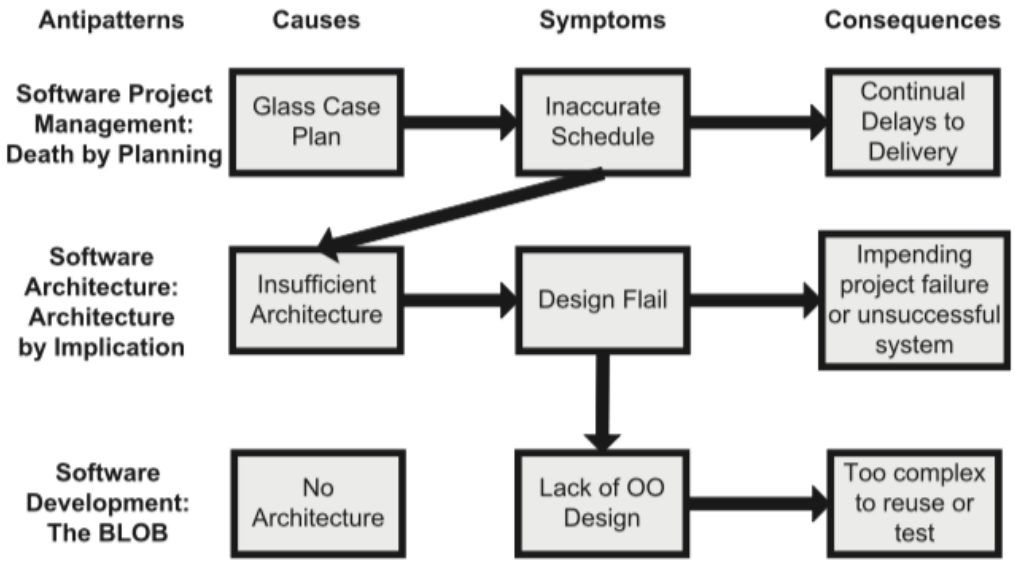

FIGURE 2.4: Relationship B/W three different antipatterns [19]

Above figure illustrates an example of relationship between three different antipatterns, through their causes, symptoms and consequences. In this example, the "Death by Planning" project management antipattern interacts with the other two antipatterns through the symptom "Inaccurate schedule". SPARSE operates with a symptom based approach. If a project manager has selected the "Inaccurate schedule" symptom, SPARSE can report the directly matching "Death by Planning" antipattern but also detects the other two related antipatterns of the example and other semantically related antipatterns[1] even though different authors have used different template to document antipatterns. Although many of them are documented through some kind of informal template, others are described in plain text.

- Templates are extremely useful for remembering antipattern details, but lack vigor and do not provide quantitative information that may be used by managers for improving tangible project indicators.
- Causality is a strong component of any antipattern description.For an antipattern to be convincing, it is crucial to produce causes and effects, with justified bonds between pairs of causes-effects.
- Antipatterns are not fully deterministic. Ambiguity comes into play when a cause is linked to an effect; a cause may produce an effect with some probability and to some extent.

Antipatterns can either appear in isolation or can be related with other antipatterns (interacting antipatterns). When studying antipatterns from a project management point of view we can observe that in IT projects, consequences are one of antipatterns which are often a cause of other antipatterns in different stage of the project.

Chapter 3

Tree Structure

In this chapter, we will discuss the problem related to selected antipattern. Antipatterns problem description includes definition, objective and possible negative consequences of it. This section of the chapter will be followed by an investigation of techniques which can be used to solve the problem as well as understand what leads to the presence of the antipattern in projects.

3.1 Problem

To define the problem we should first look into the definition of tree structure. Trees are graphs which have the following properties [2]:

- A tree is a connected graph (connected graph is a graph where there is a path between any two nodes and no node is disconnected from the rest of the graph) which has no cycles.
- Every node is the root of a sub tree.
- Every two nodes in a tree are connected by only one path. It has one less edge than the number of nodes.

The need to store hierarchies is a common requirement. However, there are multiple ways to store these relations in a relational database. Choosing a right solution that would suit our given case, best can be counter intuitive and should be determined by looking into most common use cases. In practice, developers often choose solution which may seem most logical in terms of structuring the data. This can lead to an antipattern because we solve a problem of storing tree hierarchy but we may complicate records creation, updating, deletion or data retrieval by choosing the wrong solution.

To understand the problem better, we will use an example in which we want to store relations between posts and their comments (which can be nested). Suppose, we have a website where users can add posts and comment on both posts and other comments. We need to store this relationship in our database. The design problem, this requirement introduces is that comments can possibly have an infinite number of nested comments under them. We need to be able to add, update, and delete comments.

3.2 Objective

- Be able to store and query hierarchies
- Be able to query the tree structure in efficient way

3.3 Antipattern

An antipattern in this case is that we may design the database in a way where we always depend on nodes parent. That is because a solution which is commonly used is to add a column parent_id. This column references another record in the same table. This design however, is not perfect and complicates querying data in some scenarios that will be discussed in the functional and information viewpoint subsections.

3.3.1 Problems With The Antipattern

There are two main problems We would like to focus on while investigating this antipattern:

- Maintaining the tree - We need to be able to store, update and delete records with the hierarchical relationship. This means that when we delete a record, all of its nested records should either also be deleted or should be moved to another tree level. Querying all nested records is not simple when using antipattern solution. Therefore, maintaining the tree also becomes complicated.
- Querying the tree - If there is an use case which requires querying whole tree structure or all records nested under given record disadvantages of the antipattern solution become obvious. We cannot query all nested records with just one simple SQL query because nested records only know about its closest parent. Therefore, in order to fulfill this use case we would have to build recursive query. Querying whole tree structure is a common use case.

As we can see in the solutions part of this section(3.4) design which completes these objectives can contradict themselves. We can use techniques from [14] to help us in determining which of the mentioned objectives will be more common for a given IT system.

3.4 Informatics

3.4.1 Systems Development and Design

One reason leading to the usage of this antipattern is developers not understanding main use cases the tree structure needs to fulfill. There are various techniques that can be used to improve that. We believe that this could be done using architecture definition process[14].

- The architecture definition process - it is an iterative process which consists of following steps:
  - Consolidate input
  - Identify Scenarios
  - Identify relevant architectural styles
  - Produce candidate Architecture
  - Explore architectural options
  - Evaluate Architecture with stakeholders

Applying this approach can help us in avoiding antipattern design decision because it gives us time to identify scenarios in which database will be used. After that we can explore candidate design approaches and test/evaluate each of the with stakeholders.

- Architectural Decisions - need to answer 'what', 'how' and 'with what'. Must be traceable from concerns provided. Should be specific to avoid confusion. If this approach is followed, we decrease a chance of spontaneous decision which can influence our design negatively in the future. It does not mean that wrong decisions will not be made however investigating the problem and mapping concerns into decisions gives us tools to argument and analyze each solution in more detail.
- Identifying and engaging stakeholders - this can be helpful when trying to understand how the system we are designing will be used. Stakeholders should influence the database design decisions. In our example (storing relations between comments), interviewing stakeholders could be important in understanding most commons use cases when it comes to fetching comments from database.

3.4.2 Functional Viewpoint

By applying this viewpoint we can establish use cases which system needs to fulfill. This can influence design of the database and help us to decide which approach to take when solving tree structure relationships problem. If most common use case is to display whole tree, we should introduce a design which supports fetching of the whole tree. If most common design is to add, update or remove the node from tree most of the time won't be displaying whole structure we can follow another design approach. Design should reflect functional scenarios described in this viewpoint.

Functional Scenarios

- Display all comments for given comment

  This scenario requires a SQL query which selects all comments nested under given comment. If we decide to model database in a way described in 'Antipattern' section of this chapter then a simple task of selecting all nested comments becomes complicated because each record has only reference to its direct parent. With antipattern solution, we see two ways to query all needed comments.

```
SELECT c1.*, c2.*, c3.*, c4.*
FROM Comments c1                     -- 1st level
  LEFT OUTER JOIN Comments c2
    ON c2.parent_id = c1.comment_id  -- 2nd level
  LEFT OUTER JOIN Comments c3
    ON c3.parent_id = c2.comment_id  -- 3rd level
  LEFT OUTER JOIN Comments c4
    ON c4.parent_id = c3.comment_id; -- 4th level
    ....
```

  This way of querying records is problematic because in order to add more nested records we need to include join for each nested level. Aggregate queries such as COUNT( ) are also difficult to compute.

```
SELECT * FROM Comments WHERE post_id = 1;
```

This is an easy way of querying comments but we would have to organise records in the tree hierarchy before they would be ready to be used.

- Add comment

  This scenario is the one that benefits the most from the antipattern solution. It only requires one INSERT query and we do not need to update other records in the database.

```
INSERT INTO Comments (post_id, comment_id, ..., ...)
  VALUES (1, 2, '...', '...');
```

- Delete comment

  Depending on the business logic, when we delete a comment all of its nested comments have to either also be deleted or moved to a higher level. However, each case requires checking if a comment to be deleted has nested comments. To do that, we need a similar query to the ones described in scenario 'Display all comments of given comment'.

3.4.3 Information Viewpoint

This viewpoint mostly resolves around database as it describes ways in which system stores and processes information.

Storing information about post is easy. Comment can reference a post by id. Storing relationships between comments becomes more complicated. We have to deal with recursive relationships where data is organised in a treelike way. This section investigates possible solutions to this problem and analyzes each of them. This should help us in understanding problems that arise when using an antipattern solution but also investigate whether there are alternative solutions which could be used.

Comment ID As A Foreign Key (Antipattern Solution)

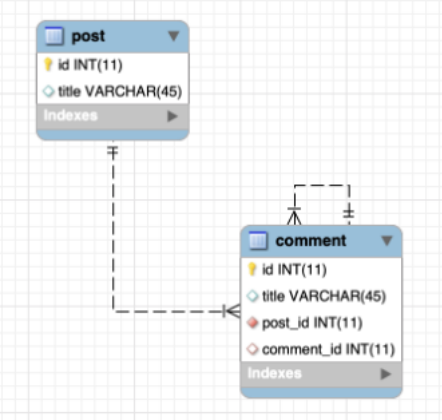

FIGURE 3.1: Comment ID as a foreign key solution design

- Explanation

  We are storing reference to parent comment record in a comment_id column. This is a foreign key relationship. If comment has a comment_id value it means this record is a comment nested under the comment it is referencing.

- Advantages

  Adding new comment, changing parent of existing comment is easy to implement. Probably, the most common solution when storing hierarchical relations because the relations are normalized and logical.

- Disadvantages

  Deleting a comment , querying all descendants will be problematic. If we want to implement count query it will also be difficult.

Path Enumeration

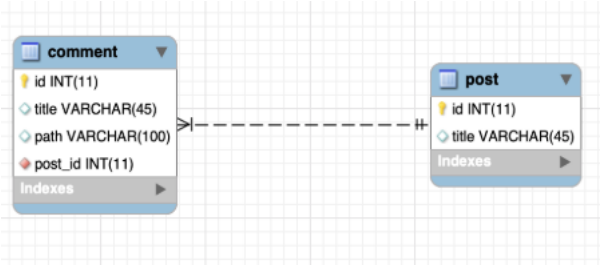

FIGURE 3.2: Path Enumeration solution design

- Explanation

  Storing a string of ancestors path as a varchar column in comments table (example: '1/2/3')

- Advantages

  Easy access to the tree path which makes querying post and all its comments easy. If fetching whole tree is a most common use case then this solution should be strongly considered.

- Disadvantages

  Creating, deleting and updating a single comment becomes complicated because every time we change the tree structure, we need to update other comments as well to make sure path is up to date for all records.

Nested Sets

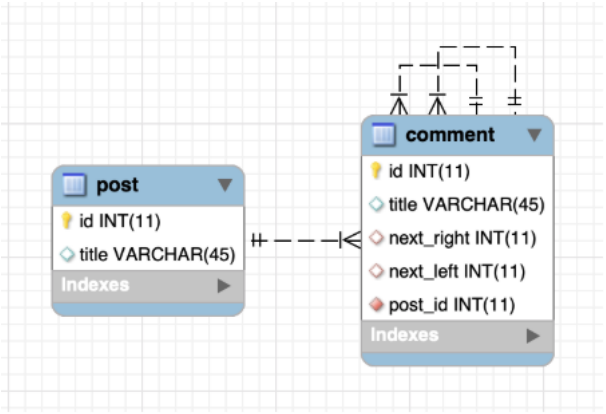

FIGURE 3.3: Nested Sets solution design

- Explanation

  This solution stores ids of comments that follows a given comment.

- Advantages

  Easy access to comments 'children'. Still fetching all comments of given post can be problematic as we need to look at the next record until given record has no children. Recursive query would be needed.

- Disadvantages

  If there can be an unlimited number of comments following given comment, storing this information can be problematic. Keeping children reference up to date requires checks and possible updates on the multiple records every time we want to update/delete single record. Should be used, if you need to mostly fetch the comments tree and you would modify it rarely.

Closure Table

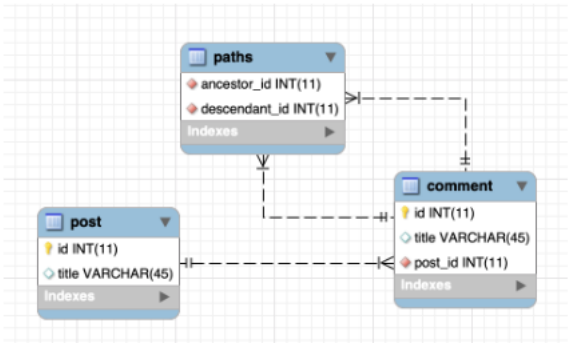

FIGURE 3.4: Closure Table solution design

- Explanation:

This is a way of storing hierarchies done by introducing new ‘paths’ table. In the ‘paths’ table, we will store not just direct relations but all relations even if they are separated by multiple levels of comments in between.

- Advantages:

  This allows us to easily fetch all comments for a given post. Fetching all comments for a given comment should not be complicated.

- Disadvantages:

  Increased space consumption as we store all relations not just direct ones which results in additional records being created.

Chapter 4

SQL Injection

In this chapter, we will discuss the antipattern- SQL injection. We will begin by introducing the problem and how does it work in practice. Then, we'll try to establish SQL injection as one of the antipatterns. Finally, the chapter will conclude with an investigation of techniques that can be used to understand what leads to the presence of the antipattern and how this antipattern can be addressed.

4.1 Problem

SQL injection is one of the most commonly used web hacking techniques [20]. It is about manipulating a SQL query, and the results of executing it on the server would not be desired. The vulnerability to SQL injection is very big, and this is a huge threat to the web-based application. The hackers can easily hack the system and obtain any data and information that they want anytime and anywhere[11].

4.1.1 How Does It Work?

A hacker(or users without database knowledge) typically supplies a malicious code in SQL statements via UI. Then, the application combines the malicious code(strings) received from UI with the application variables. As a result, the original or intended SQL query turns to a second query. It can lead to a various threats like denial of services, loss of data, and other security-related threats.

4.2 Antipattern

Execute unverified input as code.

4.2.1 Problems With The Antipattern

Often, we need to write a dynamic SQL queries for our applications. In the process, we concatenate user inputs with an application variables, which exposes the application to a SQL injection attack. A potential SQL injection attack is the price that we pay for building the SQL statements dynamically.

In January 2009, Heartland announced that the computers that they use to process payment card transactions had been breached in 2008[3]. 134 million credit cards were exposed through SQL injection attacks used to install spyware on Heartland's data systems.The method used to compromise Heartland's network was ultimately determined to be SQL injection[3].

In fact, the breach was a very slow moving event. It started with an "SQL Injection" attack in late 2007 that compromised their database. An SQL Injection appends

additional database commands to code in web scripts. Heartland determined that the code modified was in a web login page that had been deployed 8 years earlier, but this was the first time the vulnerability had been exploited [3].

4.3 Informatics

To begin with, let's review informatics definition and understand its field. Understanding the heart of any field and its scope helps to identify the valuable information and concerns related to it.

"Informatics studies the interaction of information with individuals and organizations, as well as the fundamentals of computation and computability, and the hardware and software technologies used to store, process and communicate digitized information. It includes the study of communication as a process that links people together, to affect the behavior of individuals and organizations" [5].

Some of the keywords in the definition are information, technology, people, and communication. Technologies or IT systems are being used to collect, store, and process information by people(or sub-systems).

An architectural perspective is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views[14].

4.3.1 Business Impact

SQL injection is one of the most common threats to a web-based applications. It is a problem that many people - including developers and organizations, are aware of, but due to various reasons, does little to prevent it. There are many reasons for this negligence:

- Not many organizations have resources, skills, and competences to detect SQL injection attacks.
- Investing less in security-related features and testing.
- Tight delivery deadlines with a limited budget- leads to a security compromise.
- Developers not following the standards to address the issue.
- Organizations not monitoring the applications. Besides, there should be regular code assessments.
- Organizations lack peer programming and code review practices in their software development process.
- Advanced hacker tools for finding vulnerable applications are available on the internet.

A result of a SQL injection attack opens a number of possibilities, which are limited only by the configuration of the system and the skills of the attacker [18]. If an attack is successful, a host of problems could result. The following are a sample of the potential outcomes [18]:

- Identity spoofing through manipulating databases to insert bogus or misleading information such as email addresses and contact information.
- Alteration of prices in e-commerce applications. In this attack, the intruder once again alters data but does so with the intention of changing price information in order to purchase the products or the services at a reduced rate.
- Alteration of data or outright replacement of data in existing databases with information created by the attacker.
- Escalation of privileges to increase the level of access an attacker has to the system, up to and including full administrative access to the operating system.
- Denial of service, performed by flooding the server with requests designed to overwhelm the system.
- Data extraction and disclosure of all data on the system through the manipulation of the database.
- Destruction or corruption of data through rewriting, altering, or other means.
- Eliminating or altering transactions that have been or will be committed.

All of these could lead to severe damage to the organization's reputation and will have an adverse effect on daily business. It is an organization's duty to protect its customer's data and ensure privacy while providing services. Usually, these rights are protected by data and privacy protection laws, like GDPR. Any organization failing to comply with these laws could face a lawsuit and expensive financial penalty.

4.3.2 Security Perspective

Security can be defined as the set of processes and technologies that allow the owners of resources in the system to reliably control who can access which resources[14]. For this, we need to identify stakeholders and their concerns right from the beginning. Also, we have to keep in mind that a security is about risk management that balances likely security risks against the costs of guarding against them. It can be quite difficult to accommodate all the concerns from the stakeholders, therefore, a careful analysis is required. According to the book "Software Systems Architecture"[14] by Nick Rozanski and Eoin Woods, a security perspective deals with the concerns like resources, principles, policies, threats, confidentiality, integrity, and availability etc.

The security perspective includes the following desired quality, concerns and activities(figure 4.1).

Desired Quality	The ability of the system to reliably control, monitor, and audit who can perform what actions on which resources and the ability to detect and recover from security breaches
Applicability	Any systems with accessible interfaces, with multiple users where the identity of the user is significant, or where access to operations or information needs to be controlled
Concerns	Resources, principals, policies, threats, confidentiality, integrity, availability, accountability, detection and recovery, and security mechanisms
Activities	Identify sensitive resources, define the security policy, identify threats to the system, design the security implementation, and assess the security risks
Architectural Tactics	Apply recognized security principles, authenticate the principals, authorize access, ensure information secrecy, ensure information integrity, ensure accountability, protect availability, integrate security technologies, provide security administration, and use third-party security infrastructure
Problems and Pitfalls	Complex security policies, unproven security technologies, system not designed for failure, lack of administration facilities, technology-driven approach, failure to consider time sources, overreliance on technology, no clear requirements or models, security as an afterthought, ignoring the insider threat, assuming the client is secure, security embedded in the application code, piecemeal security, and ad hoc security technology

FIGURE 4.1: Security perspective [14]

4.4 Solutions To The Antipattern

- Trust no one- User inputs must always be sanitized before it is used in dynamic SQL statements.
- Filter input- Regular expressions can be used to detect potential harmful code or force users to provide input in a specific format.
- Use appropriate privileges: We don't want all the users to have the same(admin) access rights as it can be misused.
- Parameterize dynamic values- A parameterized query is a query in which placeholders are used for parameters and the parameter value is supplied at the execution time.
- The following SQL query(figure 4.2) ensures that only the insert statement will be executed. Unlike, a SQL query created by concatenating user input, the query execution plan for the parameterized SQL is constructed on the server before the query is executed.

```
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$params = array($name, $email);
$sql = 'INSERT INTO CustomerTable (Name, Email) VALUES (?, ?)';
```

FIGURE 4.2: Parameterize dynamic values example

- Use stored procedures- It encapsulates the SQL statements and treats all input as parameters.
- Monitor SQL statements from database connected applications.
- Code review

Chapter 5

Summary

Hopefully our preceding chapters have stressed the importance of avoiding antipatterns and it is this avoidance that is at the heart of the project.

We wanted to get in contact with people maintaining databases in the real world, to examine which antipatterns had been implemented and why. However, this desire came as a result of a question that we originally had when we came across the concept of antipatterns. That is, there is all this literature about what antipatterns are, why they exist and why should they be avoided. Furthermore, they also talk about why are antipatterns so common?

Our initial response to that question was to want to answer it, but we missed that we already had a possible answer: Yes there's a lot of literature on antipatterns, but usage of the antipattern concept is limited to that literature. We had been taught systems architecture previously, and in that context we had learnt about viewpoints and perspectives, but not antipatterns.

It could be said that the subject would be too broad if we were taught multiple frameworks, rather than focusing on one. However, only viewpoints and perspectives are frameworks. Antipatterns are nothing more than an inversion of the common principle of "best practices" that feature as a part of all frameworks. As such, that antipatterns could have been taught within the framework of viewpoints and perspectives but weren't, illustrates that it's unnecessarily closed off as it's own separate concept.

In this manner, antipatterns are a tool that can be used within any other theory and design framework, just as patterns themselves are. In fact, it could be argued that antipatterns are an even more useful tool than regular patterns.

From our own experience with semester projects, things rarely go according to a predetermined plan, and there often arises unforeseen circumstances. We have a problem where there isn't a cut and dry best pattern to apply, or there are external pressures from deadlines, stress and parts not working out as they're supposed to. Not to mention needs changing during the project, or completely new needs being added. This has even been mentioned in our lectures on agile development as a flaw of the waterfall model, suggesting this is more than just personal experience.

All of this can combine to a project experience where the members aren't concerned with living up to the best practices prescribed by patterns, but just with making it work at all. It is particularly in these circumstances that awareness of antipatterns can be useful. Antipatterns provide clear guidelines of what to avoid doing when perfection is an impossible luxury, and everyone is just focused on getting it "good enough". Awareness of antipatterns could also prevent implementing bad code that complicate the development process, in addition to spotting already existing problems.

Of course, awareness of antipatterns might not be needed if the project never needs to diverge from what is planned. Any proper pattern should avoid antipatterns when applied correctly, so a project where everything goes smoothly and there's never any problems with following coding guidelines and best practices, would avoid antipatterns without being aware of the concept.

This is where empirical evidence and case study could provide data that this report can't on its own: how often does a project go smoothly vs. how often are there problems in development; how often antipatterns are the result of a troubled development, and how often do they cause it. As antipatterns can both be the cause and consequences of a troubled development cycle, another study could be on "death-spirals". This term referring to where to solve the problems caused by antipatterns, new antipatterns are used, which then causes more problems that need to be solved until the entire project is a patchwork "temporary" solutions that ended up being permanent.

With how useful we judge the antipattern terminology to be, it could be interesting to empirically test whether that judgement is correct, by examining real world projects through that lens. That would however require closer cooperation with a business or organization than we were able to acquire during this project.

Chapter 6

Glossary

1. **Ambiguity:** a situation that is unclear, but can be understood in more than one way
2. **Amelioration:** the process of making bad or unpleasant situation better
3. **Architecture:** the design and interaction of components of a computer or computer system
4. **Bound:** to go or to plan to go especially to a certain destination
5. **Computation:** the method of calculating
6. **Consolidate:** to bring together or unite things that were separate
7. **Coupling:** the act of bringing or coming together
8. **Dependency:** something that is dependent to others
9. **Deterministic:** forces and factors cause things to happen in a way that cannot be changed
10. **Enumeration:** the act of naming things separately; one by one
11. **Fuzzy:** not clear or not easily heard, seen or understood
12. **GDPR:** General Data Protection Regulation
13. **Generic:** relating to or shared by a whole group of similar things
14. **HCI:** Human Computer Interaction
15. **Integrity:** the quality of being whole and complete
16. **Interface:** a device or program enabling user to communicate with a computer
17. **IT:** Information Technology
18. **IS:** Information System
19. **Malicious:** intended to do harm
20. **Mapping:** matching process where the points of one set are matched against the point of other set
21. **Metrics:** a system or standard of measurement
22. **Nodes:** a point in a network or diagram at which lines or pathways intersect or branch
23. **Perspective:** cross-structural quality concepts
24. **SQL:** Structured Query Language
25. **Scenarios :** a description of possible actions or events in the future

26. **Spyware :** a description of possible actions or events in the future
27. **Stakeholder :** a person or group of person with something at stake, interest and concern
28. **Tangible :** real, existing that can be seen or touched
29. **UI:** User Interface
30. **UML:** Unified Modeling Language
31. **View:** description of one aspect of a system's architecture
32. **Viewpoint:** a template for developing the views
33. **Vigor:** strong feeling; enthusiasm or intensity

Bibliography

[1] Dimitrios Lettas Georgios Meditskos Ioannis Gtamelos Nick Bassiliades. "SPARSE: A symptom-based antipattern retrieval knowledge-based system using Semantic Web technologies". In: (2010).

[2] Joe Celko. "Trees and hierarchies in SQL for smarties". In: Morgan Kaufmann, 2004. Chap. 1, pp. 3–6.

[3] Julia S. Cheney. "Heartland Payment Systems: Lessons Learned from a Data Breach". In: (2010).

[4] Paula Kotzé KarenKostas Koukouletsos Babak Khazaei Andy Dearden. "Patterns AntiPatterns and Guidelines – Effective Aids to Teaching HCI Principles?" In: (2010).

[5] Michael Fourman, Michael Fourman, and Centre For Intelligent Systems. *informatics*. 2002.

[6] Nadia Bouassida Hanêne Ben-Abdallah Rahma Fourati. "A Metric-Based Approach for Antipattern Detection in UML Designs". In: (1970).

[7] IU Informatics. *What is informatics IU*. https://www.youtube.com/watch?v=bYour9b86Ys&fbclid=IwAR2nAXRY8a27gaFY1KbheDC32FEeshBzoHfcMBxvNABz8p_BQkMzqHlAEZ8. [Online; accessed 3-December-2019].

[8] Christopher Alexander Sara Ishikawa and Murray Silverstein. "A Pattern Language". In: Oxford University Press, 1977. Chap. 1, p. 10.

[9] Erich Gamma Richard Helm Ralph Johnson and John Vlissides. "Design Patterns: Elements of a Reusable Object-Oriented Software". In: Addison-Wesley, 1994. Chap. 1, p. 2.

[10] William Brown Raphael Malveau Skip McCormick and Tom Mowbray. "AntiPatterns Refactoring Software, Architectures, and Projects in Crisis". In: John Wiley & Sons, 1998. Chap. 1, p. 6.

[11] Mohd Amin Mohd Yunus et al. "Review of SQL Injection : Problems and Prevention". In: *JOIV : International Journal on Informatics Visualization* 2.3-2 (2018). ISSN: 2549-9610.

[12] Phillip A. Laplante & Colin J. Neill. "antipatterns Identification, Refactoring, and Management". In: Taylor & Francis Group, LLC, 2006. Chap. 2, pp. 13–30.

[13] Francesca Arcelli Fontanaa Valentina Lenarduzzi b Riccardo Rovedac Davide Taibi. "The Journal of Systems and Software 154 (2019) 139–156". In: (2019).

[14] Nick Rozanski and Eoin Woods. *Software systems architecture, working with stakeholders using viewpoints and perspectives*. eng. 2. ed. Upper Saddle River NJ: Addison-Wesley, 2012. Chap. 7, 25. ISBN: 032171833X.

[15] ALenarduzzi V Tosi D Lavazza LMorasca S. "Why Do Developers Adopt Open Source Software". In: (2019).

[16] Anabela Sarmento. “Issues of Human Computer Interaction”. In: IRM Press (an imprint of Idea Group Inc.) 701 E. Chocolate Avenue, Suite 200, 2005. Chap. 1, p. Vii.

[17] Dimitrios Settas. “Software Project Antipattern Knowledge Management”. In: (2011).

[18] “SQL Injection”. In: *CEH^{TM}v9*. John Wiley & Sons, Ltd, 2017. Chap. 14, pp. 389–408. ISBN: 9781119419303.

[19] Ioannis Stamelos. “Software project management antipatterns”. In: (2010).

[20] w3schools. *w3schoolsSQL Injection*. https://www.w3schools.com/sql/sql_injection.asp. [Online; accessed 4-November-2019].

[21] Wikipedia. *Wikipedia Informatics*. https://en.wikipedia.org/wiki/Information_system?fbclid=IwAR1mRcDeK119iBL-F2VFjL9njEgFxPYcXjRw8xvOhrVGUXie3VR6tfAygIk. [Online; accessed 3-December-2019].