Master Thesis

Database centric software test management framework for test metrics

for the fulfillment of the academic degree
M.Sc. in Automotive Software Engineering

Submitted by: Parawee Pleehajinda
Matr.-Nr. 326472
February 15, 1989
Bangkok, Thailand

Supervisors: Univ.-Prof. Dr. Wolfram Hardt
Dipl.-Ing. (FH) Matthias Schmidt

Date of submission: June 24, 2015
Parawee Pleehajinda

**Database centric software test management framework for test metrics**

Master Thesis, Faculty of Computer Science, Department of Computer Engineering
Technische Universität Chemnitz, June 2015
Declaration

I hereby declare that this master thesis, topic “Database centric test management framework for test metrics”, is entirely the result of my own work and it has been written by me in its entirety. Also, certify that I elaborated this research independently. The work is based on foundation of the information sources and literatures used in the thesis that I have faithfully and properly cited.

Chemnitz. June 24, 2015

Parawee Plechajinda
Abstract

Big amounts of test data generated by the current used software testing tools (QA-C/QA-C++ and Cantata) contain a variety of different values. The variances cause enormous challenges in data aggregation and interpretation that directly affect generation of test metrics. Due to the circumstance of data processing, this master thesis introduces a database-centric test management framework for test metrics aims at centrally handling the big data as well as facilitating the generation of test metrics. Each test result will be individually parsed to be a particular format before being stored in a centralized database. A friendly front-end user interface is connected and synchronized with the database that allows authorized users to interact with the stored data. With a granularity tracking mechanism, any stored data will be systematically located and programmatically interpreted by a test metrics generator to create various kinds of high-quality test metrics. The automatization of the framework is driven by Jenkins CI to automatically and periodically performing the sequential operations. The technology greatly and effectively optimizes and reduces effort in the development, as well as enhance the performance of the software testing processes. In this research, the framework is only started at managing the testing processes on software-unit level. However, because of the independence of the database from levels of software testing, it could also be expanded to support software development at any level.

**Keywords** – software testing, test management framework, test data, software testing tools, test metrics, database-centric, data aggregation and interpretation, automatization.
Acknowledgement

I would like to express the deepest appreciation to my professor Univ.-Prof. Dr. Wolfram Hardt and advisor Dr. Ariane Heller for their kindly patience, motivation, and knowledge and continuous support in my master study and research. The guidance suggested by them helped me in all the time of research and writing of this thesis.

Besides my professor and advisor, I would like to thank the rest of my thesis committee: Dr. Mirko Caspar and Dipl.-Inf. Norbert Englisch for their encouragement, insightful guidance and questions during my study and research.

My sincere thanks also goes to the department In Vehicle Systems and Services, Bosch Engineering GmbH, the mentor Dipl.-Ing. (FH) Matthias Schmidt and all of colleagues at BEG-VS/EEA2 for offering me the master internship and continuously master thesis opportunities in their teams and leading me working on diverse exciting developments.

Last but not the least, I would like to express sincere gratitude to my parents: Asst. Prof. Suchin Pleehajinda and Mrs. Tipbha Pleehajinda for giving birth to me and spiritually supporting me in every step throughout my life.
Table of Contents

Declaration .................................................. I
Abstract ........................................................ II
Acknowledgement ............................................... III
List of figures .................................................. VII
List of tables ................................................... IX
Abbreviations ................................................... X

1 Introduction .................................................. 1
  1.1 Motivation ................................................ 1
  1.2 Scientific challenges ..................................... 2
  1.3 Thesis structure .......................................... 6

2 Research ..................................................... 8
  2.1 Collaborative framework .................................. 8
    2.1.1 Process integration .................................. 9
    2.1.2 Framework environment ............................. 10
  2.2 Database in software engineering ......................... 14
    2.2.1 Major properties of databases .................... 14
    2.2.2 Requirements of database systems ................ 16
    2.2.3 Challenges in database technology ............... 18
  2.3 Database models .......................................... 20
    2.3.1 Relational model .................................... 20
    2.3.2 Object model ........................................ 26
    2.3.3 Object-relation model .............................. 31
  2.4 Database systems ......................................... 36
    2.4.1 Database system hardware ........................ 36
    2.4.2 Database system software .......................... 37
    2.4.3 Database system users ............................. 40
    2.4.4 Database system architecture ..................... 41
    2.4.5 Database management systems ..................... 44
  2.5 Framework automatization ................................ 47
2.5.1 Automatization in software development ........................................ 47
2.5.2 Interface of system components ...................................................... 49
2.5.3 Automatization management ............................................................... 51

3 Discussion .......................................................................................... 53
3.1 Framework conceptualization ............................................................... 53
3.2 Definition of collaboration ................................................................... 57
3.3 Database model selection ..................................................................... 60
  3.3.1 Comparison of alternative models ................................................. 60
  3.3.2 Compromising database models .................................................... 62
3.4 Database system selection .................................................................... 63
  3.4.1 Comparison of alternative database systems .................................. 63
  3.4.2 Compromising database management systems ............................... 64
3.5 Development tools selection ................................................................. 65
  3.5.1 Environment of the database .......................................................... 65
  3.5.2 Environment of raw data parsers .................................................... 66
  3.5.3 Environment of front-end user interface ........................................ 66
  3.5.4 Environment of test metrics generator ......................................... 67
  3.5.5 Summarization of the environment ............................................... 67

4 Implementation ................................................................................... 69
4.1 Environmental setup ............................................................................ 69
4.2 Development of the database ............................................................... 73
  4.2.1 Setting up a database using PostgreSQL 9.3 .................................... 73
  4.2.2 Implementing the database structure ............................................. 75
4.3 Development of raw data parsers ......................................................... 80
  4.3.1 Developing programming parsers .................................................. 80
  4.3.2 Transferring parsed data to the database ........................................ 81
4.4 Development of metrics generator ....................................................... 83
  4.4.1 Generating metrics in PNG format ................................................ 83
  4.4.2 Generating metrics in CSV format ................................................ 84
4.5 Development of user interface .............................................................. 85
  4.5.1 Generating the front-end user interface ......................................... 85
  4.5.2 Synchronizing data with the database .......................................... 87
4.6 Automate the framework ......................................................... 90

5 Conclusion .............................................................................. 93
  5.1 Development summary ...................................................... 93
  5.2 Future research ................................................................. 96

6 References ............................................................................. 98

7 Appendices ............................................................................. 102
  Appendix A – Evolution of database systems
    A.1 Evolution timeline .......................................................... 102
    A.2 Benefits of the evolution ............................................... 104
  Appendix B – Programming diagrams
    B.1 Programming dependency .............................................. 107
    B.2 Database middle interface ............................................. 108
    B.3 Data synchronizer ........................................................ 109
    B.4 Database updater ........................................................ 110
    B.5 Interface updater ........................................................ 111
  Appendix C – Programming codes
    C.1 SQL statements to create user-defined types ................. 112
    C.2 SQL statements to inherit existing tables .................... 113
    C.3 Python functions as the Database API ......................... 114
    C.4 Python functions as raw data parsers .......................... 115
    C.5 Python function to insert data to the database .............. 116
    C.6 Python with Matplotlib to generate graphic metrics ....... 117
    C.7 Configure a protection of the Microsoft Excel interface .... 118
List of Figures

1.1 Current framework of software unit testing ................................................. 2
1.2 Researching database-centric software testing management ......................... 5
2.1 Fundamental integration concepts ................................................................. 9
2.2 Relationship in a database system ............................................................... 18
2.3 Relationship types in relational database model ........................................... 21
2.4 Relationship of primary key and foreign key ............................................... 23
2.5 An operator created by a user-defined type ................................................. 26
2.6 Inheritance of an existing data object in object model .................................. 27
2.7 Encapsulation of a data object in object model ........................................... 28
2.8 Realization of polymorphism in object model ............................................. 29
2.9 Object-relational impedance mismatch in data representation ..................... 32
2.10 Transition states in database transaction ..................................................... 38
2.11 Functional architecture of DBMSs ............................................................... 41
2.12 Architecture of three-tier database systems ................................................. 43
2.13 An overview of typical database management systems .................................. 45
2.14 Alternative approaches of system interface ............................................... 49
2.15 Pipelining execution as a basis of continuous integration .............................. 51
3.1 An overview of conceptual framework ......................................................... 53
3.2 An overview process of raw data parsing using decentralized parsers ............ 54
3.3 An overview process of generating a user interface using Microsoft Excel .......... 55
3.4 An overview process of interpreting data though a centralized interface ........... 55
3.5 An overview process of manipulating data though database interface ............ 56
4.1 The concrete structure of development workspace ......................................... 70
4.2 The global configuration file used in administrating the framework ............... 71
4.3 The data object model used in handling data in object-oriented way ............... 71
4.4 Selecting the locale of PostgreSQL to support European characters ............... 73
4.5 A new database created for development of the framework ........................... 74
4.6 The structure of database implemented using user-defined types .................... 76
4.7 The structured directories used to store raw test data ................................ 80
4.8 Examples of raw data need to be parsed ..................................................... 81
4.9 The processes of transferring parsed data to database ........................................ 82
4.10 Graphics generated by the metrics generator using Matplotlib ............................. 83
4.11 Plain-text metrics generated in CSV format ..................................................... 84
4.12 The object-oriented structure used in handling queried data .............................. 86
4.13 Flow diagram of data synchronization .............................................................. 88
4.14 The realization of security mechanism embedded in the user interface ................. 89
4.15 Sequential execution flow driven by Jenkins CI .................................................. 90
4.16 The implemented framework controlled by Jenkins CI ....................................... 91
4.17 The configuration of software testing processes in Jenkins CI ............................ 92
5.1 A future framework expanded to support various applications ............................ 96
5.2 A future framework expanded to support various formats of test metrics ............... 97
List of Tables

2.1 Comparison of decentralized and centralized interface approaches .................. 50
3.1 Definition of the framework based on three-tier architecture ....................... 58
3.2 Comparison of alternative database models .............................................. 61
3.3 Comparison of alternative database management systems ........................ 63
3.4 Alternative PostgreSQL-API driven by Python .......................................... 65
3.5 Alternative Python packages for Microsoft Excel ...................................... 66
3.6 Development tools for framework development ....................................... 68
4.1 Descriptions of class attributes defined in data object model ..................... 72
4.2 Information of database tables used in handling development data ............ 77
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomicity Consistency Isolation Durability</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standard Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ASIL</td>
<td>Automotive Safety Integrity Level</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkley Software Distribution</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CMS</td>
<td>Configuration Management System</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create Read Update Delete</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-Separated Values</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DOM</td>
<td>Data Object Model</td>
</tr>
<tr>
<td>FK</td>
<td>Foreign Key</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>LGPL</td>
<td>Lesser General Public License</td>
</tr>
<tr>
<td>OODBMS</td>
<td>Object-Oriented Database Management System</td>
</tr>
<tr>
<td>OQL</td>
<td>Object Query Language</td>
</tr>
<tr>
<td>ORDBMS</td>
<td>Object-Relational Database Management System</td>
</tr>
<tr>
<td>PK</td>
<td>Primary Key</td>
</tr>
<tr>
<td>PNG</td>
<td>Portable Network Graphics</td>
</tr>
<tr>
<td>QM</td>
<td>Quality Managed</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-Only Memory</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SVN</td>
<td>Sub Version Control</td>
</tr>
<tr>
<td>SWUT</td>
<td>Software Unit Testing</td>
</tr>
<tr>
<td>UTF</td>
<td>Unicode Transforming Format</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

1.1 Motivation

Development of embedded systems in automotive industry is rapidly and continuously advancing both in hardware and software. A high complexity of development is evoked in every phase throughout the lifetime. Several development processes become more complicated, e.g. the initialization of the project, general operations, testing processes or the finalization of the project. In this circumstance, the completeness of each operation must be strictly ensured and validated. One of the essential processes that directly indicates the correctness and quality is testing which is involved in several phases throughout the development, e.g. reviewing, testing, integrating, etc.

In software development, an effective software testing yields a high quality of the development. It greatly facilitates developers in identifying progress and potentially risks or bugs arising during the development. Since any issue can be quickly detected, the developers will be able to handle it effectively and accurately steer the development into the right way. Thus, software testing should be considered, placed, and performed during the development as early and regularly as possible.

Therefore, in consequence, the continuous of the activity will hugely generate a variety of big data. Those artifacts will be separately stored in different locations before being dynamically transferred and processed in further operations. The decentralization and untracking of the stored data directly affect several processes of data aggregation and interpretation. Hence, without an effective big data handling mechanism, many related processes, e.g. data tracking, test metrics generation, etc., can be severely impaired and consequently degrade the quality of the development.

Furthermore, when software is being developed, any result and movement of every on-going phase must be assessed and visualized regularly through test metrics. The metrics are the essential keys for measurement that provide developers intuitive methods for progressive estimation and issue identification in any state of the development. As previously mentioned, the lack of big data handling mechanism will directly affect the processes of data aggregation and interpretation and causes enormous challenges in generating various kinds of test metrics.
1.2 Scientific Challenges

Although several tools for big data handling are available in software markets nowadays and used broadly in organizations to manage their essential data, but those are still not fully integrated and driven as a single collaborative system. In this case, the system with a high capability of data management that effectively supports the amounts of data is required, especially to facilitate a process of various test metrics generation in software development.

In the current software testing framework, illustrated by figure 1.1, each existing SWUT tool is responsible to analyze and perform software testing. In the areas of this research, the software testing is only based on software-unit level. Each software unit will be tested by the tools before getting results, i.e. QA-C/QA-C++ are applications used for static code analysis, and Cantata is mainly used for dynamic code analysis. Microsoft Excel is used to collect and summarize the data gathered from the testing activities by using the built-in features to generate graphics and metrics.

![Figure 1.1: Current framework of software unit testing.](image)

In the CMS, TortoiseSVN is used to handle each version of development artifacts. After a stored release has been checked out into a sandbox, it will be prompt to be transferred to code editors
and the used tools. Anyway, at the current, after finished a testing process, a large volume of complicate raw data, e.g. processors, subsystems, atomic components, test scripts and results, etc., will be hugely produced. It causes several challenges in data handling, especially in generating test metrics that requires several related data to be interpreted.

Decentralization of development artifacts

Due to a huge increase in development complexity of embedded system, a software or hardware component may be repeatedly used in several developing units involved in different platforms. For instance, a single project may contain multiple processors produced by different suppliers. Each of developing unit may be under control of several responsible developers, or even integrated to other systems. In this situation, the lack of co-relation of development artifacts significantly leads the software testing processes more complex.

At the current, test results of each sourcefile is separately stored in several directories in the CMS. By storing the artifacts in spread locations, the data will be stored without having any relation with others related artifacts. Due to the decentralization, any further process that requires those artifacts will not be effectively performed. The current framework becomes more complicated since huge amounts of data are stored in this manner. Thus, in practice, processes or developers that require the stored data for any further operation need to check the scratched data out from the CMS to their local workspaces, then manually categorize and summarize the data as desired.

Transformation of raw data

After a software testing activity finished, a big data of development artifacts, e.g. test scripts and test results, will be generated and sequentially conveyed to the CMS after being checked in. Those generated artifacts are fully raw that cannot be simply used in data processing. Thus, before being processed in further operations, the data that currently contain different values must be transformed into a particular format first in order to unify the variances.

As previously mentioned, several current SWUT tools currently generate huge amounts of data in various formats and contents depending on the configurations specified in the CMS. Hence, a transformation of the raw data will be an essential process that parses the data to be usable and understandable artifacts that can be effective used by any participating data-related operations.
Complexity in test metrics generation

As maintaining each version of development artifacts using the CMS as well as the impact of the decentralization, a process of metrics generation is slightly ineffective. Creating metrics to visualize the development in a specific period of time requires an effective integration of the stored data. Without the integration, the process may consume a huge effort to capture the required data that takes several high-cost processes.

The process of generating metrics should be simply performed. That means, the framework must be able to simply and effectively retrieve any needed stored development artifacts to generate test metrics. The simplicity can beneficially help to reduce effort in processing the uncategorized artifacts as well as result in a high quality of metrics.

Furthermore, instead of generating development metrics using provided tools of the Microsoft Excel, programming scripts can be alternatively used to automatically and programmatically interpret the data and transform into various kinds of metrics.

Project-independent framework

The use of a database-centric architecture in handling big data is one of the popular approaches that greatly accounts to the challenging situation that helps in maintaining every version of data securely and effectively.

Therefore, several points of engineering must considered before implementing the architecture. Those points of challenge entail all components in the researching framework, e.g. structure of the database, operating engines, system interfaces, automatization of the framework, etc. Furthermore, the developed framework must be fully independent from used SWUT tools or projects. The independence will globally support any level of software development to handle a huge amount of data continuously generated by testing activities and facilitate a generation of metrics.

The figure 1.2 presents the areas of this research to account all of the scientific challenges in managing software testing processes and test metric generation based on the current framework showing in the previous figure:
The researching framework aims to partially transform and extend the capability of the current one that becomes more complicated due to a lack data aggregation and interpretation. The database-centric architecture will be researched and implemented using a centralized database to electronically store test data in software testing. Various kinds of test metrics will be programmatically generated to help developers in visualizing the development as well as optimizing efforts. A worksheet of Microsoft Excel will be developed to be a friendly channel for authorized users to interact with the data stored in the database. Furthermore, an automatization of the framework will also be realized to support the development in automatically and periodically executing any task or operation.
1.3 Thesis Structure

Before the implementation of the database-centric software test management framework for test metrics could be started, an intensive study had been necessary. The study involved the relevant modules, the collaborative components, and the alternative approaches. This document will present the knowledge collected while studying the former framework of software testing. Also discusses the principles used to enhance the capability in test data handling, especially to automatically generate various kinds of test metrics.

Chapter 1

The introduction presents an overview of the existing software testing framework. It expresses the motivations that initiate this research, as well as determines the problem statements. A number of scientific challenges is explained. The goals of this research that are mainly emphasized in developing a collaborative framework to generate high-quality test metrics is also addressed.

Chapter 2

The chapter presents the research of the formal collaborative framework used in software engineering. It presents the relationships of all related component that are used to build up a complex integrated platform. The research covers studying of roles of database systems participating in software development. The evolution of database systems since the first system had been introduced is summarized. It gives a detailed research of several alternative database models and famous DBMS. The internal mechanisms of the systems necessary in driving the framework are explained. Furthermore, a deep study about the framework automatization, based on Jenkins CI, is also introduced in this section to present a number of potential approaches that can be used to operate the system in automatic.

Chapter 3

Detailed discussions of the alternative approaches, tools and environments is placed. As explained about the essential components in a collaborative framework in previous section, each of them will be considered and selected. The conceptual development is formed up and intuitively illustrated in form of specification and realization comparisons. Thus, the framework will be completely conceptualized and concluded in the part.
Chapter 4

The implementation and realization of the researched theoretical approaches are held in this chapter. Each programming part to build up framework is also detailed. This part will not present only the implementations of relative components, but also the interfaces and integrations of each developed modules that implement the selected approaches. Especially, the implementation of a mechanism that programmatically generates various kinds of test metrics which is one of the major parts of this research.

Chapter 5

This chapter gives a summarization of the whole development by concluding each step of the research, discussions, and implementations. As well as presenting an idea of future research in order to extend the capability to support other levels of software development.

In conclusion, as the structure of this document, the research will be presented by starting at identifying the scientific challenges that motivate the master thesis. Several alternative theoretical approaches are discussed before being selected to be used in the development. Implementations of the chosen approaches will be precisely explained. Furthermore, a number of possible additional researches in future is also introduced at the end of the document.
Chapter 2
Research

2.1 Collaborative Framework

In the last decades, there has been a major change in the way project development entities interact with their connecting applications and each other. The spread collaborative modules that are responsible to process their own tasks may away from centralized developing site, e.g. software testing tools may stand separately away from centralized data repository regarding to this master thesis. A new paradigm of distributed project engineering was emerged in order to hugely enhance the flexibility of the development chains as well as initiating customized solutions based on consumer demand.

The key to achieving flexibility, efficiency, and responsiveness is data that is available to serve the right data and ready to be accessed by the right person at the right time. In project development, to be able to capture and deliver the right data, modern developments are consuming data management technologies at ever increasing rates\(^1\). And each of these technologies improves the precise capture, and flow of data management, as well as automated delivery of the data that enables some activities in the development to function more effectively.

Therefore, to achieve agility across the engineering goals, data must be manipulated seamlessly and effectively from application to application without loss or corruption. The manipulation entails flowing from the software testing tools that generate a huge amount of software test results to the system that centrally store development data, and later delivery to the metric generator to generate various kinds of desired metrics used for development assessment, as well as transferring the data to developers or customers in different sites of working area.

In software engineering, the basis of collaborative operation greatly helps in improvement in the capabilities and performance of the project development, especially in term of exchanging of increasingly complex data continuously generated during the time.

\(^1\) Ct. [BARKMEYER et al. 2003] pp. 1-11.
2.1.1 Process integration

Typically, “integration” is the engineering and management process that creates or improves flows between collaborative processes designed for different purposes. What actually flows between the systems is data. As mentioned, the most critical to every phase of the project development is that all of the right data flows in the right ways among the processes, as well as the people who interact it, to interpret the data correctly.

Process integration aims to investigate relationships to produce classifications and merge activities into a standardized system. Integration is the process of merging elements from 2 similar antecedent processes to create a single process that can be used to replace the original processes. Figure 2.1 represents a static structure diagram formulated in UML, showing the relationships among the principal concepts defined in this section:

Figure 2.1: Fundamental integration concepts.

---

3 Ct. [BARKMEYER et al. 2003] pp. 3.
2.1.2 Framework environment

In a view of system integration, referring to the figure, this section presents detailed definitions of the fundamental concepts for systems integration and the terms used to describe them in order to build up collaborative framework, in the context of this research, that are capable of centralizing and integrating data generated by different sources, as well as jointly participating in accomplishing the major goals of aggregation and interpretation of software development data, as follows:

**System**

A complex and pre-designed combination of a number of sub modules of software, hardware, resources, or even humans to accomplish either general or specific functions. In order to specify the structure of the system, a process of system design is performed. The process includes several view of system captures, i.e. the component view, the information view, and the computational process view⁴.

Furthermore, the system functionality can be broken down into several sequences to illustrate its collaborative functions corresponding to each component subsystems. To accomplish the designing of the system, a specification for each granularity must also be available in order to initiate interfaces of the system.

**Component⁵**

A subsystem of the higher-level system that provides some or all of its functions as sub functions. In this context, a component is a member of the framework which is an application connecting and interacting to each other among the framework.

**Integration**

A state of a set of components or agents, as previously mentioned, that enables them to act jointly in one or more sub-functions of a function of a larger system.

To be precise, integration is also one of the essential systems engineering processes of design and development consisting a number of operations, i.e. identification of joining operation of 2 or

---

⁴ Ct. [BARKMEYER et al. 2003] pp. 3.
⁵ Might be called “Agents” in other literatures.
more independent agents in order to enable particular communications to enhance the functionality of the larger system, an operation of role assignment in functions of the larger system to nominal resources, determination of the suitability of specific resources to fulfill those roles, the configuration or modification of the resources to enable them to perform those roles, and the creation or acquisition of additional resources where needed\(^6\).

In this context, process of integration is taken mainly at the center of the framework to join each independent applications, e.g. software testing tools, parsers of test results, metric generator, etc., to the central repository. As mentioned, the integration results in improvements of the larger view of the system. That means, the functionality of the entire framework is enhanced by the integration.

**Behavior**

The activities of a single component, what the system or component exactly and precisely performs. Typically, a behavior is visualized by various means of system measurement through measurable attributes of the component itself. Therefore, technically, the behavior of any component or even the entire system can also be influenced by environmental conditions, past actions, or configuration.

**Function**

As previously mentioned, activities of a component is realized as several parts of behavior, called function, which satisfies the specific purpose of the activities. Technically, a function is modeled as a subset of state transitions corresponding to the purpose of the system. Precisely, terms of behavior and function are closely related which both of them are modeled in different views of system perspective.

**Resource**

In any execution of a function, the system may require several collaborative components or information set, called resources, to perform execution in order to accomplish the expect results. The resources can be classified active resource or actor which is a resource that performs one or more functions, e.g. manpower or person, devices, etc., to makes decisions of one or more function. On the other hand, a passive resource is used by active resources in the performance of

---

\(^6\) Ct. [BARKMEYER et al. 2003] pp. 5.
a function. In this context, numbers of passive resource can be referred to the resources that are used to represent the raw data or any static system component that is indirectly used in a function\(^7\).

**Process**

A process is typically used to represent a sequence of collaborative procedures that are linked to build up intended activities in order to achieve any expected results. Process is basically classified using level of human involvement in initiation or execution, i.e. automated process which is a process that is executed without any intervention of human, semi-automated or computer-assisted process that is executed with a partial support of human, and manual process that is fully executed with intervention of human or human or human-intelligence.

**Role**

In system collaboration, a role is basically characterized by desired behaviors. A role is used to characterize a resource to imply its ability in order to exhibit those behaviors. As illustrated in the figure, a role can be classified in either activity or system role, as follows:

- **Activity role**: The participation of a nominal resource in the activity, in level of function and realized component behaviors.
- **System role**: In system level, the role broadly involves a set of all activity roles processing and using resource supported by the system.

**Joint action**

The action entails all participations of collaborated resources in accomplishing a single function, e.g. coordination, collaboration, cooperation, etc. The action involves interactions among the resources in order to jointly establish communication on the part of each actor in either activity level or system level.

**Communication**

Technically, one actor may communicate with another for the purpose of obtaining specific behaviors of the other party that are relevant to a function the actor is performing, as well as obtaining specific behaviors from either or both communicating actors. The operation is said to

---

\(^7\) Ct. [BARKMEYER et al. 2003] pp. 5.
be an activity of exchanging data between actors resulting a flow of information establishing from one actor to another.

The flow broadly refers to any process of process mechanisms, e.g. rendering and transmitting of internal information, and, vice versa, interpretation of received data into internal information.\(^8\)

**Protocol**

Any communication must be established by precisely constructed higher-level abstraction of communication, called protocol, to initiate a way in which information flows between participating resources.

**Interface and Presentation**

Interactive channels allow user or even interacting parties to communicate with internal system as well as interacting with any possible interactive internal components. In the context of this thesis, the interface is emphasized in interaction between incoming parsed software data and outcomes of interpretation of the stored data in repository displaying on interface in various forms of presentation.

In summary, a systems integration is an engineering process for constructing systems. One of the first questions that arises is: What is meant by “automating systems integration”? And in order to answer that question, it is necessary to introduce and define a larger set of terms at first based on the mentioned essential fundamental system components.

\(^8\) Ct. [BARKMEYER et al. 2003] pp. 5.
2.2 Database in Software Engineering

Software engineering is an application of complex processes involving a variety of collaborative tasks driven by software development teams to develop numbers of methods and products on the various production phases. Huge amounts of developed objects and units of information generate challenges of the development that require an appropriate tools to persistently store them as well as effectively manage the archiving system to meet the expected goals of development.

Database technology has provided a variety of possible solutions that exactly address the challenges since the first database system was introduced in over the past 30 years⁹. Nowadays, numbers of database management systems are used widely with a capability of extending the range of application integration to support various business requirements. Obviously, the wide-range integration indicates that database technology and software engineering are related in several collaborative ways and support each other to appropriate services to the requesting processes.

2.2.1 Major properties of databases

A database is a set of interrelated data which a known data can be stored with a structured relationship. Various properties of a database beneficially help in the entire process of software engineering, e.g. a database represents a structure of real development structure which is a logically coherent set of potentially huge amounts of data with concrete implicit or explicit meaning to facilitate development lifecycle to achieve expected goals whether in quality aspect or data consistency and integrity as well as maintain security of the stored data, the independence of data offers effective mechanism of manipulations of the stored data that can be operated independently as well as accessing database can also be simply operated over a user-friendly and flexible interaction channels which maximizes the operation performance of users.

Database technology constitutes modeling of real-world aspects of software development by comprising concepts and system methods for a construction and an operation of the software system which definitely are essential parts to lead the software development to success as well as response demands of the organization and desired achievements of the development. The

---

important properties of database system, that are essential for developing any secure and effective systems, are discussed as follows\(^{10}\):

**Data integrity**

Mechanisms to control semantic integrity in order to ensure the consistency of the stored data during its manipulation are provided in a manner of data constraints. The automatic detection mechanism greatly helps in checking the correctness of configurations and implementation.

**Data persistence**

Once a data object has been persistently recorded in a database, its life spans beyond the execution time allowing it to be manipulated anytime. This manner corresponds to a goal of software engineering in order to develop a system object that can be reused by other processes within or even across the different system environments.

**Data security**

Any malicious or accidental access can be detected and protected by security mechanisms to prevent access by unauthorized users. User privilege and rules can be constructed in database configuration to grant rights to known users to access any authorized part of the database.

**Data querying**

Stored data can be retrieved by users or even through programming by specifying a user-defined condition or criteria. The querying can be simply performed over the system interface using a declarative query specification with a relationship among the specified data. Various standard query languages have been developed and commonly used nowadays including Structured Query Language (SQL) for relational database, and Object Query Language (OQL) used in object-oriented database.

In fact, each database management system also provides specific features to support the specific purposes of software development. Therefore, the general purpose of all systems is to manage huge amounts of system data in a consistent manner as well as providing comfortable and effective

---

\(^{10}\) C\(t\) [DITTRICH et al. 2000] pp. 2-3.
mechanisms of data manipulation in the database. The context of database system is currently stated an integration of a database and database management software which mainly provides more functionality to facilitate every individual task of software engineering.

2.2.2 Requirements of database systems

Databases are used as the data interconnection as integration medium to be central interfaces of various applications and subsystems. Several functional and operational requirements are generated aimed at exploring effective solutions to support more complex archiving models of software engineering environment to handle huge amounts of system artifacts which are system or human generated data objects, e.g. diagrams, programming codes, executable objects, test data, or human-readable documents. The engineering requirements of databases for software engineering are discussed following:

Persistent repository of artifacts

The data model of a database must be able to store different kinds of artifacts as well as allowing the database schema including the description of potentially required artifact type and the defined artifact descriptions to be dynamically evolved during the lifetime to support a variety of artifacts.

Artifact integrity and consistency

The integrity and consistency of the stored data are driven by the mechanism of static constraints in order to validate any state of artifacts in the system while dynamic constraints maintain an integrity and consistency of state transition and development history over time. The mechanisms provide a strong and secure system in order to main the integrity and consistency of the stored data that users are able to operate system without being aware of any state of individual stored data elements in the system.

Artifact retrieval

The techniques of data retrieval depend on the implemented data model in the database system. Therefore, as using a database in business allows every stored artifacts can be recalled and reused anytime in a manner of reusable software libraries which is a prerequisite for software engineering to enhance the capability of the system to handle any demand on manipulating data.
Artifact configuration management

During the lifetime, numbers of artifacts generated in execution states and reflect progress of development. Various versions of the artifacts as well as the relationships among the artifacts at each state must be stored to effectively support software development.

In complex software system, the file-based mechanism to operate versioning of system artifacts is called software configuration management which provides functionality of system artifact management to support the versioning of a granularity of system artifact. Therefore, the database must be able to support a customization of version management granularity as well in order to maximize the flexibility of the artifact storage system.

System integration

In software engineering environment, aka. SEE, the integration of collaborative development tools is important, e.g. a data artifact generated by one tool must be usable by other tools in the environment. The requirement can be achieved by constructing a common interface to access the infrastructure to facilitate interconnection of the tools and provide services to each other.

Evaluation of development metrics

Assessing software development using software metrics has been considered for several years. Various kind of metrics are used to effectively assess progress of development throughout the lifecycle as well as identifying issues occurring during the development. Thus, the requirement is emphasized on the capability of the database that must be able to collect and persistently store any specific data used to generate metrics to provide possibility of subsequent evaluation in any point of time during the development timeline.

As mentioned above, the requirements are mainly based on ease and effectiveness of data manipulation in a way that securely and correctly maintain the characteristics as well as the correctness of the affected data. Furthermore, the database systems must also be able to serve a possibility to any additional process in term of data and process interaction to establish a dynamic connection in processing the stored data with minimal limitations. If the requirements are taken into account, any database system will be able to grow up as a versatile and more powerful collaborative system that greatly responses the needs of organization.
2.2.3 Challenges in database technology

In recent years, several database approaches have been introduced and used to develop powerful and appropriate file storing systems. In fact, each approach provides several advantages and also disadvantages depending on the development environment as well as applications which it is integrated with. Various special-purpose systems have been developed to response demands of specific business purposes as well as satisfy requirements of database system. The systems are based on the principle of supporting versioning of development artifact as well as extending in data and transaction models to extend the functionality of the system.

Meanwhile, several general-purpose database systems have also built up to support a wide range of extending of big data and mainly to support effective modeling of software artifact, software configuration management, dynamically evolving of development and cooperative transactions.

Up to this point, obviously, one database system whether it is developed for special or general purposes, it is definitely based on the common infrastructure with common basic mechanisms. A system is commonly comprised of a relationship of multi-level subsystems, i.e. file systems at the lowest, object management systems, and database management systems at higher levels respectively, described as the figure 2.2\(^\text{11}\).

---

\[\text{Figure 2.2: Relationship in a database system.}\]

Software engineering is a term of application dealing with non-tangible products which can be widely evolved over time or even manipulated across different platforms or organizations. Since an appropriate approach has been implemented in database, the further considerations in several aspects must be evaluated.

In order to optimize the infrastructure, the use of system resources must be minimized together with a functionality of the system allowing components to be accessed across different environments from different software engineering organizations as well as providing mechanisms to share the persistent stored artifacts transparently to optimize computational resources by operating sharing artifacts thought a common interface which can also be connected to a variety of interconnected tools as needed without being aware of physical infrastructure.

Furthermore, a mechanism that retrieves the stored data objects which is one of the major functionalities of database systems must focus on aspects of data integrity and security by providing a powerful mechanism of fault detection and tolerance to the system. Since the functional and operational requirements have been identified and responded effectively, database systems with high data storing technology are now enough to be implemented in general purposes of a wide-range interconnected business applications\(^\text{12}\).

2.3 Database Models

In database systems, a data model is used to explicitly specifying the data structure of a database in which each data element associates one another as well as providing the definition, format, and relationship among the data to realize the aims of a business. A model is a realization of a formalization and documentation of existing or ideal events and processes interpreted during the system design phase to visualize and translate complex system specifications and technical requirements into a precise and understandable representation of event and process flows. The representation is generally generated as formal diagrams or flowcharts to illustrate the relationship between data elements and also ensures all of the specifications and requirements have been identified and structured.

Basically, a data model is used only to analyze the requirements of a business mainly emphasizes in generating system representations and visualizing operating framework. Database modeling which is an operation of interpreting the realized representations will be constructed after the conclusion of data modeling takes place to formalize a logical structure and also implement the relationship among the system elements by forming a linkage between each data element to one another. Since a data model has been generated, a database model will be selected to implement the system of the data model as well as used to determine how the system is structured and specifies the manner of data manipulation with a relationship between each part of data.

In system development, there is a number of database models to implement a database system. This research focuses only on following common logical data models presented in the figure 2.2, in previous section, which are widely used in businesses nowadays.

2.3.1 Relational model

A relational model is based on a collective set of multiple data sets represented in a tabular form consists of general model components, i.e. tables, records and columns to establish a concrete-defined relationship among database tables and components. Relationships of components to establish communication between tables and also allows every component to share information

---

between each other that facilitates data searchability, organization and reporting when performing system analysis using mathematical techniques\textsuperscript{15}.

The relational model organizes data in several different ways. A table contains numbers of record which each record is identified by a unique data instance. Each table has its internal relation to identify each table columns. In database systems, there is a number of data relationships used in real systems to establish linkages of every data elements stored in database, in figure 2.3:

\begin{figure}[h]
\centering
\subfigure[]{
\includegraphics[width=0.4\textwidth]{a.png}
\caption{Relationship types in relational database model. (a) One-to-one relationship.}
\label{fig:a}
}
\subfigure[]{
\includegraphics[width=0.4\textwidth]{b.png}
\caption{Relationship types in relational database model. (b) One-to-many relationship.}
\label{fig:b}
}
\subfigure[]{
\includegraphics[width=0.4\textwidth]{c.png}
\caption{Relationship types in relational database model. (c) Many-to-one relationship.}
\label{fig:c}
}
\subfigure[]{
\includegraphics[width=0.4\textwidth]{d.png}
\caption{Relationship types in relational database model. (d) Many-to-many relationship.}
\label{fig:d}
}
\end{figure}

Each relationship defines a specific manner of the linkages amount tables in database that provides different advantages and functionalities in database organization, as follows:

**One-to-one**

In figure 2.3(a), the relationship refers to relationships of 2 items which one is able to only relate to another one. In a meaning of database, means one table record exactly relates to another record in another table.

One-to-many

In figure 2.3(b), the relationship allows frequently mapping of items which one is stored in one place and referenced by multiple other items. In a meaning of database, means one table record has relationship with many records in another table.

Many-to-one

In figure 2.3(c), the relationship is a vice versa of the one-to-many relationship. It mainly presents the mapping of the side of multiple of items that referenced by a single item, whereas the one-to-many focuses on the side of a single item. In a meaning of database, means one or more than one table record relates to a single record in another table.

Many-to-many

In figure 2.3(d), the relationship is placed among more than one item which relates to multiple items in another place. In database term, means one or more table record can be related to multiple records in another table.

In data organizing, the database system generates a new unique value every time when a new data record is written to the table, the unique value is called PK. The primary key is used by the system primarily for accessing the table to uniquely identify a single data record in a table. At least one column of a table must be specified and registered to be the primary key of the table.

Whereas the primary keys are used to uniquely identify a data record in a table referenced by other tables in the system, another important key of the system which is used to reference a data record of another table is called an FK. A table which has a foreign key referencing to another table is called “referencing or child table” and a table which its primary key is referenced by the foreign key is called “referenced or parent table”. The relationship of the primary key and foreign key is demonstrated as the following.

---

16 Ct. [BALTER 2007]
The figure presents a relationship between the primary key of table “Product” which is referenced by a foreign key in table “Component” that uniquely identifies a data record of the referenced table. If the value of the referenced key is updated, the meaning of the referencing table will also be automatically modified. Thus, developer is highly responsible to handle those keys securely.

In practice, there is also a possibility, provided by the relational model to support the reality of real-world database implementation in organizations, that the primary key can be composed of a combination of multiple columns in a table in case of a unique data record must be identified by a set of columns e.g. a unique product model must be identified by the product name together with a revision or build number, or a car must be identified by its brand model name.

The relational database model greatly provides several advantages in developing database systems. These following characteristics of this model also serve the major benefits to developers as well as users in various point of views in software engineering:

**Ease of use**

A set of collaborative components consisting tables with a number of rows and columns representing in tabular form is easy to understand and can be simply customized by various established open-source technologies.

**Flexibility**

By implement the tabular form, a new component whether it is a new table or record can be easily modified and manipulated by operators as desired. Once the primary key of a table has been generated as well as a relationship is established among tables, any transaction or operation between tables can be simply and flexibly operated under the defined relationships and constraints.
Data independence

In the relational model, a process to refactoring a table to decrease any potential redundant of the stored data is called “Normalization”\textsuperscript{17}. The process is used to maintain than complicated nested structure without losing of data and also makes the structure is more easily to be operated.

Precision

Since the structure is normalized, any manipulation of the relational model using the unique primary key and foreign keys together with the structure designed based on a precise mathematical data set concept will precisely ensure unambiguity of the system which beneficially support the linkages in a complicated database schema.

System integrity

The authorization can also be implemented easily by moving sensitive attributes in a given table into a separate permission-required relation. Furthermore, the system administrators can also grant a right or permission to any user who is allowed to access the database to limit operation permission in the system.

Data manipulation

The possibility of responding to query by means of a language based on relation algebra and relation calculus, e.g. SQL is simple and useful in the relational database approach. Manipulation of the stored data can also be operated easily by specifying a table or any system component want to execute the desired operations over.

Due to the fact that the relational model is based on a flat-tabular structure that is generally normalized by the normalization process, thus any risk of data duplication or even inconsistency of data records will not be occurred in the system to ensure system security in term of performing operations or system maintenance.

Anyway, by using a complex technique in data management, several drawbacks also emerges as disadvantages of the model, as follows:

\textsuperscript{17} Database normalization is the process of organizing the attributes and tables of a relational database to minimize data redundancy by decomposing a table into less redundant tables but without losing information.
Performance

The relational model is highly flexible, but the flexibility of the system is able to effect the performance of the system. If the number of tables or even collaborative components in which established relationships are large, any operation made to the system will also takes more computational effort and directly effect to overall system performance e.g. complex queries consume sophisticated processing resources, if a database is designed and connected by external data sources or other interconnected applications any execution may requires more system performance to execute the queries and quickly response.

Physical design

Furthermore, performance of interactive system of the model also directly depends upon the physical storage consisting of data tables, records, or even data elements stored in the database. This means the physical data layout and system’s schema must be chosen properly to maximize the system performance which directly effects to resource consumption and system utilization in most frequently run operations.

Extraction of the meaning of data

The data organized by several separated tabular form is naturally slower than data organized in a hierarchical manner in term of meaning each data. An extraction of the meaning requires a numbers of operations to interpret the meaning of a relationship among database components.

Key and constraints handling

The model requires key constraints in order to establish and interpret relationships between tables, if a table lacks a unique key, e.g. the primary key was modified without being aware of another executing transaction, the database may result incorrect querying outcomes. This means if only a key is broken, it may impact every relationship among the entire system\(^\text{18}\).

In summary, the relational model provides a huge advantages that to support wide ranges of application, but it also comes with several disadvantages in key handling which is the main process of this model. Thus, before implementing the model, developers must carefully consider about it.

2.3.2 Object model

An object model is a representation of an OODB, which information is represented in concept of object-oriented paradigm which visualize every unit of components or even a single system element as an object as used in object-oriented programming. The object model is completely different from the relational model which is a table-oriented approach that system components are represented in tabular form, whereas the object model presents as objects that makes the model suitable for complex applications. Most object-oriented databases are operated by OODBMS which allows objects to be manipulated using OQL, which is a language based on declarative programming approach.

Theoretically, accessing of data by using the object query language is faster than an implementation of relation database which is based on a tabular approach, because the direct relationship of each object under the object-oriented structure does not requires any extraction and interpretation of relationship among stored objects, the database stores data and logically establish direct relationships among the objects without requiring a process of relationship interpretation in order to extract the meaning of relationship between data which is required in the relational model\(^\text{19}\).

Using object query language to operate an object-oriented database system offers a beneficial possibility to developers to create own UDT or composite object types from any built-in database types or even inherit from any previously created object types which is the main property of the object-oriented concept in order to reference to other object classes to provide higher-level ways to organize and access data in the system. The following figure demonstrates an implementation to take the advantage of creating own user-defined object types in object-oriented database.

\[\text{Component}\]

<table>
<thead>
<tr>
<th>ComponentID</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partnumber</td>
<td>Integer</td>
</tr>
<tr>
<td>Supplier</td>
<td>PERSON</td>
</tr>
</tbody>
</table>

\[\text{Person}\]

<table>
<thead>
<tr>
<th>PersonalID</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Text</td>
</tr>
<tr>
<td>Contact</td>
<td>Text</td>
</tr>
</tbody>
</table>

Figure 2.5: An operator created by a user-defined type.

The figure 2.5 demonstrates how a user-defined type is used in an implementation of object model. A user-defined type “Person” is created first by combining 3 existing built-in data types provided by database system, and then a database table “Component” is constructed by using 2 built-in data types to specify attribute names as well as using the created user-defined type as the type of “Supplier”. This advantage beneficially helps developers to reuse their user-defined type as needs and also able to create any new type as much as business application require.

Furthermore, numbers of user-developed methods can be included in objects which is encapsulated by the object-oriented properties. When user fetch or manipulate a set of related objects as a single unit, the entire nested structure with other objects directly connected will be automatically proceed. On the other hand, with the behavior of object model, when any single unit, called parent, in the structure has been modified, the entire underneath structure or nodes, called children, connected to it will be automatically updated as the inheritance behavior of object-oriented concept.

The major characteristics of object-oriented approach that related to database system development are illustrated as follows:

**Inheritance**

In object-oriented programming, inheritance enables new objects, called children, to inherit the certain properties of parent objects. A child inherits visible properties and methods from its parent with a capability of adding additional properties and methods to itself, as the figure 2.6:

![Figure 2.6: Inheritance of an existing data object in object model.](image)

---

**Encapsulation**

A simplified and understandable way to use the created objects without being aware of their complexity inside. By hiding the internal details and properties of the objects, developers can only operate the objects by normally and easily accessing over the existing methods of the objects.

For instance, to get the current status of an engine equipment, developers only query through an operator instead of executing several operations in the objects. When the operator is called, the internal methods will be executed and return value to the operator. Thus, developers can get the current status of the engine component without being aware of the internal hidden methods, illustrated as the figure below:

![Figure 2.7: Encapsulation of a data object in object model.](image)

**Polymorphism**

In the object-oriented programming, polymorphism or pseudo-typing is used to make applications more modular and extensible by providing a powerful way to create interchangeable objects which can be manipulated as needs. For instance, an interchangeable operator is create without being specified its data type. This makes a number of connecting objects able to access the operator at the same time and manipulate data over it without being aware of the specified data type, the realization of polymorphism is demonstrated in the figure 2.8:
In the figure, a polymorphic type “BuildStatus” is created as an interchangeable operator and also simultaneously assigned completely different values by tables “Product: A” and “Product: B”.

By taking the advantage of polymorphism, when developers perform querying to get the current status of all tables currently existing in the structure, the developers only execute a query command through the operator of the polymorphic type, then both different stored data assigned by the 2 tables will be automatically returned to the developers.

The techniques of object-oriented programming provided by the object database model greatly offer developers a huge advantages in application development as well as data management in developing database systems. These following characteristics of this model also serve the major benefits to developers as well as users in various point of views in software engineering:

**Ease of maintenance**

Most of the processes in the implemented system are fully encapsulated that can be reused and safely incorporated into new behaviors. It greatly reduces maintenance costs, in term of programming efforts and also complex problem solving. Furthermore, system resource consumption can also be minimized, on the other hands, as well as maximize the lifetime of the database system.
Reliable and flexible

New behaviors within an object-oriented system can be built by a reuse of existing created objects by dynamically called and accessed to create new objects anytime. The new created objects, called children, may inherit data attributes or even internal user-developed methods from existing one or many other objects, called the parent or parents, without effecting the existing systems functions. Furthermore, new user-developed methods can also be added to a single new created object without effecting others.

Reusability

A new created data object will automatically inherit everything from the parent class where is was spawned. The data attributes and object characteristics will be inherited to the new object. Meanwhile, the object will also automatically inherit the data and behaviors from its parent classes in which it participates to the entire data object layers underneath.

Real-world modeling

The object model tends to effectively reflect and response demands of businesses in order to develop database system using a model which supports a real world data structure of the organizations which are generally designed by using hierarchical structure. As following the object-oriented concept, the stored data are represented as objects and organized into several multi-level classes, and each object is associated with own behaviors.

By using more logical mechanisms in data handling, several disadvantages of the object model also emerge that hugely effect several process in database organization. These below points of view present the main drawbacks caused when implementing the model in real applications:

Schema modification

Although the implementation of object model offers several advantages, but any modification of the database schema in an OODBMS typically involve recompiling the entire persistent schema. A modification of object or class instance, e.g. inserting, deleting, or re-designing, entirely effects other related objects in the same class that interact with the modified one.

---

Language dependence

Operating an OODBMS is dependently executed by a specific object-oriented programming language via a specific Application Programming Interface (API), also known as a wrapper, which programatically connects both sides of interacting partners, i.e. the internal database system and external connecting applications. Also modifying the API may technically require a huge redesigning or recompiling of the entire developed programming components.

Broken encapsulation

Because of the encapsulation of the object model, any manipulation performed on objects in an OODBMS is definitely highly dependent on the design of the schema. The structure allows users to execute an operation only in the way the system is designed without breaking the relationships among the stored data objects.

In contrast, the relational nature of the data, in the relation model, allows users to query any data element in any component of the database depending on the user’s rights. Also joining tables or classes of the stored objects is not possible in the object model, whereas the operation can be simply performed in the relational model. Thus, developers must compromise both relational and object models carefully in several perspectives of application development.

2.3.3 Object-relational model

In previous sections, the relational and the object models were discussed. Several advantages offered by the relational model powerfully facilitate the implementation of database system by providing numbers of strong mechanisms in order to maintain several aspects of storing data in the database. Anyway, the major weaknesses of implementing the relational model in database for business applications are the effort and expertise of developers which require a high system designing skills to properly construct the system to support huge amount of incoming data with various different data types\(^{22}\).

Meanwhile, the systematic and concrete relationships of the stored data objects of the object model lead a high complexity in data handling. Any modification made to the stored may break the entire structure or at least the structure must be recompiled. Also, in favor of accessing data objects, the

\(^{22}\)Ct. [BAROUDIBLOOR 2004] pp. 5.
well-structured database schema only allows users to accessing a data object by strictly following the encapsulated processes.

Throughout the evolution of database systems, a new hybrid data modeling approach, called the “object-relational model”, which is operated by an ORDBMS, was introduced to model a database system as a middleman between relational and object models to minimize the conceptual gap of the different data organizational approaches which is called the “object-relational impedance mismatch” by bridging the strengths of both models to enhance the performance of database system as well as facilitates a higher performance for the system. The major points of the object-relation impedance mismatch are presented as follows:

**Data representation**

The relational model represents data in tabular form which a table contains a number of records, and each of record contains a number of attributes, as the figure 2.9(a). In contrast, the object model represents data in the form of objects that each single object has its own properties, as the figure 2.9(b). The impedance mismatch is illustrated as follows:

![Image of data representation](image)

Figure 2.9: Object-relational impedance mismatch in data representation.
Navigational model

Accessing a stored data in a database implemented using the relational model is performed by extracting and interpreting the relationship among the keys. In the object model, to access data in objects, developers need to follow the procedures restricted by the structure to access each object level through the precisely constructed relationships.

Transaction management

In general, the object model has no characteristic of isolation and durability. Whereas transactions and concurrency manipulation with relational model are flexible. When stored data with this model are manipulated in a database table by different transaction in parallel; the transactions will be executing separately without effecting another (will be detailed in later sections). In contrast, the object model is based on creating an instance of a parent node, when 2 instances are executing in the same time, any shared operators may be impacted by the concurrency.

The minimization of the object-relational impedance mismatch, also call bridging, is based on an integration of the strengths of both relational and object model. The enhanced model is built by generating a new model based on a relational database with the extended characteristics of object model on the top. These following advantages are results of the invention of the object model that are mostly based on the advantages of using object-oriented techniques in developing systems:

Inheritance

The inheritance offers developers a possibility to develop classes for their own data types by inheriting certain object properties from other parent classes to be used in other functions of the database. As taking the advantage of inheritance, the costs of system development can be reduced because the structure will be constructed based on the real-world modeling and automatically effectively inherit all properties and capability of one class easily to other inheriting classes to reduce programming effort of the developers. Also when querying data from the parent, all of the stored data of its children will be return to users to decrease accessing costs of the operation.

Polymorphism

Polymorphism in the object-relational model beneficially allows a number of different tables within the same database can connect to the same operator by only establishing own relationship to the
operator. This behavior beneficially helps developers in order to simply connecting tables by the operator name and perform querying as normal, the database will return all records with different stored data in the same operator.

**Encapsulation**

Encapsulation of objects provided by the implementation of the object-relational model is in the form of tree-like data accessing. This characteristic allows users to only access data objects by following the constructed procedures to maintain the meaning of data objects, e.g. querying data by accessing a child operator cannot be directly performed without accessing through its higher classes. This behavior provides data integrity and security which are 2 major properties of database system in software engineering.

**Extensibility**

Object-relational model capabilities are extended with the addition of user-defined data types, user-developed methods, and beneficial characteristics found in object-oriented programming. The features offers a huge possibility to users e.g. in a column which is defined the data type as numeric data types also allows users to insert string characters by the support of polymorphism of object-oriented characteristics. Furthermore, any already existing built-in data types can be combined to create a complex user-defined data types as a new data object which can be opaque or distinctive without any limitation\(^\text{23}\).

Although the combination of 2 different approaches greatly help developers in several ways in data management, i.e. the relational model provide a huge simplicity in data handling, the object model offers a number of techniques in object-oriented ways in programming, but several disadvantages of each approach still possibly exist even the mismatch is considered as well as the gap has been minimized. Thus, developers must have a very high experience in handling the model in both ways of the bridged approaches. The main disadvantages of the hybrid model are discussed as follows:

**Key and constraints handling**

Because of the any relationships in the object-relational model is establish based on the basic of relational model. The important major weakness of this model definitely is a mechanism to handle

all the existing key constraints whether it is the primary key or foreign keys of each table. A modification of any key may still sensitively impact the relationships established among database components. If any key has been incorrectly modified or even a new key is generated by incorrect procedure, the entire relationships may be suddenly broken that severely causes a huge failure in the running system.

**Effect of the inheritance**

Although one of the object-oriented characteristics which is the inheritance of objects offers a huge benefit to developers in order to extend the database structure, but, as the known behavior, the entire properties of the parent will be inherited to its children. This makes the children have no possibility to select which property they do or do not need to inherit. Once a number of children is holding several unneeded properties inherited from their parent, the system resources may be unnecessarily consumed.

**Expensive maintenance**

Any children table inherited from a parent which is being removed from the structure will be automatically cascade removed as well. This requires necessary additional processes to handle the operation in case of the children table must not be removed, e.g. a general process is creating a new type, create new individual tables to handle the children, and then transfer the existing stored data to the new tables.

In conclusion, each database model provides several advantages and also disadvantages in different development perspectives. Therefore, according to the scientific challenges in developing a huge collaborative framework that must be able to dynamically support various levels of software testing, the object-relational (hybrid) model will be selected to be the data model of this development due to its flexibility and functionalities in extending the capabilities of the system in a powerful object-oriented way that, especially, also satisfyingly response the challenge and requirements of the organization.
2.4 Database Systems

In previous sections, the evolution of database systems is discussed to show a number of database systems and data models introduced over the period as well as showing a number of available data models that can be used to implement a database as intended by this research. This section is mainly focused on a discussion of major components of the database systems which are necessary for developers to understand in order to develop a powerful database to response demands of businesses and also to be able to initialize the system with the most appropriate environments. The major components of database systems are categorized into 5 components, i.e. hardware, software, users, architecture, and management system.

2.4.1 Database system hardware

Database management systems are typically installed on general-purpose computers. Computer memory is categorized into 2 classes, i.e. internal memory and external memory. Although some internal memory is permanent, e.g. ROM\textsuperscript{24} that the persistent stored data can be changed by program, and RAM\textsuperscript{25} which is a volatile memory that electrical interruption causes the loss of data. In examining the memory needs of a DBMS, the following issues must be carefully considered\textsuperscript{26}:

**Persistent character**

Data of a DBMS must have a persistent character; in other words, any data item stored in the database must remain available long after any program that is using it has completed its work. Also, data must remain intact even if the system breaks down. The persistency of data covers any point of time that a piece of data is created and stored in the system.

**Data retrieval performance**

A DBMS must access data at a relatively high rate or not slower than a given time that satisfies users. Therefore, operational time to retrieve any data item normally depends on database and data structure as well as programming procedures created in order to access any piece of stored data.

\textsuperscript{24} ROM stands for Read Only Memory; it is memory that must be written using special equipment or special procedures, and for our purposes is considered unchangeable.

\textsuperscript{25} RAM stands for Random Access Memory.

\textsuperscript{26} Cfr. [SIMOVICI 2011] pp. 2-3.
Cost of memory

Such a large quantity of data need to be stored that the storage medium must be low cost. These requirements are satisfied at the present stage of technological development only by magnetic disks. To be precise, a powerful database to handle huge amount of data does not need only an appropriate designed data structure, but also need a physical data storage to collect all data items using in application developments.

2.4.2 Database system software

Users possibly interact with database systems through query languages the running systems. A query language involves 2 major tasks, i.e. to define the data structures that serve as receptacles for the data of the database, and provide users for the speedy retrieval and modification of data. The data manipulation has its broad tasks which are data retrieval and data updates. Data retrieval involves obtaining data stored in the database that satisfies a certain specified retrieval conditions formulated by the user in a query. Whereas, data updates entirely entails data alteration, deletion and insertion. The updates are executed by programming in query languages of DBMSs that are typically written in C, Pascal, or PL/1 directly implements an algorithm for solving a problem. 27

In term of database operation, a transaction is a sequential way of representing a state change. Operations a transaction usually consists of updates, with possible retrievals. Transactions ideally have 4 properties, commonly known as ACID: 28

- **Atomicity**
  A database considers all transaction operations as an atom that represents one whole unit of the entire operations. When the database processes a transaction, the atomic transaction will be ensured by the property that the operation must be either fully completed or not executed at all.

- **Consistency**
  The Consistency property ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the transaction and after the transaction is over.

---

28 Ct. [YUSHIWEI 2009] pp. 4-10.
• **Isolation**

Ensures that all transactions are independently processed at the same time securely without any interference without maintaining the order of each transaction in process. Thus, any transaction which is not yet committed must remain completely isolated from others.

• **Durability**

Ensures that any transaction committed to the database will be stored securely made through the use of database backups and logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures that is available in its correct state.

As centrally operating database systems using transactions, developers are highly responsible for securely starting and ending at all points of the initialized transactions to logically enforce the data consistency by precisely defining the sequence of data manipulations occurring in the transaction, called interim status\(^{29}\), that moves the execution state into a consistent status, as illustrated by figure 2.10:

---

**Figure 2.10:** Transition states in database transaction\(^{30}\).

---

\(^{29}\) Ctr. [HELPsAP 2005]

\(^{30}\) The whole processes is also called “Database Logical Unit of Work (LUW)”.
The developers are also responsible to commit the each transaction in order to confirm the changes made to the data stored in the database. Otherwise, the transaction will automatically rollback the execution state back to the previous status. Thus, by using transactions, the DBMS can effectively enforce the physical integrity of the transaction and enhance any process of data manipulation performs effectively and more secure.

An intuitive demonstration is the modification of the specified ASIL level of a sourcefile which is used prioritize a software unit in term of software integrity for any secure application\textsuperscript{31}. The example transaction consists of the following simultaneous operations:

- Decreasing the ASIL level from QM to be A.
- Increasing the ASIL level from B to be A.
- Querying value of the current ASIL level.

If only the first operation is executed, then the ASIL level of the software unit will be decreased to be QM. If only the second is executed, then the ASIL level will be immediately increased to be A. In either case, the consistency of the database will be compromised if both operation are executed at the same time which directly impact the query of the current value. Once the consistency is compromise, the operation to query the current value of ASIL level might not be correct. Thus, a transaction transforms one consistent database state into another consistent database state using procedures of begin and commit each transaction separately, a property of transactions known as consistency\textsuperscript{32}.

Typically, a large number of transactions will be initialized and securely and independently processed in the database system. The transaction management, which has been mentioned as the central of the system, ensures that the execution of one transaction will not be influenced by other existing transactions, corresponding to the property of transaction isolation. After the execution of any transaction, the completed outcome state must persist in the database, referring to the durability property of transactions\textsuperscript{33}. Hence, a database system without the ACID properties, everyday occurrences such using computer systems to buy products would be difficult and the potential for inaccuracy would be huge.

\textsuperscript{31} A risk classification scheme defined by the ISO 26262.
\textsuperscript{32} Ct. [SIMOVICI 2011] pp. 4.
\textsuperscript{33} Ct. [SILBERSCHATZ et al. 2011] pp. 2-7.
2.4.3 Database system users

The community of users of a DBMS involves a variety of individuals and organizational entities. These users are precisely classified based on their roles and authorization in terms of data accessing and administrating the databases. When a database is created, the database administrator\textsuperscript{34} must immediately make decisions about the specific kinds of data to be stored in the created database, as well as constructing the access policies to be used in order to enforce any user who is going to access any parts of the database, monitoring and customizing the database, etc.

Furthermore, the community also includes either anonymous or known end users. These users have limited access rights, depending on the rights granted by the ruled policies, and they need to have only minimal technical knowledge of the database.

The particularly important categories of users of DBMSs consists of following:

- **Developers**
  The persons who are responsible to interpret the requirements and specifications provided by the system analyst in order to implement new database functionalities for the end user.

- **End users**
  The persons who directly interact with the application without technical knowledge of the internal system. They are generally allowed to insert, delete and update data in the database as well as retrieve data from the database in certain parts that they are authorized.

- **Sophisticated end users**
  Include engineers, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

- **Stand-alone users**
  Maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces\textsuperscript{35}.

\textsuperscript{34} An IT professional responsible for the installation, configuration, upgrading, administration, monitoring, maintenance, and security of databases in an organization.

\textsuperscript{35} Ct. [BANGIA 2006] pp. 10.
Furthermore, a role will be precisely assigned by the administrators for each of classified user to group together privileges or other roles, e.g. the super administrators are allowed to create and drop some tables in the database, but normal users are only allowed perform data retrieval without making change to the data, or a right to access any objects belong to other users. Normally, a single individual may have only one role in the system or possibly have more than one depending on the configuration of database security which can be configured by the responsible administrators\textsuperscript{36}.

2.5.4 Database system architecture

The architecture of a DBMS can be examined from several views, i.e. the functional architecture that identifies the main components of a DBMS, the application architecture that focuses on application uses of DBMSs, and the logical architecture that describes various levels of data abstractions. Functionally, a DBMS contains several main components which are shown in figure 2.11 that presents the relationships of main 3 architectural components, i.e. the memory manager, the query processor, and the transaction manager.

\begin{figure}
\centering
\includegraphics[width=0.7\textwidth]{figures/fig2_11.png}
\caption{Functional architecture of DBMSs.}
\end{figure}

\textsuperscript{36} Cited [SIMOVICI 2011] pp. 4.
The query processor converts a user query into instructions the DBMS can process efficiently, taking into account the current structure of the database, also referred as metadata. The memory manager obtains data from the database that satisfies queries compiled by the query processor and manages the structures that contain data, according to the DDL directives. Finally, the transaction manager ensures that the execution of possibly many transactions on the DBMS satisfies the ACID properties and also provides facilities for recovery from system and media failures.

The standard application architecture of DBMSs is based on a client/server model. This architecture is also known as two-tier architecture. The client, which can be a user or an application, initiates a connection to the server and generates a query, which is a process involves parsing, generating executable codes, and executing the optimized codes transmitted to the server. The server processes the query and then returns an answer to the client.

In large project developments, it is often necessary to create more layers of processing to effectively handle the number of clients that may vary over time. The layers of processing mean layers of software to concentrate the data manipulation of all connecting modules. In project development, data generally have broad meaning covering various data sources as well as granularity artifacts that could be database systems, or processed data elements, etc. Furthermore, the layers also act as mediums in order to organize the communication between the connecting applications and the main data repository of the development that may stand at center of the framework. This concept is known as multi-tier architecture.

The architecture conceptually includes the lowest tier which is closest to the data stored in the repository. The intermediate tiers, also known as the middleware, that may be consist of development repository, shared data storages, and may be considerably complex. In general, the tiers stand at the middle as to establish communications between the highest and the lowest tiers of the architecture. The highest tier consists of users that directly interact with the system through user interfaces or front-end applications retrieve required data from the repository. The multi-tier architecture is also known as three-tier architecture that was developed by John J. Donovan in Cambridge, Massachusetts, is illustrated by the following figure:

---

38 Ct. [KAREEM 2012] pp. 1-4
The figure 2.12 represents the 3 collaborative layers of data abstraction distinguished by the architecture as following:

**Presentation tier**

The highest tier of the application which is called user interface or system’s front-end interface, also known as Graphic User Interface (GUI). The tier displays interactive information related to such services provided by the system, e.g. searching, modifying, or committing. Furthermore, the tier is responsible to communicate with other tiers by conveying user assigned commands to the internal system as well as pulling out the results back to the user or even transmitting to all others in the network.

**Application tier**

The tier is also called middleware or logical tier. The tier is pulled out from the higher tier responsible to contain the configured detailed information of the system to describe the nature of data and how they are stored in the system, e.g. addresses of components, data lengths, etc. All application programs are connected to the database system though this layer by being aware of structure of data accessing to process and move data to other tiers.
Data tier

The tier consists of the data persistence mechanisms, i.e. database servers, shared data repository. The persistence mechanisms are fully encapsulated by the data access layer which is also responsible to expose the requested data.

By following the concept of this architecture, the data access layer also provides an Application Programming Interface (API) to its higher tier, the application tier, to effectively manage the stored data without any data dependencies on the data repository. The benefit of operating data without having dependencies is allowing developers to update or change any method in the system without being aware or affected by any higher tiers to enhance performance in order to improving system scalability and maintainability as well as optimizing costs of the development.

2.5.5 Database management systems

To operate the previously mentioned components, a centralized system is required in aggregating all internal and external related components to implement a database, which is the major point required by this research, as intended.

In general, operations made by either users enquiries or interactive system are directly conveyed to the DBMS in order to accomplish the required data manipulations. Thus, communication between the question unit and the system always have 2 directions, enquiry and response or answer, as the concept of client/server architecture.

By following the architecture, any incoming new data transmitted though editing software will be conveyed to the DBMS and get responses generated by the system to complete the entire desired operations. Precisely, because the DBMS is responsible to aggregate all connecting modules to the internal database, hence any operation must be operated though the DBMS in order to execute or even simply connecting to the main database elements39.

The overview perspective of a typical DBMS can be illustrated as the following figure:

---

The figure 2.13 shows that a database management system (DBMS) is an aggregate of data, hardware, software, and users that helps developers manage data artifacts using in software development by providing efficient and reliable methods of data retrieval to any development activities. For instance, in software unit testing, there are thousands of developing software unit being tested every day to generate test results of each used for test analysis. If a test result of each unit contains a parsed information of 1000 lines every day, then over a year, the development storage must handle huge amount of data of more than 1 million lines of test results. It is not easy to extract records satisfying certain criteria from such a set, and by current standards, this set of records is quite small. Given the current concern for lines of code of a specific sourcefile, a typical question to answer is determining the evolution of the coding averages in introductory software unit testing over the development lifetime. Therefore, it is clear that efficient data retrieval is an essential function of the system.

Most DBMSs deal with several users who try simultaneously to access several data items and, frequently, the same data item as well as querying a specific data using a given condition to retrieve data in different timestamps. For instance, an organization want to introduce a system to collect data artifacts generated from software testing activities in a centralized database which can be accessed anytime if any uses want to retrieve their required data items which may be stored in several timestamps or data layers. If an item has been updated by a user, then the same item that is being accessed at the same time by other users will take the effect.
In this section, a number of well-known database management system that can be used to implement the intended framework of this research are discussed as follows:

- **Oracle**
  One of the leading SQL database management systems that support implementations of relational data model to solve any complicated problems in term of data organization.

- **Microsoft SQL Server**
  Mainly designed for various goals and targets at different levels, e.g. web development, business enterprise applications, etc. Able to create powerful multimedia database as require that effectively support manipulation of various kinds of possibly big data.

- **MySQL**
  Broadly used for web applications which are widely written in PHP programming language and operates with the database elements for the platforms, i.e. Linux, BSD, Mac, Windows. The system is very popular and suits for project developments using well-known programming languages.

- **PostgreSQL**
  An object-relational DBMS that are widely preferred by many application developers to use as the back-end data management system for wide-ranging and critical applications throughout its history. PostgreSQL also provides lots of advanced feature set and key advantages sharing globally through open source community support. Furthermore, it beneficially serves cost optimization, as well as effective methods of system administration to maximize the system performance.\(^{40}\)

Although the introduced DBMSs are interesting to be used as a database management engine for the development, but there is also a lot of aspects still need to be considered and discussed, which will be held in next chapters.

\(^{40}\) Ct. [MOMJIAN 2012]
2.5 Framework Automatization

Basically, initialization of an operation in framework can be done manually by executing programming scripts or applications through available user interfaces by authorized users.

Therefore, since the framework is built up to mainly response the highly need of software test management that is also operated anytime in order to aggregate and interpret the development data into desired metrics, a powerfully effective mechanism that helps in handling the operations in a manner of automatic execution is required in accomplishing the major requirement of the test management framework41.

In 1913, the first series production was introduced in order to enable a higher productivity in automotive production. The science behind was based on the principle of dividing the entire work to be performed into many simple tasks before start performing. After all tasks have been done, an integration of the entire work will be placed down to create a continuous chain of automated execution in the system42.

2.5.1 Automatization in software development

Since the principle has been implemented in a wider economic section with a huge increased in its popularity in engineering development, a new term of the principle is constructed, which is called automation technology.

In software development, term of automatization characteristics of any system also refers to automating software development processes that beneficially help developers in performing the intended tasks as need.

Repeatability

Scripts can be repeated, and, unless your computer is having a particularly bad day, you can be reasonably certain that the same instructions will be executed in the same order each time the same script is run.

41 Ct. [KITCHENS 2006] a part of “The Business Argument for Investing in Test Automation”.
Reliability

An automated system can significantly help to reduce chances for human error, as well as optimize development efforts that covers a broad term of execution, e.g. correctness of system operations, timing of programming scripts execution, sequencing of collaborative tasks in the framework.

Efficiency

Generally, an automated task is executed faster than performed manually\(^43\). Therefore, automatization of tasks or even the entire framework may result in a higher cost the development more time than it saved, depending on development circumstances and situations, hence this is also a point that system administrators must take into account.

Testability

Scripted processes undergo testing throughout the development cycle, in much the same way the system code does. This greatly improves chances for successful process execution as the project progresses. Automated scripts eventually represent a mature, proven set of repeatable processes.

Versioning

Scripts are artifacts that can be placed under version control. With manual processes, the only artifacts that can be versioned and tracked are procedure documents.

Leverage

Another big benefit to automating is that developers and testers can focus on the areas where they add real value to a project; developing and testing new code and features instead of worrying about the underlying development infrastructure issues\(^44\).

The characteristics and benefits of framework automatization beneficially enhance capabilities of the framework by mainly replacing tasks that are usually performed manually by users, or must be operated in a fix sequence, by automated processes which are performed automatically and periodically with the same order. Furthermore, chances of error occurring in the automated

---

\(^{43}\) In practice, the result actually depends on how often the process is executed.

\(^{44}\) Ct. [KITCHENS 2006] a part of “The Business Argument for Investing in Test Automation”. 
framework will also be decreased by the mechanism provided by the automatization that removes all human efforts in performing execution.

2.5.2 Interface of system components

Software testing is an essential process of software development. It is related in a huge amount of development artifacts. A framework which is built to handle the artifacts must be able to respond any requirements of the development to maximize the system capabilities as well as decreasing chances of failure in the system.

As a concept of modularity, any of the connecting modules in the system will be freely managed in any action, which greatly helps in avoiding effect to other parts of the framework. Basically, each of collaborative application is interacting with each other through available interfaces in order to performing communication or interaction. To establish this connection, at least one interface must be built up at the middle of the affected parties. The following figures show 2 different approaches in interfacing a central repository:

![Alternative approaches of system interface.](image)

Figure 2.14: Alternative approaches of system interface.
Figure 2.14(a) illustrates an implementation of an individual interface connected by only single application and the repository, also called decentralized interface, while figure 2.14(b) showing implementation of a centralized interface connected commonly by both connecting applications. In comparison, this below table presents a comparison between both different implementations of the alternative interface approaches in different aspects:

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Decentralized</th>
<th>Centralized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>Differences of technology driving each communicating parties results in a higher system complexity.</td>
<td>High complexity in development, but simple in a view of system perspective.</td>
</tr>
<tr>
<td>Administration</td>
<td>In heterogeneous systems, updating any separated interface significantly consumes high effort and costs.</td>
<td>Requires a high knowledge and experience in maintenance. Anyway, the control can be simply performed at a single point.</td>
</tr>
<tr>
<td>Performance</td>
<td>Distribution of payloads running on communication channels.</td>
<td>All communications are centralized without unnecessary switching.</td>
</tr>
<tr>
<td>Scalability</td>
<td>Flexible when expanding the communication channels over the interface to support the system.</td>
<td>Growing up of the system directly effects the common interface that must be immediately updated.</td>
</tr>
<tr>
<td>Security</td>
<td>Defects arising on a fully separated module is simpler to be detected and located, and fixed.</td>
<td>Single point of failure; an arising defect possibly impacts the entire communication channels.</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of decentralized and centralized interface approaches.

According to the table, each approach provides its advantages in different views of development. However, to select the approach to be implemented in the system, term of ease of development must also be considered. In general, each module in the system should be independent and separate from the others to facilitate processes of system maintenance, as well as minimizing the complexity of a single module. Once the complexity in modules is minimized, any additional processes can be

---

45 In practice, ease of administration is generally relies on the point of the interface in the system.
performed much simpler and more effective, e.g. framework expansion, process configuration, and process of documentation. According to the implementation of a common interface, discussed in the figure, the infrastructure of the framework, that is fully independent from software testing levels, can be widely expanded to support more participating applications.

### 2.5.3 Automatization management

Nowadays, the term of automatization is interpreted broadly depending on the use of the principle in any specific part of software development. Continuous Integration\(^{46}\) is broadly used as a concept of software testing which based on testing regularly and continuously.

A basis of the principle is dividing the whole operation into several chunks of task, then assigns each task to a local environment. Each task will be executed and chained by others in a manner of pipelining execution, as following figure:

![Pipelining execution in continuous integration principle](image)

**Figure 2.15:** Pipelining execution in continuous integration principle\(^{47}\).

---

\(^{46}\) A practice, in software engineering, of regularly merging all developer working copies with a shared mainline.

\(^{47}\) Also called a continuous delivery process diagram.
As showing in the figure 2.15, the pipeline provides various benefits developers in developing and manipulating data artifacts throughout the framework. Therefore, several critical points in driving software development using a pipeline need to also be considered, as follows:\(^\text{48}\):

**Data dependency**

Any shared data artifact that can be simultaneously manipulated by multiple processes must be securely handled. In case of update, the meaning or contents of artifact might severely effects the further execution in the same pipeline.

**Control dependency**

Pipelining is based on dividing tasks and sequencing them in the mainstream of pipeline. All possibly conditional branches that can be occurred during the execution must be precisely and primarily set to avoid uncertainty to instruction sequencing.

**System performance**

Operating a huge execution pipeline requires a huge amount of computational resource in order to accomplish the tasks as well as allocating environments for other processes running simultaneously in background.

**Risk management**

Collaborative mechanisms for dynamic hazard resolution must be available. Thus, any risk arising at runtime must be detected and handled effectively.

In summary, the continuous integration can be used as a conceptual driver to establish an automatization of the framework, or even manual operation, of the processes of the framework. By following the concept, the framework will be able to start an execution of an individual configured task periodically and then continuously perform the next scheduled tasks in automatic, also able to set a specific execution time that greatly helps development in decreasing efforts on monitoring the system. Hence, the automatization involving an automation and periodic executions in the collaborative framework can be completely fulfilled by this idea of this concept.

\(^{48}\) Ct. [HUMBLE et al. 2010]
Chapter 3
Discussion

In this section, a conceptual framework of software test management for test metrics is constructed using the current development circumstances, scientific challenges, and collected knowledge previously mentioned in details in previous chapters. At first, conceptualization of the entire system will be discussed to build up a concrete architecture and environmental perspectives of the aimed framework. Also each of essential and appropriate components to be implemented in this development will also be selected.

3.1 Framework Conceptualization

According to the situations along with the on-going scientific challenges in the current framework of software unit testing, a new mechanism of big data handling to support all of development artifacts generated in software testing becomes a great alternative idea. As mentioned in the figure 1.2, the current system will be partially transformed by placing a centralized database at the middle that connects other modules through the APIs. Jenkins CI is used to realize the framework automatization. This following figure shows an overview of conceptual framework:

Figure 3.1: An overview of conceptual framework.
The figure 3.1 presents the connectivity of all participating modules in the framework made to a centralized database. A number of code editors and tools for software unit testing get a sandbox from the CMS to proceed their tasks and then perform a check in once to return the outcomes as a new version. A tabular user interface is connected to the database to be a friendly front-end user interface to interact with the current stored data. After that, the modifications will be synchronized to maintain the consistency of data. A metrics generator is used to programmatically interpret test data retrieved from the database into various kinds of high-quality test metrics, e.g. graphics, office documents, etc., depending on the requirements of organizations.

Nowadays, a variety of data formats becomes a big challenge in data processing, e.g. the current SWUT tools generate their own artifacts in different formats. In this situation, increasing in number of connecting modules in the system leads the system more complex. APIs become a typical solution to handle such of those challenges. Technically, an interface is mainly responsible to be a glue at the middle of communication by transforming all of incoming data into a desired and specified format. A number of interfaces or APIs are used in developing the framework, as the figure. Each of them is used as the following purposes:

**Parsers**

Each subversion of raw data stored in TortoiseSVN consists of several formats and contents, e.g. header files (*.h), include files (*.inl), metrics files (*.met), source files (*.c, *.cpp), etc., that are generated by different software testing tools, e.g. Cantata and QA-C/QA-C++. To transfer the data into the database, the raw data must be parsed to refine only desired contents, illustrated as following figure:

![Figure 3.2: An overview process of raw data parsing using decentralized parsers.](image)
Excel API

In a process of generating a front-end user interface in tabular form, Microsoft Excel is chosen due to its friendliness that most users familiar with. The process requires a precise data, retrieved from database, to be filled in worksheets. Thus, an API of Microsoft Excel is required to get and place each data into specific cells as well as format its appearance and properties, e.g. position, color, cell protection, etc. An overview of the interface is generated is illustrated as following figure:

Figure 3.3: An overview process of generating a user interface using Microsoft Excel.

Metrics API

As the major process of the framework, a unit of data can be interpreted into various kinds of test metrics, e.g. graphics, office documents, browser-based files, etc. A metrics generator is used to programmatically transform the data into specific metrics. A unit of data retrieved from database will be independently interpreted into several different forms of test metrics that developers have specified in programming scripts. The interpretation is done as the following figure:

Figure 3.4: An overview process of interpreting data though a centralized interface.
Database API

A centralized database is the most essential component of this research used to store all development artifacts to serve for any further processes that continuously request the data. Refer to one of the scientific challenges which is also an important requirement and specification of this framework, the database must be developed independently from any participating modules, furthermore, and projects which the database is integrated in. A database-application programming interface is used in handling any manipulation of data in both directions between requesting processes and itself. Thus, any communication between the database and participating parties will be established through the interface, as the figure:

![Database API Diagram]

Figure 3.5: An overview process of manipulating data through database interface.

As illustrated by the above figures, both approaches of decentralized and centralized interface are considered and implemented in the framework. Refer to the table 2.1, a decentralization definitely suits for development of several heterogeneous participating parties, e.g. a number of different software unit test tools, that also offer developers higher ease of administration. Whereas, the second approach is more effective for development of a common interface with a number of parties that resemble like implementing in a communication between database and other connecting modules.
3.2 Definition of Collaboration

Since a conceptualization of the software test management framework for test metrics has been constructed in the previous section, as well as a number of interfaces was intuitively illustrated to present an overview of each underlying process driven in the framework, every granularity of those components must be fulfilled by their precisely defined roles, properties, behaviors, etc., referring to the figure 2.1 that presents a conceptual collaborative framework in term of definition of each element inside the framework.

In this section, each element in the conceptual collaborative framework will be concretely defined and presented in detail to fulfill the collaboration of the framework based on three-tier architecture by starting at the lowest layer of the framework, which is a database, and move up to data aggregation and integration that are the major tasks of the middle layer, then finalize the highest layer that establishes an interaction between a user and the system. The following table presents details of each layers in this conceptual collaborative framework:

<table>
<thead>
<tr>
<th>Views</th>
<th>Data Tier</th>
<th>Application Tier</th>
<th>Presentation Tier</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Database system</td>
<td>Raw data parsing,</td>
<td>MS Excel interface,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interface generation,</td>
<td>High quality test metrics</td>
</tr>
<tr>
<td>Component</td>
<td>Data to be stored,</td>
<td>Raw data parsers,</td>
<td>MS Excel workbook,</td>
</tr>
<tr>
<td></td>
<td>Database model,</td>
<td>Database interface,</td>
<td>Archive of artifacts,</td>
</tr>
<tr>
<td></td>
<td>Database structure,</td>
<td>MS Excel generator,</td>
<td>Web browsers.</td>
</tr>
<tr>
<td></td>
<td>Database system</td>
<td>Metrics generator</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>Data collector, Data</td>
<td>Data transformer,</td>
<td>Data visualizer, Data</td>
</tr>
<tr>
<td></td>
<td>manipulator</td>
<td>Data conveyer</td>
<td>synchronizer</td>
</tr>
<tr>
<td>Behavior</td>
<td>Electronically stores</td>
<td>Transforming data</td>
<td>Presents current state</td>
</tr>
<tr>
<td></td>
<td>development data</td>
<td>transferred between</td>
<td>of the system and</td>
</tr>
<tr>
<td></td>
<td>and provides to other</td>
<td>participating parties</td>
<td>allows users to</td>
</tr>
<tr>
<td></td>
<td>collaborative processes.</td>
<td>into specific formats.</td>
<td>manipulate artifacts.</td>
</tr>
<tr>
<td>Function</td>
<td>Based on CRUD as well as serving the stored data to other processes.</td>
<td>Data aggregation and interpretation to generate test metrics.</td>
<td>Displays data on the MS Excel interface and updates changes to the database.</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>Resource</td>
<td>Development artifacts, Database engines, Relevant development tools and packages.</td>
<td>Scripts and Application Programming Interfaces (APIs) of all interactive modules.</td>
<td>MS Excel application, Archive for artifacts, Web browsers, Graphic generator.</td>
</tr>
<tr>
<td>Process</td>
<td>Imports, categorizes, stores, and exports data through the API.</td>
<td>Retrieves data from source, format using APIs, and transfer to the interacting parties.</td>
<td>Imports data to be presented from the APIs and display on user-friendly channels.</td>
</tr>
<tr>
<td>Joint action</td>
<td>Manipulation of data between applications connected to database.</td>
<td>Transformation and formation of different data formats.</td>
<td>Update current data and synchronize with other applications.</td>
</tr>
<tr>
<td>Communication</td>
<td>Connect and manipulate data through the API.</td>
<td>Connect and format heterogeneous data.</td>
<td>Synchronize data states through the API.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Participating modules are connected via API.</td>
<td>Participating modules are connected via API.</td>
<td>Participating modules are connected via API.</td>
</tr>
<tr>
<td>Interface</td>
<td>Back-end interface, Command prompt</td>
<td>Application Programming Interfaces (APIs)</td>
<td>MS Excel interface</td>
</tr>
</tbody>
</table>

Table 3.1: Definition of the framework based on three-tier architecture.
As summarizing in the table, all of main system modules, i.e. test management database, raw data parsers, Microsoft Excel user interface generator, and test metrics generator, have their own properties in numbers of view. Basically, a communication between each pair of participating parties is established through specific Application Programming Interfaces (APIs) in order to maintain the connection as well as statically transforming heterogeneous data transmitted across different formats. Furthermore, as known, the database takes a role of electronically storing development artifacts that will be dynamically requested by other collaborative processes executing in participating modules, e.g. the Microsoft Excel user interface requests the stored data to display on worksheets, or the metrics generator retrieves the data and programmatically interpret into various kinds of high quality test metrics.

Since the conceptual collaborative framework, mentioned in the research, is precisely fulfilled by defining numbers of view to each system component, a suitable development tools that will be used in developing the framework must be properly considered and chosen to realize the conceptual framework to be a working system as expected, in both perspective of system architecture and also desired system specifications and behaviors.
3.3 Database Model Selection

As the purpose of this research, a centralized database will be developed based on basis of database-centric architecture by visualizing the system into several collaborative layers referring to the three-tier architecture. The drive the system with success, a database model which is the most essential part of the database of the test management framework must be properly selected as a suitable database model for the development.

3.3.1 Comparison of alternative models

A comparison of three alternative database models, i.e. relational model, object model, and object-relational or hybrid model, will be held to precisely present their differences strengths and weaknesses in a number of aspect based on the technical requirements, as below table:

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Relational</th>
<th>Object</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query facilities</td>
<td>Separately access each table to get the stored data using SQL.</td>
<td>Pointing at parents is able to get all data of their children.</td>
<td>Querying can be done either entirely or individually.</td>
</tr>
<tr>
<td>Data visualization</td>
<td>Tabular form provides ease of categorization and binding relations.</td>
<td>Real-word data model intuitively represents the real structure.</td>
<td>Based on tabular representation of the relation model.</td>
</tr>
<tr>
<td>Schema evolution</td>
<td>Separately handle each component in schema. Updating in a single table will be secure and effective.</td>
<td>An inheritance makes updating in parents consequently affect all of their children.</td>
<td>Evolution of inherit tables are based on object model. Rest of tables is individually handled and evolved.</td>
</tr>
</tbody>
</table>
### Table 3.2: Comparison of alternative database model

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Established by mean of referencing keys.</th>
<th>Logical pointers are assigned to establish object relationships.</th>
<th>Based on the concept of referencing keys, in relational model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data types</td>
<td>Only built-in data types can be used.</td>
<td>Customization of user specific data types.</td>
<td>Customization of user specific data types.</td>
</tr>
<tr>
<td>Knowledge</td>
<td>Simple model that basically requires experiences and knowledge in SQL.</td>
<td>High experienced developers in a deep knowledge of object-oriented are required.</td>
<td>Deep knowledge in object-oriented programming and SQL is required.</td>
</tr>
<tr>
<td>Supporting tools</td>
<td>Supported by several database engines and relevant tools, e.g. Oracle, MySQL, PostgreSQL, and MS SQL Server.</td>
<td>Supported by several languages for object-oriented programming, e.g. C++, Python, etc.</td>
<td>Supported by numbers of database engine that facilitate operating the database using object-oriented programming languages.</td>
</tr>
<tr>
<td>Independency</td>
<td>The internal structure is independently constructed from type of projects. But, still having limitations of built-in data types.</td>
<td>The internal structure is independently constructed with precise behaviors that are programmatically controlled.</td>
<td>The internal structure is flexible and fully independent from type of projects. UDTs can be implemented to enhance functionality.</td>
</tr>
</tbody>
</table>

The comparison of 3 alternative database model presents various advantages and disadvantages. Therefore, the selection of the most suitable model to be used as a data model of this database-centric framework will be based on numbers of technical requirements derived by the scientific challenges. A discussion in selecting the model will be held in next section.

---

49 Ctr. [ROBIE/BARTELS 2012] pp. 1-15
3.3.2 Compromising database models

According to the comparison, each of alternative database model offers its strong benefits as well as showing the limitations in developing a database in different aspects. The relational model greatly offers a simplicity of database development based on a basis of related physical tabular storages driven by SQL and several supporting tools. Anyway, the technical requirements analyzed previously require a possibility of creating specific user-defined data types, which is not offered by the use of this model. In object model, several flexible functionality are offered that beneficially provide developers more possibility in developing databases, e.g. customization of data types, and inheritance of object data based on object-oriented approach. Nevertheless, the functionality and powerful internal mechanisms are entirely controlled by numbers of logical pointer which are assigned programmatically. Thus, a database developed with the object model hugely requires a high knowledge of developers in order to develop and maintain the system in any case.

As detailed the research, the object-relational or hybrid model has been introduced mainly in bridging the impedance mismatches of conventional models, i.e. relation and object models. The object-relation model completely accounts all of the technical requirements with a possibility of supporting a variety of development data by mean of creating user-defined types. Furthermore, the model physically organizes data in categorized tables that greatly help developers in term of data management. As well as being extended by object-oriented approach, the model is able to organize the stored data also in a real-world model to represent the actual structure of project development. Hence, with those compromises, the object-relational database model will be selected to build up the software test management database for test metrics to handle a huge amount of data in software testing.
3.4 Database System Selection

In this section, a database management system will be discussed and selected to drive the object-relational database model. A numbers of necessary criteria is used in comparing a number of well-known database management systems to compromise all of them in different views of specification as well as supports for further implementations.

3.4.1 Comparison of alternative database systems

Selecting a right database system must be done carefully by comparing using specific based on a realization of the selected database model. The below table presents a comparison of 4 DBMSs:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Oracle 11g</th>
<th>Microsoft SQL Server 2014</th>
<th>MySQL 5.6</th>
<th>PostgreSQL 9.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>Freeware</td>
<td>Commercial</td>
<td>Freeware</td>
<td>Freeware</td>
</tr>
<tr>
<td>ACID compliant</td>
<td>Fully</td>
<td>Partially</td>
<td>Partially</td>
<td>Fully</td>
</tr>
<tr>
<td>Support hybrid model</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support UDTs</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Support data partitioning</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Programming with arrays</td>
<td>Support</td>
<td>No</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td>EU character encoding</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td>Concurrency control</td>
<td>Provided</td>
<td>Provided</td>
<td>Provided</td>
<td>Provided</td>
</tr>
<tr>
<td>Max. database size</td>
<td>11 GB</td>
<td>524,272 TB</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Main query language</td>
<td>PL/SQL</td>
<td>T-SQL</td>
<td>SQL</td>
<td>PL/pgSQL</td>
</tr>
<tr>
<td>Implementation language</td>
<td>C/C++</td>
<td>C/C++</td>
<td>C/C++</td>
<td>Native C</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of alternative database management systems\(^50\).

\(^50\) Ct. [WIKIPEDIA 2015]
3.4.2 Compromising database management systems

According to the comparison presenting above, Oracle 11g Express Edition, MySQL 5.6, and PostgreSQL 9.3 are distributed as freeware, whereas Microsoft SQL Server 2014 is licensed. Regarding a restriction in selecting development tools issued by this research, any licensed software or development tools will not be considered. Thus, only the three mentioned freeware will be considered and discussing in further aspects.

In term of ACID, a concurrency control is more effectively supported by Oracle and PostgreSQL that both provide a strong mechanism in organizing the processed data by mean of transactions that securely controlled by a transaction manager, also called a transaction monitor.

With a higher flexibility in creating user-defined composite types using available built-in data types that is only supported by Oracle and PostgreSQL, fully achieving all of the development purposes as well as supporting the technical challenges can be more completely satisfied. Furthermore, a huge amount of test data will be continuously generated overtime that definitely requires a huge database size to support them. In this case, only MySQL and PostgreSQL are capable of storing all of the data with lesser limitations.
3.5 Development Tools Selection

In this section, the conceptual framework will be completely fulfilled by selecting the suitable environment for it. All essential development tools, i.e. tools for creating the parsers, interfaces (APIs), programming languages, and necessary libraries, will be selected based on the sub modules comprising the conceptual framework.

3.5.1 Environment of the database

As the selections have been done since previous sections, an object-relational database model and PostgreSQL will be used to develop the database that is basically driven by SQL and PL/pgSQL. In a part of the database interface or database API, a suitable environment must also be set up.

According to a number of on-going projects currently running, most of them are written in Python 2.7. This circumstance leads the development of the framework focuses on developing also in Python to provide a flexibility in further integrations. Thus, the API will be considered by selecting a tools that fully matched between the PostgreSQL 9.3 and Python 2.7.

The below table shows a number of Python drivers can be integrated with PostgreSQL:

<table>
<thead>
<tr>
<th>Software</th>
<th>License</th>
<th>Platforms</th>
<th>Python Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psycopg</td>
<td>LGPL</td>
<td>Unix, Win32</td>
<td>2.4 - 3.2</td>
</tr>
<tr>
<td>PyGreSQL</td>
<td>BSD</td>
<td>Unix, Win32</td>
<td>2.3 - 2.6</td>
</tr>
<tr>
<td>py-postgresql</td>
<td>BSD</td>
<td>Any (pure Python)</td>
<td>3.0+</td>
</tr>
<tr>
<td>bpgsql</td>
<td>LGPL</td>
<td>Any (pure Python)</td>
<td>2.3 - 2.6</td>
</tr>
<tr>
<td>pg8000</td>
<td>BSD</td>
<td>Any (pure Python)</td>
<td>2.5+ / 3.0+</td>
</tr>
</tbody>
</table>

Table 3.4. Alternative PostgreSQL-API driven by Python\textsuperscript{51}.

According to the table, the only one Python driver which is supported by version 2.7 and distributed as a freeware (with LGPL) is Psycopg. Thus, it will be used as the database API.

\textsuperscript{51} Python-PostgreSQL wiki: https://wiki.postgresql.org/wiki/Python
3.5.2 Environment of raw data parsers

Regarding the current framework of software unit testing, all of the test data are generated in text file format. Generally, accessing the contents containing in text files can be done by a read/write built-in function provided by Python. The parsers are also visualized as a single interface standing at the middle between the raw test data and the database API, as figure 3.1. Thus, Python which is the language that will be used in developing the framework will also be used to create the parsers to transform the raw test data generated in various formats before transferring to the database.

3.5.3 Environment of front-end user interface

Visualizing any related data stored in the database is widely made through a tabular document, e.g. Microsoft Excel, CSV, or though web browsers. In this development, Microsoft Excel will be selected by its known features that most of users is familiar with. By using the Microsoft Excel as front-end user interface, any visualization, modification, as well as interpretation using its provided built-in functions can be executed effectively.

In term of programming, a number of Python packages use to drive a development of the Microsoft Excel is broadly provided, as following table:

<table>
<thead>
<tr>
<th>Packages</th>
<th>File Formats</th>
<th>Python Versions</th>
<th>MS Excel Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>win32com</td>
<td>All formats in MS Excel</td>
<td>2.2+</td>
<td>Yes</td>
</tr>
<tr>
<td>openpyxl</td>
<td>xlsx / xlsxm / slsx / xltm</td>
<td>2.6 - 3.4</td>
<td>No</td>
</tr>
<tr>
<td>xlsxwriter</td>
<td>xlsx</td>
<td>2.5 - 3.4</td>
<td>No</td>
</tr>
<tr>
<td>xlrdd</td>
<td>(Read) xls / xlsx</td>
<td>2.6 / 2.7 / 3.2+</td>
<td>No</td>
</tr>
<tr>
<td>xltwt</td>
<td>(Write) Excel 95 - 2003</td>
<td>2.6 / 2.7 / 3.3+</td>
<td>No</td>
</tr>
<tr>
<td>xlutils</td>
<td>xlrdd + xltwt</td>
<td>2.6 / 2.7 / 3.2+</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.5. Alternative Python packages for Microsoft Excel.

As discussing by the criteria, all of them are supported by Python 2.7. But, the main features that will fully meet the technical requirements of creating a front-end interface is the supported file
formats. In this case, win32com is most suitable with its functionality in handling all of Excel file formats. Furthermore, by using an instance of Microsoft Excel application, developers will be able to take control over all of built-in functions of the application that greatly helps in creating a better documentation as a friendly user interface.

### 3.5.4 Environment of test metrics generator

Currently, the required types of test metrics that will be programmatically generated consist of graphic metrics, tabular worksheets, and possibly in other formats. Referring to the previous discussion, win32com will be used to generate various test metrics in required formats. Whereas other formats will be generated by built-in functions provided by Python.

In a part of graphic generation, additional graphic libraries for Python are necessary. Currently, several on-projects are developed by using a powerful plotting package, called Matplotlib 1.4.3. Thus, this package will be used to generate the graphic metrics to help develops in creating better test metrics in term of visualization as well as optimizing development efforts.

### 3.5.5 Summarization of the environment

In conclusion, the selected development tools that have been selected and that will be used in realizing the conceptual framework are summarized as below table:

<table>
<thead>
<tr>
<th>Components</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code editors</td>
<td>Independent, but using Eclipses as current.</td>
</tr>
<tr>
<td>Software testing tools</td>
<td>Independent, but using QA-C/QA-C++ and Cantata at current.</td>
</tr>
<tr>
<td>Subversion control</td>
<td>TortoiseSVN</td>
</tr>
<tr>
<td>Automatization driver</td>
<td>Jenkins CI</td>
</tr>
<tr>
<td>Database</td>
<td>PostgreSQL 9.3 / SQL / Psycopg 2.6 / Python 2.7</td>
</tr>
<tr>
<td>Raw data parsers</td>
<td>Python 2.7</td>
</tr>
</tbody>
</table>

---

52 Matplotlib—Python plotting: documentation http://matplotlib.org/
Table 3.6. Development tools for framework development.

Since all of suitable environments have been properly considered to implement the selected theoretical approaches that have been discussed and concluded since previous chapters, implementations of the approaches can be started to build up a powerful database-centric framework to support software testing activities.
Chapter 4
Implementation

In this chapter, the conceptual database-centric test management framework for test metrics will be implemented to realize the theories collected during the researches since the early period of this thesis. Development of each underlying component in the framework will be deeply explained to present the instructions to build up the complete module of the component as well as completing the entire collaborative framework.

Prerequisite Requirements:

- The framework is implemented on Windows 7 Professional (64-bit).
- The main development language which is Python 2.7 must be installed.

4.1 Environment Setup

The first process before going into the details of implementation is a preparation of the environment. An idea in developing this project is to make the framework portable that means the framework can be relocated and used to support any project as needed. Anyway, a concrete development structure, aka. Development workspace, must be precisely defined and clearly understood by developers. The workspace is responsible to categorize and collect all developing stuffs in a proper way.

This below is the structure that is used in organizing the artifacts using in the developing of the framework:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRVC_MainSW_Review</td>
<td>10/03/2015 14:32</td>
<td>File folder</td>
</tr>
<tr>
<td>SRVC_MainSW_UT</td>
<td>13/03/2015 16:17</td>
<td>File folder</td>
</tr>
<tr>
<td>SRVC_MainSW_UT-DB</td>
<td>13/03/2015 16:18</td>
<td>File folder</td>
</tr>
<tr>
<td>SRVC_MainSW_UT-Jenkins</td>
<td>10/03/2015 14:33</td>
<td>File folder</td>
</tr>
</tbody>
</table>
The figures present the structure of this development, the figure (a) shows the main workspace of the framework containing a folder of test data checked out from the SVN, a folder of Jenkins CI that will be implemented in the final process, and a folder that will be used in development of the database. The figure (b) (under: SRVC_MainSW_UT-DB) shows the workspace where each category is placed whereas (c) shows the internal structure under folder Scripts that is used to collect all of programming scripts using in the implementation of the research.

Due to a high complexity of development workspace, a global configure file is created (under: \SRVC_MainSW_UT-DB\Scripts\Global) to collect all relevant directories as well as defining the configuration of the framework, as the figure below:
In the level of data administration, any piece of data flowing in the framework will be handled in object-oriented way that provides a lot of advantages in manipulating the data in a systematic manner. To implement the idea, a Database Object Model (DOM) is structurally defined, under the same directory that will be used in creating a new data. The following figure show the contents presenting how the data object model is defined:

As showing in the figure, two object classes are defined. The class “cls_data_object” is used to handle the data existing in the system, e.g. transferring between the database and raw data parsers.
or the user interface. The class “cls_interface”, which is a sub class of the above class, will be used to handle objects of the user interface in header section.

The classes are used in the whole life of data, i.e. data defining, handling, and manipulating. Each attribute in the classes is detailed in below table:

<table>
<thead>
<tr>
<th>Classes</th>
<th>Attributes</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cls_data_object</td>
<td>table</td>
<td>Collects name of the table where the data is stored in the database.</td>
</tr>
<tr>
<td>cls_data_object</td>
<td>attribute</td>
<td>Collects name of the column where the data is stored in the table.</td>
</tr>
<tr>
<td>cls_data_object</td>
<td>value</td>
<td>Collects the current value of the data.</td>
</tr>
<tr>
<td>cls_data_object</td>
<td>type</td>
<td>Collect type of the data corresponding to type specified in the SQL command used when creating database tables, e.g. text, integer, numeric, etc.</td>
</tr>
<tr>
<td>cls_data_object</td>
<td>interface</td>
<td>An instance of the class cls_interface.</td>
</tr>
<tr>
<td>cls_interface</td>
<td>label</td>
<td>Collects a header text where the data is written in the user interface.</td>
</tr>
<tr>
<td>cls_interface</td>
<td>order</td>
<td>Collects the index of a column where the data is written in the user interface.</td>
</tr>
<tr>
<td>cls_interface</td>
<td>locked</td>
<td>Collects the status of cell protection used to block an access or modification of data made by unauthorized users.</td>
</tr>
</tbody>
</table>

Table 4.1: Descriptions of class attributes defined in data object model.
4.2 Development of the Database

The development will be started by constructing and realizing the implementation of the database first. These following instructions present how the database is developed.

4.2.1 Setting up a database using PostgreSQL 9.3

At first, download the installer of the DBMS that supported by the 64-bit environment. Then begin the installation as usual (under: /SWUT-DB). When a dialogue asking the locale to be used by the new database cluster is shown, please select the locale from default to “C” to support the further implementations, as showing in the following figure:

![Figure 4.4: Selecting the locale of PostgreSQL to support European characters.](image)

Locale support refers to an application respecting cultural preferences regarding alphabets, sorting, number formatting, etc. PostgreSQL uses the standard ISO C and POSIX locale facilities provided by the server operating system. Hence, the ISO C must be explicitly selected to support a variety of European characters operated in the database.

---

53 Download Postgresql-9.3.6-2-windows.exe from http://www.postgresql.org/download/windows/
54 Explicitly set the locale that further supports Latin-1 character encoding. See ISO/IEC 8859-1:1998.
After the installation is completed, then access that the interface of PostgreSQL, called pgAdminIII, with the set password. In this step, a new database that will be used as the main workspace of the test management framework must be created and named as “SWUT-DB”, using the following SQL command:

```
CREATE DATABASE "SWUT-DB"
    WITH
        OWNER        = postgres
        ENCODING     = 'LATIN1'
        TEMPLATE     = template0
        TABLESPACE   = pg_default
        LC_COLLATE   = 'C'
        LC_CTYPE     = 'C'
        CONNECTION LIMIT = -1;
```

The command assigns the encoding of the new database in Latin1 that effectively supports all of the special European characters, as well as explicitly selects types of locale corresponding the one selected in the previous step. The result shows as the figure:

![Figure 4.5: A new database created for development of the framework.](image)

Up to this point, the new database, in the figure 4.5, is now ready to be implemented in further steps with a selected locale to support a variety of data in any character format.
4.2.2 Implementing the database structure

Since the SWUT-DB is ready, a database structure containing tables and relevant components will be created to fulfill the schema. According to the technical requirements, the structure must provide a possibility to track any stored data as well as organizing data in the way that facilitate all of further processes, e.g. maintenance, evolution, and integration. Refer to the research and discussions, the object-relational model will be used to handle the big data of software testing by means of object-oriented approaches that provide several techniques in handling object data.

These procedures present the realization of object-oriented approaches in developing the database of the software test management framework:

1) Create user-defined types

A user-defined type, aka. Composite type, is a structure that collects a number a built-in data types in as a single type. In this development, information of several embedded system components are commonly described by the same structure, i.e. component name, supplier, and the owner of the component. In this case, 2 UDTs are required to handle those information in a common way.

```
type_component  : Handle basic information of each component
type_file        : Handle specific information of each file using in development
```

Thus, the created types will be implemented to handle each embedded system component and development file in the system.

2) Create database tables

Since the data types have been created and available with support of object-oriented techniques provided by PostgreSQL. Database tables will be constructed to satisfy the requirements in handling test data in several categories.

The following figure presents the relationship of tables in the SWUT-DB that reflects the real structure of current software development in the organization:

---

55 See Appendix C.1: SQL statements to create user-defined types
Figure 4.6: The structure of database implemented using user-defined types.

The user-defined types are used as framed in the figure 4.6 to create a number of tables that contains the same table structure. Each table is responsible to store a huge test data of each sourcefile in a specific category, e.g. development information such as the project or baseline that the sourcefile belongs to, and test data of the sourcefile, as the table below:

<table>
<thead>
<tr>
<th>Tables</th>
<th>Details</th>
<th>Implemented Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Information of be the project.</td>
<td>User-defined type</td>
</tr>
<tr>
<td>Baseline</td>
<td>Information of be the baseline.</td>
<td>User-defined type</td>
</tr>
<tr>
<td>Processor</td>
<td>Information of be the processor.</td>
<td>User-defined type</td>
</tr>
<tr>
<td>Subsystem</td>
<td>Information of be the subsystem.</td>
<td>User-defined type</td>
</tr>
</tbody>
</table>
Chapter 4. Implementation

<table>
<thead>
<tr>
<th>AtomicComponent</th>
<th>Information of be the atomic component.</th>
<th>User-defined type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sourcefile</td>
<td>Information of be the sourcefile.</td>
<td>User-defined type, Create table from type</td>
</tr>
<tr>
<td>StaticCodeAnalysis</td>
<td>Result of static code analysis (Cantata)</td>
<td>-</td>
</tr>
<tr>
<td>CodeReview</td>
<td>Result of code review (QA-C/QA-C++)</td>
<td>-</td>
</tr>
<tr>
<td>UnitTest</td>
<td>Information of be the unit test.</td>
<td>-</td>
</tr>
<tr>
<td>TestScript</td>
<td>Information of the test script used to test the sourcefile.</td>
<td>User-defined type, Inherit from sourcefile</td>
</tr>
<tr>
<td>TestResult</td>
<td>Test result of the sourcefile tested by the above test script.</td>
<td>User-defined type, Inherit from sourcefile</td>
</tr>
<tr>
<td>Mapping</td>
<td>Collects referenced indexes of each table during an execution.</td>
<td>-</td>
</tr>
<tr>
<td>Execution</td>
<td>Collects information of each execution used for data tracking.</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2: Information of database tables used in handling development data.

PostgreSQL allows user-defined types to be used in many of the same ways that simple types can be used. As showing in the table, the third column, not only the technique of user-defined type which is implemented in the database to increase numbers of available data types in the database, but also a technique of table creation using a data type that allows tables can be declared to be of the created user-defined type.

As showing in the figure 4.6, several tables, e.g. Project, Baseline, and Sourcefile, are created using the technique by referencing their structures to the created user-defined types, i.e. type_component and type_file. Then, the structure and all properties of those tables will be the same as the
referenced types. The following SQL command is used to implement the technique in creating tables of Project and Sourcefile by declaring to be of the created user-defined types:

```sql
-- create table "atomiccomponent"
CREATE TABLE atomiccomponent OF type_component (  
    CONSTRAINT pk_atomiccomponent PRIMARY KEY (component_id)
);

-- create table "sourcefile"
CREATE TABLE sourcefile OF type_file (  
    CONSTRAINT pk_sourcefile PRIMARY KEY (file_id)
);
```

Furthermore, the inheritance provided by object-oriented approach is also implemented in the creation of tables. Table TestScript and table TestResult are the outcomes of implementing this technique by inheriting table Sourcefile\(^{56}\).

As the behaviors of inheritance, if any modification is made to the structure of inherited table, which is `Sourcefile` in this case, the structures of all inheriting tables will also be updated automatically. As well as altering the user-defined type that is used to create a table, e.g. `type_component` that is commonly used to create table `Project` and `Baseline`, will also consequently effect the inheriting tables.

Thus, by implementing the inheritance and user-defined types in developing the database, the effort spent in development and maintenance can be significantly reduced. Furthermore, the integrity of the database structure will be improved and ensured by the mechanisms of the DBMS that provides several object-oriented techniques for the development.

### 3) Create table sequence

The sequence is responsible to create unique values for all of primary keys. Once a generated unique number has been assigned to a primary key, the sequence will be incremented for a new value and returns it. There is no transaction isolation in the mechanism of automatic sequence, so several transactions cannot get the same value simultaneously, as well as without a possibility to roll back in any case.

\(^{56}\) See Appendix C.2: SQL statements to inherit existing tables.
In this development, a sequence is individually created and assigned for each table in order to generate a unique number for the primary key of each record. Thus, when a new record is created, a sequence will immediately generate a unique number and serve it to the record to be used as the primary key. This following SQL command is used to create a number of sequences for the existing tables, i.e. table Mapping and table Execution:

```
-- create sequence "sq_mapping"
CREATE SEQUENCE sq_mapping MINVALUE 1;
ALTER TABLE mapping ALTER COLUMN mapping_id
SET DEFAULT nextval('sq_mapping');

-- create sequence "sq_execution"
CREATE SEQUENCE sq_execution MINVALUE 1;
ALTER TABLE execution ALTER COLUMN execution_id
SET DEFAULT nextval('sq_execution');
```

The command will create sequences starting at 1, and then alter the tables in the column of primary key by setting the default that every increment of the primary will directly controlled by a specific auto-incremented sequence.

4) Develop database API

The API, also called interface or wrapper, is a main gate to access the database to manipulate the stored data. The interface must be support all of possible operations in manipulating the stored data that any connecting application just simply passes required data to the interface to create a complete SQL statement to be executed in the database.

The database API helps an application that is communicating to the database able to perform a specific operation in the database. A number of required arguments must be passed into the parameters of a specific function. The function concatenates the given arguments to create a complete SQL statement, then connect to the database via Psycopg2, to execute the prepared statement. Thus, instead of manually composing a query statement, the API can beneficially help developers and any participating application to simply interact with the stored data as desired\(^{57}\).

Since all of essential components of the database have been developed completely. Thus, the database SWUT-DB is now fully ready to support any task of data handling.

---

\(^{57}\) See Appendix C.3: Python functions as the Database API.
4.3 Development of Raw Data Parsers

In this section, the implementation of raw data parsers, that are used in refining and transforming a huge amount of data checked out from the TortoiseSVN, is presented. Before going into details of the implementation, the current environment and how the raw data is stored by subversion control should be explained first.

4.3.1 Developing programming parsers

Inside the CMS, every piece of raw test data is stored under a well-defined structure. The following figure presents a checked-out structure of directories that is used in collecting a version or release of test data (under: \SRVC_MainSW_UT):

As showing in the figure, the amount of raw test data that need to be parsed is separated stored in several folders, i.e. Cantata, include, and source. The Cantata folder collects all of dynamic code analysis results of each sourcefile, generated by Cantata in several file formats e.g. (test scripts) *.c, *.cpp, (result files) *.ctr, *.cov (under: Cantata\tests). In folder include, there is a lot of include files, i.e. *.h, *.hpp, *.inl, used in the developing embedded system, whereas the source folder collects all developing source files that are used in the development.

For static code analysis, the results that are generated by the tool, i.e. QA-C/QA-C++, will be stored in another folder in as *.met files.
The major task of the parsers is parsing each file, which are now generated in several formats that is stored in those folders to a needed data that will be later transferred to the database. The needed data that the parsers must get from the amount of files, e.g. filename, author, revision, lines of code, code coverages, etc., will be systematically stored in the database tables that has been created since the previous section. Examples of raw data residing in the files are shown in below figure:

![Figure 4.8: Examples of raw data generated by SWUT tools.](image)

As the area framed in the figure 4.8, the parsers must get the interested data from each file. The procedures to get any specific content in a file are basically based on silently accessing the file, locate the interests, then readout the value written in that file to a memory to be forwarded to any further process\(^{58}\).

### 4.3.2 Transferring parsed data to the database

Once the interested data of each sourcefile is ready to be transfer to the database, a middle script will be developed and used as a wrapper to communicate to the database API, which has been created since previous section. The figures below illustrate the protocol established by the middle wrapper in transferring the parsed data to the database:

---

\(^{58}\) See Appendix C.4: Python functions as raw data parsers.
The external processes of transferring parsed data is illustrated by figure 4.9(a). The processes required the middle wrapper in analyzing a result returned by the database that is collaborated with the database API in order to execute the query. Figure 4.9(b) presents a flow the algorithm used in transferring data to the database as an object. The algorithm is implemented by Python 2.7 (under: `\SRVC_MainSW_UT-DB\Scripts\Parser`)⁵⁹.

---

⁵⁹ See Appendix C.5: Python function to insert data to the database.
4.4 Development of Metrics Generator

As discussed, Matplotlib which is a plotting libraries for Python 2.7 will be used use in the implementation. The development of the generator begins at downloading the library\(^60\) and the wrapper functions\(^61\), then continued by installing the downloaded libraries and programming part.

The current outcomes generated by the metrics generator that will be presented in this document consist of only two file formats, i.e. a graphic formatted in *.PNG, and another format that can be later imported by further process, that is *.CSV.

When the metrics generator is executed, a communication between the generator and the database will be established via the database-API in order to retrieve the stored data with a specific period, e.g. stored data between years 2014 and 2015, or even the data that is stored since yesterday, to the generator to perform an interpretation and generate test metrics.

4.4.1 Generating metrics in PNG format

A graphic metric can be used in several ways of presentation, e.g. inserted in a spreadsheet used in conferences, showing on web browsers, or used for development assessment. The following figures are an example of two graphics generated by the metrics generators:

Figure 4.10: Graphics generated by the metrics generator using Matplotlib.

\(^{60}\) Download Matplotlib-1.2.0.win-amd64-py2.7.exe from github.com/matplotlib/matplotlib/downloads.

An overall development status in a comparison between amount of lines of code and line-of-code coverage is showed in 4.10(a). The period of time used in generating the graph is during a year of the project. While the figure 4.10(b) showing the current status in view of different subsystems within only a specific date. Thus, table Execution and Mapping are mainly used to implement a data tracking mechanism which is used in summarizing a status of development that can be queried independently in any specific period of time.

A generation of the graphics requires a number of complex Python scripts to analyze the data that will be generated to be metrics, calculate graphic properties, as well as rendering the metrics in PNG format (under: \SRVC_MainSW_UT-DB\Scripts\Graphics)\textsuperscript{62}.

### 4.4.2 Generating metrics in CSV format

One of a file format that is widely used to store tabular data in plain-text form is CSV. In this development, the formatted file will not be generated directly to be test metrics, but rather be artifacts that can be simply imported and processed by other operations. The following figure demonstrates a tabular worksheet that the CSV is imported by Microsoft Excel\textsuperscript{63}:

\textsuperscript{62} See Appendix C.6: Python with Matplotlib to generate graphic metrics.

\textsuperscript{63} The implementation uses a semicolon “;” as a separator to avoid potentially crashing to input data.
4.5 Development of User Interface

As discussed in the research and discussion chapters, Microsoft Excel is chosen to be used as an application that allows authorized users to monitor or manipulate the data stored in the database through the interface. In this context, the interface is called front-end user interface.

The development of the interface requires some additional packages for Python 2.7 to be the wrapper that control a created instance of the Microsoft Excel application as well as accessing COM object of the application. After the application and the two required additional packages have been installed completely, the development of the front-end user interface will be started.

4.5.1 Generating the front-end user interface

A generation of the interface consists of several procedures, e.g. retrieving the current data from the database, extracting and writing into a worksheet, and formatting the interface. The main procedures to generate the interface are detailed as follows:

1) Retrieve data from the database

The specification of data that will be written into the interface must be the latest one which has been input to the database by the latest execution. That means, the data contents showing on the interface are definitely up-to-date.

The script (under: \SRVC_MainSW_UT-DB\Scripts\Interface) will get the timestamp of the latest execution from table Execution in the database first. Then a tracking process will be started to get all of data which are stored in the database under the timestamp before putting in a huge data object that is structured in object-oriented way defined in the DOM, see table 4.1 for information about the objects.

This following figure presents a structure that the retrieved current data is organized by the object-oriented structure, also called a data tree:

---

64 The implementation has been tested under version 2007 and 2013.
65 Download Win32com (pywin32) from http://sourceforge.net/projects/pywin32/.
66 Download xlwt 1.0.0 from https://pypi.python.org/pypi/xlwt.
Figure 4.12: The object-oriented structure used in handling queried data.

A tree of the structure, showing in the figure 4.12, is created by a recursive function that first collects the primary key of all data records where data will be retrieved. Then recursively iterate over attributes of each object in each level of the structure.

2) Extract and write a worksheet

Since the data that will be written into the interface has been organized and stored in a data tree, a process of tree extraction is required. Manipulating data in the tree can be done like a procedure of data accessing in object-oriented way.

This below demonstrates how to access data stored in the tree-like structure in different classes:

```
# get a detail of the supplier who manufactured the subsystem.

# get the data type of owner which the atomic component belongs to.

# get the order of column where name of the baseline will be written.
$DATA = Project.Baseline.Name.Interface.Order
```
3) Create and format the user interface

Once all of data queried from the database can be accessed, a worksheet of Microsoft Excel interface will be created. The following part of Python code is used together with xlwt library to create a new interface:

```python
    #create a new workbook and worksheet.
    o_workbook           = xlwt.Workbook(encoding="LATIN-1")
    o_worksheet          = o_workbook.add_sheet(conf.excel_sheet)

    #set a password to protect the worksheet.
    o_worksheet.protect  = True
    o_worksheet.password = conf.excel_password
```

After the creation of a new workbook to be used as the user interface is completed, all of data will be written into the worksheet. By a security mechanism embedded in the worksheet, a number of columns will be protected by password defined in the global configuration file to block any access from unauthorized users, but the rest is unprotected that allows anyone to modify the contents showing on the interface.

4.5.2 Synchronizing data with the database

As the main objective in creating the front-end user interface, all of data stored in the database will be interacted through the interface instead of accessing to the back-end interface of PostgreSQL. Anyway, once the user interface has been modified by an authorized user, a synchronization of the changes is immediately required to apply the modification of data in the interface to the database. On the other hand, if the database is in a more up-to-date state, based on its latest execution’s timestamp, the interface must also be updated. Thus, when the synchronization is completed, data stored in both sides, i.e. the database, and the front-end user interface, will be exactly equal. However, the synchronization will be completed once the timestamps of both sides are equal, by considering in level "%Y-%m-%d %H:%M:%S".

The algorithm that is used in driving the synchronization of data in the database and the front-end interface is illustrated by the following figure:

---

67 See Appendix C.7: Configure a protection of the Microsoft Excel interface.
As shown in the figure, the latest timestamps of the database and the user interface will be compared to get the one which is more up-to-date. If the timestamp of the latest execution of database is not less than the timestamp of the user interface, all of data currently existing in the interface will be immediately overwritten. But, otherwise, all of current data in the user interface will be read out and submitted to the Database Uploader which is a Python script used to collect data and forward to the database to perform updates.

The following figure shows the front-end interface that is developed using Microsoft Excel. Furthermore, the figure also presents the realization that security mechanism that protect the interface from any unauthorized users to modify data contents in the columns that contain sensitive data. The protected columns will be accessible once they are unprotected by filling the password which is known by only the super administrator:\footnote{The password is securely stored in the configuration file that can only be accessed by the administrators.}
CHAPTER 4. IMPLEMENTATION | 89

Figure 4.14: The realization of security mechanism embedded in the user interface\textsuperscript{69}.

The user interface will be persistently stored under a specific directory, defined in the global configuration file. As already known, a synchronization must be performed every time when any data stored in any side has been modified to maintain a consistency of data stored in the system. The additional process that will handle the entire process of synchronizing data flowing in the system will be deeply explained in the next section.

\textsuperscript{69} The realization is made on Microsoft Excel 2013 under the configuration set by the code in appendix C.7.
4.6 Automate the Framework

Since all of individual modules have been developed completely, each of them now will be able to perform its functionality as expected. The database is ready to store all of data that are input by raw data parsers after finished refining the checked out data. Metrics generator is ready to publish graphic metrics and other kinds, as well as the front-end user interface that has also been generated.

Although the modules are currently workable, but each of the underlying processes is now only performed individually. An integration of the components must be implemented to support the deployment of the framework to drive the whole system to reach the targets. The integration is actually occurred when every communication is established to exchange data between them. According to the figure 3.1 that presents the conceptual framework of this development, all of modules that have been developed are integrated together to build up a huge collaborative framework used for handling a huge amount of data in software testing.

In this section, as mentioned since the introduction, Jenkins CI is currently used in the organization to support the continuous integration processes in software development. Thus, the engine will also be used to implement an integration of the framework, which is also called collaborative framework in this context. The tool will be responsible to automate the underlying processes to perform each task, aka. Job, in a proper static sequence that will also be triggered by the engine to execute the jobs in a specific time. This following figure presents the sequential execution flow of the collaborated jobs that will be driven by the Jenkins CI:

![Sequential execution flow driven by Jenkins CI](image)

Figure 4.15: Sequential execution flow driven by Jenkins CI.
As presenting in the figure 4.16, Jenkins CI is responsible to administrate the both lines, i.e. the first line that performing software unit testing and generate a huge amount of data which will be checked in to the TortoiseSVN at the final process, and the second line that is continuously handling the amount of test data checked out from the repository by means of parsing and transferring to store in the database. Technically, the two lines are able to execute their collaborated jobs simultaneously in a manner of pipeline execution, demonstrating in the figure 2.21. But, in a practice of this development, the line of software unit testing will be executed first at the specific starting time, e.g. at 10:00 AM in every day, or even executing the framework only in weekday at noon, and immediately continued by the underlying processes of test data management. This following figure presents a real implementation of the framework in a productive system which is currently used to support the software testing processes:

Figure 4.16: The implemented framework controlled by Jenkins CI\textsuperscript{70}.

\textsuperscript{70} The interface of Jenkins CI version 1.610 configured by BEG-VS/EEA2 Bosch Engineering GmbH.
By automating the framework using Jenkins CI, the administrator is able to simply customize the execution flow through the configuration interface provided by the engine, as well as configuring the execution time and also defining conditions to handle any failure that might occur during the execution. The figure below presents the interface that allows the administrator to customize the configuration of this framework:

![Configuration Interface](image)

**Figure 4.17:** The configuration of software testing processes in Jenkins CI.

Up to this point, the automatization of the framework is now completely realized that makes all of collaborated modules consisting in this collaborative framework able to timely and automatically support the software testing together, as well as fully satisfying the scientific challenges in develop a framework to handle a huge amount of data generated by the processes of software testing.
Chapter 5

Conclusion

5.1 Development Summary

According to the several challenges in the conventional process, management of big data of
software testing is mainly operated by the CMS and Jenkins CI. The data is collected in spread
locations without an appropriate handling mechanism. Generation of test metrics is also
performed manually that potentially causes an inaccurate data interpretation.

The organization aimed to develop a framework that is able to partially transform the existing one
into a collaborative framework that is capable to effectively handle the huge amount of test data.
An idea to organize the data in a single repository is introduced by using a centralized database.
Based on the implementation of the database-centric architecture, the engine supports the
requirements in management of test data satisfactorily. Furthermore, it can also be expanded by
integrating with other development modules to enhance its functionality and capability to fully
support the software testing processes at any level.

The knowledge collected during the research is summarized as follows, to present the scientific
and the technical solutions that are used in supporting the challenges:

Collecting data in a centralized database

Database-centric architecture is used as a basis of managing the data that can be connected by
several collaborated modules. The three-tier architecture is also integrated with the first approach.
The integration mainly classifies the system components into separated layers. The classification
greatly helps in developing and administrating the framework in term of system organization and
visualization. A database is developed based on the approaches using PostgreSQL 9.3, together
with the API written in Python 2.7 and Psycopg2 that have been properly discussed and selected
as the most appropriate development tools.

The object-relational model is implemented in the database structure. The realization provides a
lot of advantages to satisfyingly support the complexity of the test data. By developing the database
using this model, the object-oriented techniques can be widely used in several operations, e.g. data organization, visualization, maintenance, etc., as well as offering a simply way to access the stored data in an object-oriented way. That means, the structure is totally based on a relational model, but the mechanisms of schema administration are derived from the characteristics of object model.

**Transforming raw data using parsers**

After a process of software testing has finished, a large volume of raw data that has been generated by the used SWUT tools and Jenkins CI will be located in the system. It generally contains several types of artifact, e.g. source files, test scripts, test results, etc. The programming parsers, written in Python 2.7, will statically transform the data through localizing and refining processes to get specific values which are specified in the parsers. The extracted values will be transferred to the database before being dynamically retrieved and processed in other data-related modules.

Thus, by transforming the data using programs, the test data will be unified in a particular format. Each required value will be automatically and accurately readout that also directly optimize the development efforts as well as minimizing human errors in data handling.

**Generating test metrics by programming scripts**

By systematically storing a huge amount of data in a centralized database, several processes that have been previously used in generating test metrics have been taken out. At the current, numbers of office documents that have been used to collect and summarize data, as well as used in pivoting charts or generating graphics are not required anymore. A metrics generator is developed as programming scripts, written in Python 2.7 and Matplotlib. Since the module has been deployed to the framework, it fully replaces the former processes. Currently, data stored in the database will be accurately interpreted and quickly generated into various kinds of test metrics as aimed.

Thus, without any manual process, the interpretation of data to generate the test metrics will be faster and more accurate. The results greatly help the organization in several development phases, e.g. process tracking, control, planning, etc. Furthermore, by implementing the automated process, any potential incorrect data processing, that may occurred when processing data in manual, will be cleared out.
Automating the framework

Since each modular component has been developed, it performs own functions individually without any collaboration with others. An automatization is implemented using Jenkins CI to automatically and periodically execute the tasks related in software testing. The static sequence is configured by administrators and sequentially triggered by the tool. The implementation results in a continuous flow of sequential executions involving all operations of the components. It also builds up an automated collaborative framework which is currently deployed in the productive systems. The framework satisfyingly supports the on-going operations as expected. Furthermore, nightly executions are systematically and effectively handled by the framework that significantly reduces the efforts spent on manually performing each testing activity.

At the current, a number of SWUT tools is used for testing activities, i.e. QA-C/QA-C++ and Cantata. Each tool generates a huge amount of test data in own format depending on the configuration. The parsers will unify the variances of contents by transforming into a particular format. After that, all of the specific values will be programmatically parsed and continuously transferred to the database. At this point, the stored data is fully prompt to be retrieved and processed by further operations, especially in test metrics generator.

In conclusion, all of the scientific challenges have been solved completely throughout the implementations of the approaches mentioned above. The automated collaborative framework greatly supports the running complex processes of software testing. The centralized database effectively stores the huge amounts of test data as well as provides the possibility of granularity tracking. A generation of test metrics is automatically performed by the generator that interprets the stored data into specific formats. Furthermore, the generated test metrics are also embedded on web browsers at the current. It provides a great solution to the developers and customers which are in different sites to monitor the development progresses more closely and effectively\textsuperscript{71}.

\textsuperscript{71} The graphics are embedded on the interface of Jenkins CI by using “Section View Plugin”, developed by Timothy Bingaman (https://wiki.jenkins-ci.org/display/JENKINS/Sectioned+View+Plugin).
5.2 Future Research

In this research, the database-centric test management framework for test metrics is developed based on testing activities on software-unit level. The centralized database is fully independent from any application or project which is related with. Thus, it can be widely connected by various tools and expanded to support other software development levels, e.g. system level, vehicle level, etc. By handling a communication between the database and each connected component through the APIs, any application will be supported and allowed to connect to the database. As well as transferring data to the database, the process can be performed securely without affecting the stored data though the parsers. That means, any connection of software testing tools requires only a modification in functions of the parsers that have also been designed to support this requirement.

![Future framework expanded to support various applications.](image)
The figure 5.1 presents a conceptual expanded framework that is connected by several applications used in software testing in various levels of software development. In order to integrate new applications to the system, necessary methods must be inserted into the parsers to individually parse and readout specific values from the tools.

Furthermore, the capability in generating various kinds of test metrics can also be enhanced as well as creating customer specific reports based on templates given by customers. Although the metrics generator is currently developed to publish only two types of document, i.e. graphic and office document, but an interpretation of the stored data to create other formats is also possible in future.

Any additional type requires a modification in the generator, which is actually an API for test metrics generation, by adding necessary methods to support a rendering of the desired extension, illustrated by figure 5.2:

![Figure 5.2: A future framework expanded to support various formats of test metrics.](image)

In summary, the framework is prompt to be expanded to fully support other applications in various software development levels. The APIs only need to be modified to extend their functionalities and capabilities in aggregating and interpreting the test data. Thus, the entirety of this development can be a foundation of software test management framework that aimed at generating high-quality test metrics for other developments in future.
References

Books:


Journals/Reports:


Web-Documents:


Appendix A

Evolution of Database Systems

Development of database systems emerged since several decades ago aimed at storing data which are raw building blocks of system information in a persistent database which is a collection of information with a mechanism to manipulate data in the system for specific purposes by providing a storage to facilitate the processing of a big amount of data\(^\text{72}\). After that, Database Management System (DBMS) which is a set of software used to operate data in a database working with machine file systems had been developed as well as a number of approaches was introduced and released to build up database systems to satisfy various business requirements\(^\text{73}\).

1 Evolution Timeline

Various database systems had been introduced throughout a few decades, to response amounts of huge demands of business to build up the most effective storage systems to support their applications. The timeline of database systems, since the first conceptual model was introduced till the cutting-edge database models which are widely used nowadays, is summarized as follows\(^\text{74}\):

1940s

The evolution of computer system has begun by using the first non-proprietary programming language, COBOL, to create enterprise computer systems. The systems were developed with a need of storing data in the systems. That was the beginning of database systems which globally used up to the present time.

1960s

The computerized database to store information of government offices and business organizations was started with a number of data and database models was introduced to be options for private organizations. There were two popular data models used in commercial purposes, a network model

---


\(^\text{73}\) Ct. [HAADI 2010]

\(^\text{74}\) Ct. [DANIELSEN 1998]
called CODASYL and a hierarchical model call IMS. The most commercial success of the decade was the SABRE system that used by IBM in corporation with American Airlines.

1970-1972

The standard principle for database systems was emerged, Edgar E.F. Codd proposed the use of relational database model in ab article which introduced a fundamentally different approach by constructing database system which the logical organization, and physical information storage are fully disconnected from each other, the principle called database schema and still being considered by database developers up to the present time.

Furthermore, Codd suggested to store data in database by representing the structure as a tabular form, consisting tables, column, rows, and relations, operated by a high-level non-procedural (or declarative) language to identify the desired records or columns which need to access. Several RDBMS products were produced during the decade e.g. Oracle, Informix, Ingres, and DB2.

1980s

A motivation to handle more complex data objects in the database were emerged through a number of research that was focused to accomplish the tasks of database to store classes, association, and methods of objects in systems. In late of the decade, several object-oriented DBMS (OODBMS) had developed e.g. ObjectDesign, Versant, and O2, to support a wide-range of applications.

1990s

After several approaches of database had been introduced and developed over decades, new client tools for application development were released at high prices e.g. ODBC and Excel/Access. The first prototype for Object Database Management Systems or ODBMS were also created with a rise in demand for internet database connectors.

2000s

Database applications continued growing up during the period and several new interactive applications were researched and developed by focused on higher complex database technologies. In the decade, a hybrid structure or semi-structured database was developed to bridge the existing
approaches, the relational and object database, to provide more performance and decrease gaps of data processing\textsuperscript{75}.

The following figure 2.3 presents the evolution timeline of database systems at each state since the first database system was developed up to the present time which several functionalities are provided by numbers of database management systems to response demands of business application requirements.

![Timeline of the evolution of database models and systems](image)

Appendix A-Figure 1: Timeline of the evolution of database models and systems.

2 \textbf{Benefits of the Evolution}

By this time, databases are capable of electronically storing up to billions of data records and powerfully facilitate all needed user operations. The systems had been developed since the beginning of the evolution timeline to release database systems to support a diversity of business while several approaches also introduced and realized to build up suitable possible solutions for business requirements\textsuperscript{76}.

Up to the present time, there is still no conclusion indicated to specify the best approach to build up a database system. Throughout the evolution timeline, each approach offered different own strengths and also reflect its own weaknesses comparing to others, depending on which application is integrated with and a circumstance of implementation surrounding the system.

\textsuperscript{75} Ct. [ELMASRI/NAVATHE 2009]

\textsuperscript{76} Ct. [BURTON/THOMAS 2009]
Therefore, all introduced approaches were aimed at the same direction to manage data in a persistent data storage to facilitate the processing of data. Since the database systems had been evolved, every single data element can be stored permanently in a persistent database instead of on papers or in spreadsheets. These follows are the major benefits of databases in several aspects:

**Substantial time saving**

A database saves time to get a record of information which may take several minutes for searching data on spreadsheets or paper-based filing system. Operating via a computerized database to merge records and generated needed documents can be done in seconds, as well as simply modifying structure of storage by defining new relations of data elements that are needed.

**Data accessing**

Database is designed to support effective data visualization by giving searching results, called query, in a pattern of list in the database which allows users to programatically query data and store programmatic procedures, called routines, in the system that can be re-called anytime to maximize and utilize the performance of the system.

**Data integration**

Database supports an integration of different sources of information by linking each related data element using a defined relationship to produce a new information to extend the visualization of data. The integration offers users to maximize the working performance by operating querying over the database to get a single result by using only one-time access to the database.

**Driving business application**

Database offers the ability to provide an individualized response to millions of business application by standing at the middle of the system and manipulate data as well as administrate transactions made by potential business demands.

Furthermore, not only the evolution of database systems occurred during the last decades. There was also the evolution of internal core structure of a database, called schema evolution. The

77 Ct. [BUSINESSLINK 2007]
evolution refers to the problem of evolving a database schema to reflect the demands in real-world business application. The evolution is not focused only on the problem of schema modification. But, in fact, it also covers the evolution which affects the stored data within the schema as well as any operation will is related to the database.

During a few years ago, general schema of current database systems are able to accept any change in the requirements of business applications. Because of the evolution of database systems and also schema, database developers are widely realized that the data management, either in a level of database or its internal structure which is the schema, of any business application is one of the most difficult and critical engineering task to implement. Because of the sensitiveness of evolving database schema, a huge problem caused by the schema evolution will severely impact on queries and any operation performs on the applications. Hence, researching for the best appropriate database system and the schema is the first step that all database developers should carefully and properly perform to analyze the requirements and specification set by the businesses.
Appendix B
Programmng Diagrams

Appendix B.1: Programming dependency diagram.
Appendix B.2: Programming diagram of database middle interface.
Appendix B.3: Programming diagram of data synchronizer.
Appendix B.4: Programming diagram of database updater.

1. Start
2. Create data instances
3. Initialize the data instances
4. Analyze update in each source file
   - Insert current interface data to data objects
   - Check if the current object has data attribute
     - Yes: Has the object data attribute?
       - Yes: Get current value from data object
         - Get column order in Excel interface from data object
         - Create class instances
         - Find and collect mapping keys
         - Database/data_uploader.py
         - Sort and unify the mapping list
         - Query the latest mapping list of the baseline
         - Compare the queried and current lists
           - Yes: Are the mapping lists different?
             - Yes: Update changes to database
               - Global/data_object_model.py
             - No: Return to a calling function
           - No: Recursion over each sub classes
   - No: Recursion over each sub classes
Appendix B.5: Programming diagram of interface updater.
Appendix C

Programming Codes

Appendix C.1: SQL statements to create user-defined types.

```sql
-- create type "type_component"
CREATE TYPE type_component AS (
    component_id integer,
    name text,
    supplier text,
    owner text
);

-- create type "type_file"
CREATE TYPE type_file AS (
    file_id integer,
    name text,
    location text,
    revision text,
    revisiondate text,
    type text,
    author text,
    supplier text,
    priority numeric,
    status text,
    asillevel text,
    needtobetested text
);
```
Appendix C.2: SQL statements to inherit existing tables.

```sql
-- create table "testscript"
CREATE TABLE testscript (  
    lineoffile integer,  
    lineofcode integer,  
    numberofactivatedtestcase integer,  
    numberofdeactivatedtestcase integer,  
    selectedcoverageruleset text,  
    hostbasedtestable text,  
    CONSTRAINT pk_testscript PRIMARY KEY (file_id)  
) INHERITS (sourcefile);

-- create table "testresult"
CREATE TABLE testresult (  
    overallresult text,  
    entrypointcoverage numeric,  
    statementcoverage numeric,  
    decisioncoverage numeric,  
    booleanoperandeffectivenesscoverage numeric,  
    testexecutionstartdate text,  
    testexecutionfinishdate text,  
    scripterror integer,  
    checkfailed integer,  
    checkpassed integer,  
    checkwarned integer,  
    callsequencefailure integer,  
    CONSTRAINT pk_testresult PRIMARY KEY (file_id)  
) INHERITS (sourcefile);
```
Appendix C.3: Python functions as the Database API.

#function: connect to database.
#the secret is stored in the configuration file.
def begins(self):
    try:
        #connect database and create a new session.
        o_session = psycopg2.connect(SECRET)
        #get the cursor object.
        o_cursor = o_session.cursor()
    except psycopg2.DatabaseError, e:
        print "error %s" % e
    finally:
        #return cursor object to the call function.
        return o_session, o_cursor

#function: terminate session of transaction.
def ends(self, o_session):
    b_success = False
    try:
        o_session.close()
    except:
        #if not succeed, return error flag.
        b_success = False
    finally:
        #return result of operation to the call function.
        return b_success

#function: commit a transaction.
def commits(self, o_session):
    b_success = True
    try:
        o_session.commit()
    except psycopg2.databaseerror, e:
        print "error %s" % e
        b_success = False
    finally:
        #return result of operation to the call function.
        return b_success
Appendix C.4: Python functions as raw data parsers.

#function: get information about the source file.
def get_sf_information(file_path, keyword):
    content = open_sf_file(file_path)
    s_result = None  
    #find the lines with the information.
    for s_line in content:
        if keyword + ":" in s_line and s_line.startswith("*"):  
            s_result = s_line.split(':')[1].replace('$','').strip()  
            break  
    #return the interested value.
    return s_result

#function: get all test scripts.
def get_all_ts_files():
    #get all test script files and the attendant paths.
    l_ts_files, l_paths = get_files_of_type(s_test_path,(".c", ".cpp"))
    for s_path in l_paths:
        l_ts_paths.append(s_path)
    #return lists containing file names and paths.
    return l_ts_files, l_paths

#function: get name of test script.
def get_ts_name(file_path):
    if file_path.endswith((".c",".cpp")):  
        #check for the right test script.
        if not l_ts_paths:
            get_all_ts_files()
        file_path = os.path.basename(file_path)
        if not file_path.startswith("test_"):
            file_path = "test_" + file_path
        s_ts = get_file_path(file_path, l_ts_paths)
        file_path = os.path.basename(s_ts).split('.')[0]
    else:
        file_path = None
    #return the file path.
    return file_path
Appendix C.5: Python function to insert data to the database.

```python
# get a matched primary key of the current table.
i_pkey_id = database.manipulation.select(o_cursor, cls_ins.primary_key,
    cls_ins.table, s_qr_condition)[0]

# check the result, if no matched key then insert as a new record.
if i_pkey_id == None:
    i_pkey_id = insert_new_record(o_cursor, i_pkey_id, cls_ins)

# insert a new record into the database.
def insert_new_record(o_cursor, i_pkey_id, cls_ins):
    # create an insert statement.
s_inserting_column = ""
s_inserting_value = ""
i_attribute_index = 1
for attr_ins, addr_ins in cls_ins.__dict__.items():
    # get column name from data object.
    if hasattr(addr_ins, "attribute"):
        s_updating_attr = addr_ins.attribute
        s_updating_value = database.manipulation.cast(addr_ins.value,
            addr_ins.type)

    # get current value from data object.
    if i_attribute_index != 1:
        s_inserting_column += ", "
        s_inserting_value += ", "

    # concatenate to the string of insert statement.
    s_inserting_column += s_updating_attr
    s_inserting_value += s_updating_value
    i_attribute_index += 1

# insert into the table.
i_pkey_id = database.manipulation.insert(o_cursor, s_inserting_column,
    s_inserting_value, cls_ins)[0]

    # return the primary key of the new record.
return i_pkey_id
```

Appendix C.6: Python with Matplotlib to generate graphic metrics.

```python
# plot the graphic and draw all bars on it.
l_rects = []

# set the text for all rects inside of them.
i_counter_rect = 0
for i_counter_rect in range(len(l_rects)):
    auto_text(l_rects[i_counter_rect], l_len_bars[i_counter_rect])

# hide the right and top spines.
axis.spines['right'].set_visible(False)
axis.spines['top'].set_visible(False)
axis.spines['bottom'].set_visible(False)

# format the layouts.
axis.yaxis.set_ticks_position('left')
axis.xaxis.set_visible(False)

legend = plot.legend(l_rects, conf.l_labels, loc='upper center',
    bbox_to_anchor=(0.5, 1.10), ncol=4, fancybox=True,
    shadow=True, title = conf.s_legend_title, prop={'size':11})

# render the graphic.
plot.savefig('SRVC_MainSW_UT-DB\Metrics\graphics\ASIL_Metrics.PNG')
plot.close()
```
Appendix C.7: Configure a protection of the Microsoft Excel interface.

```python
#access com object of microsoft excel.
o_interface = win32com.client.Dispatch("Excel.Application")
o_interface.DisplayAlerts = False
o_interface.Visible = False
o_interface.Interactive = False
o_workbook = o_interface.Workbooks.Open(conf.excel_interface)
o_worksheet = o_workbook.Worksheets(conf.excel_sheet)

#unprotect the worksheet using the stored secret.
o_worksheet.Unprotect(conf.excel_password)

#freeze panes at the header row.
o_worksheet.Application.ActiveWindow.SplitRow = 1
o_worksheet.Application.ActiveWindow.FreezePanes = True

#enable autofilter and autofit.
o_worksheet.EnableAutoFilter = True
o_worksheet.Cells.AutoFilter(1)
o_worksheet.Columns.AutoFit()

#disable or enable the features for normal users.
o_worksheet.Protect(Password = conf.excel_password,
      DrawingObjects = True,
      Contents = True,
      Scenarios = True,
      AllowUsingPivotTables = True,
      AllowFormattingCells = True,
      AllowFormattingRows = True,
      AllowFormattingColumns = True,
      AllowSorting = True,
      AllowFiltering = True,
      AllowInsertingHyperlinks = True,
      AllowInsertingColumns = False,
      AllowInsertingRows = False,
      AllowDeletingColumns = False,
      AllowDeletingRows = False)

#save and close the protected excel interface.
o_workbook.Close(SaveChanges=True)
o_interface.Quit()
```
– End of the Thesis –