SE 3XA3: Test Report
GoDBMS

Team #7, Databased
Faiq Ahmed, ahmedf46
Eesha Qureshi, qureshe
Kevin Kannammalil, kannammk

April 12th, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
April 11, 2022	1.0	Initial Document
April 11, 2022	1.2	Started Test Cases
April 12, 2022	1.2	Finished Test Cases
April 12, 2022	1.2	Completed Document

The purpose of this document is to provide the outcome of the testing methods that were laid out in the Test Plan document. This document includes the assessment of how successful the Functional and Non Functional requirements that were created during the development process and the traceability of those requirements throughout the process.

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirements Specification; a document that outlines important information about the software project such as stakeholders, functional requirements, non-functional requirements and project scope
POC	Proof of Concept; a prototype, demo or design that provides evidence of the project's feasibility
DBMS	**Database** Management System; software that allows a user to perform **database** transactions and operations
CLI	Command Line Interface; the point of communication between the software and the user
Go	GoLang

Table 3: **Table of Definitions**

Term	Definition
Database	A set of **record**s that store information organized by keys
Command Line	The point of communication between user and internal computer instructions
Table	An organized collection of **record**s in a **database**
Record	A single row of data in a **database**
Primary Key	A unique identifier for a **record** in a **database**
Struct	A modular collection of fields in GoLang
Schema	An outline of fields, table names, and relationships in a **database**
Columns	Vertical aggregations of data in a table
Catalog	A list storing the names of all the tables in the database. This list can be encoded and decoded from a file to allow persistent storage.
Tuple	An immutable finite sequence of elements
Message Passing	The method of invoking a process in a computer
Concurrency	The use of threads to run multiple computer processes at the same time
GoLang	A compiled programming language

1 Functional Requirements Evaluation

1.1 Input Parsing

Test #1:	**IP-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started and is waiting for a user input
Input:	User enters an input that is not a create table, insert, delete, list or search statement
Output:	Invalid query error appeared
Expected:	The system must print an error back to the user stating that their query syntax is invalid
Result:	PASS

Test #2:	**IP-BE1-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to create a table will be passed in to the parser testing function with a missing table name
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table name is missing
Result:	PASS

Test #3:	**IP-BE1-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to create a table will be passed in to the parser testing function with a missing columns
Output:	Missing columns error appeared
Expected:	The function must return an error stating that the table columns is missing
Result:	PASS

Test #4:	**IP-BE1-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to create a table will be passed in to the parser testing function with a missing primary key
Output:	Missing primary key error appeared
Expected:	The function must return an error stating that the table primary key is missing
Result:	PASS

Test #5:	**IP-BE1-T4**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to create a table will be passed in to the parser testing function with the correct syntax
Output:	Struct of the query was retrieved
Expected:	The function must return a struct that contains the schema of the create table statement
Result:	PASS

Test #6:	**IP-BE2-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to delete a table will be passed in to the parser testing function with a missing table name
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table name is missing
Result:	PASS

Test #7:	**IP-BE2-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to delete a table will be passed in to the parse testing function with the correct syntax
Output:	Received a struct with the schema of the delete table statement
Expected:	The function must resturn a struct that contains the schema of the delete table statement
Result:	PASS

Test #8:	**IP-BE4-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to insert a record will be passed in to the parser testing function with a missing table name
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table name is missing
Result:	PASS

Test #9:	**IP-BE4-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to insert a tuple will be passed in to the parser testing function with the missing columns
Output:	Missing columns error appeared
Expected:	The function must return an error stating that the record columns are missing
Result:	PASS

Test #10:	**IP-BE4-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to insert a tuples will be passed in to the parser testing function with missing record data
Output:	Missing record data error appeared
Expected:	The function must return an error stating that the record data is missing
Result:	PASS

Test #11:	**IP-BE4-T4**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to insert a record will be passed in to the parser testing function with the correct syntax
Output:	Received struct with the schema of the insert record statement
Expected:	The function must return a struct that contains the schema of the insert record statement
Result:	PASS

Test #12:	**IP-BE5-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to modify a record will be passed in to the parser testing function with a missing table name
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table name is missing
Result:	PASS

Test #13:	**IP-BE5-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to modify a record will be passed in to the parser testing function with a missing missing primary key to uniquely identify the tuple
Output:	Missing primary key error appeared
Expected:	The function must return an error stating that the record primary key is missing
Result:	PASS

Test #14:	**IP-BE5-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to insert a tuple will be passed in to the parser testing function with the missing data to update
Output:	Missing record data error appeared
Expected:	The function must return an error stating that the record data to update is missing
Result:	PASS

Test #15:	**IP-BE5-T4**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to modify a record will be passed in to the parser testing function with the correct syntax
Output:	Received the struct of the schema for the modify record statement
Expected:	The function must return a struct that contains the schema of the modify record statement
Result:	PASS

Test #16:	**IP-BE6-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to delete a record will be passed in to the parser testing function with a missing table name
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table name is missing
Result:	PASS

Test #17:	**IP-BE6-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to delete a record will be passed in to the parser testing function with a missing primary key to uniquely identify the tuple
Output:	Missing primary key error appeared
Expected:	The function must return an error stating that the record primary key is missing
Result:	PASS

Test #18:	**IP-BE6-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to delete a record will be passed in to the parser testing function with the correct syntax
Output:	Received a struct of the schema for the delete record statement
Expected:	The function must return an error stating that contains the schema of the delete record statement
Result:	PASS

Test #19:	**IP-BE7-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to search for records will be passed in to the parser testing function with missing table names
Output:	Missing table name error appeared
Expected:	The function must return an error stating that the table names to search from is missing
Result:	PASS

Test #20:	**IP-BE7-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to search for records will be passed in to the parser testing function with missing table columns to return
Output:	Missing table column error appeared
Expected:	The function must return an error stating that the table columns to return are missing
Result:	PASS

Test #21:	**IP-BE7-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The parser has been initialized
Input:	A string query to search for records will be passed in to the parser testing function with the correct syntax
Output:	Received struct of the schema for the search record statement
Expected:	The function must return a struct that contains the schema fo the search record statement
Result:	PASS

1.2 Saving Data

Test #22:	**SD-BE1-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started and is waiting for a user input, and the database has been initialized
Input:	User enters an input to create a table with the correct syntax and a unique table name
Output:	Message appeared indicating that the table was created successfully
Expected:	The system must print a success message back to the user informing them that their table has been created
Result:	PASS

Test #23:	**SD-BE1-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized
Input:	A struct to create a table has been passed in with the correct syntax and a unique table name
Output:	Verified that the table does exist
Expected:	The system must create the specified table in the catalog
Result:	PASS

Test #24:	**SD-BE1-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables added to it
Input:	A struct to create a table has been passed in with the correct syntax and a table name already exists
Output:	Received duplicate table name error
Expected:	The system must return an error stating that a table by the specified name already exists
Result:	PASS

Test #25:	**SD-BE2-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables added to it
Input:	User enters an input to delete a table with the correct syntax and an existing table name
Output:	Received success message and verified that table does not exist
Expected:	The system must print a success message back to the user informing them that their table has been deleted
Result:	PASS

Test #26:	**SD-BE2-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables added to it
Input:	A struct to delete a table has been passed in with the correct syntax and an existing table name
Output:	Verified that the table does not exist anymore
Expected:	The system must delete the specified table in the catalog
Result:	PASS

Test #27:	**SD-BE2-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized
Input:	A struct to delete a table has been passed in with the correct syntax and a table name does not exist
Output:	Received error indicating there is no table with that name
Expected:	The system must return an error stating that a table by the specified name does not exist
Result:	PASS

Test #28:	**SD-BE3-T1**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized
Input:	A string query to list all tables has been passed in to the parser testing function with the correct syntax
Output:	Received the existing tables
Expected:	The function must return all tables in the catalog
Result:	PASS

Test #29:	**SD-BE4-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables added to it
Input:	User enters an input to insert a record with the correct syntax, an existing table name, and a primary key that does not already exist
Output:	Received success message
Expected:	The system must print a success message back to the user informing them that their record has been inserted
Result:	PASS

Test #30:	**SD-BE4-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables added to it
Input:	User enters an input to insert a record with the correct syntax, an existing table name, and a primary key that does not already exist
Output:	Received success message and verified that the record exists
Expected:	The system must print a success message back to the user informing them that their record has been inserted
Result:	PASS

Test #29:	**SD-BE4-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables added to it
Input:	A struct to insert a record has been passed in with the correct syntax, an existing table name, and a primary key that does not already exist
Output:	Verified that the record exists in the table
Expected:	The system must insert the specified tuple in the database
Result:	PASS

Test #31:	**SD-BE4-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized
Input:	A struct to insert a record has been passed in with the correct syntax and a table name that does not already exist
Output:	Received error indicating the table does not exist
Expected:	The system must return an error stating that a table by the specified name does not exist
Result:	PASS

Test #32:	**SD-BE5-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables and records to it
Input:	User enters an input to modify a record with the correct syntax, an existing table name, and a primary key to uniquely identify the record
Output:	Verified that the record has been modified correctly
Expected:	The system must print a success message back to the user informing them that their record has been modified
Result:	PASS

Test #33:	**SD-BE5-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	A struct to modify a record has been passed in with the correct syntax, an existing table name, and a primary key to uniquely identify the record
Output:	Verified that the record has been modified correctly
Expected:	The system must modify the specified tuple in the database
Result:	PASS

Test #34:	**SD-BE5-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	A struct to modify a record has been passed in with the correct syntax and a table name that does not exist
Output:	Received incorrect table name error
Expected:	The system must return an error stating that a table by the specified name does not exist
Result:	PASS

Test #35:	**SD-BE5-T4**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	A struct to modify a record has been passed in with the correct syntax, an existing table name, and a primary key that does not exist
Output:	Received incorrect primary key error
Expected:	The system must return an error stating that a table by the specified primary key does not exist
Result:	PASS

Test #36:	**SD-BE6-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables and records added to it
Input:	User enters an input to delete a record with the correct syntax, an existing table name, and a primary key to uniquely identify
Output:	Received a success message and verified that the record has been deleted
Expected:	The system must print a success message back to the user informing them that their record has been deleted
Result:	PASS

Test #37:	**SD-BE6-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	A struct to delete a record has been passed in with the correct syntax, an existing table name, and a primary key to uniquely identify the record
Output:	Verified the record does not exist in the table
Expected:	The system must delete the specified tuple in the database
Result:	PASS

Test #38:	**SD-BE6-T3**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	A struct to delete a record has been passed in with the correct syntax and a table name that does not exist
Output:	Received missing table name error
Expected:	The system must return an error stating that a table by the specified name does not exist
Result:	PASS

Test #39:	**SD-BE7-T1**
Type:	Functional, Dynamic, Manual
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables and records added to it
Input:	User enters an inputs to search for records with the correct syntax, and an existing table name
Output:	Received all the records in the table
Expected:	The system must print back all records found from their search query back to the user
Result:	PASS

Test #40:	**SD-BE7-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The command line interface has been started, is waiting for a user input, and the database already has tables and records added to it
Input:	A struct to search for records has been passed in with the correct syntax and a table name that does not exist
Output:	Received all the records in the table
Expected:	The system must return an error stating that a table by the specified name does not exist
Result:	PASS

Test #41:	**SD-BE8-T1**
Type:	Functional, Dynamic, Manual
Initial State:	Testers have access to the source code for the transactions in the database management system
Input:	The testers do a code walk through to ensure that the system correctly locks the record during a write
Output:	The testers were able to confirm that the locking mechanism is programmed correctly
Expected:	We are able to determine whether structurally the code will prevent writes and other transactions at the same time
Result:	PASS

Test #42:	**SD-BE8-T2**
Type:	Functional, Dynamic, Automated
Initial State:	The database has been initialized and already has tables and records added to it
Input:	The testers do a code walk through to ensure that the system correctly locks the record during a write
Output:	The testers were able to confirm that the locking mechanism is programmed correctly
Expected:	We are able to determine whether structurally the code will prevent writes and other transactions at the same time
Result:	PASS

Test #43:	**SD-BE8-T3**
Type:	Functional, Dynamic, Manual
Initial State:	The database has been initialized and already has tables and records added to it
Input:	The testers will manually pass in multiple read and write queries to the database through multiple CLIs
Output:	The testers verified the result of the queries were correct
Expected:	The system will output the result of all passed in queries
Result:	PASS

2 Non Functional Requirements Evaluation

2.1 Look and Feel Testing

Test #44:	**NFR-1-LF1**
Type:	Dynamic, Manual
Description:	A manual test to determine whether the appearance of the client for the database is up to par
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	Average survey score of at least 70%
Result:	PASS

Test #45:	**NFR-2-LF2**
Type:	Dynamic, Manual
Description:	A manual test to determine whether the styling of the client is good
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	Average survey score of at least 70%
Result:	PASS

2.2 Usability and Humanity Testing

Test #45:	**NFR-3-UH1**
Type:	Dynamic, Manual
Description:	A manual test to determine whether the database is intuitive enough for beginners
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	Average survey score of at least 70%
Result:	PASS

Test #46:	**NFR-4-UH2**
Type:	Dynamic, Manual
Description:	A manual test to determine whether how easily comprehensive the language in the database client is
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	Average survey score of at least 70%
Result:	PASS

2.3 Performance Testing

Test #47:	**NFR-5-P1**
Type:	Dynamic, Automated
Description:	An automated test to determine whether the database provides responses to the user in a reasonably fast time. This is done by timing how long it takes for responses to multiple requests to be made in the database
Input:	Multiple search queries are run on the database
Expected:	The time recorded is less than or equal to the maximum accepted response time of 5 seconds
Result:	PASS

Test #48:	**NFR-6-P2**
Type:	Dynamic, Manual
Description:	A manual test to determine whether the database maintains the precision of the numbers that are stored
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	The testers confirm whether or not the precision of the initial input data has been maintained
Result:	PASS

Test #49:	**NFR-7-P3**
Type:	Dynamic, Manual
Description:	A manual test to determine whether the *.data* file has been generated in the Data folder of the local directory to indicate that the data is persistent and being stored locally
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	The testers confirm whether or not the data file has been generated
Result:	PASS

2.4 Performance Testing

Test #50:	**NFR-8-OE1**
Type:	Dynamic, Manual
Description:	A manual test to determine whether all the functionalities of the database still work as expected without an internet connection
Tester(s):	Group of students with at least a basic understanding of databases
Pass:	The testers confirm the database still works as normal without an internet connection
Result:	PASS

3 Comparison to Existing Implementation

The original implementation was well tested but there were no documentations to support the tests. Additionally the original implementation also did not have any non functional tests. Though the original implementation was well tested from a functional testing standpoint, the lack of traceability to the requirements did not raise confidence in the correctness of the software. In our implementation, the requirements were well documented and divided up into modules that were tested to an equal degree to the original implementation. Due to the use of documentation and explicitly identifying our requirements, which allowed us to trace our tests back to the requirements and be confident that we had successfully achieved our requirements. Our implementation also included various NFR tests which also increased our confidence in the correctness of the implementation.

4 Unit Testing

Unit testing was heavily implemented to ensure confidence in the functionality of each component of our code. As the individual software units comprised a much larger system, it was important to build confidence in the lower level functionality so that error tracing efforts would be efficient for system wide testing.

5 Changes Due To Testing

Testing did not uncover a cause to deeply reframe the architectural decisions or design patterns. However, rigorous test cases revealed local code bugs such as syntax errors, type mismatches and minor logic errors that were promptly addressed. Testing also revealed edge cases and necessary error handling.

6 Automated Testing

Our test plan was heavily automated testing using the Go test library. This reduced manual labour efforts and time resource consumption, and provided instant results right in the IDE. We used nearly 700 lines of automated testing scripts to ensure thorough testing.

7 Trace to Requirements

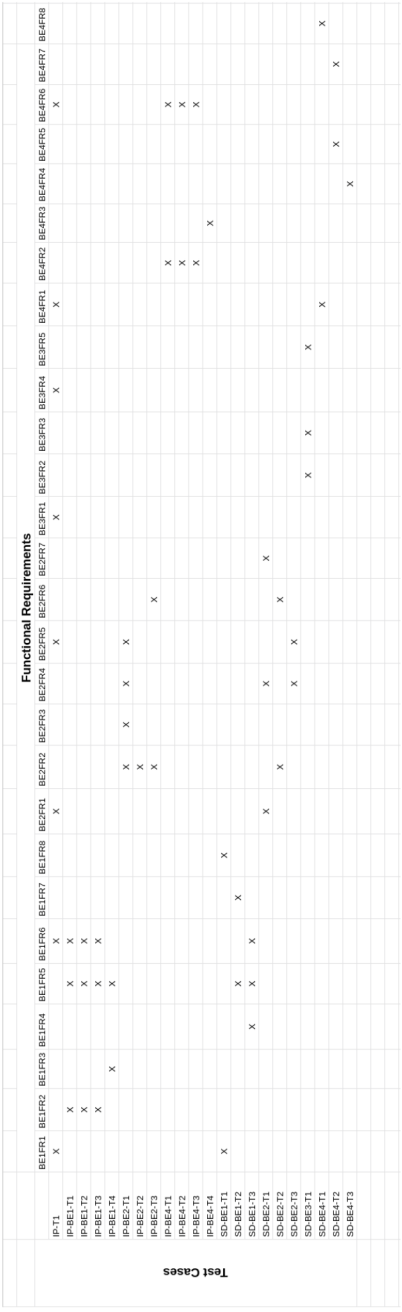

Figure 1: Traceability Matrix for Functional Requirements in BE1 to BE4

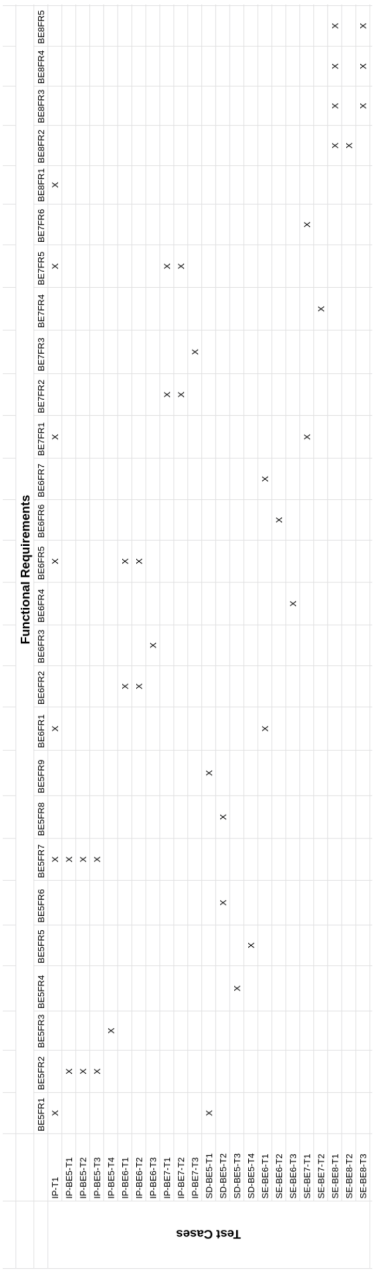

Figure 2: Traceability Matrix for Functional Requirements in BE5 to BE8

		Non Functional Requirements										
		3.1.1.1	3.1.1.2	3.1.2.1	3.2.1.1	3.2.2.1	3.2.3.1	3.2.3.2	3.3.1.1	3.3.3.1	3.3.6.1	3.4.1.1
Test Cases	NFR-1-LF1	X	X									
	NFR-2-LF2			X								
	NFR-3-UH1				X		X	X				
	NFR-4-UH2					X		X				
	NFR-5-P1								X			
	NFR-6-P2									X		
	NFR-7-P3										X	
	NFR-8-OE1											X

Figure 3: Traceability Matrix for Nonfunctional Requirements in NFR-1 to NFR-8

8 Trace To Modules

AC	Modules
ac1	m16
ac2	m13
ac3	m1
ac4	m1
ac5	m3, m10, m17
ac6	m13
ac7	m14

Table 4: Trace Between Anticipated Changes and Modules