Software Testing Using Context Maps Test Cases

Farooq Ali Khan

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montreal, Quebec, Canada

November 2007

Your file Votre référence
ISBN: 978-0-494-40942-8
Our file Notre référence
ISBN: 978-0-494-40942-8

Abstract

Software Testing Using Context Maps Test Cases (CMTC)

Farooq Ali Khan

In software industry the greatest amount of expenditure is in maintaining software developed poorly or specifications not met properly, this result in failures, budget overruns and unmet timelines. Software testing, if carried out properly, could actually resolve or mitigate these issues. However, in many cases the testing methodologies as well as the testing tools consider testing as a post development effort, which is conducted in isolation of the analysis and worse still, development itself. The efficacy of the testing as a guard against software and project failure is therefore not always realized, partly because of the absence of a tool based support for test cases, a tool which could be simple and yet effective.

We propose that contexts should be considered for testing and as such testing should not be considered as an isolated activity, but an umbrella or integrated activity, intrinsically linked and uses the output from different phases of software development lifecycle. For instance, the fact that context for testing is present in the test case would only make the testing easy and will help in creating a role for non testing teams including the users to participate in the success of the tests and a better product.

We use Context Maps to create Context Maps Test Cases (CMTC), for User Acceptance Testing in small customization projects to test defect fixes and enhancements in already developed software. CMTC helped in contriving the test case artifacts like test data, expected and actual output not just by testing team on the basis of documents, but also provide a role or channel for the end users as well as other stakeholders to share their thoughts, as to what constitute the best data to test the product with.

To hundreds of thousands of people killed since 911

Acknowledgements

I would like to thank all my teachers, my examiners Dr Li and Dr Radhakrishnan, my mother who called me everyday to inquire about when I will finish my studies, my father, my brothers, each and every one of my friends. I also would like to thank the Department of Computer Science, especially, Ms Halina Monkiewicz, Ms Edwina Bowen and all other staff and security members of the Concordia University. I also thank all the organizations and individuals who helped me and provided me the opportunity to work on different projects during my thesis research.

However, I cannot thank Dr Peter Grogono enough for his indefatigable support, enormous amount of help, both technical and moral. Whenever I met him, all the problems no matter how humungous they might seem before meeting him would just *peter out*. Dr Grogono is the most precious and important lesson I have learnt in my academic career. Very few people I have met are like him, open hearted in his support to his students, magnanimous, and above all always smiling.

Dr Peter Grogono, I thank you with all my heart and hope and wish the very very best for you and your family.

Index

Index

List of Figures

List of Tables

Important Notice:

The names of individuals and projects are used with permission.

Chapter 1

Introduction

1.1 Introduction

Most businesses need software for their daily operations, from hospitals to universities there is hardly a business environment that can operate in this day and age without software tools. It is important that these tools are tested properly for the business to continue its daily operations. Software testing is perhaps the most important phase of software development life cycle [14, 17]. It is an assurance that the software will meet the requirements and function properly. Because of the complex nature of communication between the technical and business professionals, requirement gathering is never a simple task and often the requirements are not properly documented. In such a situation proving that the resulting software is fulfilling the business requirements is difficult. Testing or more precisely user acceptance testing [17] is supposed to prove that the software fulfills the requirements of the business for which the software is being developed. This is not a reclusive or remote problem but is one that various studies have pointed out [1, 2, 3, 4, 5, 6 and 13] and almost all software organizations are familiar with. Many studies point out that the major cost in a software project is software maintenance [7, 8, and 13]; one factor that can reduce this cost is proper testing.

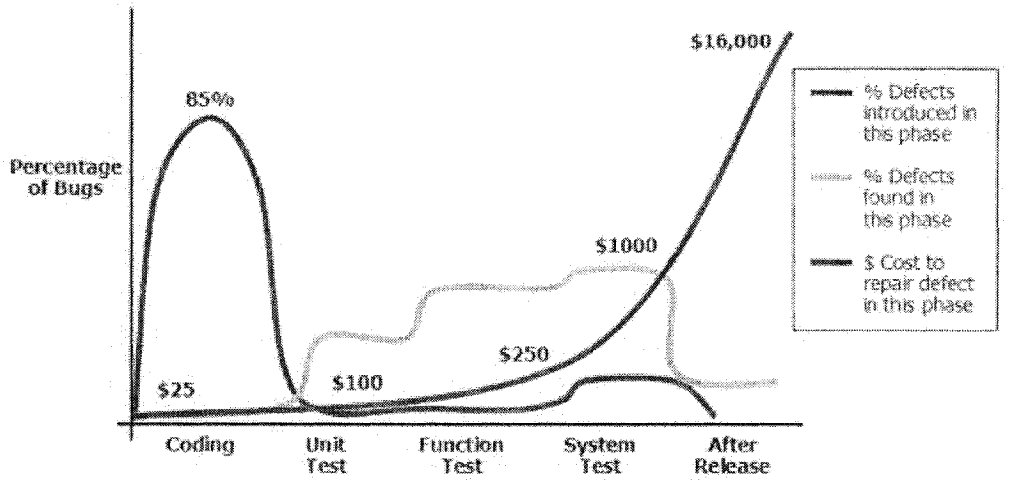


Figure 1.1 Results from Agitar Study [13]: Most of the defects are a direct result of the software not matching the real requirements

Context and Contextualized Testing

There is a very important relation between the test cases, the system being tested, related systems and the people who would identify the contents of the test cases, this relationship could be identified as the context of the test. There are studies that have pointed out the efficacy of contexts in systems analysis and design, such as the fish eye view [9, 10], however, there is little mention of software testing.

For the purpose of this thesis we define Context as, "the multiple Complex dynamic systems in which the project is situated".

The emphasis in contextualized testing is to encompass the relationships that exist in everything that goes on in a software development process which has a bearing on the effectiveness and value of the software. It is important for testing to be integrated in the software life cycle [15, 16, and 31] for testing process to be mature and for testing to be contextualized.

According to the IEEE Standard for Software Test Documentation: IEEE Standard 829-1998 [33], the test case document should have the following information

- An identifier to uniquely identify the documents from others
- Items to be tested
- Input / Output Specifications
- Hardware / Software environmental needs
- Test case specific procedure requirements e.g. setup, execution, cleanup
- Related test cases or pre requisites, which must be executed prior to the implementation of this test case

We observed from studying projects that the standard has the following very important information missing, which prevents it from capturing the contexts properly in the test case documents

- The programmer, user / business analyst, test analyst or managers have an important role respectively, in creating the above mentioned contextual elements of a test case document, **the roles are missing**.

- Also a tool that will help each one of the roles (mentioned in 1) to participate in identifying and documenting these related or contextual information, in a way so that the relation could be preserved and presented, **tool is missing.**

- At what stage in the testing life cycle each of these artifacts is documented, a **temporal dimension** is missing.

There are various tools from different vendors that help capture the context for software testing and defect tracking. Examples of such tool suites are IBM Rational (ClearQuest for Defect Tracking, Test manager for test case management and authoring, Manual Tester for test case authoring), Microsoft Team Systems and Tenrox among others [10, 19 and 34]; we will discuss our work with these tool suites in Chapter 3.

During our research we worked with customization departments of software companies, such as Perfapps, Visual 2000, Tenrox and Liberty Mutual, for user acceptance testing [17] on projects related to defect fixes and small enhancements. While testing these short term and small user acceptance testing projects, we found the current set of defect tracking, test case management and authoring tools to be inadequate because of the information related to the test is spread across various tools and within each tool across various tabs. We found that in order to compare the information the tester must browse through these tabs and tools, it is also time consuming to trace the data used for testing to the requirements or to identify the participants, such as Business Analysts or Programmers who could help understand or add the information available to create test cases. For instance, with our experience of working with a company where IBM Rational tool suite is being used, the defect tracking information is stored in IBM Rational ClearQuest whereas the test case itself or the input specification and output behavior of the system is captured using tools, such as IBM Rational Test Manager. There were incidents in which the business analysts did not capture or transferred all the information from IBM Rational Requisite Pro (used to document requirements by business analysts in that company) to the attached requirement documents in the ClearQuest (used as a defect tracking

tool), and when testing of the defect started the Analyst was on his vacations, so a test that would have finished in a day took a week to finish.

The discussion highlights the need of a simple tool that could help in both documenting test cases and defect tracking especially for user acceptance testing in small customization projects; this will eliminate the need to use multiple testing tools with many tabs to capture information for test cases in small customization projects effectively saving time and will also reduce the cost of testing by replacing the more expensive testing tool suites. Such a testing tool will also help Business Analysts, End Users and Programmers to add or review the test case contents easily without much help from test team members to understand the structure and mechanism of the tool.

This is our motivation in this research to study the aforementioned issues and develop a tool to support software testing needs in small and short term customization projects.

1.2 Scope of Our Work

We use Context Maps Test Case (CMTC), will be introduced in chapter 2, for documenting test cases to test customizations in existing projects. These customizations could be a result of defects found in existing software or a small enhancement required to fulfill the needs of the client or business using the software.

The main area of study in our thesis is the user acceptance testing [14, 15 and 17] along with regression testing [3, 14] usually carried out using Black Box Testing technique [17]. The customization projects we participated during our thesis are small customization projects for testing enhancement in current functionality and defect fixing; as a result the test cases were rather small in terms of the input and output data specification, which usually constitutes an important part of any software test case document [33]. Therefore scalability of the test case document was not an

important issue as we use Context Maps to create test case documents. However, a tool **Context +** [12] could be used to address the scalability of the CMTC in case where the Context Maps are larger and difficult to read.

1.3 Contributions of the thesis

This thesis offers the following contributions:

- It proposes a new approach to test customizations in existing projects, with a tool called CMTC (CMTC will be discussed in the next chapter). CMTC can present both defect tracking and testing information in one simple map without the need to browse across various testing and defect tracking tools and their respective tabs to compare the related information. This simple presentation of information also facilitates management control, especially in small scale customization projects for testing defect fixes and enhancements.

(The customizations in existing projects that are being tested could be a result of defects found in existing software or minor enhancements to fulfill the need of the customer. The main area of study in our research is the user acceptance testing [14, 15 and 17], which is usually carried out using Black Box Testing [17]).

- It provides CMTC as a simple tool which is easy to use and helps integrate user acceptance test planning and execution in the software development process [16] for small customization projects. This integration coupled with simple presentation format, which will be discussed in the next chapter, helps participants from various phases of Software Development Lifecycle [14] to participate in development of test case documents. For example, End user and business analyst during the requirement specification phase [15] could add information, such as which error or defect is more severe. Similarly, the programmer could specify the input data (like testing boundary

value conditions) which he might have overlooked during unit testing [14] because of time constraints.

We found that CMTC prevents ignorance towards testing and as such we consider the following points as contribution of our thesis though they also point out why CMTC are difficult to implement in some companies

- In small customization projects for testing defects and minor enhancements, the CMTC provide a way to create complete test cases with accurate information, and help organizations to follow the testing life cycle, which extends from software specification to successful software delivery [17, 18].
- Many organizations don't want to spend on training their people, effectively obviating the training aspect of the most important of four Ps (People, Process, Product, Project) [20]. CMTC require people to be at least familiar with the Context Maps terminology and gain a certain level of proficiency in using it, hence underlying the importance of training People.
- CMTC contradict a widely held belief that software metrics as well as testing results are not to be used for appraisals of various members involved in software development [20] as it might create a negative competition, for instance, programmers might start counting or writing more comments in source code just to be considered more productive. However, the CMTC do not just give the error but the precise and complete context. We found that the context makes it easy to identify and correct the problem.

1.4 Thesis Outline

The thesis is organized as follows

- Chapter 2 provides the background, the chapter introduces Context Maps and CMTC, with explanation and references to help the reader read Context Maps based CMTC.
- Chapter 3 presents related work and our work with other testing and defect tracking tools. This chapter highlights the problems and provides the motivation.
- Chapter 4 compares CMTC with the related work presented in Chapter 3, and presents our solution.
- Chapter 5 presents the CMTC documents that were created for various projects with their explanation as well as the results obtained. It also compares the effectiveness of CMTC and other related tools that we used in our work, with the help of graphs.
- Chapter 6 includes an insight as to what we think is the future of Context Maps in software testing and what needs to be done to further facilitate the usage of Context Maps as a software testing tool, this chapter also includes a summary of the thesis results.

Chapter 2

Introduction to Context Maps

In this chapter we describe the background, concepts and techniques used in the development of Context Maps. We will also show how to read Context Maps.

Context Maps is defined as follows:

Context Maps is a set of sets that describe information and enable visualization. The relationship among the set members is delineated through the Map with the help of notations (as will be described later in the chapter).

Context Maps is a collection of disparate knowledge components. These components are brought together in a logical way to form a map and to present contextual information to the user.

Based on *Context Maps* technology, separate but related pieces of knowledge can be merged into one view, which could be read horizontally and vertically to elicit different pieces of information.

Context Maps allow the capture of contexts, as described in Chapter 1, as well as the modeling of these concepts, making them ideal for presenting information in groups. The application of Context Maps could include

- All phases of software development life cycle
- Software process improvement
- Software Testing and Debugging
- Software Team Management
- Risk Management and Quality Assurance
- Measurement and estimation of software projects
- Data comprehension and conversion techniques

In the next section we will discuss how to read Context Maps. We will discuss the notation set, the terminology and what constitutes the Context Maps by illustrating a Context Map used to describe the different types of team structures and the projects that best suits each type.

2.1 How to read Context Maps

2.1.1 Context Maps Terminology

Besides some changes in the notation set, which we felt were necessary to support the software testing, the Context Maps terms used in this thesis are the same as in previous works [11, 12, 21]. These terms include the following definitions, acronyms and abbreviations. We will describe these terms with an example in section 2.1.3.

- **Context Tuple**: A generic association of set members cast in roles.
- **Sets**: Sets are shown inside curly brackets "{}" in the leftmost column of a Context Map.
- **Set Member**: The members under each set are the Set Members.
- **Set Roles**: the upper case letters in spreadsheet cells.
- **Set Member Roles**: the lower case letters in spreadsheet cells.
- **Cardinality of Roles**: the amount of non empty Roles in every row.
- **Cardinality of Set Member**: the amount of set members under each set.

2.1.2 Context Maps Notations

Context Maps provide a very flexible and adaptable set of notations. This set is expandable and editable, depending on the underlying business. This is one of the benefits of Context Maps and is similar to the tags in XML, which are defined and customized according to the underlying business. During our research, which used Context Maps to create CMTCs, we added some more notations to the existing notation set of the Context Maps [11, 12, 21], to help us better capture and represent the concepts involved in software testing (like state of the system before testing starts or pre-state, input data, output data, incorrect output, severity level of a problem or bugs found during testing). The additions are listed in bold in the following table and will be described in greater detail as we will use them in chapters 4 and 5.

Some symbols can be associated with multiple concepts [12]; similarly different users can define symbols by themselves. This is because the defined regulations are flexible.

Cateory	*Name*	*Description*
Notation of sets	A	(A)ggregation of columns - context tuples
	E	- (E)dge properties
	F	- (F)low graph nodes
	L	- (L)flow graph with cycles
	N	- (N)ode properties
	V	- (V)alue
	S	- (S)equence
	G	- (G)uard
	R	- (R)esource
	O	- (O)bject
	I	- (I)dentifier
	X	- Cartesian Product
	T	**- Test Cases**
	P	**- Pre State of system (before testing starts)**
	B	Business Rule
	I	**Input Data**
	E	**Expected Output**
	O	**Actual Output**
	?	- unknown value
Notaion of set members	V	- marker
	?	- unknown value
	M	- (m)iddle of 'arrow'
	F	- tail of 'arrow'
	T	- head of 'arrow'
	B	- both f/t
	F	- (f)rom node
	T	- (t)o node
	L	- (l)oop
	B	- f/t - both nodes component
	F	- (f)rom node component

	T	- (t)o node component
	L	- (l)oop node component
	Y	– yes
	O	- otherwise
	I	- input item
	OE	**- Expected Output**
	OC	**- Correct Output**
	OX	**- Incorrect Output**
	G	**- Bugs**
	R	- (r)ead
	U	- (u)pdate
	D	- (d)elete
	X	- component of Cartesian Product
	C	- concurrence
	J	- (J)oin from fork
	K	- (K)ey

Table 2.1 Context Maps Notation

2.1.3 An Example

In order to explain Context Maps terminology and how to read the Context Maps we will present the Context Map shown in Figure 2.1. The Context Map of Figure 2.1 was developed to define the organization policy describing the relation between team structure and project type at Perfapps Solutions, a company that provides business software solutions.

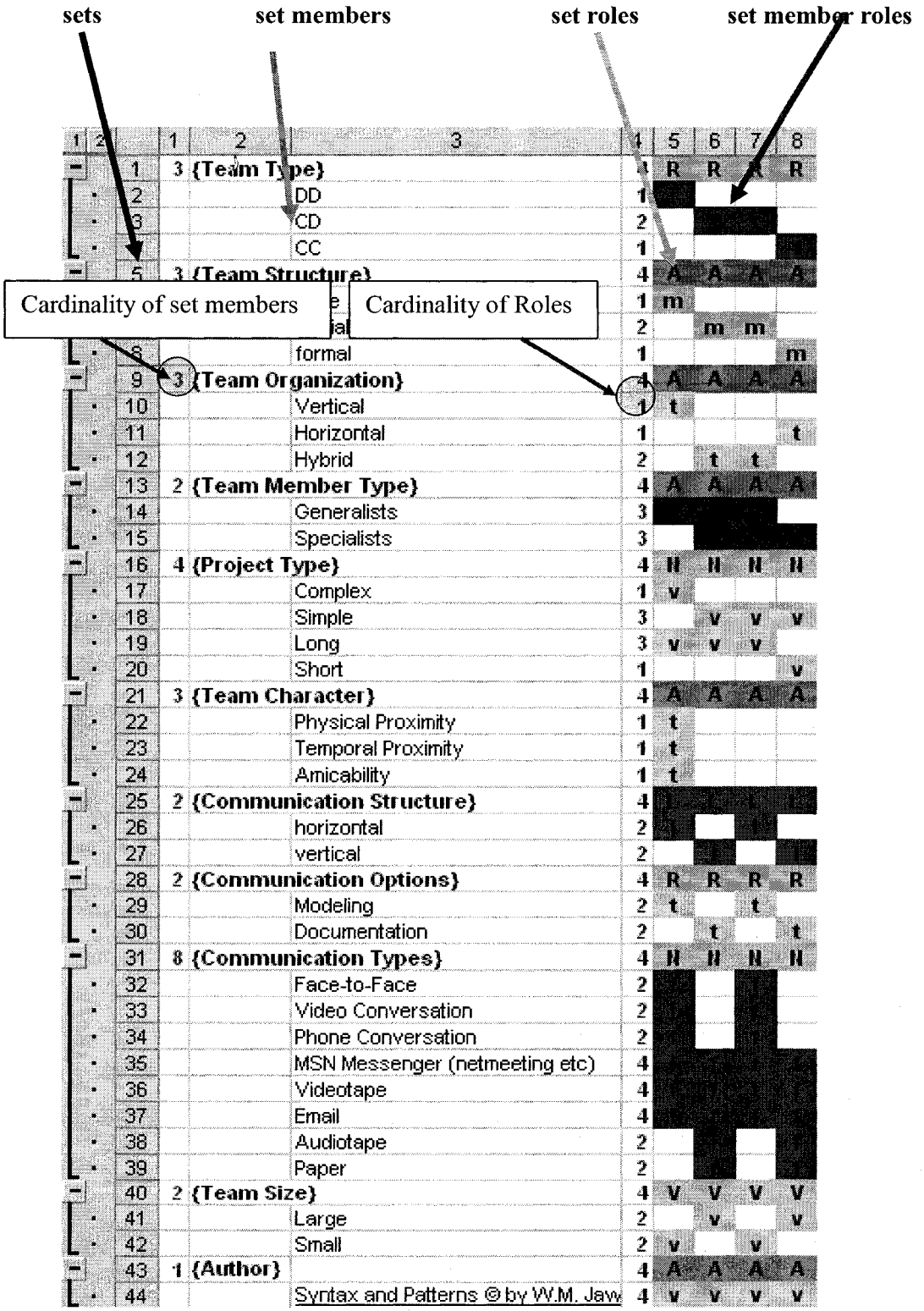


Figure 2.1 Context Maps for Software Team

1. We used the appropriate notations based on the basic elements of Context Maps notation set. The choice of which notation is to be used for a set or set member is up to the discretion of the creator of the Map.

2. We expand each set by filling in its members. The following describes the syntax of the Context Maps:

- The bold terms in column 2 within curly brackets {} are called sets, such as {Team Type}
- The elements under the set in column 3 are called set members, such as "DD" and "CC".
- The contents in columns (5-8) are called Context Tuples.
- The single uppercase capital letters in Context Tuples are Set Roles, such as the letter R, A, N, and L
- The lower case letters or digits in Context Tuples are Set Member Roles, such as f, m, t, v, and b
- The column 4 with numbers is the count of the set member roles.
- The column 1 with numbers is the count of the set members.

For every value in the spreadsheet, user can read up or down a column to see related values and towards the left of the map to find which set member the value is referring to. Now we will see how to read the Context Map in Figure 2.1.

2.1.4 Reading Context Maps

To explain how to read the Context Maps in Figure 2.1, let us consider the sets Project Type and Communication Structure.

- In Row 16, "Project Type" is represented with "N" or as a Node in the map, as mentioned earlier, it is up to the creator of the Map to decide which notation to use, much like the column names in the Database table or tag names in XML. Project Type could be represented as a resource, "R", but the creator of this Context Map chooses Project Type as a Node in the Context Map to be represented as "N".

In Rows 17-20, the set members of this set are "Complex", "Simple", "Long" and "Short" type of projects. In the map these set members are considered as values or "v". Observe that the notation for set members is in lower case and for sets in uppercase.

- In Row 25, "Communication Structure" represents iterative communication between project team members and end users to clarify the project requirements. In order to capture this iterative nature of communication, the creator of the Context Map in Figure 2.1 choose loop or "L" as a notation to represent Communication Structure in the Map.

The set member of this set are "horizontal" and "vertical", also represented as loop or "l", pointing out the iterative nature of communication whether the communication is horizontal (team members are free to communicate) or vertical (communication among member of different teams is through the managers only)

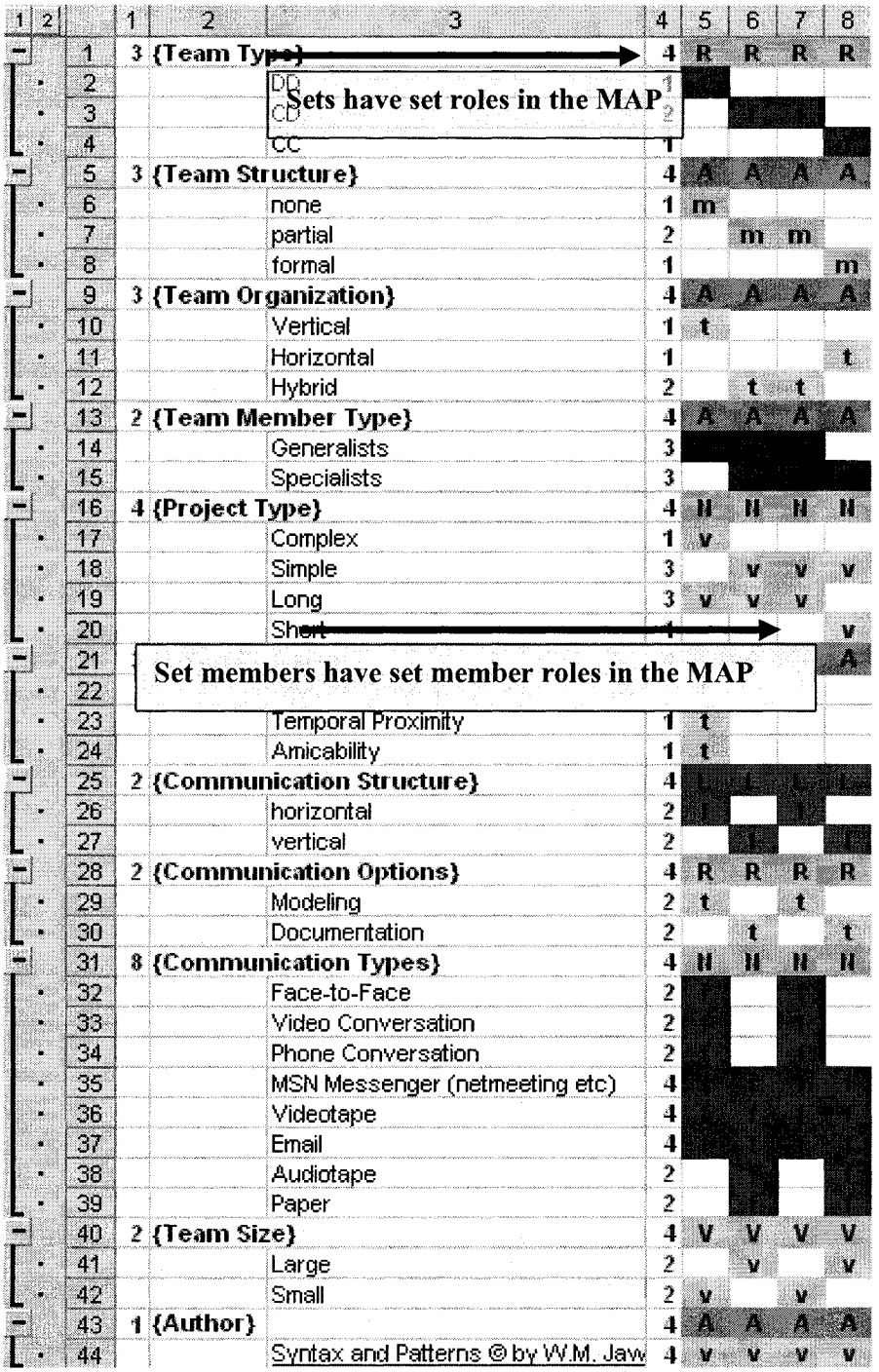


Figure 2.2 Context Maps for Software Teams

- The relation between the set members could be read by reading in the map vertically. For example, when we read the set "Project Type" and the member of this set "Complex", we see the notation "v" in this row (row # 17 of the Context Map) appears in the 5th column in the Context Map. Similarly when we read the set "Communication Structure" and the member of this set "horizontal", we see the notation "l" in this row (row # 26 of the Context Map) appears in the 5th and 7th column of the Context Map. The fact that both Project Type "Complex" and Communication Structure "horizontal" have values in the map under the same column (i.e. 5th column) shows the relation between these two sets. The relation could be read as, "when the Project type is Complex the Communication Structure is horizontal" or when it is a complex project, the organizational policy is that the communication among the team members and among different teams is allowed without any management restriction or control; this open communication will reduce the complexity of the problem. The relation between Project Type "Complex" and Communication Structure "horizontal" is further explained in the following Figure 2.3 (an excerpt of Figure 2.1).

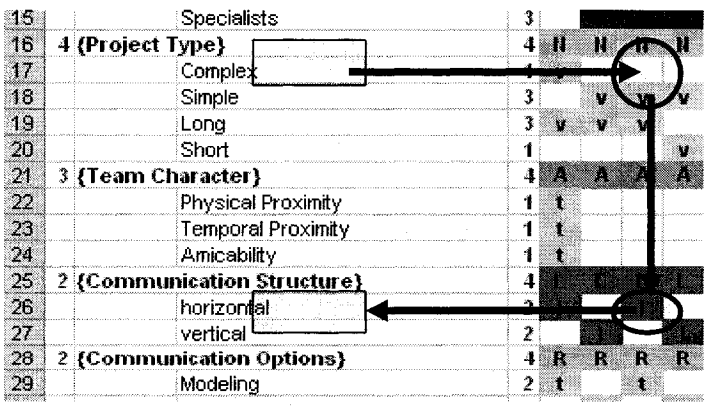


Figure 2.3 Relation between set members

- In the above Figure 2.3, the notation "v" appears only once in the row where the Project Type "Complex" appears, therefore, the Cardinality of set member in this row (row # 17) under the column 4 is 1, as shown in the following Figure 2.4. Whereas the notation "l" appears twice in the row where the Communication Structure "horizontal" appears, therefore, the Cardinality of set member in this row (row # 26) under the column 4 (of Figure 2.4) is 2. This provides another way of checking if in actuality the set member is showing these many relations to other set members in the Context Map, which helps keep the Map consistent.

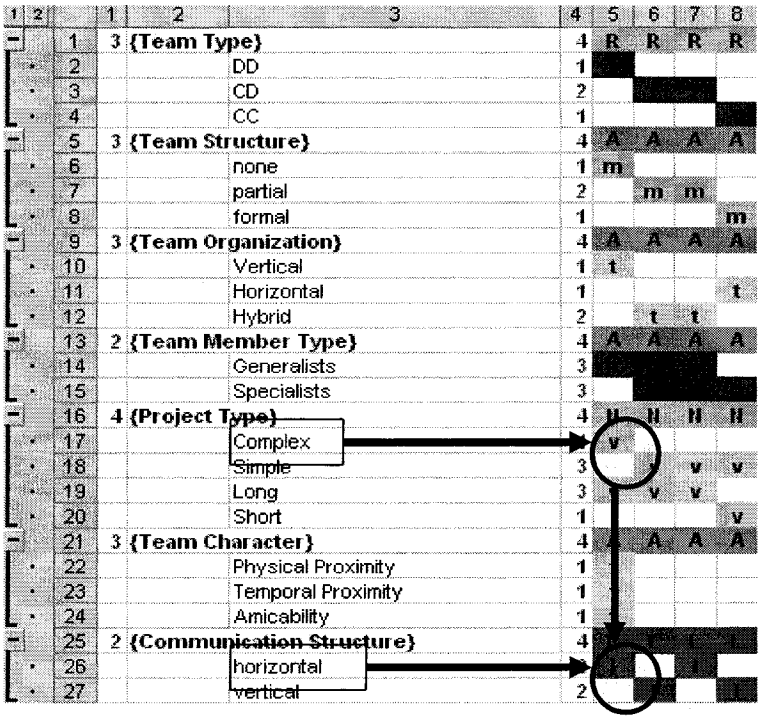


Figure 2.4 Cardinality of set members

2.2 Context Maps Test Case

In this thesis research we use Context Maps to develop Context Maps Test Case (CMTC). The developed CMTC, their comparison with other testing tools, and the benefits obtained from their usage will be discussed in Chapter 4 and 5.

The requirements for a software project are not necessarily understood in the beginning of the project. At every phase of the Software Development Life Cycle (SDLC) [14, 17] the requirements are clarified by the people who are involved. For instance, in one of the projects the programmer identified that the space to replace special characters asked by the user to use in the search criteria of a form, was creating problems when the special character is adjacent to a space itself. The programmer identified that the requirements be modified and instead of space a wild card like "%" should be replacing the space. This shows how the Requirements are clarified as we move from one phase of SDLC to another. The user acceptance testing [17] which is conducted at the end of the testing life cycle and provides validation that the resulting software fulfills the requirements, needs test case documents which are developed throughout the SDLC and across its various phases with input from the participants of each phase as the requirements are understood in greater detail.

Therefore, in user acceptance testing it is important to provide a structure for the test cases that is easy to understand. This will help the participants, not necessarily testers but also end users, programmers and managers, to fully participate and comment on the decisions taken, like choosing the proper data with which to test the application.

Since these stakeholders are not always technically equipped to handle complex tools, they might not be able to handle complex interfaces to read or change the test cases, for instance, those provided by toolsets like IBM's Rational or Borland's Silktest. The simple Spreadsheet-like structure of the CMTC and the knowledge that it imparts by browsing through it, without the need to click any tabs or open various windows, makes CMTC easy for those involved to understand the information presented through the test cases. As oppose to creating a diagram to delineate a test case or merely describing a test case, one of the benefits we observe of using CMTC is that they are not just easy to create but also easy to read and update. Moreover, the information presented in the form of sets could be related easily to other sets within the map and more information could be added easily without changing the current Map Structure.

From the management perspective, the managers get all the information related to the test they would require by simply browsing through one simple map. In our experience by virtue of CMTC, managers were able to identify the number of errors, the related module, the version or build being tested, the programmer to assign the issue for debugging, the end user or business analyst to contact for clarification of the problem. The presence of all this information in one simple map helps managers with limited time to make important management decisions without wasting time to gather information from across the various tabs of test management and defect tracking tools. The CMTC and the resulting benefits will be presented in greater detail in Chapters 4 and 5.

Chapter 3

Problems with testing and testing tools

In this chapter, we look at contextualized testing. Testing or, more specifically, user acceptance testing is considered as an umbrella activity of the software process. As such, it starts from the beginning of the software process, and involves participants from all phases of Software Development Life Cycle (SDLC) [14]. We will also discuss various existing approaches and tools for software testing. We will show how we use them during our research, and the results we obtained. We will see the problems in existing approaches and tools while performing user acceptance testing.

3.1 Testing Process

Many authors and researchers have recognized that testing should not be considered as an activity but a process, which starts with software specification and requirements gathering [2, 15, 16, 17, 22, 23, 24, and 25]. During the initial phase of a Project, that is, Requirements gathering, testing activities include determining verification approaches (black box or white box [14, 17]), generating test data and identifying a test plan which will result in clearer and fewer test case documents.

There are several testing processes specified by different researchers, Figure 3.1 shows a typical testing process.

→Specification *(acceptance test plan)*

→Design *(system and sub system integration test plan)*

→Implementation *(integration and unit testing)*

→ Testing *(acceptance testing)*

Figure 3.1 A typical testing process

The test process in Figure 3.1 shows that testing phases are dependent on SDLC phases, such as 'planning user acceptance testing', depends on 'requirement elicitation' phase of SDLC. According to these testing processes [2, 3, 4, 5, 14, and 17] during the requirement gathering the test case document for the user acceptance testing is prepared. Each testing phase has participants from the adjacent SDLC phase, who work with test team members to develop test case documents and monitor the results of testing. In user acceptance testing the end user should be able to identify the input and expected output, as well as elaborating the User Interface.

Now we will present examples of software test processes and will analyze how these processes help in user acceptance testing and the problems that need to be addressed.

3.1.1 The V Model

The V Model captures the testing process: see Figure 3.2 [15].

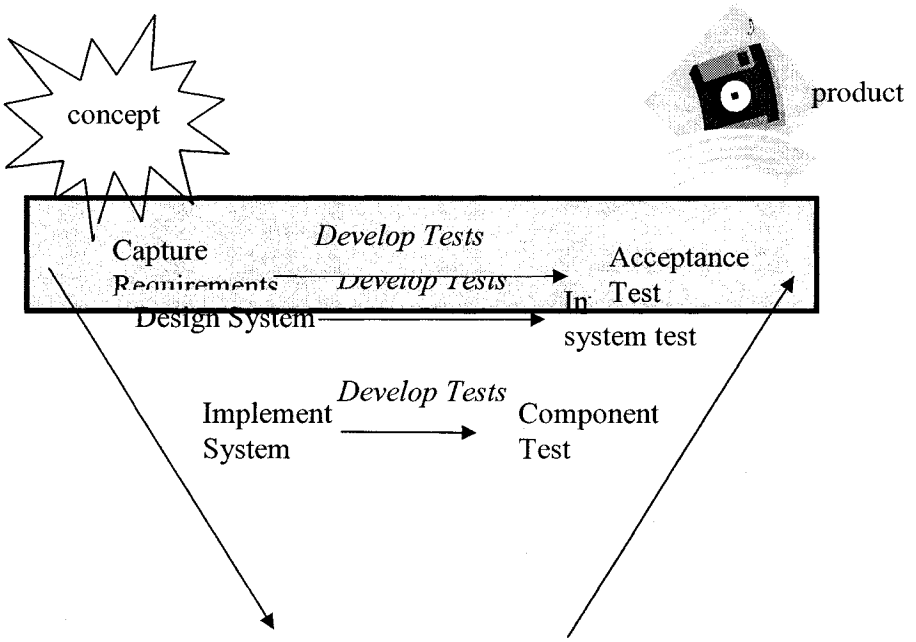


Figure 3.2 The V Model

According to Black test team is not isolated in their efforts to develop test case documents, and should work with participants from other phases as shown in Figure 3.2. The V Model and some other testing life cycle models [15, 17] suggest that user acceptance testing is carried out after the software has been developed. The model also suggests that the planning for user acceptance testing should be in the beginning of SDLC, while capturing the requirements. The planning for user acceptance testing involves activities, such as preparation of test case documents and identifying test data. Similarly, the Integration test plan is prepared during the design of the system, and the component and unit test plan is prepared during the implementation of the system as shown in Figure 3.2.

The authors of models similar to the V Model identify that user acceptance testing is an effort not only of software test team members, but also of those who are participating in capturing requirements such as end users and business analysts [2,5,17]. However, the V Model does not identify a role for developers and designers to help create user acceptance test case documents. In other words the V Model does not see the user acceptance test case planning beyond the requirement gathering phase of the SDLC and into the later phases such as Design and Implementation.

The model in Figure 3.2 shows that user acceptance test plans are developed while requirements are captured in the beginning of the SDLC. This view is similar to the deprecated waterfall model [14] where the requirement elicitation is a phase which is in the beginning of the software development process. However, just as requirements are refined and reviewed throughout the software development lifecycle in the more recent and accepted SDLC models such as the Prototyping, Iterative, and Evolutionary models [14, 17], the user acceptance test case document (which is validating the requirements against the developed software product) should also be planned and reviewed throughout the different phases of SDLC with the participation of members from each phase.

Often the developers while implementing the requirements uncover problems that could change the requirements; consequently the user acceptance test case document should change too. The business analysts and end users might ignore some details as implicit requirements or trivial. However, the testers and developers because of lack of business knowledge might not be able to readily understand certain issues; they might add more details to the requirements or in some cases even change the requirements.

In one of our experiences, we tested a search engine, which was searching for a string provided by the user. The search was against a database of customer names. The search engine (before searching for the string in the database) was replacing special characters in the string, such as "\, ", :, (, }, ?, &" with a space character. This is due to the fact that the business policy does not allow special characters in the customer names. However, there was a problem when the special character in the search string was adjacent to a space itself, for example "P & G", as this makes the search string "P G", that is, a customer name where P and G are separated with two spaces between them. To fix this problem the developer change the search criteria, and replaces "space" character used to swap special characters like "\, ", :, (, }, ?, &" in the search string with the wild card "%". In this way a string like "P & G" would be replaced with "P % G" and as such the search result would include all names with P and G within them. In this case, if the developer won't change user acceptance test plan, the test case will fail as the tester will assume that special characters are being replaced with "space" rather than "%". This example illustrates that it is important that user acceptance test case documentation should be viewed, like requirements gathering, as an umbrella activity of SDLC, and participants of any SDLC phase should be able to update or review the user acceptance test plan.

3.1.2 Testing Maturity Model

Burnstein [16, 31] suggests testing as a sub process within the software development process. She suggests that testing should be improved in levels following the paradigm set by SEI through Capability Maturity Model (CMM) [14] and proposes Testing Maturity Model (TMM) to improve the process of testing.

Level 5: Optimization

Test process optimization

Quality control

Application of process data for defect prevention

Level 4: Management and Measurement

Software quality evaluation

Establish test measurement program

Establish organization wide review program

Level 3: Integration

Control and monitor the testing process

Integrate testing into software life cycle

Establish a technical training program

Establish a software test organization

Level 2: Phase Definition

Institutionalize basic testing techniques and methods

Initiate a test planning process

Develop testing and debugging goals

Level 1: Initial

Figure 3.3 the TMM

Level three of the model suggests key process areas (KPAs) such as 'integration of testing into software lifecycle', and 'establishing a software test organization'. This shows how according to her model testing is considered an umbrella activity of SDLC, and needs to be integrated in the software life cycle.

3.1.3 Fisheye views

There have been attempts such as the fisheye views [9] to underline the importance of contexts in which a system lies — for example, its software, hardware environment and related modules. The model suggests that only by considering contextual information (that is, related modules and components) that a system could be studied properly, and even though the emphasis could be on one specific module or sub-system, but this should not lead to the isolated study of the module. However, fisheye views is focused on software analysis and design only, and suggests nothing that highlights the importance of contextualized testing.

Integration of software testing into the software lifecycle process should be viewed in the same way as fisheye views. Different phases of SDLC and their participants should be perceived as providing the contextual information for test case documents, so that software testing won't be carried out in isolation by software test team members.

3.1.4 A Standard for Software Test Case Documentation

IEEE Standard 829-1998 for software test documentation [33] defines the scripted testing approach. The Standard defines a set of eight documents for software testing; among these documents is the test case specification document. IEEE Standard 829-1998 suggests that the test case specification document should be composed of the following sections

- An identifier to uniquely identify the documents from others
- Items to be tested
- Input / Output Specifications
- Hardware / Software environmental needs
- Test case specific procedure requirements e.g. setup, execution, cleanup
- Related test cases or pre requisites, which must be executed prior to the implementation of this test case.

The IEEE standard stipulates that contextual elements such as 'Hardware/Software environment', and 'related test cases' should be mentioned in the test case document for it to be comprehensive and complete. The addition of the names of responsible people in the test case document will complement the test case further, as it will help in the traceability, debugging, error correction, and will also save time and resources.

We observed from studying projects that the IEEE Standard 829-1998 has the following very important information missing, which prevents it from capturing the contexts properly in the test case documents

- The programmer, user / business analyst, test analyst or managers have an important role respectively, in creating the above mentioned contextual elements of a test case document, **the roles are missing**.

- At what stage in the testing life cycle each of these artifacts is documented, a **temporal dimension** is missing.

- There is also a need for a tool that will help each one of the roles (mentioned in 1) to participate in identifying and documenting the above mentioned contextual or related information, in a way that the relation could be preserved and presented.

3.2 Testing Tools

Tools make methods more efficient [20]. Testing methods include documenting test case, specifying test data, reporting errors, identifying the severity of a problem, tracking defects, assigning and monitoring roles and responsibilities of those involved in testing. Testing tools help in effective and efficient completion of these testing methods. However, the efficacy of software testing tools depends on how mature the organization is in implementing the testing process. As we have seen in Section 3.1, testing is an umbrella activity in a software project. Therefore, it is important for testing tools to be accessible by not just test team members, but also by analysts, end users, and developers, for the tools to be useful. As all these stakeholders are documenting requirements and defining constraints, as discussed in Section 3.1, they should be able to verify the data used for testing by accessing the testing tools themselves.

We will now present our research on currently available software testing tools.

3.2.1 Criteria for Testing Tools

Software testing tools should be evaluated according to the following criteria [16]:

- **Ease of Use and Insertion**
- Functionality, Power and Robustness
- Quality of Support
- Cost of the tool, Organizational value

A good testing tool is one which is easy to use so those who are involved in testing could use the tool without the help of expert users. This is more important in small customization projects for defect fixing and small enhancements, as users do not have time to learn complex tools. A good testing tool also helps user to insert and retrieve data easily from test case documents. This will help members from other phases of SDLC, including Business Analysts, End Users, Programmers, and Managers, to add their input, as well as review the contents of test cases. This inter-phase communication while developing the test cases, will improve chances of finding errors and testing remote and hidden scenarios.

3.2.2 Defect tracking tool (Rational ClearQuest)

Defect tracking tools log and keep track of defects and their status [16]. We studied Rational ClearQuest as a defect tracking and test case management tool along with Rational Test Manager as a test case authoring tool. In the company we worked

Rational ClearQuest was used to keep track of test cases being tested by Development Support and User Acceptance test teams, for small enhancements and defects.

Rational ClearQuest allows monitoring the status of the test, as to how much of it is has been completed, who it is currently assigned to, and who passed or failed it, along with other related information and test cases. It also helps in identifying the importance of the test case in terms of severity and priority of the issue, as shown in Figure 3.4 and Figure 3.5.

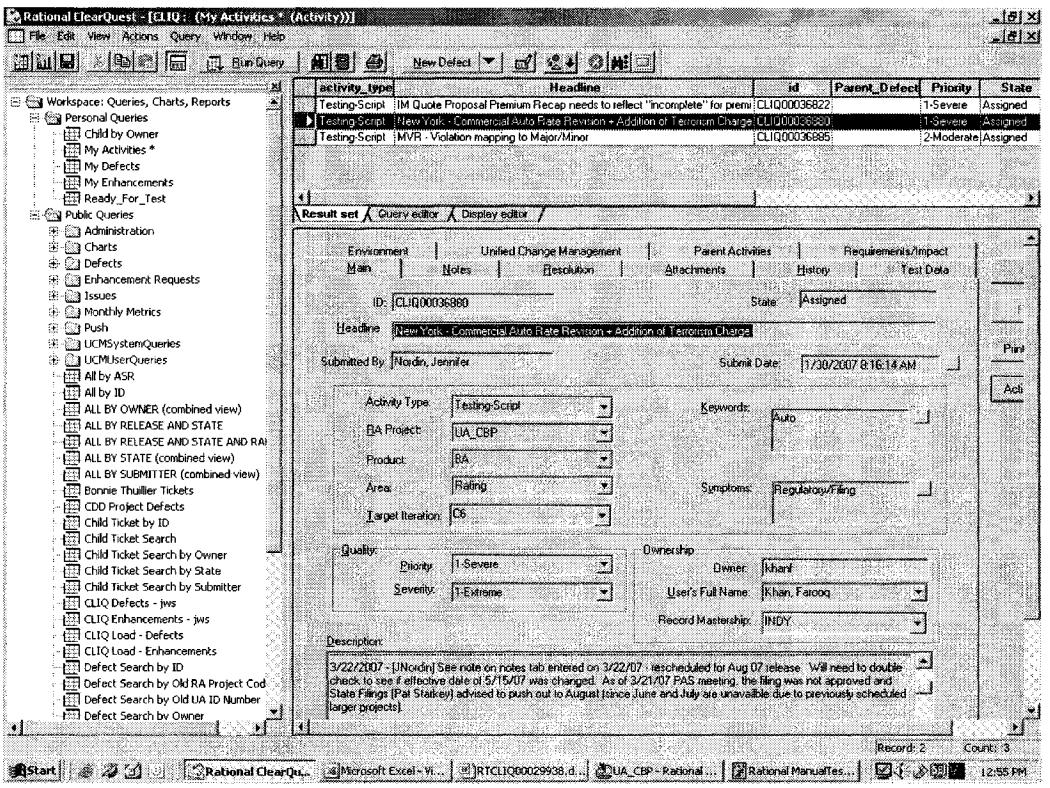


Figure 3.4 Rational ClearQuest – the Main tab

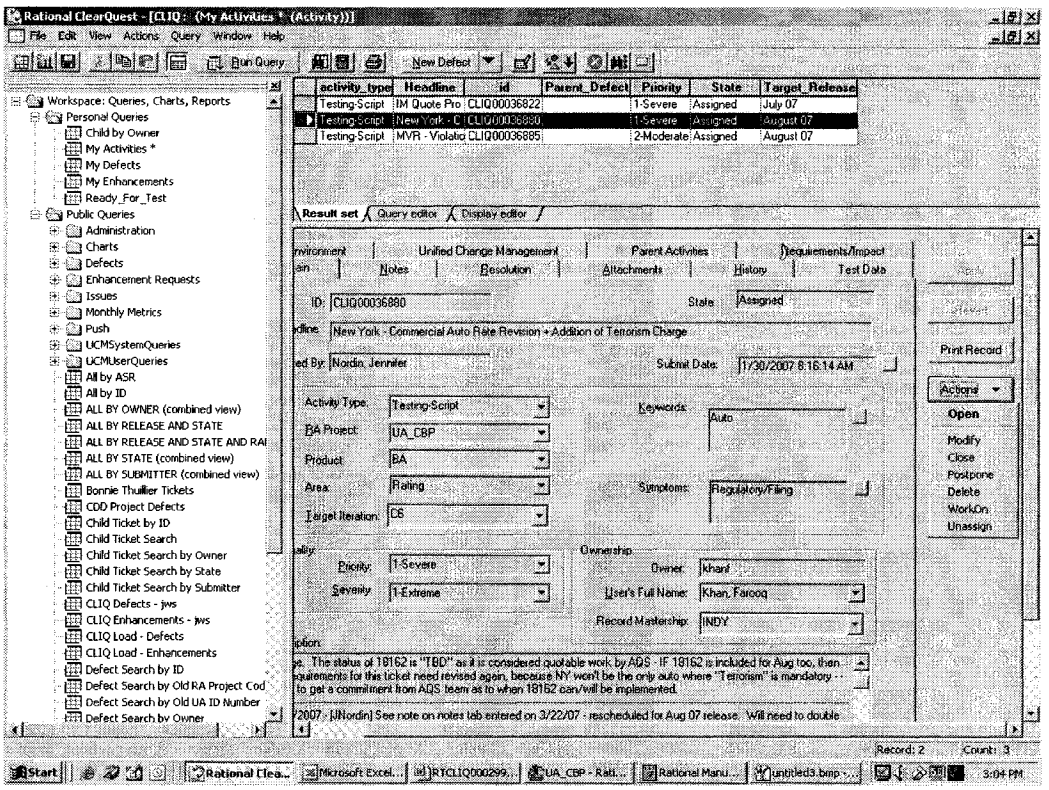


Figure 3.5 Rational ClearQuest – the Actions dropdown

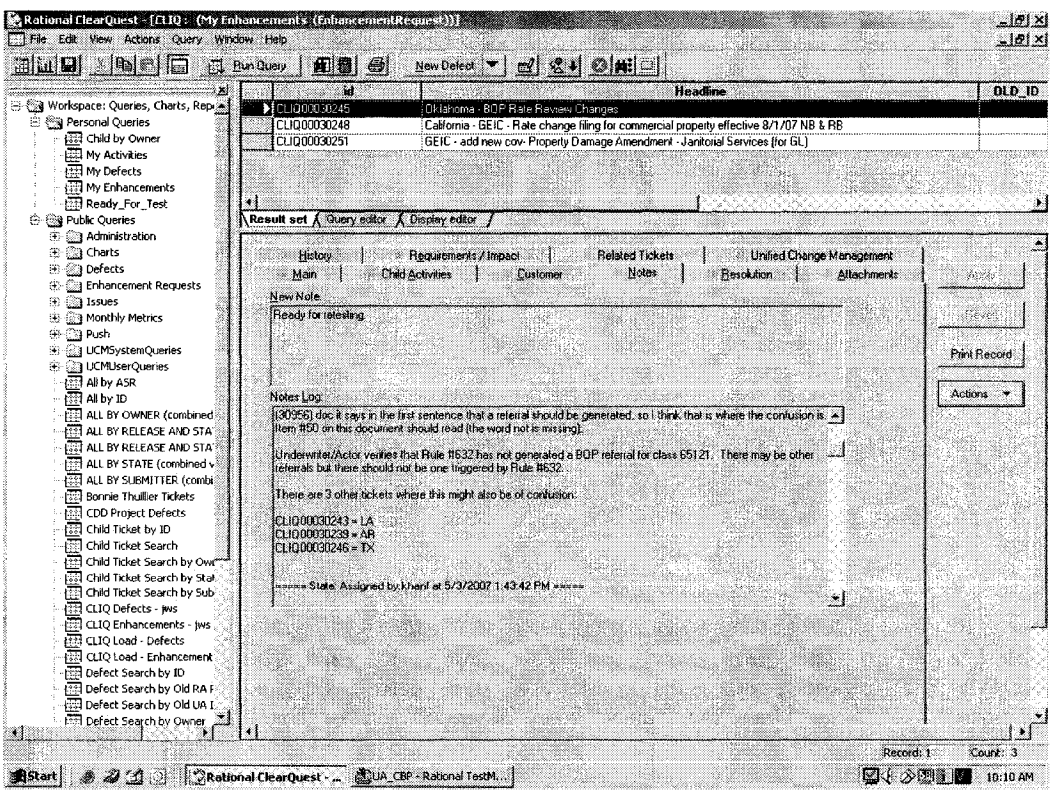


Figure 3.6 Rational ClearQuest – the Notes Tab

Rational ClearQuest also helps in documenting and relating information in hierarchic way by categorizing test cases as parent and child test cases as well as by using tabs within a test case as shown in Figure 3.6. Related documents could be associated through 'Attachments' tab. Reports and Requirement Specification documents could be added by Business Analysts, usually using some other tool such as Rational RequisitePro. All these features and tools help the user to look at the test case within its context, and also enable the user to drilldown or rollup, according to the amount of information user needs. However, the information is dispersed across the tabs or property pages, and it is difficult to compare and view the information across these different pages and tools, especially in small projects where there is no need to present information in such a scattered way. Moreover, the test cases themselves are documented using other Rational tools, such as IBM Rational Test Manager or IBM Rational Manual Tester, and, in the case of automated testing, IBM Rational Functional tester. This means that comparison with the requirements is very difficult because of the involvement of various tools. Similarly, because of the involvement of different tools (and within each tool different tabs and property pages) to capture related information, it won't be easy for the test case or the result of the test case to be associated with the programmer who has developed the program being tested, or the business analyst who has specified the requirements for which the program is developed. This means that traceability and error tracking (in case of the test case resulting in identification of a bug) to the source would involve considerable effort. Moreover, managers need information such as 'identifying or assigning the resources needed for testing or defect fixing', 'assess the severity or priority of an issue', and 'history of related defects' without depending on the test team to provide them with

this information. However, because of the information spread across multiple tabs and tools as shown in Figures 3.4, 3.5 and 3.6, it is difficult for the managers with little time to extract the information they need from tools such as Rational ClearQuest. In Chapter 4 we will see how these problems could be resolved using CMTC.

3.2.3 IBM Rational Manual Tester

The IBM Rational Manual Tester covers many aspects of the Testing Maturity Model (TMM) [16], as it helps organizations to institutionalize basic testing techniques and methods, and help them control the testing process as well as help them in integration of testing in software life cycle. The following study specifies the various aspects of IBM Rational Manual Tester (RMT) [19] as a manual testing tool, and in the next chapter we will compare its functionality with the CMTC.

The IBM Rational Manual Tester provides the following features,

- *Test planner*
- *Requirement recorder*
- *Requirement to test tracer*
- *Test case reuse*

We used the tool in the testing of a Repair management system while working for a software company. The following screenshots explain some of the features of RMT.

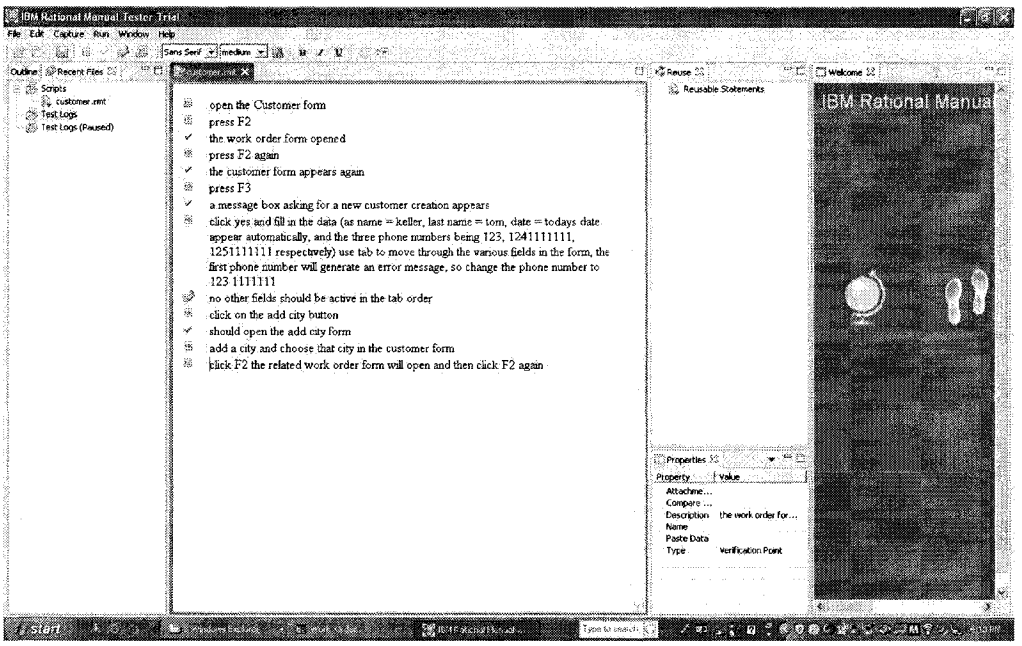


Figure 3.7 Test case for Customer form

Figure 3.7 shows how IBM Rational Manual Tester helps in classifying the various statements in the test case as steps, verification points, or reporting points.

In test case preparation using IBM Rational Manual Tester, a *step* is merely an action to be completed during the test, whereas the success or failure of the test depends on the successful execution of a verification point. The verification point makes it possible for the user to easily compare the expected output with the actual output and to observe if the verification passed or failed. A reporting point becomes part of the final report at the end of the test case execution. Rational Manual Tester adds to the report the verification point which results in identification of a bug. In this way the Rational Manual Tester provides some contextual information about the error found, such as the input data along with some additional information in the report about the bug. Testing team members have to communicate with the business analysts and end users to find out what exactly needs to be verified to identify steps as verification points. Also the test team members need to communicate with the stakeholders and

business executives to find out what needs to be reported as the reporting point. In this way RMT ensures communication between test team members and members from other phases of SDLC as well as integration of testing in the SDLC, as required by level three of TMM (discussed in 3.1.2).

Another problem in using the Rational test suite for small projects is that Rational Manual Tester and ClearQuest store information in different formats. The requirement documents in ClearQuest are not stored as steps, verification points, and reporting points as the test case documents are in Rational Manual Tester. Therefore, it is difficult to map input and output data used during testing with the requirements using Rational Manual tester, especially in small customization projects for defect fixing and minor enhancements.

Figure 3.8 shows how IBM Rational Manual tester helps in test case execution. As it allows execute the steps in the test case with the application execution. This helps in visualizing and comparing the expected output with the actual output behavior of the application.

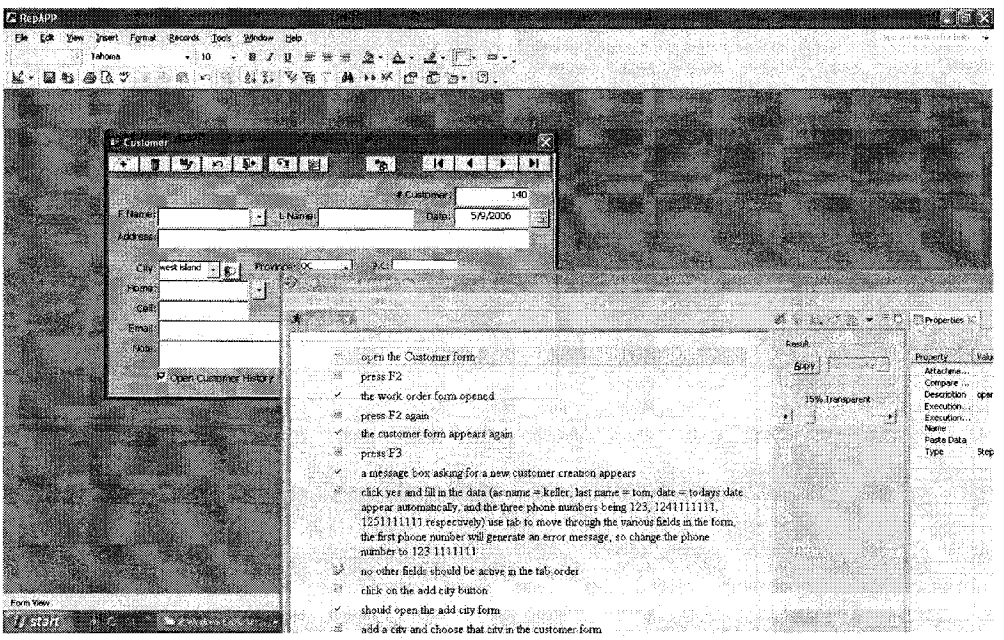


Figure 3.8 executing the test with the application

Figure 3.9 shows a test case that was created to test a module related to a customer search.

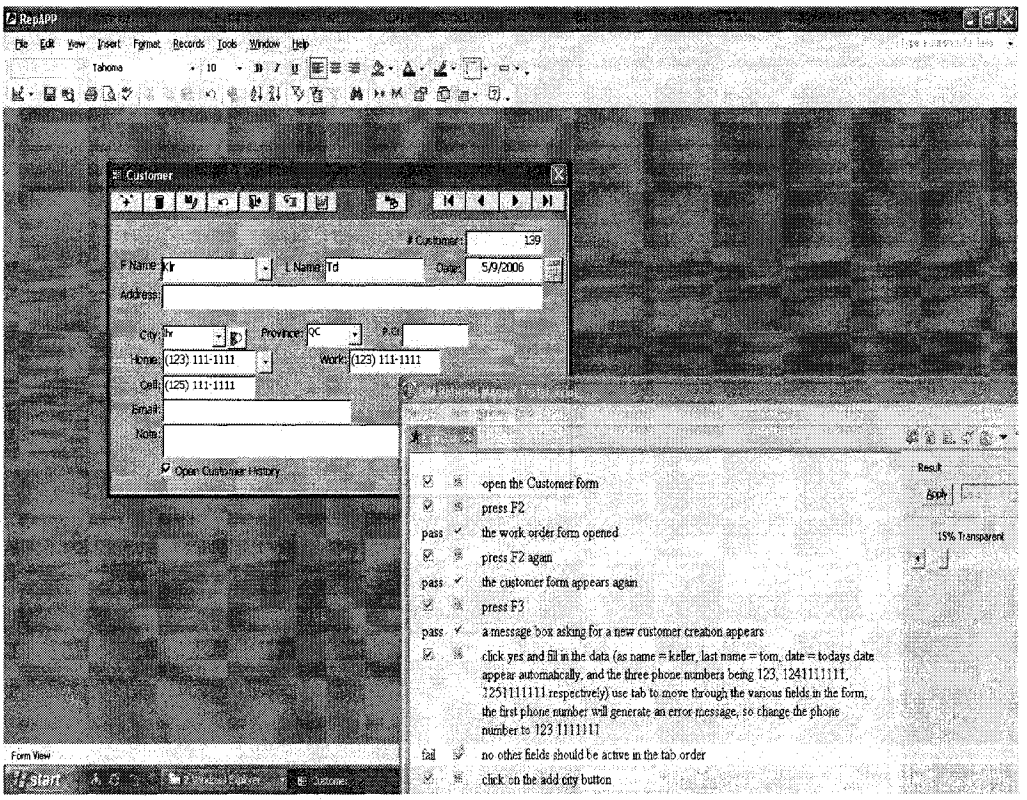


Figure 3.9 test case for searching an existing customer

The RMT also generates a report of the test case showing the bugs or assertions, as shown in figure 3.10. The report gives a brief description of the test and highlights the steps or verification points which cause the test to fail, in case of a failure. These reports are kept in the log for future reference.

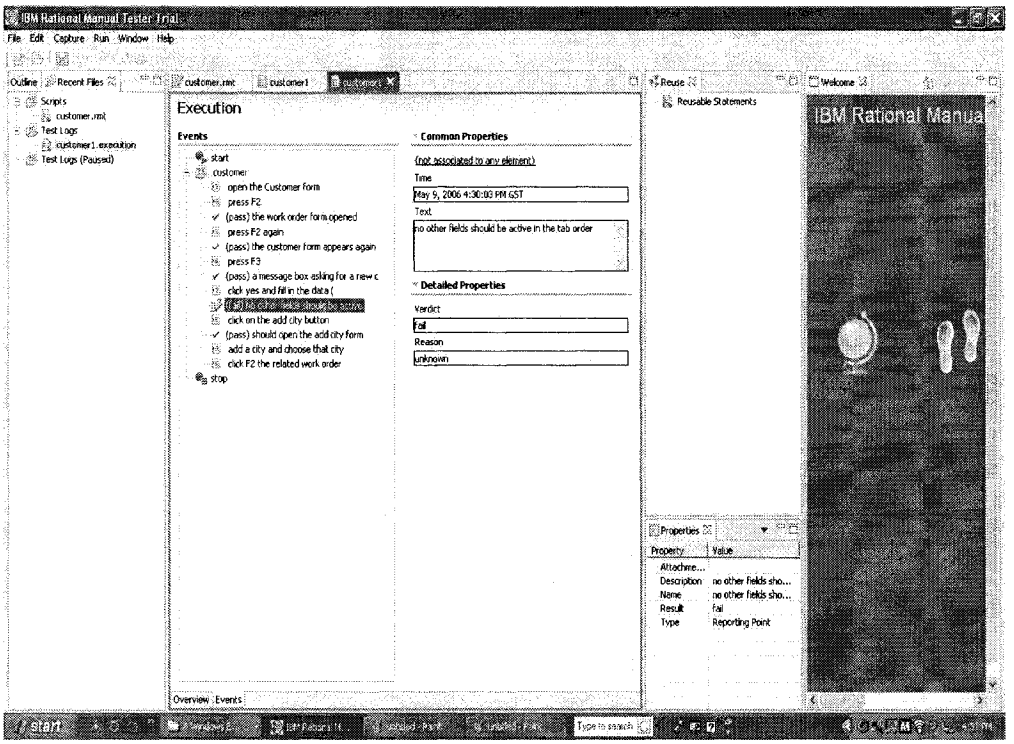


Figure 3.10 test case result summary

The result report also mentions the reason for the failure, as well as the step where the bug appears so that locating the bug would be easy.

RMT helps in reuse of the test cases, as well as documenting the results or actual outcome of the tests and matching them with the specification and expected outcome. However, the RMT provides little to link test cases to other relevant aspects of the project, including developers, testers, modules, and other test cases, in a way that is easy to understand. It does, however, provide an option of adding information through the property window with each step in the test case, by adding details in the description textbox of the property page as shown in Figure 3.10.

In short, RMT is a tool that helps capture related information to support contextualized testing with the help of tools such as Rational ClearQuest; however,

there is no direct effort to include the context in the test cases, or to motivate the test case designers to do so. The problem would be more severe in integration testing, as in integration testing different test cases need to be compared for better understanding of the testing environment, to do so with RMT multiple screens or documents need to be analyzed. We find the following weaknesses in our application of RMT

1) It is difficult to identify responsible people, for example, the programmers, or to identify sources of error, for example the failing module, without checking related details using another defect tracking tool.

2) To reuse the test cases is also a problem. Past test case scripts could be reused by including them into the test case script. However, the test cases could not be categorized or identified based on some grouping criteria, for instance, test cases for 'form validation routine'.

3) It needs a defect tracking tool such as ClearQuest to track or manage the test case. Rational ClearQuest was discussed in section 3.2.2.

In chapter 4 we will compare IBM Rational Manual Tester with CMTC.

3.2.4 Defect tracking tool (Tenrox)

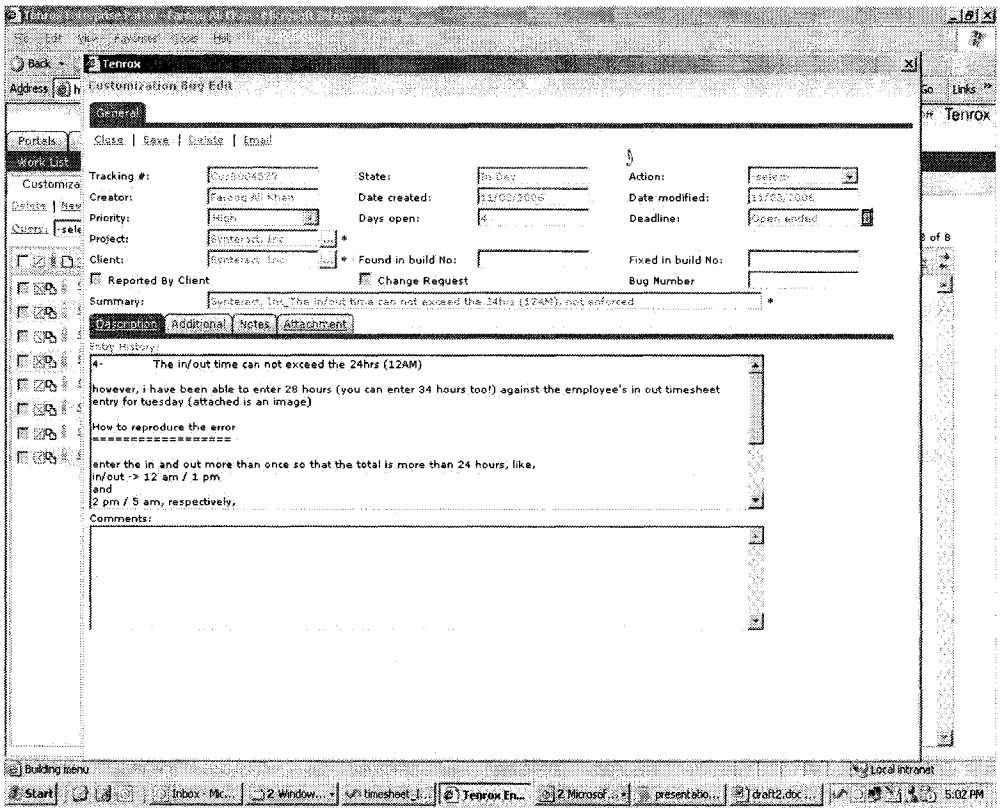


Figure 3.11 Tenrox – Defect Tracking Software

Tenrox defect tracking software [34] allows management of test cases, and also some contextual information to be preserved in the test case. This information includes the creator and other stakeholders of the test case, the version related information of the module being tested, and timelines. It also allows associating related documents to the test case as well as adding notes. However, one drawback of the Tenrox test case creation and management tool is that it is difficult to compare input and output data, other than the status changing to pass or fail for the entire test case. In order to write the test case document some other spreadsheet such as Excel is used. We will compare Tenrox with CMTC as a defect tracking and testing tool in chapter 4.

Chapter 4

CMTC as a testing tool

In the previous chapter we looked at various software testing tools and we discussed our application of these tools. In this chapter we will present CMTC as a tool for software test case management and authoring. Furthermore, we will compare CMTC documents with the tools we discussed in the previous chapter.

4.1 CMTC and the Testing Process

There are several testing processes specified by different researchers, Figure 4.1 shows a typical testing process [17].

→Specification *(acceptance test plan)*

→Design *(system and sub system integration test plan)*

→Implementation *(integration and unit testing)*

→ Testing *(acceptance testing)*

Figure 4.1 A typical testing process

The phases in Figure 4.1 are further described in *Table 4.1.*

Development Phase	**Testing Activity**	**Participants/Stakeholders**
Specification	Acceptance testing plan	Analyst / Domain Experts
Design	Integration test plan	Designers/SystemArchitects
Implementation	Unit testing	Coders / Developers
Testing	Acceptance testing	Installers / End Users

Table 4.1 A typical testing process along with stakeholders

We applied CMTC while testing various projects. CMTC documents were helpful for all the participants of the testing process. Participants of the testing process are described in Table 4.1. The participants were able to understand the test cases, add information, and manage testing easily with the help of CMTC documents, as we will see later in this chapter. The CMTC documents presents all aspects of testing including test data, test results, severity of errors, failures discovered, the names of people involved and their roles, in a single screen. As a result of this presentation, the final CMTC documents were used as reports of the testing activity. The presentation of information in one screen also helped the management to make more informed decisions, according to our findings this is especially the case in small customization projects where managers have little time to browse across various tools to gather information. Figure 4.2 shows the CMTC that we used to test a customization project.

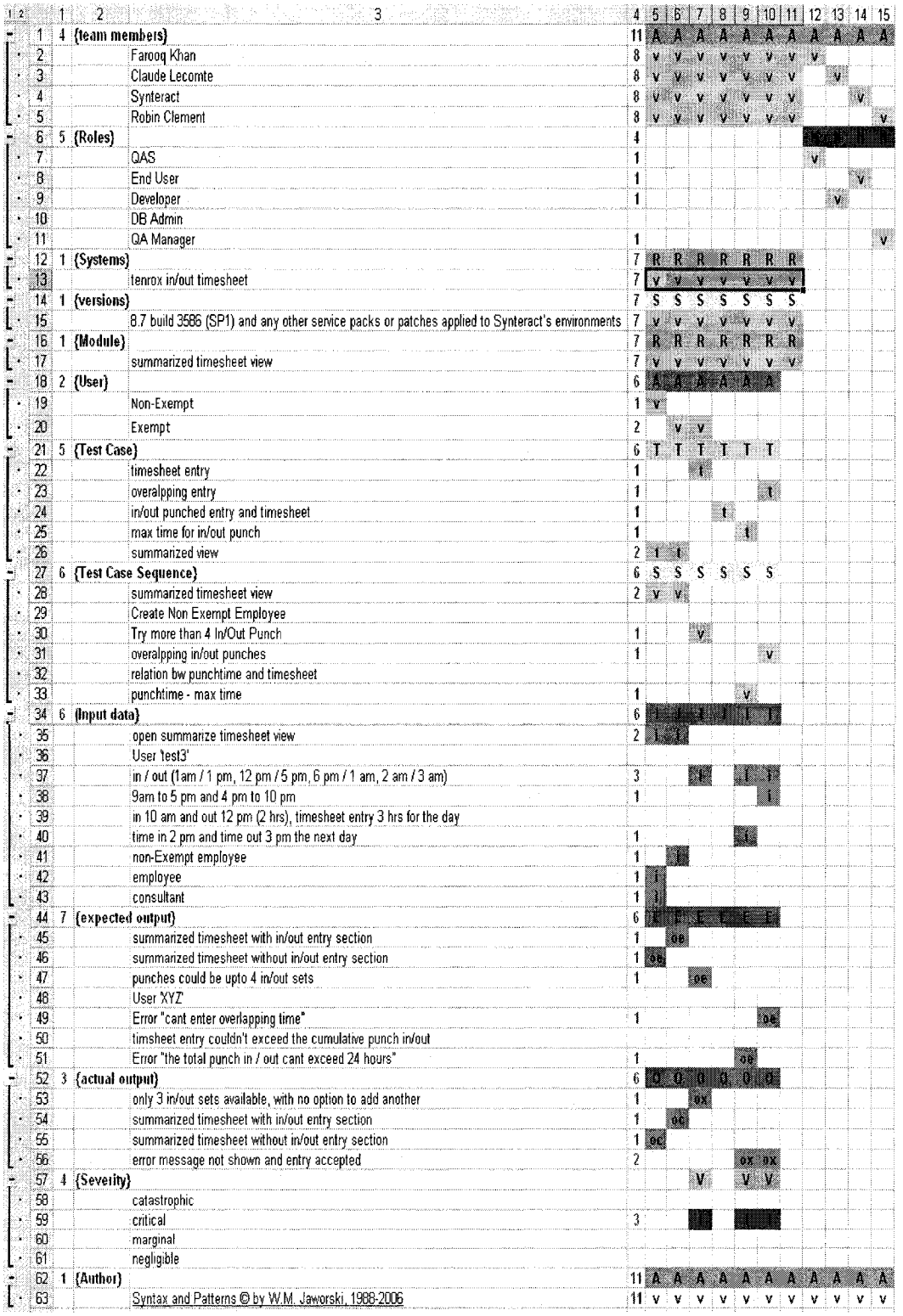


Figure 4.2 CMTC used at Tenrox to manage and document the testing on a project

The CMTC in Figure 4.2 models elements of software test case document as sets, and represents the relationship that exists among the members of these sets with the help of notations in the map. The concepts used and how to read Context Maps was described earlier in Chapter 2 of this report.

The CMTC in Figure 4.2 represents the sets 'System' and 'Module' as resources, represented by 'R', emphasizing that the system being tested is a resource to the organization. Modules are also represented as resources. Similarly, the CMTC represents 'versions' of the software being tested as Sequence or 'S', emphasizing at the existence of a sequence in the release of various versions. The CMTC in Figure 4.2 uses some new notations introduced to support testing, discussed in Chapter 2. The set 'test case' is represented as 'T', and the input data, expected and actual output for the test cases are represented with 'I', 'E' and 'O' respectively. There is a sequence that exists between the various members of the input and output data sets, and as such they could also be represented with 'S'; it is up to the discretion of the author of the CMTC document to choose the most appropriate notation. Similar to the sets in the CMTC the Context Tuples or the set members are represented with symbols or notations too. In order to make it easy for the reader, the notation set used for set members is without capitals, and for sets the notation set is with capitals.

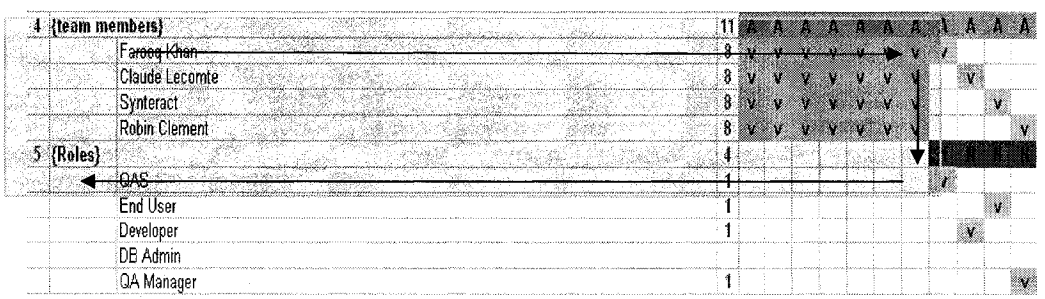


Figure 4.3 CMTC describes the people and their roles as part of the test case document

To read the relation between the members of the sets, the reader is expected to browse vertically down the column, and match the notations used, as shown in Figure 4.3. In this figure, when we compare the set "team members" with the set "Roles", we observe that "Farooq Khan" from the set "team members" is related to the member "QAS" of set "Roles", indicating that his role in this testing effort is of QA Specialist. The set "actual output" identifies the correct output from the incorrect output. The notation "oc" is used to represent correct output, while "ox" is used for incorrect output. It is easy to observe the relation that exist among the members of sets 'Module', 'test case', and 'test data', and in case of an error or incorrect output, 'the severity of the error'. The presentation of all this information in one single screen is useful for management of small projects. It helps managers to identify and allocate resources to fix a problem and to estimate the severity of the problem without having to browse and extract information from various tools and their tabs.

1	{Systems}		7	R R R R R R R
		tenrox in/out timesheet	7	v v v v v v v
1	{versions}		7	S S S S S S S
		8.7 build 3586 (SP1) and any other service packs or patches applied to Synteract's environments	7	v v v v v v v
1	{Module}		7	R R R R R R R
		summarized timesheet view	7	v v v v v v v
2	{User}		6	A A A A A A
		Non-Exempt	1	v
		Exempt	2	v v
5	{Test Case}		6	T T T T T T
		timesheet entry	1	t
		overalpping entry	1	t
		in/out punched entry and timesheet	1	t
		max time for in/out punch	1	t
		summarized view	2	t t

Figure 4.4 CMTC documents and helps relate the test case related information

The managers also find all the information, such as the system, build, and module being tested as part of the test case as shown in Figure 4.4. The information is available to the managers along with the input data used for testing and output

behavior of the system in the same screen. This is very helpful in small customization projects with limited time for management to analyze the situation and make decisions. This simple presentation of data is in contrast with the Rational test suite, which we studied in the previous chapter.

In Rational test suite the information, such as the names of participants, responsibilities and ownership, history, data used, and results of the test are dispersed across various tabs and tools. In order to elicit the information from Rational test suite, managers must browse and extract information from these different tools and tabs. This makes comparing information for decision making even more difficult. The problem with Rational test suite is even more acute in situations in which different test cases need to be compared, as multiple sessions need to be opened to compare the data. This problem of comparing various related test cases in small customization projects is mitigated by the use of CMTC, as shown in Figure 4.2, which presents multiple related test cases in one CMTC document.

The CMTC supports developing test cases gradually, and throughout the testing lifecycle. For instance, in the project planning phase the team members and their roles are specified in the CMTC document. During the requirements gathering the input and expected output data is described and is added to the CMTC document. During the design and development the actual output is added to the CMTC document. If there is a discrepancy between the expected and actual output then the severity level of the problem is assigned and is added to the CMTC document. We find that throughout this process, because of the consistent set based structure of CMTC, it is easy for the participants to read and add information to the test case documents. The result or the final CMTC as shown in the Figure 4.2 helps in reporting all important aspects of the testing effort.

CMTC supports KPAs introduced in the following Testing Maturity Model (TMM) by Ilene Burnstein.

Level 5: Optimization

Test process optimization

Quality control

Application of process data for defect prevention

Level 4: Management and Measurement

Software quality evaluation

Establish test measurement program

Establish organization wide review program

Level 3: Integration

Control and monitor the testing process

Integrate testing into software life cycle

Establish a technical training program

Establish a software test organization

Level 2: Phase Definition

Institutionalize basic testing techniques and methods

Initiate a test planning process

Develop testing and debugging goals

Level 1: Initial

Figure 4.5 the TMM

At level 3, the TMM specifies control and monitoring of the testing process. The CMTC in Figure 4.2 helps the participants to easily view the test cases as well as helping them to update the test cases, in this way giving the participants control of the

testing process. Figure 4.2 shows how CMTC helps visualize the relation between input data and output data along with monitoring the difference between the expected and actual output. The CMTC also shows test case related information in the same document, where the input and output data is presented. All this information in the same CMTC document makes management and monitoring of the testing process easy, as it eliminates the need for the user to browse across various tabs. This simple presentation is important in small customization projects where the managers due to time constraints find it difficult to browse and search for information.

The level 3 in Figure 4.5 specifies the integration of testing in the software process as a KPA. Because of the intuitive set based interface of the CMTC, the CMTC that we discussed in Figure 4.2 makes it possible for participants from other phases of SDLC to develop the test case document. This participation ensures integration of testing in the software process. During project planning those involved in planning could add information in the CMTC document as shown in Figure 4.3, where the project team and the roles were described. Similarly, in later phases such as requirement specification the data used for testing is added to the CMTC. In this way CMTC with its simple set based structure where information is added as sets and set members, provides tool based support to integrate testing into the software process. Moreover, because of the same set based interface used throughout the software process to develop the CMTC document, the test case is readable by those involved and also easy for users to add information to; this is in contrast with the Rational test suite where different tools are used for software test case authoring, defect tracking and test management. Rational test suite was discussed in Chapter 3.

4.2 CMTC and Organization Test Planning

We also use Context Maps to document and identify test planning policies. Figure 4.6 identifies the organizational rules suggesting which type and structure of team is ideal for a project, depending on project characteristics. The Context Maps in Figure 4.6 specifies the communication structure to be used for a team type [10, 32]. This is helpful because unlike the Rational Tool Suite, we studied in Chapter 3, the same tool is used for planning, policy specification and test case documentation. This consistent interface helps stakeholders and participants to easily understand all related aspects of testing.

	1	2	3	4	5	6	7	8
1	3	{Team Type}		4	R	R	R	R
2			DD	1				
3			CD	2				
4			CC	1				
5	3	{Team Structure}		4	A	A	A	A
6			none	1	v			
7			partial	2		v	v	
8			formal	1				v
9	4	{Project Type}		4	N	N	N	N
10			Complex	1	v			
11			Simple	3		v	v	v
12			Long	3	v	v	v	
13			Short	1				v
14	2	{Communication Structure}		4	L	L	L	L
15			horizontal	2	I		I	
16			vertical	2		I		I
17	2	{Team Size}		4	V	V	V	V
18			Large	2		v		v
19			Small	2	v		v	
20	1	{Author}		4	A	A	A	A
21			Syntax and Patterns © by W.M. Jaworski, 1988-2006	4	v	v	v	v

Figure 4.6 Context Maps showing organizational team structure

Figure 4.6 defines the organizational policies related to software team structure. The sets in the Context Maps of Figure 4.6 explain the important aspects to be considered while forming a test team for software projects. These sets include Team Type, Team Structure, Project Type, Communication Structure and Team Size. Team type is considered as a resource while planning for the project, it is represented with an "R", the members in this set are shown to be closely related to members of other sets and are represented with "f" or the arrow tail. Team Structure is shown as an aggregate of team members and is represented with "A"; its members are simply considered as values and are represented as "v". We can already read the relation among the set members of Figure 4.6 and the organizational policies regarding team formation with these few context tuples. The "DD" (democratic decentralized) team type when used has decentralized team structure, contrariwise, the "CC" (controlled centralized) team structure has a formal team structure. Set member from both these sets can be related to Project Type represented as a node "N" in the above Map. For instance, it can be easily discerned by the reader that DD team type with no formal team structure is ideal for complex or long Projects. Since complex projects would require discussions, as oppose to simple or short projects where the discipline of CC team type and a formal team structure would be beneficial. Also the CMTC in Figure 4.6 shows that the DD team type with no team structure and complex project is best suited for a communication structure which is horizontal. The communication structure is represented with "L", specifying iterative communication between participants. The use of Context Maps for both organizational policy documentation (as in Figure 4.6) and for CMTC documents, makes it easy for the participants in the testing process to understand the policies as well as the test cases without learning different tools.

4.3 Testing Tools

We have applied various software testing tools while testing in different organizations as discussed in the previous chapter. Now we will discuss our application of the CMTC as a test case management and authoring tool and compare it with other tools.

4.3.1 Comparing CMTC with Defect tracking and test case management tools

Tenrox

The Timesheet customization project for Synteract involves extending the normal timesheet module, so that the in/out combinations could be preserved within the timesheet. The timesheet module needs to follow certain business rules, such as the total number of hours should not exceed 24 for a given day, overlapping time entries should be prevented, certain number of in/out combinations should be allowed for a given timesheet entry date. We used Tenrox's defect management software [34] as well as CMTC to manage and document the test cases for this project.

Figures 4.7 and 4.8 show the software permitted a timesheet entry of more than 24 hours for a single day. This behavior of the software violates the business rule that the total time for a given day should not exceed 24 hours. Another business rule suggests that the time out should not exceed 11:59 pm, as at 12 am a different day will start, which stipulates a different timesheet entry for the new day.

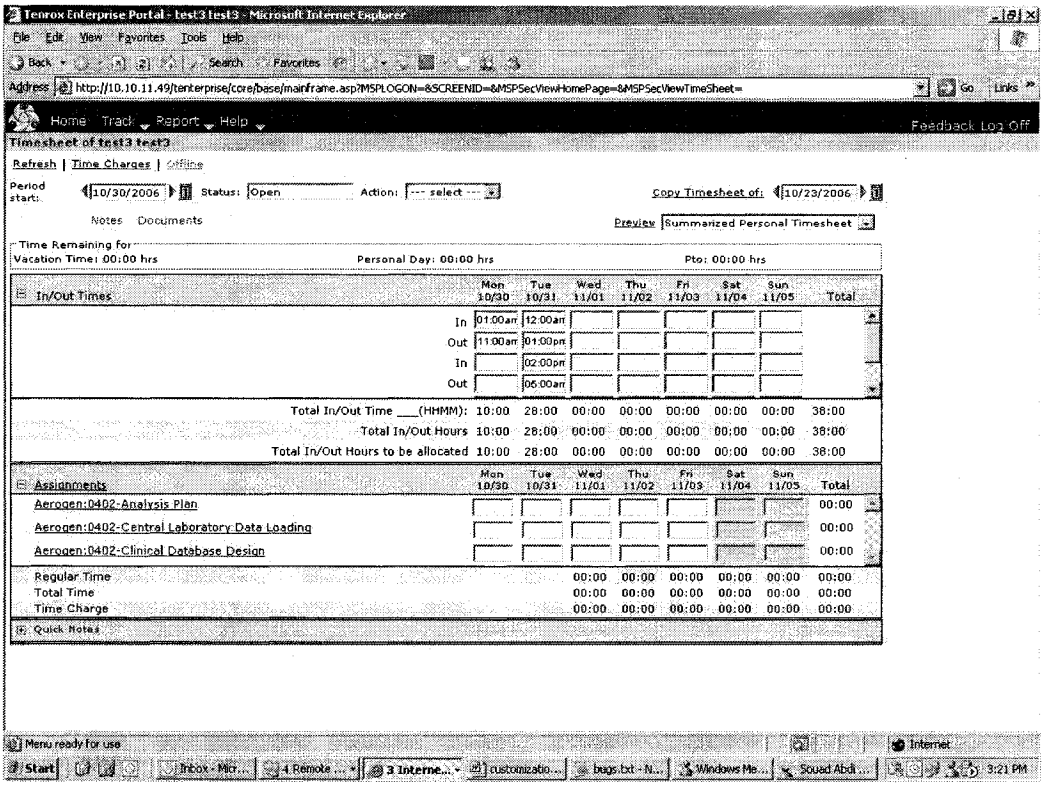


Figure 4.7 Synteract Customization testing

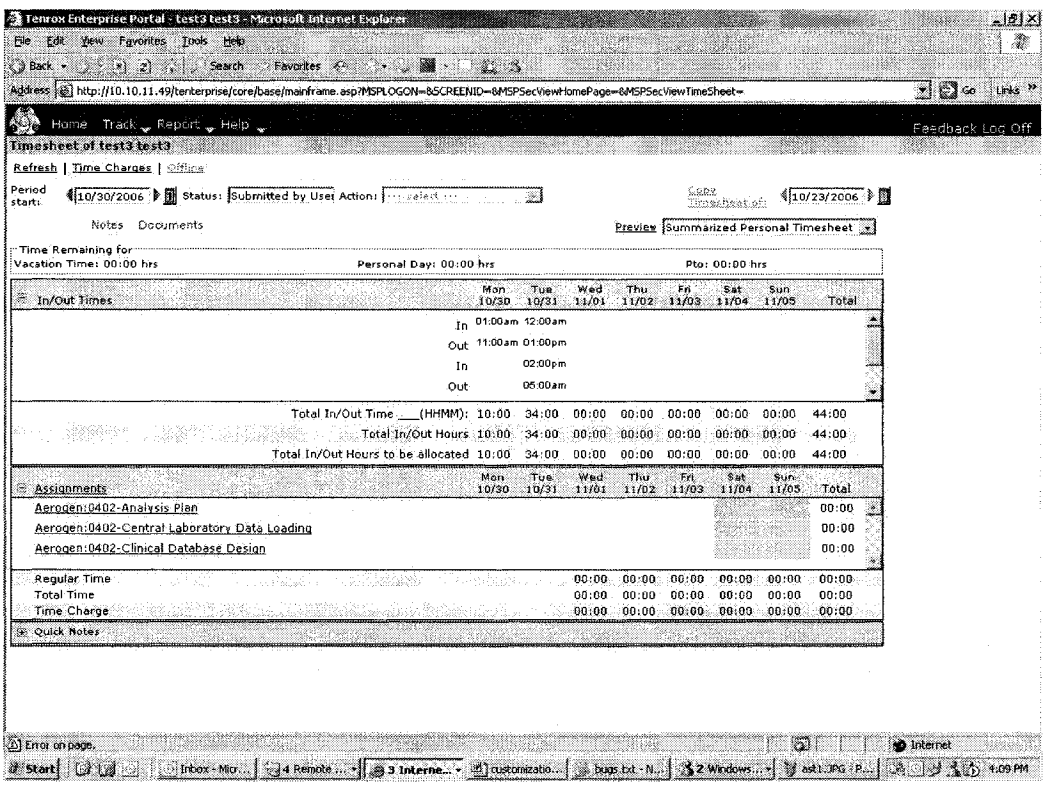


Figure 4.8 Synteract Customization testing

Figure 4.9 shows the existence of another bug in the software, one which allows overlapping in/out timesheet entries for a single employee. This Figure shows the first set of in/out time entry ends at 9 pm, whereas the second set of in/out time entry starts at 5 pm (which is before the end of the first time entry). This is clearly an abnormal behavior of the software and needs to be corrected.

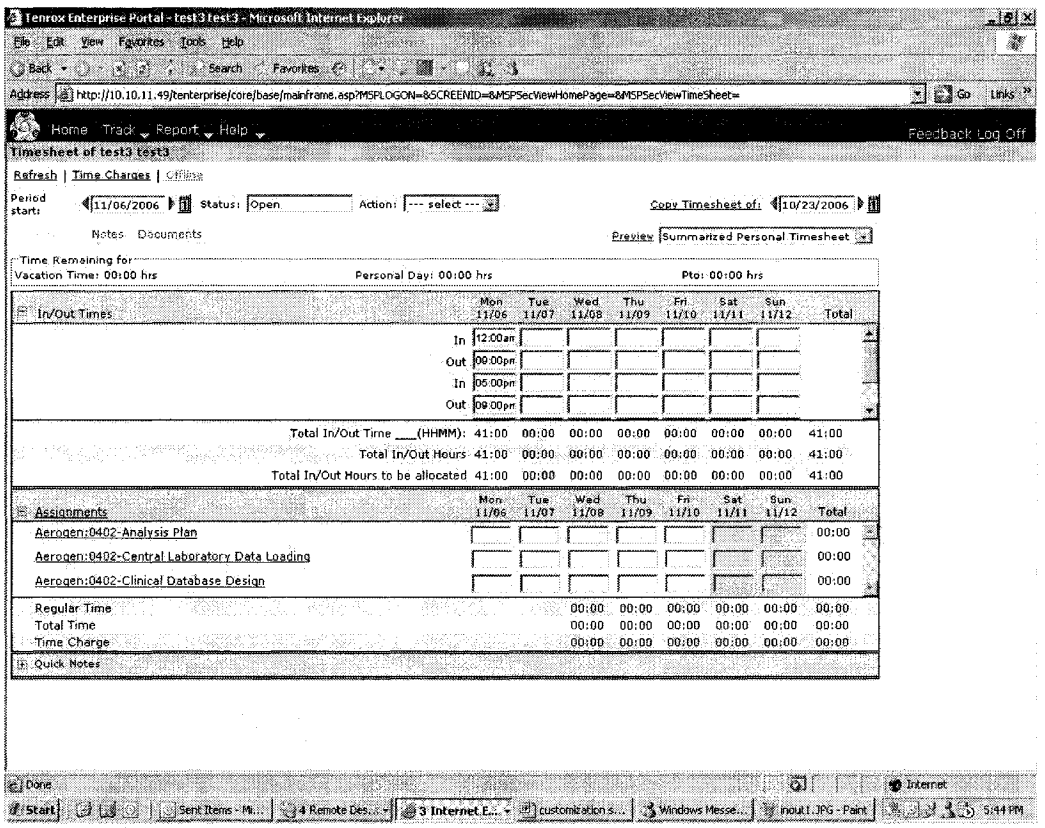


Figure 4.9 Synteract Customization testing

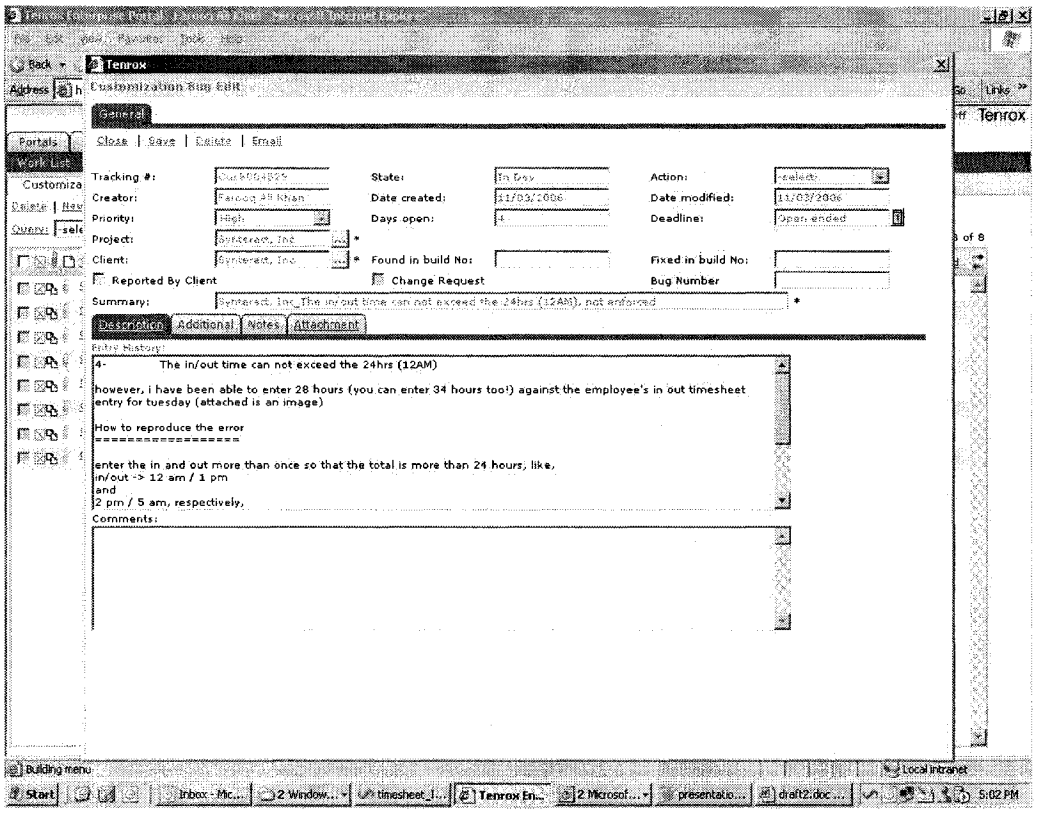


Figure 4.10 Tenrox Defect Management tracking software

Figure 4.10 shows how Tenrox workflow based software was used in order to document and track the bug related to the in/out timesheet module, which we discussed earlier in this section. We find that Tenrox helps in tracking information such as error description, the name of responsible or owner, priority and current state of the issue being tested. It also tracks temporal information (such as date created, duration for which the issue is in a particular state, deadlines), as well as version or build of the software being tested. The ownership determines the ability to update a ticket, which helps to prevent accidental changes. In Tenrox the ownership can be changed through the Action menu. The tool does not provide a way to look at the history of the issue or the participants and their roles. However, this information can

be added as text to the additional notes or the description. The test case specific information such as the input and output data needs to be documented separately and can be attached to the issue using the attachment tab.

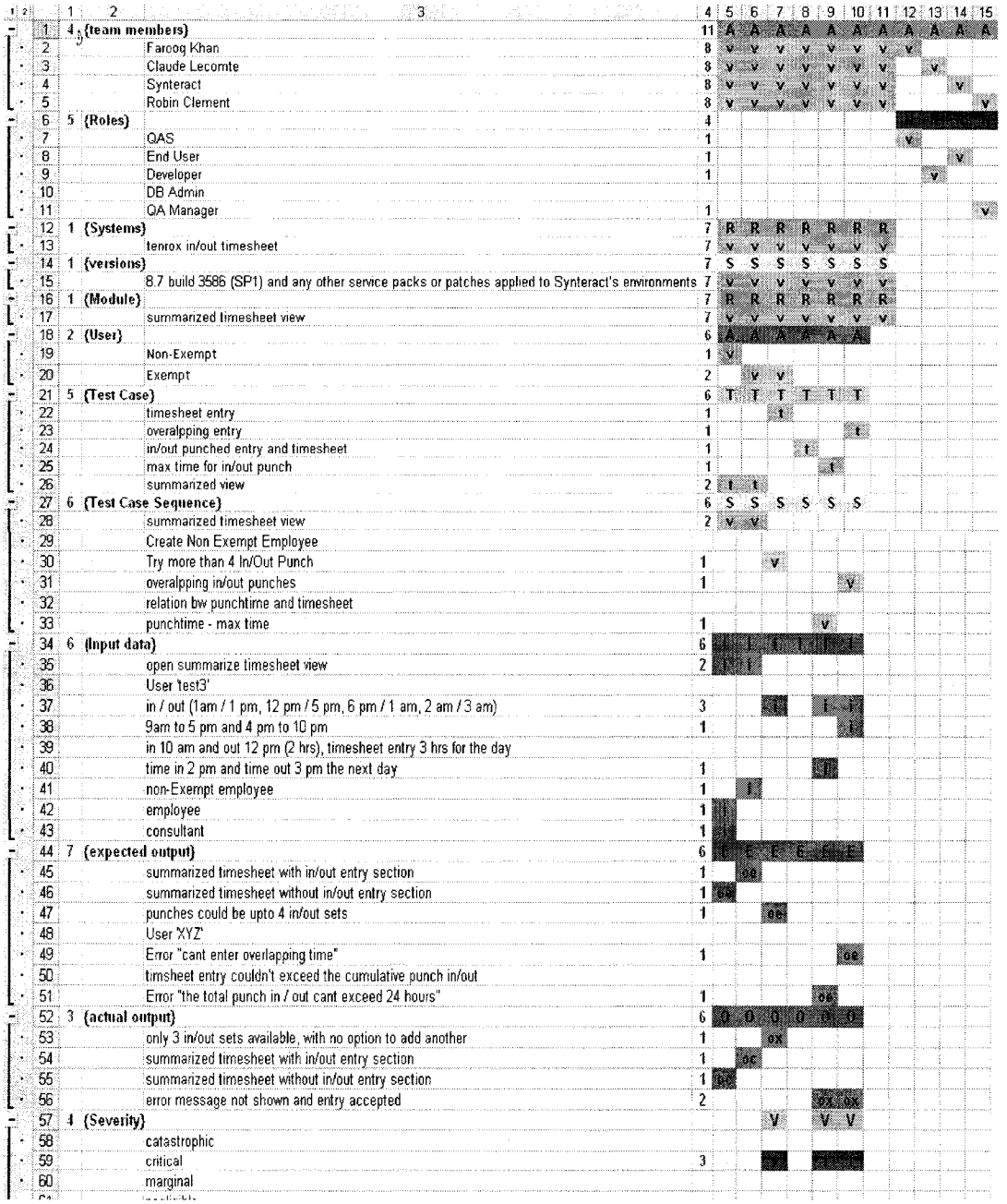


Figure 4.11 CMTC used to test Synteract customization project

Besides Tenrox Defect tracking and test case documentation software, we also used CMTC to create test case scripts, and track defects for the customization of Synteract in/out timesheet module. The CMTC provides a simple way of not just tracking and managing the defects, but also preparing and documenting the test case, which makes it easy for participants to understand the testing process. In Section 4.1 we have discussed how CMTC is used for tracking and managing defects, as well as preparing and documenting the test cases.

The CMTC in Figure 4.11 explains the test case designed to test the Synteract in/out timesheet modifications. The CMTC describes sets such as team members as aggregates "A" and the members of this set as values "v". Roles describes the role that the team member plays in the project and is represented as "N" or node in the CMTC, its members are represented as values "v". In order to read the relation between the members of the two sets, one must browse vertically up or down the symbol "v" used for the set members in the set "team members" and wherever the "v" meets a symbol in another set such as Roles, that row shows related member in that set; such as the team member "Farooq Khan" is the Quality Assurance Specialist (QAS), "Claude Lacomte" is the Developer. "Systems" is considered a resource and is referred to as "R". Different versions of software are considered as part of a sequence and are represented as "S", the members of both these sets are considered as values in the CMTC of Figure 4.11. The CMTC further identifies the existence of two different roles of users, "Exempt" and "Non Exempt". There are various test cases identified in the set "Test Case" represented with the notation "T". The sequence of tests is represented with "S" and its members are represented with "v". The CMTC also explains the data, errors found and the severity of errors. The presence of all this

information in the same map makes it easy to identify various related aspects and compare them with potential or actual problems.

The input data is represented with "I", expected and actual output with "oe" and "oc", whereas in order to identify incorrect output "ox" is used. Furthermore, the set "Severity" is used especially in order to assign the incorrect output a severity level, represented in the map with "l" (loop). According to the participants, the severity level can be represented with "l", because it alludes towards the iterative nature of problem solving and regression testing. As was specified earlier the usage of these notations is up to the discretion of the involved participants and there might be more than one correct solution, which can be used for a single entity in the CMTC.

The CMTC in Figure 4.11 is documenting various test cases, such as the different timesheet views, which is primarily a regression test to check if all the views and reports are working normally after the customization. The other test cases in the CMTC are more directly related to the customization itself. For instance, testing the maximum number of in/out timesheet entries allowed, checking the maximum number of hours allowed for a day, and overlapping timesheet entries. All these revealed errors as shown by the presence of both expected output "oe" in set Expected Output "E" and incorrect output "ox" in set Actual Output "O" in the same column. The severity in all these cases is assigned a level of critical, which can be read by browsing vertically down from each "ox" in set "O" to the corresponding "l" in the set "Severity". So the test data is presented in a way that it is easy to compare and also in case of a discrepancy the relevant programmer or business analyst can be identified without the need to browse across multiple tools or tabs. The same tasks with a tool like Tenrox would need browsing across the tabs, and in order to compare the actual testing steps another tool needs to be looked at.

Rational ClearQuest

We used Rational ClearQuest as a defect tracking and test management tool. It is integrated with Rational Test Manager, which is used for test case authoring. Rational ClearQuest helps in defining the scope and ownership of the test case. It allows the user to control the flow of the test case and provides configuration management by preventing accidental changes, because of multiple users accessing or modifying the test case at the same time. For instance, a user working on a test case has complete control over it until he/she modifies the ownership or changes the status, by choosing an action, such as test case rejected or validated. Once the status has been modified the test case will be assigned to a different person or department. In our case, while working as production support at Software testing department of a company, whenever we passed the ticket it usually goes to User Acceptance Testing or UAT, which is a group of users testing the application with their own test cases. Rational ClearQuest keeps track of events, history, and other related information as notes and attachments. Rational ClearQuest also keeps a hierarchy of related test cases and shows them as child or parent tickets (test cases). However, the tab based structure of documenting information for test cases has some problems, which could be resolved with a tool like CMTC.

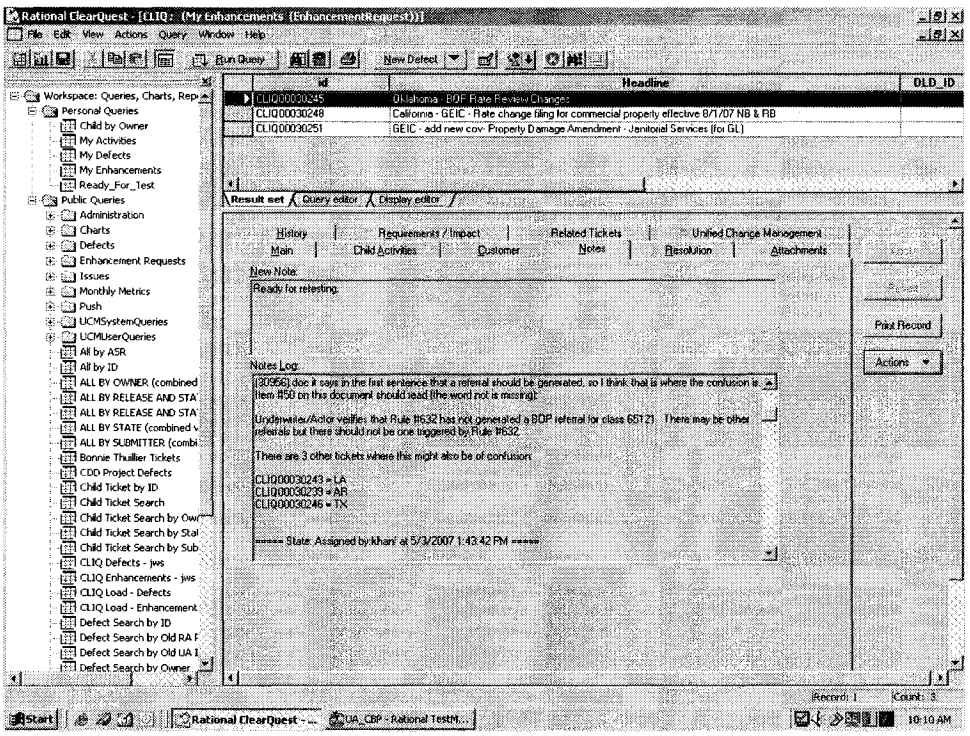


Figure 4.12 IBM Rational ClearQuest – Notes tab

Figure 4.12 shows the related information stored in the notes tab. The author is trying to show the relationship that exists between the test case being studied and three other test cases. However, using ClearQuest it is not possible to relate the contents of these test cases in the same screen. In order to compare related test cases and their contents, the user has to browse different screens in separate documents. We found that using CMTC this problem could be resolved. As in Figure 4.13 two different but related test cases are documented in the same CMTC (Pricing Bug and Available Quantity), so it is easy to compare their contents.

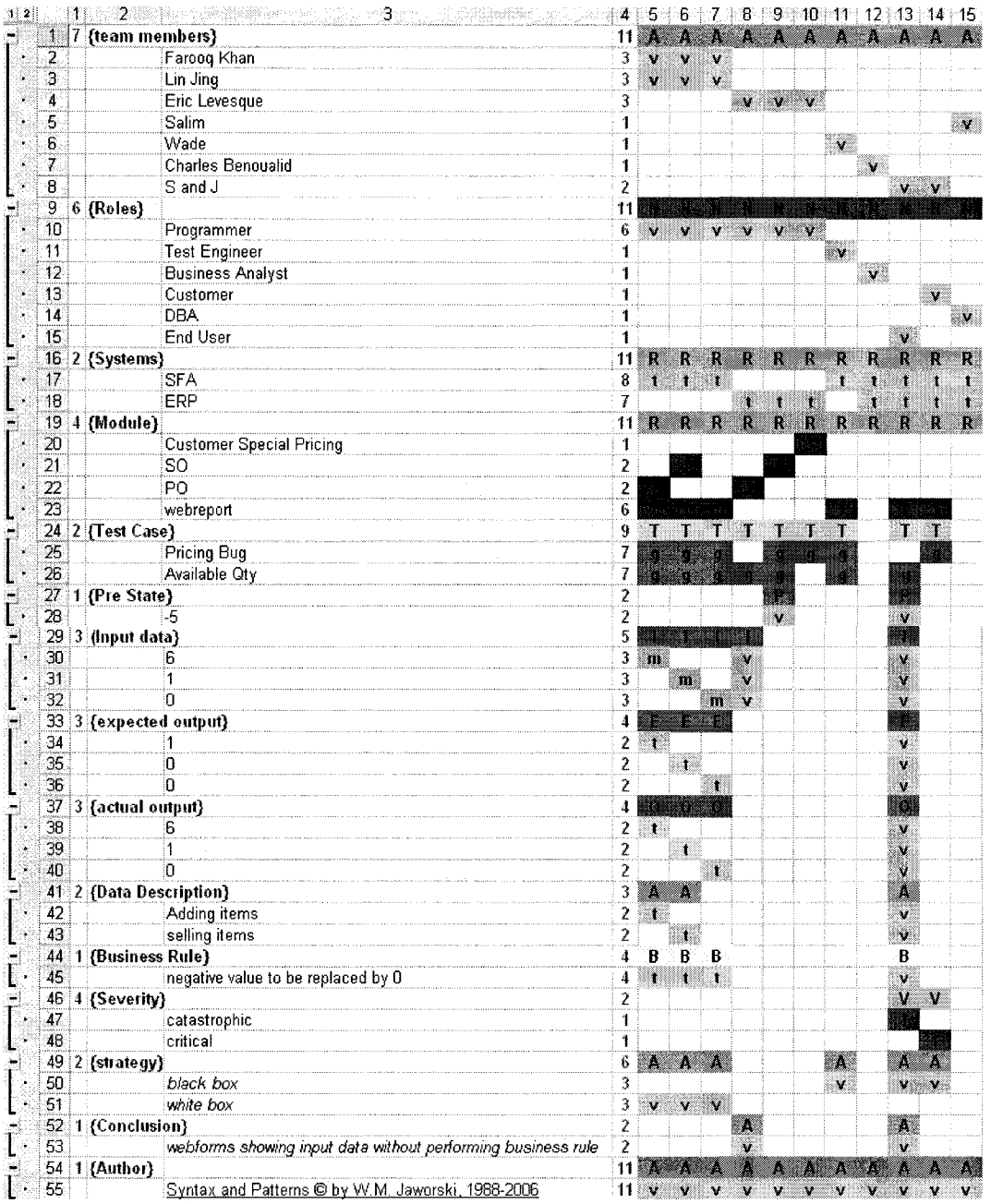


Figure 4.13 CMTC compared with IBM Rational ClearQuest

In Figure 4.13 the set of test cases represented with notation "T" has two members, that is, Pricing Bug and Available Quantity, represented by set member role "g". The CMTC provides a simple way of relating the two test cases and efficiently reusing what is common in them, such as the input data. In addition, it presents related

information, so the test case could be viewed in its proper context, and comparisons could be made easily between the input data with expected and actual output. Browsing down the map the defects could further be related to severity of the problem, as well as the system and module being tested by the test case. The names of team members and their respective roles are also documented in the CMTC document, so the assignment of responsibility is not difficult (that is, if an error is found, who should be correcting the error). The identification of members involved and their respective roles also helps in avoiding accidental changes as the roles are clearly declared. By extending the notations in the notation set presented in Chapter 2, the CMTC in Figure 4.13 could be extended to include the rights of team members to view and/or update the test case document. For instance, instead of using "v" different notations could be used to specify the access rights for the team members, for instance, "r" could be used for users with read access to the CMTC. However, in the test case documented in Figure 4.13, it was not a requirement to define access rights, as the participants involved were working in a democratic decentralized team structure, where everyone was participating fully without any restrictions to their access of the CMTC. (Figure 4.13, is explained in detail in the next chapter where we look closely at our implementation of CMTC.)

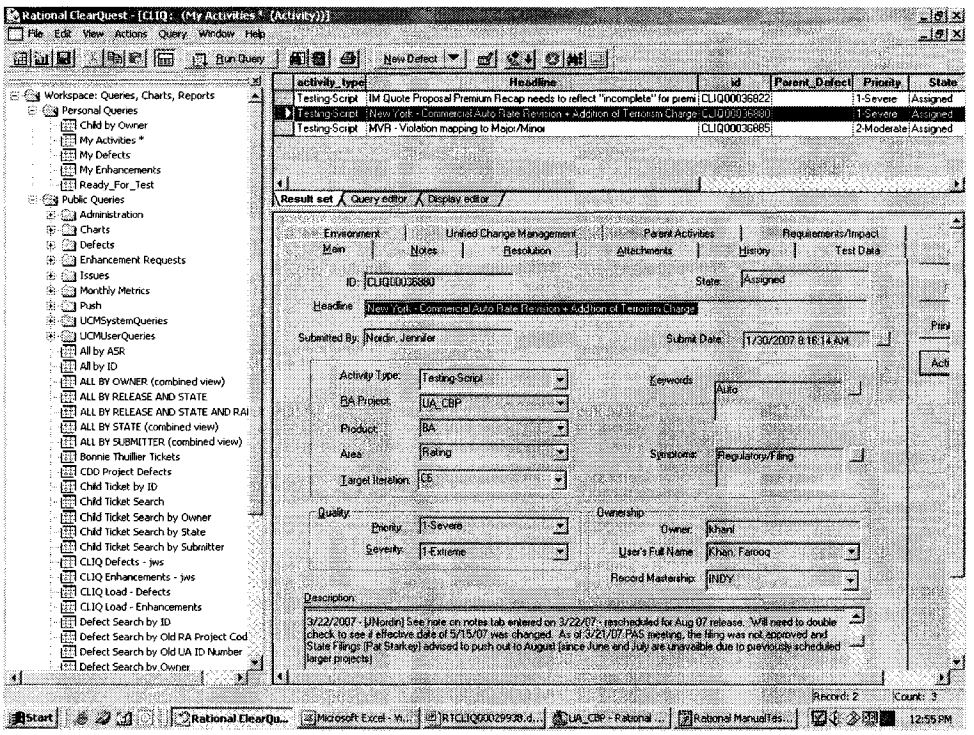


Figure 4.14 IBM Rational ClearQuest – Main tab

Figure 4.14 shows the main tab of the ClearQuest ticket. The main tab group together information, such as names of people involved including name of the submitter of the issue, current state, the date when the issue reached its current state, ownership, keywords, description etc. A problem with this type of test case documentation tool is lack of extensibility; the keyword field is used in order to mention any additional information. In Figure 4.14 the keyword field is used to mention the module name that is being tested. Figure 4.13 shows that CMTC doesn't have to work with this bottleneck of lack of extensibility. All the related modules are grouped explicitly in the set named "Module", this clarity and consistency in presenting information makes it easy for those who are not test team members to read the test case easily. This also shows that the set based structure of the CMTC can be extended to include more information at any stage of test case planning.

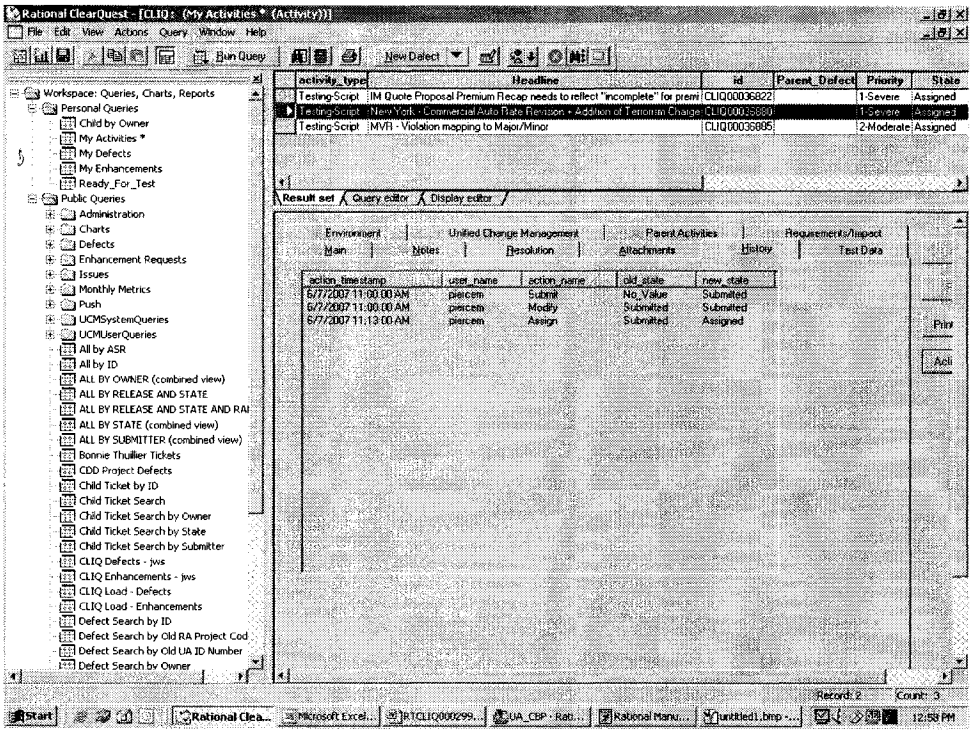


Figure 4.15 IBM Rational ClearQuest – History tab

Figure 4.14 shows only the current state and the information related to the current state for other related aspects another tab needs to be clicked, as shown in Figure 4.15, which shows only a brief summary of the assignments and some temporal information such as duration of the state. However, the information related to various states is lost, whereas in CMTC all the states are preserved. This is because in CMTC the information is added gradually throughout the testing lifecycle. In the CMTC shown in Figure 4.13, data and other related information of four different modules is preserved. The testing of these modules was performed not necessarily at the same time, as such the CMTC in Figure 4.13 is also an example of various states of information preserved within one test case.

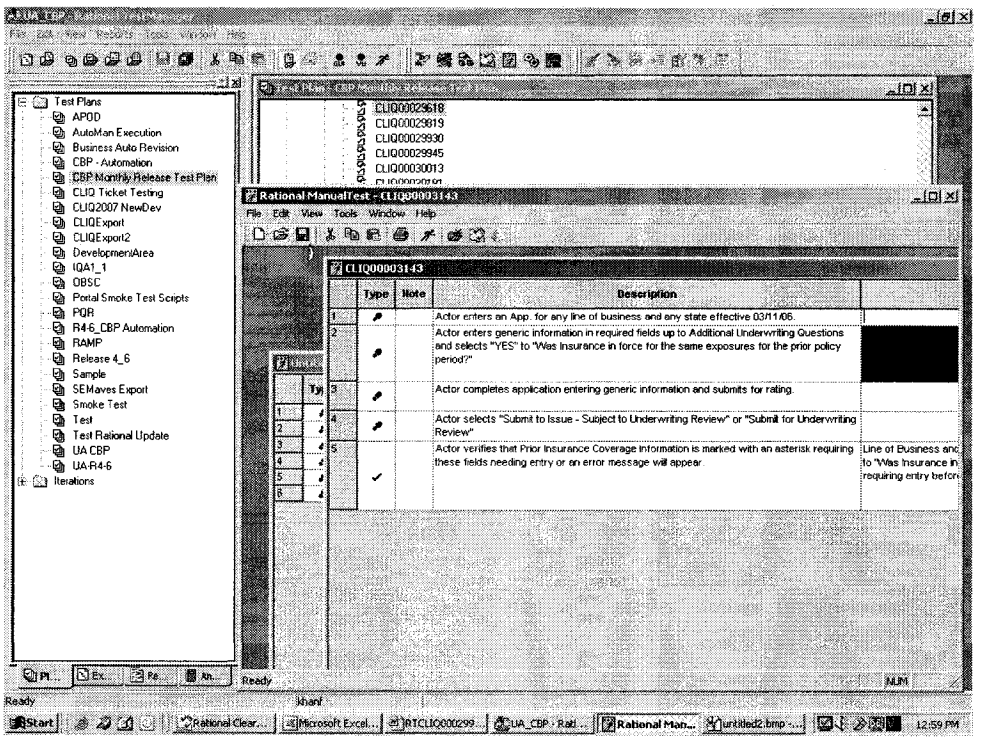


Figure 4.16 IBM Rational Test Manager

In the Rational Tool Suite the test case itself is documented in a separate tool, which is similar to Test Manager or Manual Tester, as shown in Figure 4.16. In order to compare the test case steps, data used, expected and actual output, with other related information, such as the severity of a defect, the user has to browse unnecessarily between various tabs and tools. The problem is more severe in small projects for defect and enhancement testing. CMTC addresses this problem by showing the entire test case along with all related information in one screen, grouping the information using sets. The resulting test case allows the user to compare various aspects of the test, as well as managing the testing process as information is more readily available.

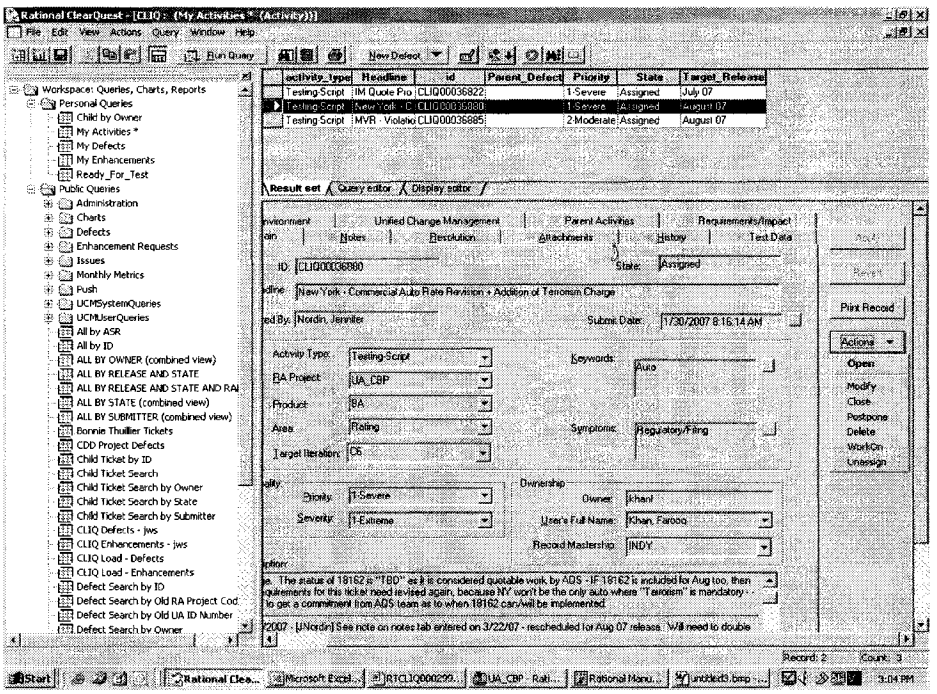


Figure 4.17 IBM Rational ClearQuest – Changing state

Figure 4.17 shows that in Rational ClearQuest information related to the transition of the state is not readily visible to the viewer. Moreover, once the test case moves from one state to another, the attributes of the previous states are not preserved, as discussed before. In CMTC the transition of state is not accompanied with hiding information, CMTC adds information to the test case document so the states are visible and reader could compare the information across various states.

IBM Rational Manual Tester

IBM Rational Manual Tester (RMT) is used to document test cases and to show the result of testing. It saves test cases so they can be added to test case documents, in future testing efforts. We used RMT in a Software firm while testing invoice management system; we also use CMTC in this project. The RMT helped us write test cases and identify steps which are crucial for the success or failure of the test.

An advantage of the IBM RMT as shown in Figures 4.18 and 4.19 is that it allows the test case to be linked with the application so the steps of the test case can be viewed as the application is being executed. It can also integrate with the application and provide a degree of automation, so the tester doesn't have to fill the form manually as shown in Figure 4.18, 4.19.

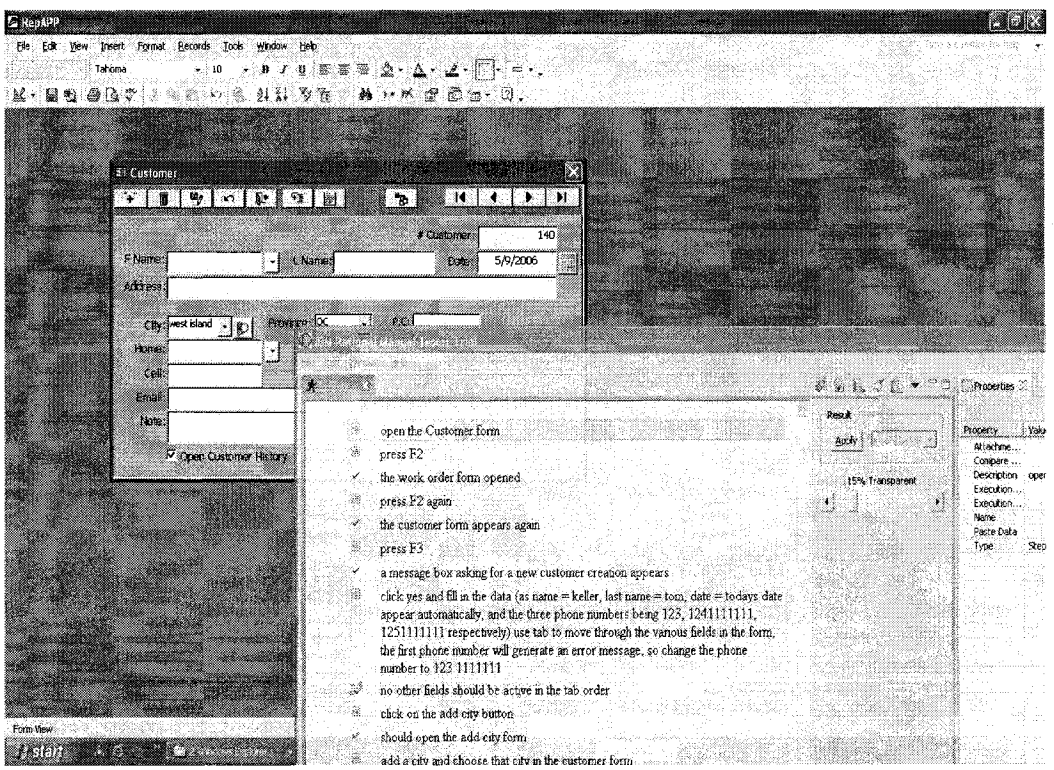


Figure 4.18 executing the test with the application

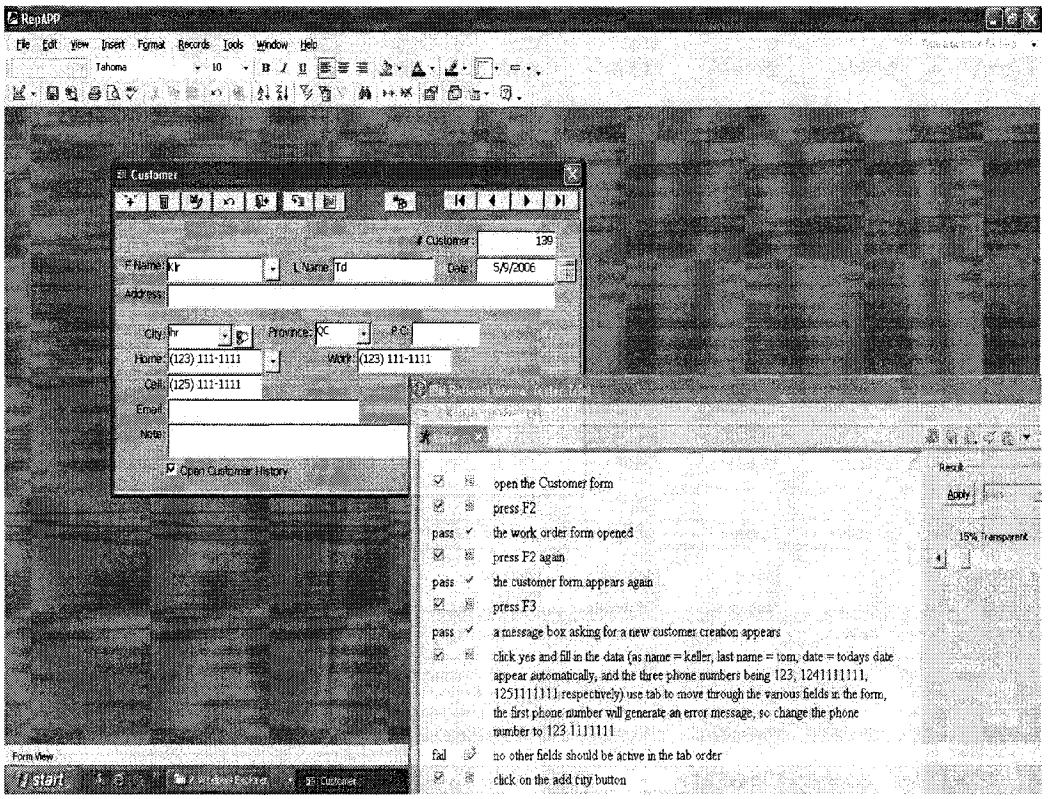


Figure 4.19 test case for searching an existing customer

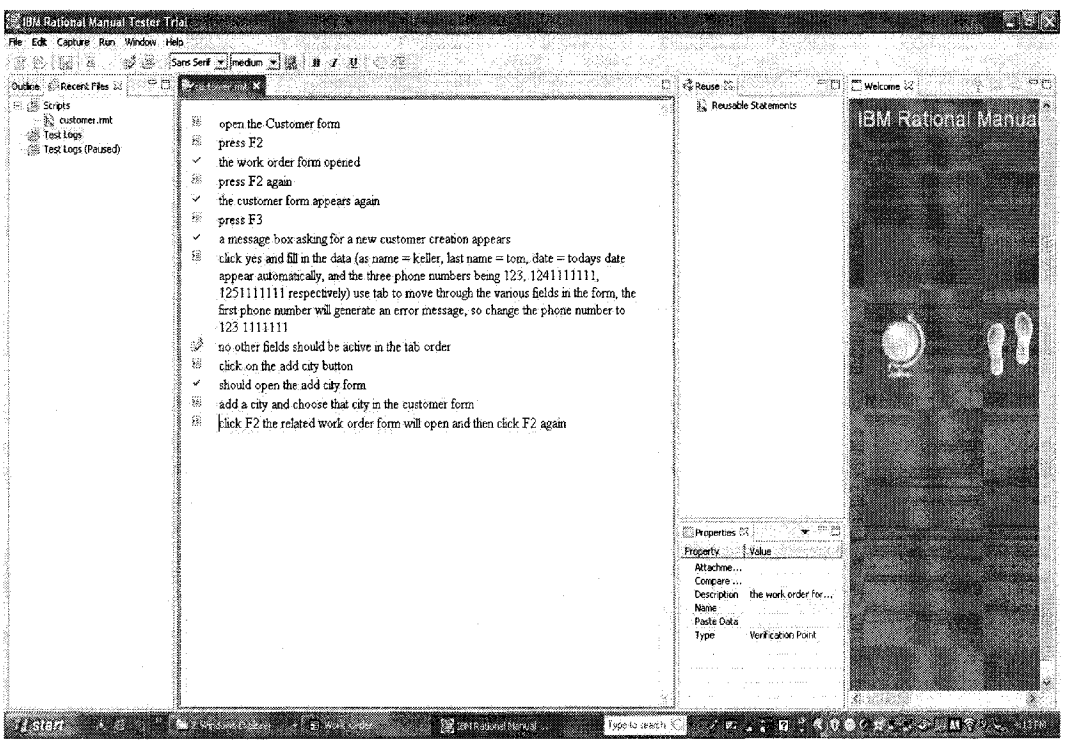


Figure 4.20 Test case for Customer form

Figure 4.20 shows IBM Rational Manual Tester uses verification points marked as✔. If the verification point fails the test fails. The reporting points are marked as and are used as the points to be included in reports as shown in Figure 4.21.

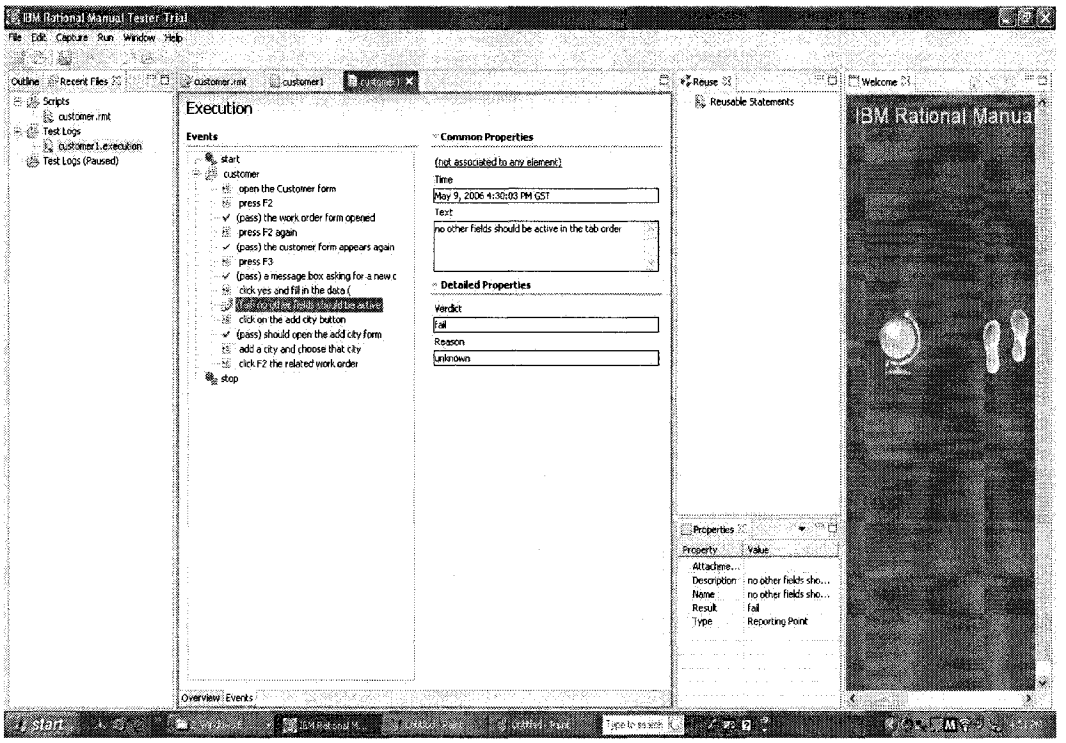
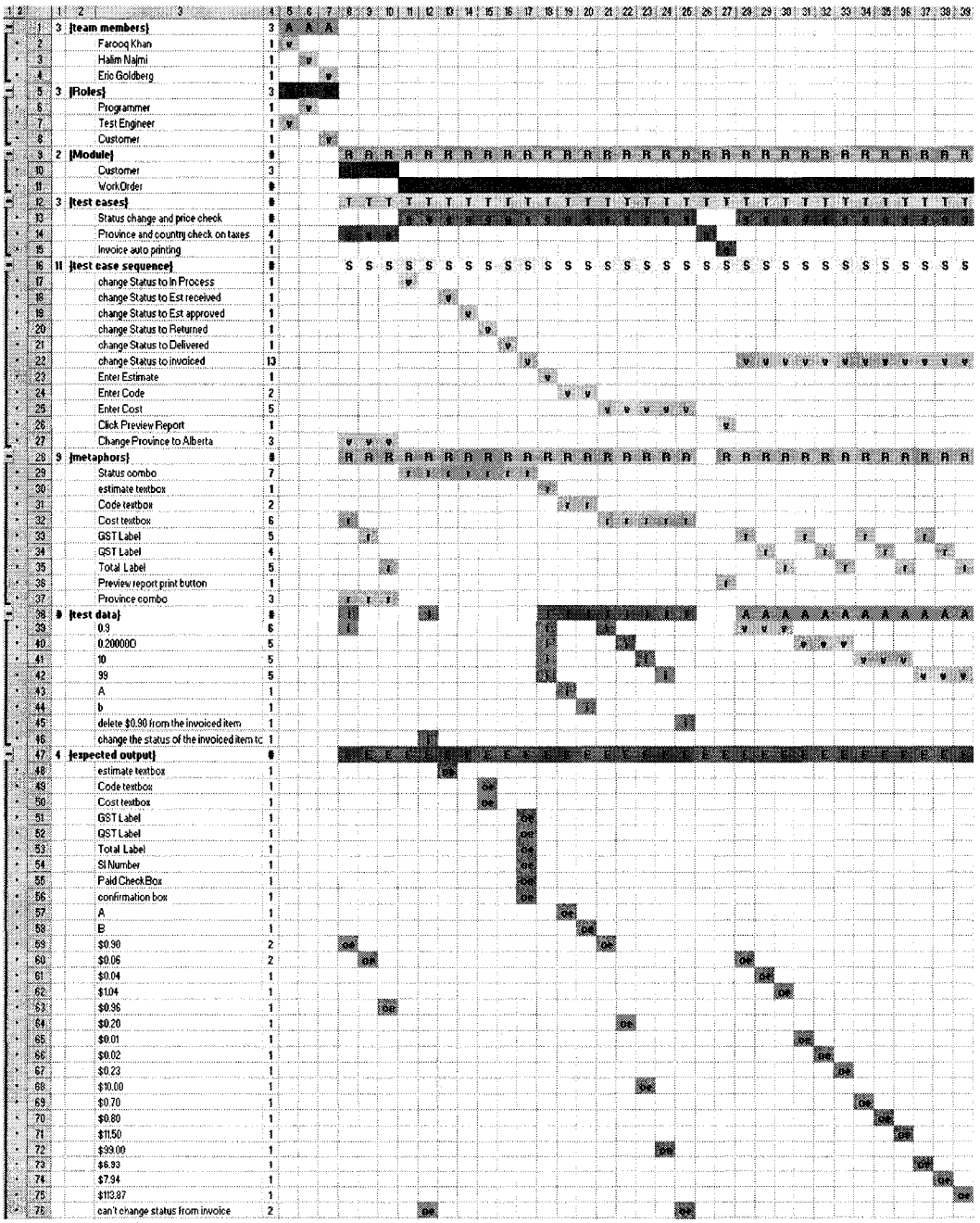


Figure 4.21 Test case result summary

With every 'step', 'verification point', or 'reporting point', additional information can be added such as file attachments, textual description etc. We find this helpful as we can attach screenshots or functionality description with every step, as additional information. However, this additional information is only visible when the step to which the information belongs is active or highlighted; this is a limitation in the tool as comparing details is difficult. From our application of CMTC on the same project as shown in Figure 4.22, we find that CMTC helps compare the details of the test case,

as all related information is visible without having to select any record or step in the test case.



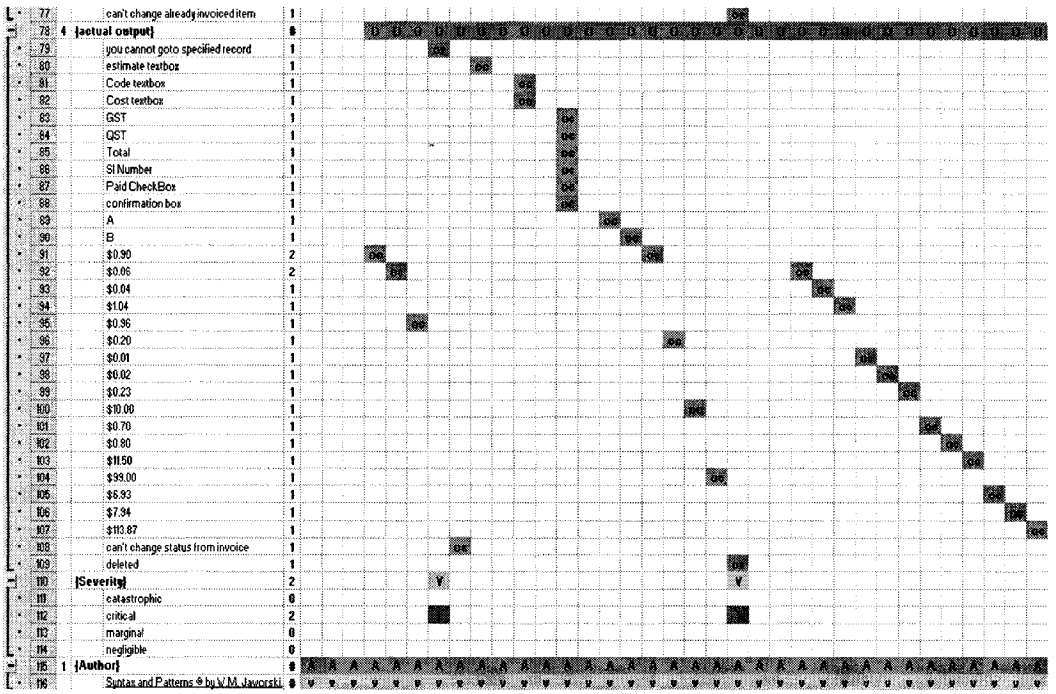


Figure 4.22 CMTC for Customer form

The project that we used IBM RMT has two forms, one involving customer information as in Figure 4.23. The other form has information related to the product being repaired and the tracking of the product along with the cost and tax calculation, as shown in Figure 4.24. The reports are all filtered depending on the criteria selected as shown in Figure 4.25. The application though small has a lot of embedded functionality, to move from one form to another *F2* key is pressed, to add a new record *F3* key is pressed; the tax calculation depends on the provinces and countries.

When we used IBM RMT to write test cases, it was mostly a sequence of steps that we wrote. The steps were identified as verification points as shown in Figure 4.20. If *F2* key is pressed the verification point is "the Work Order form is opened", if the verification point is not executed properly then the test case failed and an error is found. Similarly, those points which need to be reported in the report are marked as reporting points. For instance, when the tab is used to move across different fields

within the form the tab order must be correct. The result of this step is not just causing the test case to pass or fail, but also needs to be reported in the final report as shown in Figure 4.21.

Figure 4.20 shows that in IBM RMT the input data used for testing and the output specification are embedded within the steps to be executed for testing. As a result the relationship between the modules being tested, the steps and the errors is not clear. The relationship between the other related aspects of the test case such as the participants involved, the metaphors (text boxes, msgboxes, command buttons etc), their labels, and their relation with the error in case of a user interface error, is also difficult to identify.

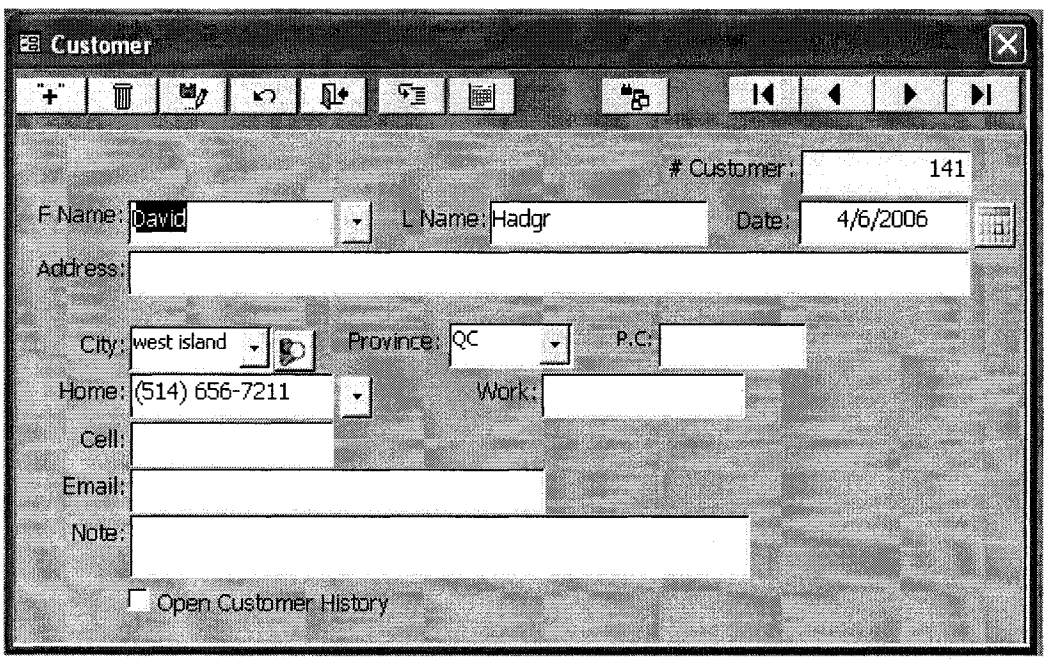


Figure 4.23 Customer Form

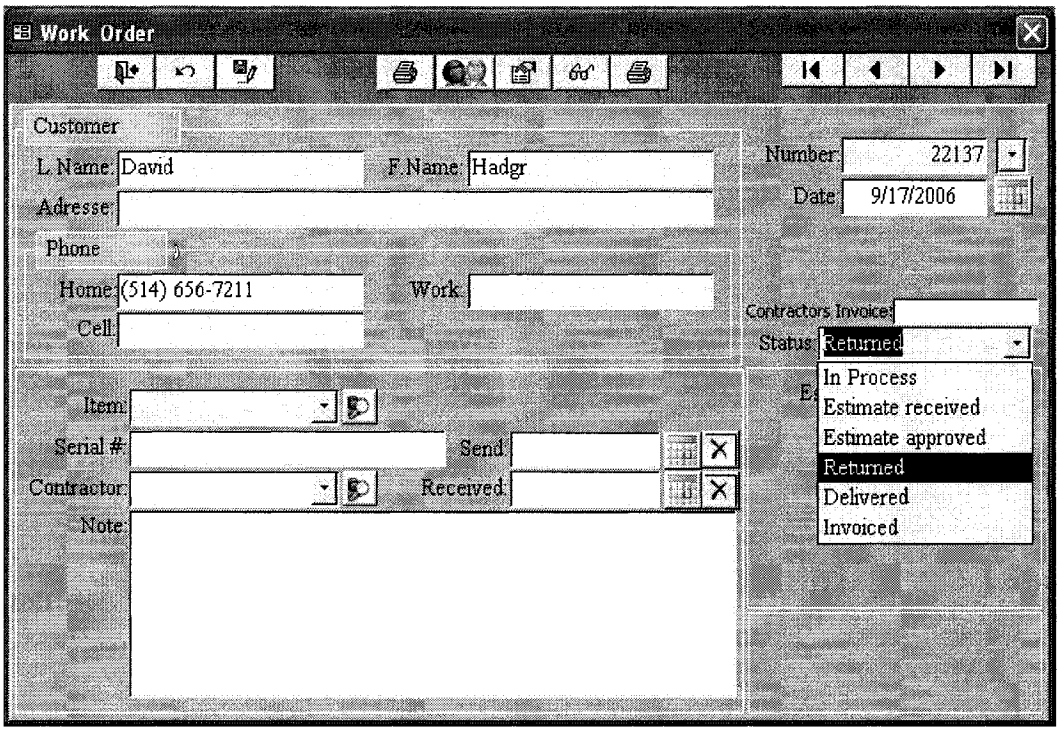


Figure 4.24 Work Order Form

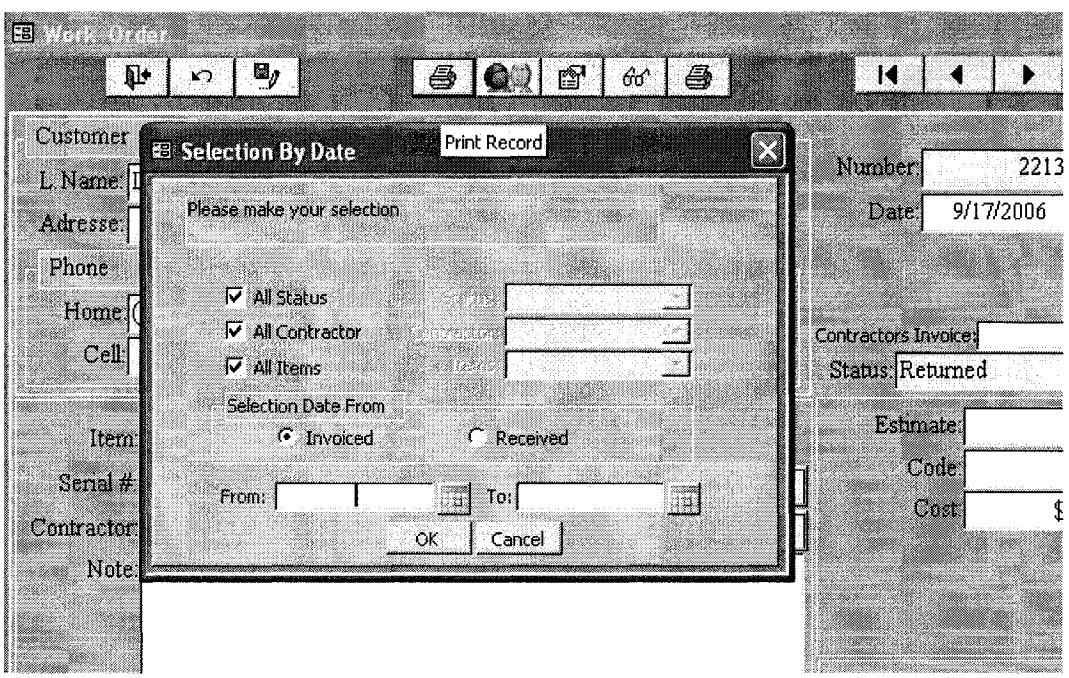


Figure 4.25 Filter for the report

When we use CMTC in the same project to test it, we find that the results helped understanding, reporting and identifying the related aspects of the test case, as well as documenting the data, and classifying it as input, output or incorrect output.

Figure 4.22 shows various sets in the CMTC used for this project including team members, their roles, modules, test cases, test case sequence, metaphors, test data, expected and actual output and the severity of the discrepancy in case of a mismatch between the expected and actual output. Test cases are represented as "T", whereas the set members are represented as guard or "g". The participants decided to use this suggestive notation to emphasis that a proper test case is a guard against software that creates problems. Test case sequence is represented with "S" a notation used to describe a set that has a sequential nature. Metaphors are also considered in this test case and are represented with "R" or a resource type. It is important to look at the relation between the user interface artifacts used and the data being used to test the application, as it points to the extensibility of the CMTC. This shows how CMTC can be used to add new details to the test case without jeopardizing the readability of the Test Case. Such additions are not easy, if at all possible, in tools discussed earlier.

The CMTC of Figure 4.22 describes test cases related to how the price is calculated and taxes included as the status changes (status options are shown in Figure 4.24), Figure 4.22 also includes test cases to test the invoice printing. The test case 'Province and country check on taxes' shows how the taxes are calculated on the basis of provinces or countries selected as shown in Figure 4.18. Figure 4.22 shows CMTC which is related to the test case sequence that has the steps "change to Alberta" and further down in the map in the set "Metaphors" we can see the effected metaphor is "Province Combo", "Cost Textbox", "GST and Total Labels". This means that when the province Alberta is selected the metaphors that appear on the

screen are Cost Textbox, GST and the total cost, while the Provincial tax labeled as QST is not relevant. The test case further documents the "input data" that should be used; in this case it is 0.9 to be used as the cost. In Figure 4.21, the "i" for "input data" appears in the same column where the notation appears for the "cost textbox" (in the set "metaphor"). The "output" set, if we continue vertically in the same column as the "i" (in the set "test data") has "oe" for the value or set member 0.9, the reader of this CMTC can easily see that there is no discrepancy between the test data and actual output and this is also shown by the notation "oe" (in the set "expected output"). Similarly, it can be seen that other calculated values for GST and total cost are also as expected. However, if you read the column where "ox" appeared (in the set "expected output") in upward direction and matching the members of various sets; it can be noticed that the test case related to invoice items where the invoice item is deleted has failed. As according to the business policy the invoiced items should not get deleted, instead, a message box with warning that invoice items cannot be deleted should appear.

Chapter 5

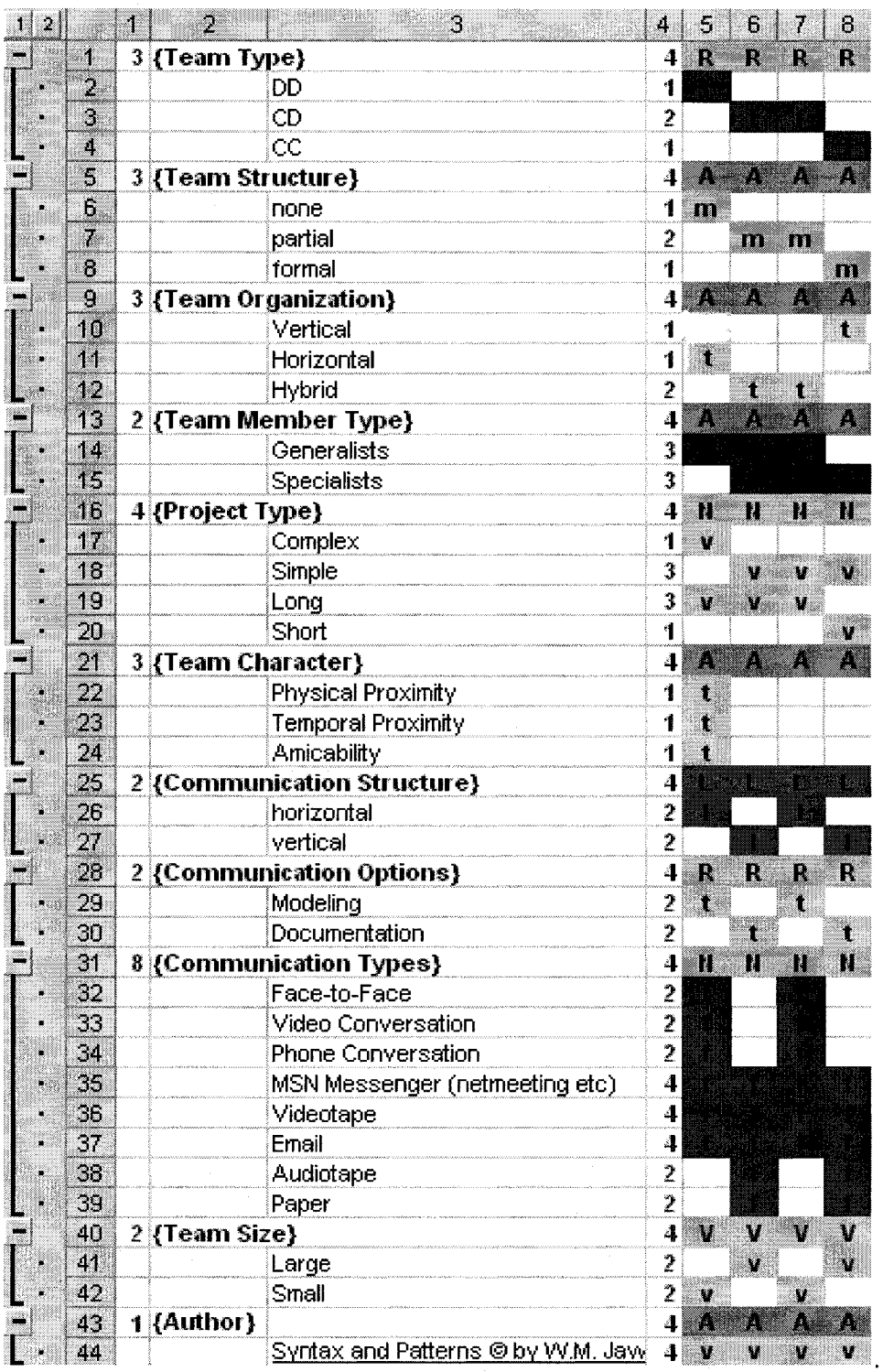
Application of CMTC in Testing Projects

In this chapter we will present the results from our application of CMTC in different projects and organizations. Later in this chapter we will compare CMTC with various test case authoring and defect tracking tools.

We worked on different projects and software companies to apply the concepts we propose in this report. Some of the organizations were already using testing tools such as IBM Rational Manual Tester, IBM Rational Test Manager, IBM Rational ClearQuest, or Tenrox Workflow based management software for software testing and defect management and tracking; others were not sure whether they want to pursue software testing as a separate activity because of the cost involved. However, CMTC was easy and not costly for even small organizations to implement as we will see in this chapter. We now present our study from the application of CMTC in various organizations.

5.1 Using Context Maps during Team Formation and Resource Planning

Context Maps can be used to model and describe any process or activity. This is useful in applying Context Maps for testing as related processes such as team organization, team communication and resource allocation can be described and modeled using the same tool, which is used for test case authoring and defect tracking. The usage of the same tool for test case authoring, management and describing related processes makes it easy for the participants of the software testing process to understand all aspects of software testing.



		1	2	3	4	5	6	7	8
1	3	{Team Type}			4	R	R	R	R
2				DD	1				
3				CD	2				
4				CC	1				
5	3	{Team Structure}			4	A	A	A	A
6				none	1	m			
7				partial	2		m	m	
8				formal	1				m
9	3	{Team Organization}			4	A	A	A	A
10				Vertical	1				t
11				Horizontal	1	t			
12				Hybrid	2		t	t	
13	2	{Team Member Type}			4	A	A	A	A
14				Generalists	3				
15				Specialists	3				
16	4	{Project Type}			4	H	H	H	H
17				Complex	1	v			
18				Simple	3		v	v	v
19				Long	3	v	v	v	
20				Short	1				v
21	3	{Team Character}			4	A	A	A	A
22				Physical Proximity	1	t			
23				Temporal Proximity	1	t			
24				Amicability	1	t			
25	2	{Communication Structure}			4	t	t	t	t
26				horizontal	2	t		t	
27				vertical	2		t		t
28	2	{Communication Options}			4	R	R	R	R
29				Modeling	2	t		t	
30				Documentation	2		t		t
31	8	{Communication Types}			4	H	H	H	H
32				Face-to-Face	2				
33				Video Conversation	2				
34				Phone Conversation	2				
35				MSN Messenger (netmeeting etc)	4				
36				Videotape	4				
37				Email	4				
38				Audiotape	2				
39				Paper	2				
40	2	{Team Size}			4	V	V	V	V
41				Large	2		v		v
42				Small	2	v		v	
43	1	{Author}			4	A	A	A	A
44				Syntax and Patterns © by W.M. Jaw	4	v	v	v	v

Figure 5.1 Context Maps for software team organization

We applied Context Maps to document resource allocation as well as team coordination and communication issues, while working at a software company. The following Figures 5.1 and 5.2 show the results. We found that Context Maps are helpful to introduce software process in small organizations because of their ease of use and low cost. The maps in Figures 5.1 and 5.2 describe organizational policies used in forming teams and modeling issues concerning team communication and coordination.

The Context Maps in Figure 5.1 presents the organizational policy regarding team formation for different types of projects. Team related sets are 'Team Type', 'Team Structure', 'Team Organization', 'Team Character' and 'Team Size'. The set 'Project Type' identifies different type of projects. The communication issues are presented in sets such as 'Communication Structure', 'Communication Options' and 'Communication Types'.

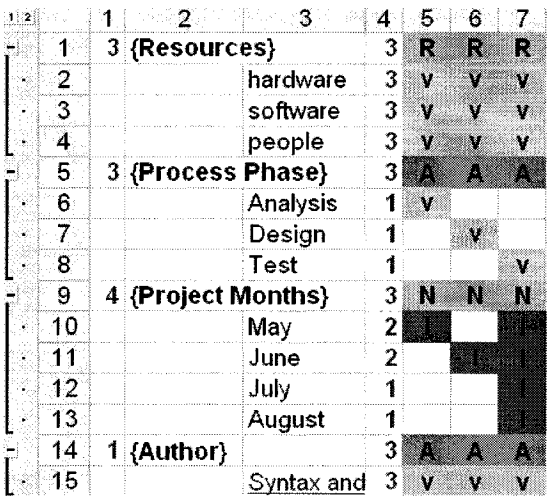
To read the Context Maps of Figure 5.1 we can look at a set and its member. For example, in the set "Team Type" the member democratic decentralized (DD) is represented with "f" or the tail of the arrow, suggesting that it is a pointer that relates the team type "DD" to other set members. If we read vertically down Column 5, the column which associates "f" with "DD", we can notice it is related to the Team Structure "none". This mapping means that for the democratic decentralized team type there is no strict hierarchy of control, unlike the formal team structure for controlled centralized (or CC) team type. If we read further down the same column, we see that the team organization for this team type is also horizontal and the team members are Generalists. The notations used for these sets and set members were based on the participants' judgment, just like the statements used in a test case using Rational Manual tester, or any other test case authoring tool. Further down we see

that if the Project Type is complex which is denoted by a "v" in front of the row where the set member "Complex" is, or if the Project Type is "Long", the project share the same attributes, such as the Team Type "DD" and Team Size "small".

Figure 5.1 describes how the management of a company we worked with views team organization, team communication issues, and the relation among different aspects of a team. It shows the conditions which identify the type of communication and coordination among team members. The Context Maps uses the notation set presented in Chapter 2 to identify the relation among the sets presented in Figure 5.1. This is a design and there is no one correct design, it depends on the creator of these Context Maps as to which notation he selects for a set or a set member. In Figure 5.1 Team Type is considered as a resource represented with "R". This is because the organization considers the identification of various team types as a resource. The team type and other related sets in Figure 5.1 show the organization's readiness to work in different situations where a team type, for instance, Democratic Decentralized is best. However, it is not wrong to consider Team Type as a Node N, as the type of team can be considered as a node in its relation with other sets or concepts being presented in the Context Maps of Figure 5.1.

The Context Maps in Figure 5.1 presents the information that an organization needs in order to plan the team type for a project type, along with information related to each type of team. For example, the team type Democratic Decentralized (DD) as opposed to Controlled Centralized represents a more loose management structure as can be viewed from the values of various related set members, such as team organization being horizontal rather than vertical. The team type DD is used for projects which are complex and relatively long, so the communication as well as team

organization is horizontal and people can discuss more among themselves to reduce the complexity of the problem. Controlled Centralized (CC) structure is preferred for simple and short projects. These projects involve more vertical structure or control for both team as well as communication. The policies explained in the Context Maps of Figure 5.1 are applicable to testing teams just as effectively as they are for software development teams.



	1	2	3	4	5	6	7
1	3	{Resources}		3	R	R	R
2			hardware	3	v	v	v
3			software	3	v	v	v
4			people	3	v	v	v
5	3	{Process Phase}		3	A	A	A
6			Analysis	1	v		
7			Design	1		v	
8			Test	1			v
9	4	{Project Months}		3	N	N	N
10			May	2	I		
11			June	2		I	I
12			July	1			I
13			August	1			I
14	1	{Author}		3	A	A	A
15			Syntax and	3	v	v	v

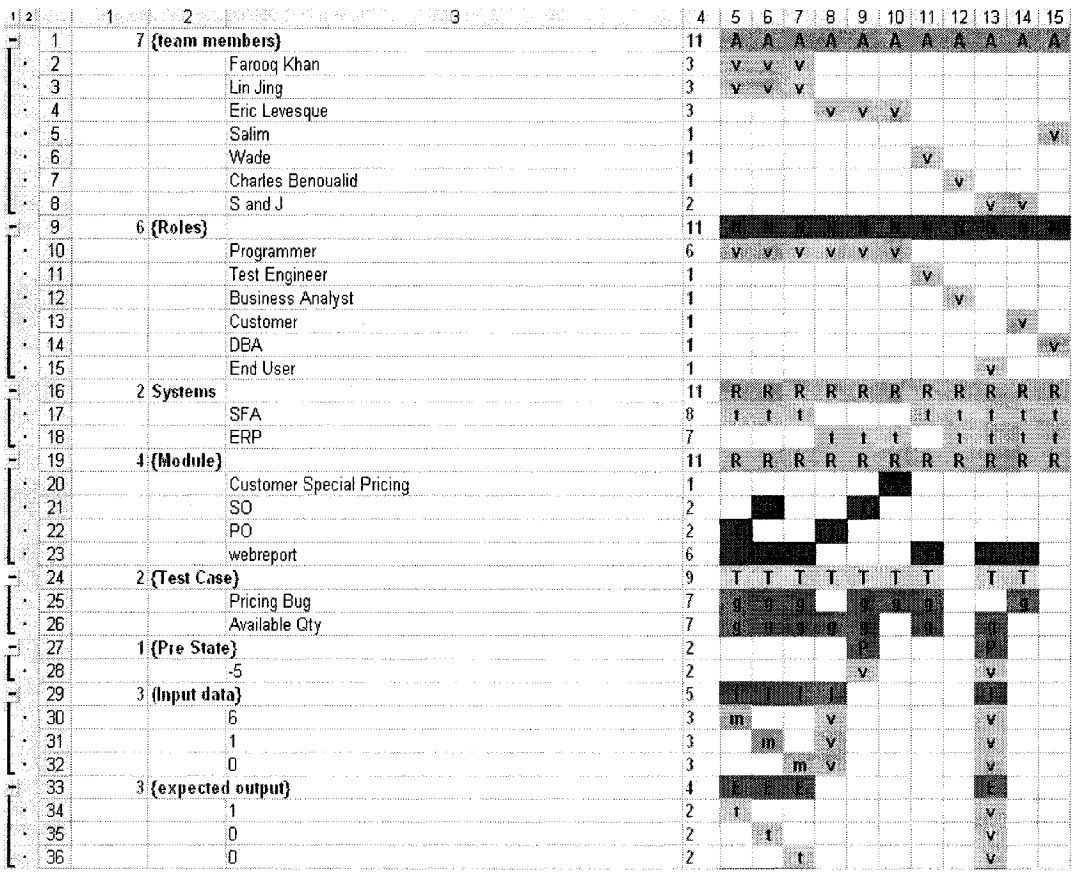
Figure 5.2 Context Maps for Resource Allocation

Figure 5.2 explains the resource allocation within the organization for a project that we worked on. As the resources should be managed efficiently to reduce losses, it is important to identify the resources needed for a particular phase of the project and the time when they are needed.

Figure 5.2 considers three resources hardware, software and people; the set itself is represented with the notation "R", whereas the set members are represented with "v" or values in the Context Maps. This Context Maps shows that the project analysis phase will take place during the month of May (i.e., during May resources will be

used by members of Analysis phase) and design will take place during the month of June, whereas the testing will be carried out as an umbrella activity throughout the entirety of the project. It considers the Project Months as nodes "N" and the months themselves as loop "l" during which the activity goes on iteratively.

5.2 Using CMTC



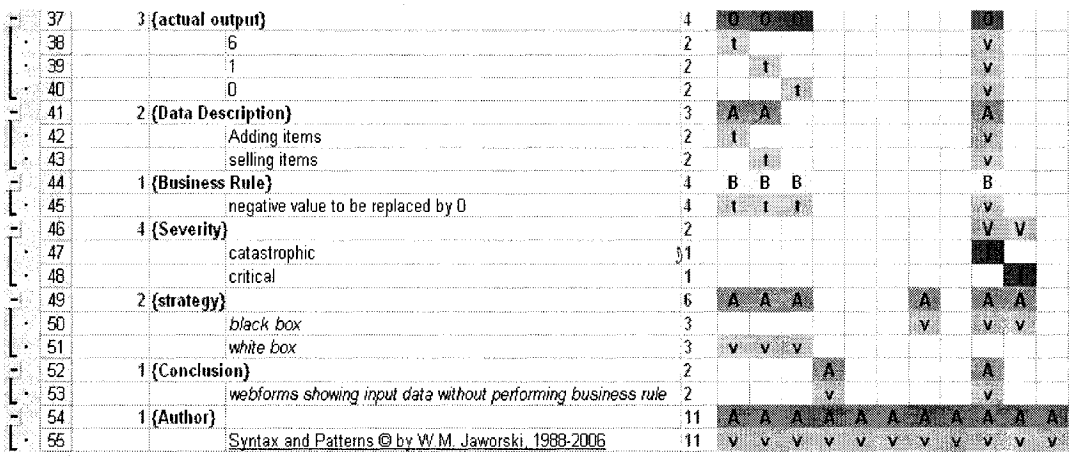


Figure 5.3 CMTC used to debug the SFA

The CMTC in Figure 5.3 was used during the testing and debugging of an ERP's web module. In order to read the CMTC the reader should consider any set member then read horizontally along the same row. The column in which the reader finds a notation, the reader should read along that column vertically up or down. Then the reader should match the notation found with any notation in any other set (in the same column of the map). For instance, in Figure 5.3 in the set "team members" both Farooq Khan and Lin Jing showed "v" in front of them, if we read along the same column we will see "v" in the set 'Roles' against the set member 'Programmer', this suggests that these two are programmers for the project being tested, whereas "Wade" is the test engineer for this project.

In the CMTC of Figure 5.3, the error can be read as a discrepancy between the related members of the sets "expected output" and "actual output". If we look at the set member "1" in the expected output, it is vertically above the set member "6" in the set actual output. The difference between the two suggests that there is an error. If we browse further up the same map in the set module we see it intersects the notation "f" in front of the set members PO (Purchase Order) and web report. This indicates that

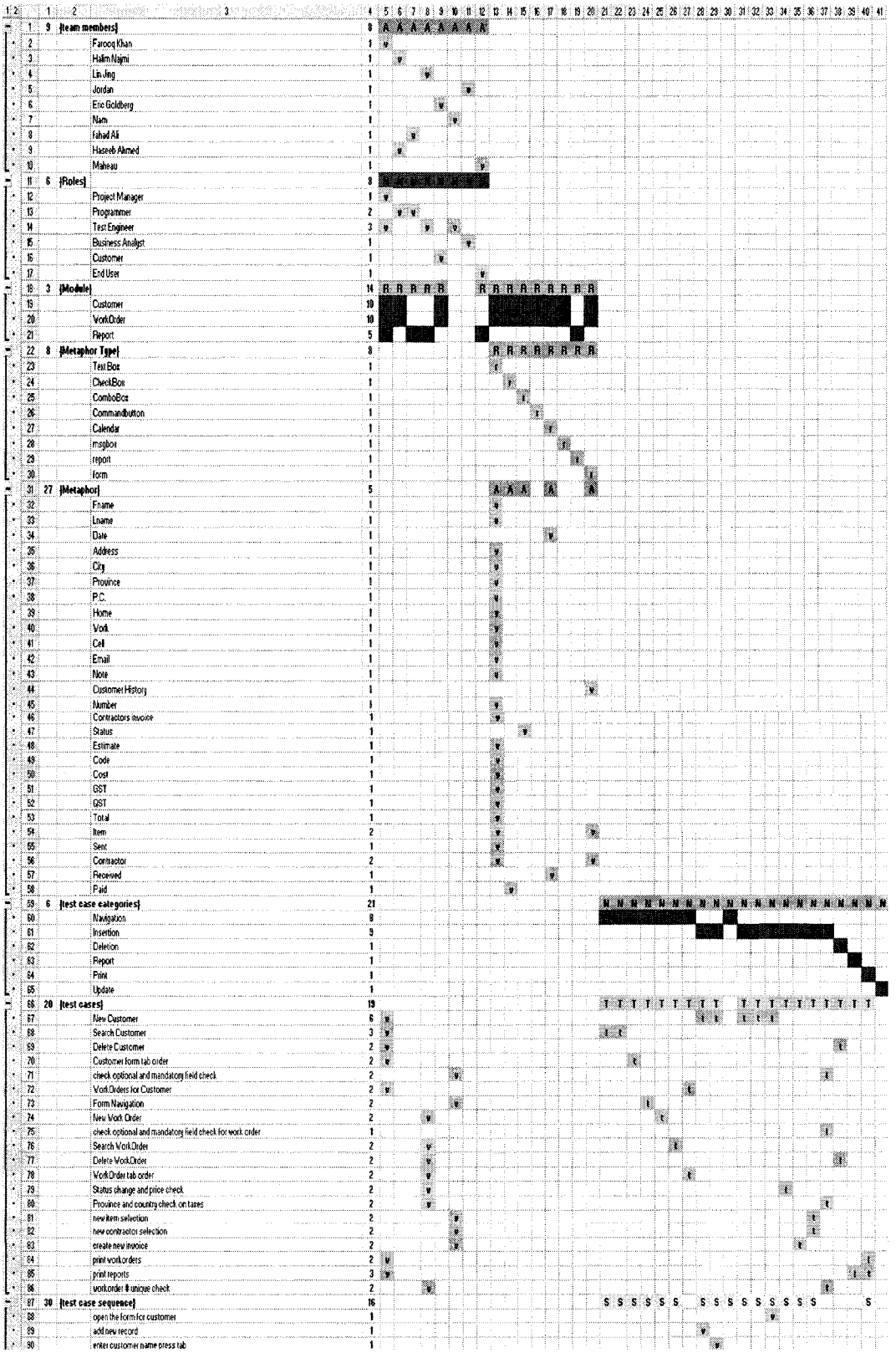
the error is related to the modules PO and web report. In the set Systems we can see the same column lead us to notation "t" in front of SFA, which is the web module of the application.

The customers of the ERP were using the light weight web version of the ERP as they travel. It was important for them not to show the available stocks as negative if they have sold more than they have. The system should use 0 as remaining stocks instead of negative values. However the system was not behaving according to the business rule and was adding newly added stocks to 0 instead of negative values in cases where the stocks were negative. Understanding these business details would have been difficult especially for people from various different teams. With the help of CMTC though the relation between the various aspects of the problems such as related modules, stakeholders and their roles, data use to capture and locate the problem was clearly demonstrated and as such understood by all involved. Moreover, the final CMTC as presented in Figure 5.3 was also helpful in regression testing and as a report of the testing activity itself.

Figure 5.3 considers the team members involved in various activities of the project as values "v" and the set team members as the aggregation "A" of these values. Moreover the CMTC relates the members with the systems considered as resources by the participants of the testing activities (such as identification of test data for testing) and represented with "R". Most of the modules are related to or execute both the test cases "Pricing Bug" and "Available Quantity". However, PO will not execute the unrelated Pricing Bug test case. In order to read this information from the CMTC of Figure 5.3, one must traverse horizontally in the row where the PO is mentioned under the set "Modules" and pick the second "f" in the row. The reader then browse vertically down the same column where the "f" is located and reaches the members of

the set "Test Case", where the blank in the row of "Pricing Bug" shows it is not related with the Module "PO". Similarly, customer special pricing will not execute the test case Available Quantity. The CMTC further explains how the test cases are associated with the data, and also shows the actual behavior of the system against the data as well as the severity level of the discrepancy or bug if one is found. For example, test case 'available quantity' when run against the module web report, under the business rule constraint "if the value or quantity is negative than it will appear as 0 on the report" is creating some problems. The pre-state or the available quantity being -5 and the report shows 0 in accordance with the business rule. The input value or addition to the quantity is 6, which should result in the new value being 1. However, the expected value appears as 6 and not 1, suggesting the system is calculating on the basis of the appeared value on the report instead of the actual value. The CMTC helps not only in identifying the defect and its all related aspects but also in identifying who is best suited to work on the defect, as well as associating a level of severity to the defect.

The following CMTC was used during testing at a software company called Perfapps for the repair management and sales system. In this CMTC we have used new notations added to the current Context Maps notation set to facilitate software testing; the notation set is discussed in Chapter 2.



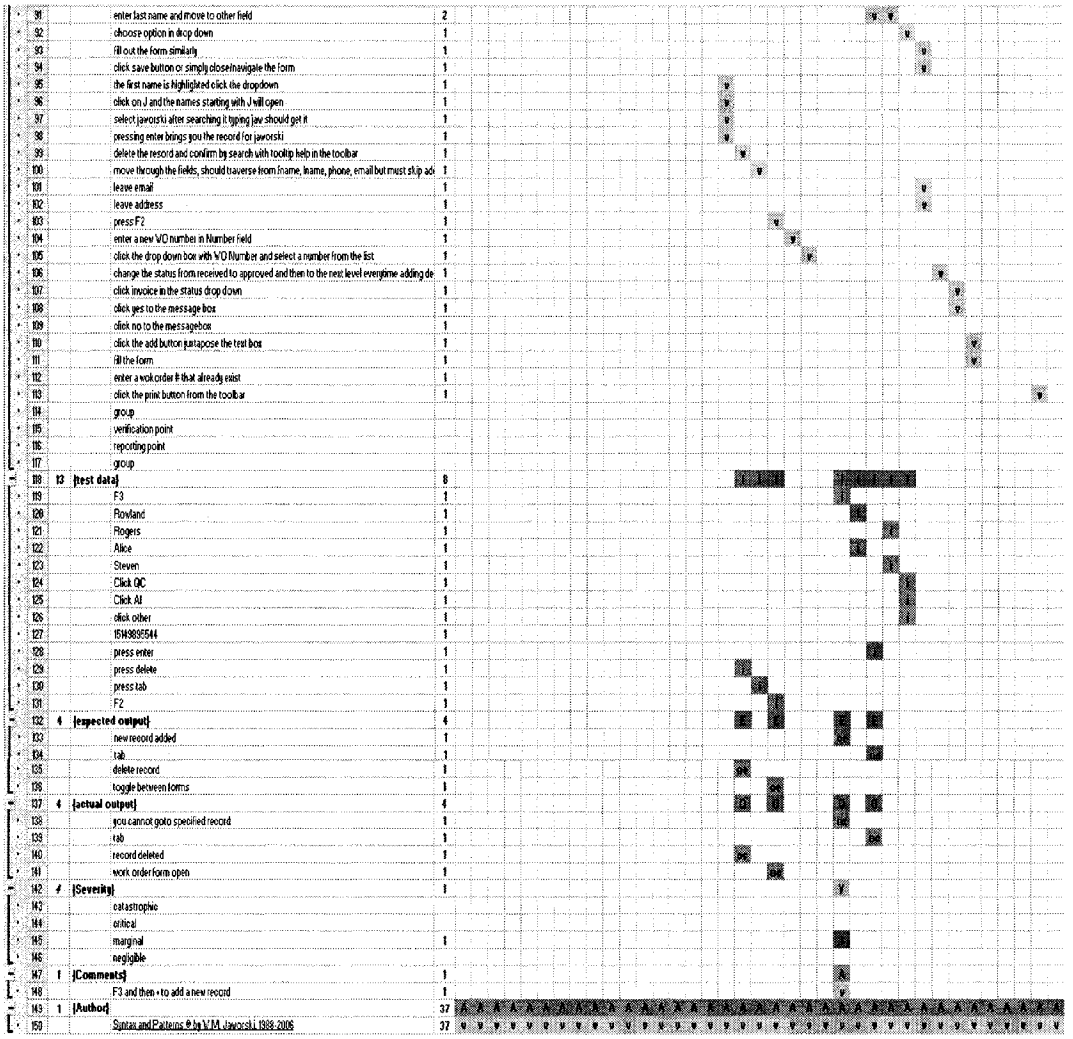


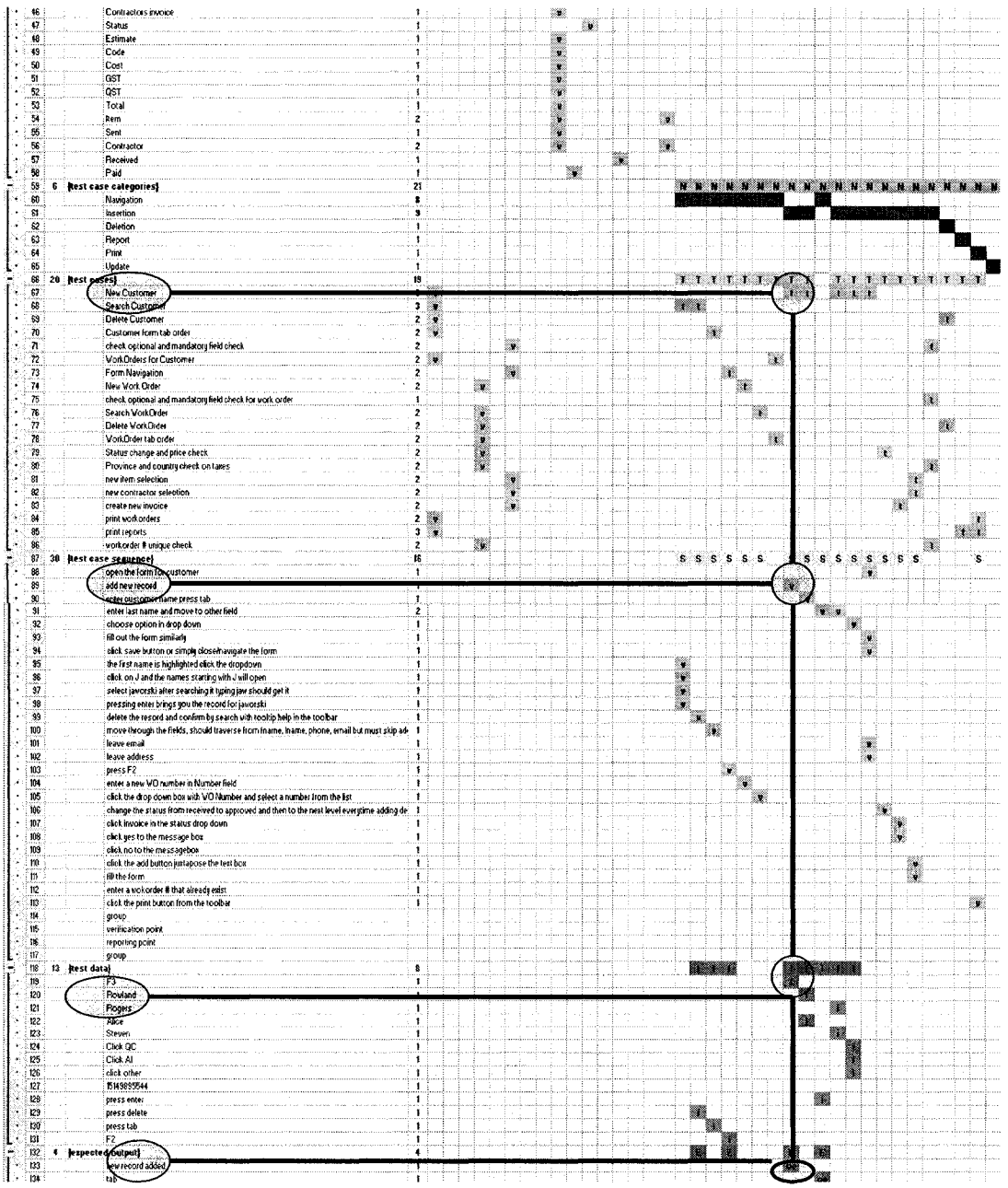
Figure 5.4 Context Maps used during testing of sales and repair management system

The CMTC in Figure 5.4 was developed to support the testing of the sales and repair management system at Perfapps Montreal. The CMTC helped in understanding the project being tested by presenting those involved in the project, their roles and responsibilities, modules, User interface components, test cases prepared, input data, expected output, actual behavior of the system, and in case of a bug the severity of the problem. The CMTC provide a simple way of relating the input data with the

expected and actual output. All of this helps in detecting the errors easily and improves understanding of the test case as well as the product being tested.

The CMTC presents Programmers, roles and the modules as well as their relationship, so it is easy to find the responsible person. For instance, if the problem is related to the reports module the programmer responsible is Fahad, whereas if it is related to the Work Order and/or Customer modules the programmers responsible are Haseeb or Halim. This makes finding and associating tasks and responsibilities (to fix defects) easy and transparent. The notations used are at the discretion of the Context Maps creator and can change, for example, the set modules is considered as a resource represented as 'R' in the CMTC, but this set could also be considered as a Node in the CMTC. Unlike the CMTC of Figure 5.3, the CMTC in Figure 5.4 also captures the metaphor such as text boxes, combo or dropdown lists, message boxes, command buttons and check boxes, identified as a resource (R). The set members are represented with the read notation "r", since there values are read from the software application forms. The presence of Metaphors in Figure 5.4 makes it less important to consult the application while trying to understand the test case. Furthermore, the CMTC uses three sets 'test case category', 'test cases within each category' and the 'sequence of steps to be followed to complete each test case' for the proper understanding of the test case hierarchy. The notation set of the Context Maps is extended to include "T" as representative of test cases. Test case sequence is represented as "S". Steps in the sequence are associated with input, expected and actual output data. The notation set is extended to deal with testing peculiarities, for example the input data, expected output, and actual output are represented with "I", "E", and "O" respectively. The set members for these sets are "i", "oe" and "oc" respectively. In case of a discrepancy between the expected and actual output "ox" is

used to represent actual output, this also shows how the notation set of the CMTC can be extended to model various concepts. The set "Severity" represented with notation "V" is used to measure the impact of the problem. For example, the only incorrect output revealed in the CMTC of Figure 5.4 is for the test case New Customer and the test case sequence "addnewrecord" as shown in the following Figure 5.4 (b).



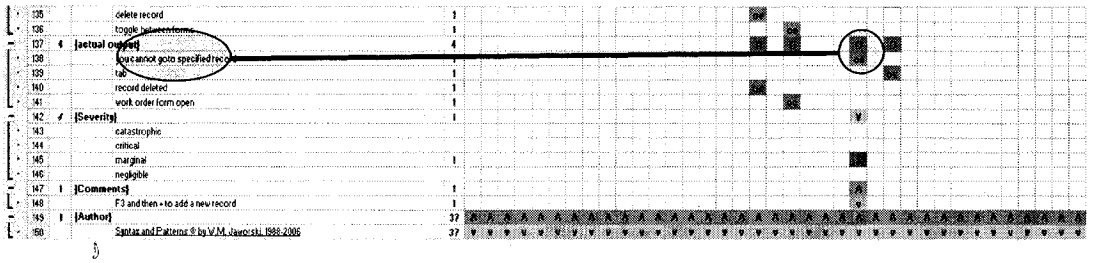


Figure 5.4 (b) Context Maps for testing of sales and repair management system

(To read this CMTC please use the same method as described in the previous CMTC in this chapter and explained in Chapter 2)

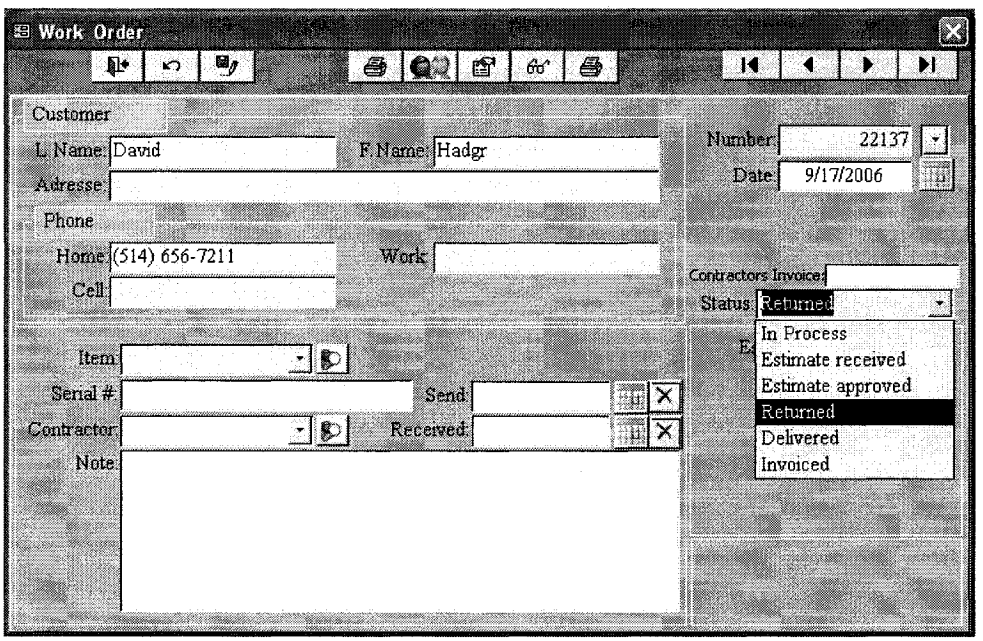
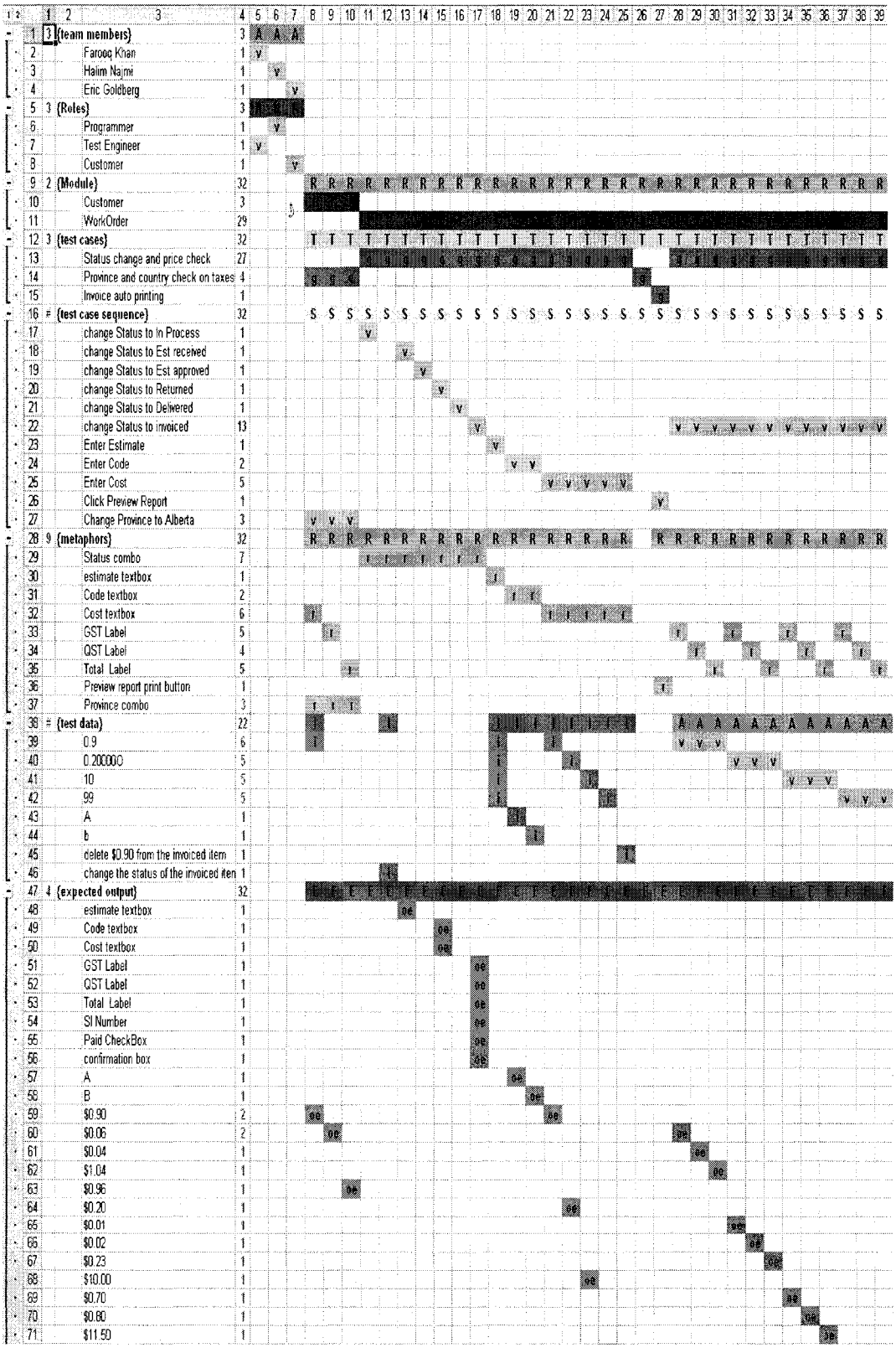


Figure 5.5 Work Order Form



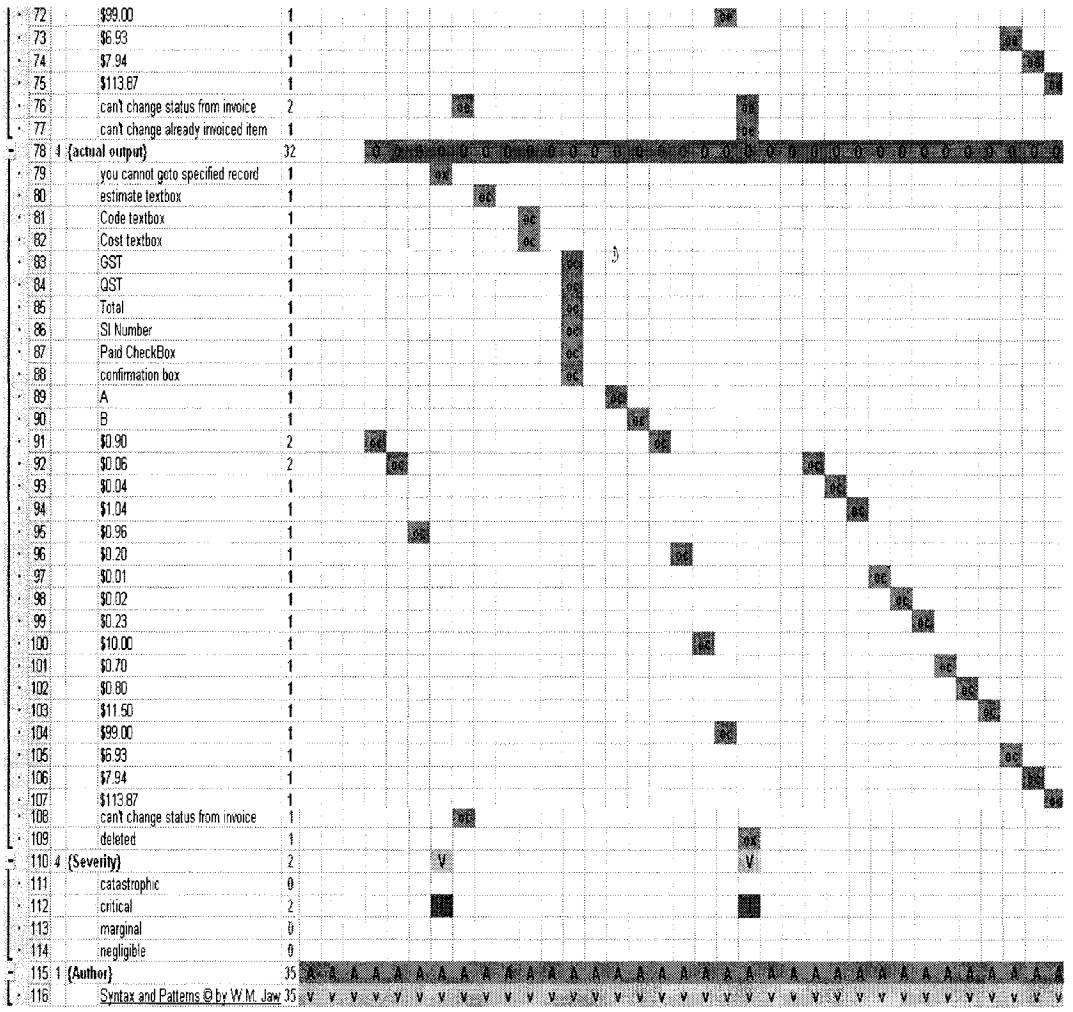


Figure 5.6 Context Maps used during debugging of sales and repair management system

The CMTC of Figure 5.6 describes test cases related to the effect of status change on price and tax calculation (statuses are shown in Figure 5.5). The CMTC also includes test cases to test the invoice printing. For instance, the Province and country check on taxes shows how the taxes are calculated on the basis of provinces or countries selected for the customer in the Customer form, discussed in Chapter 4. This test case (follow the set member notation used "g" vertically down to the set of test case sequence) is related to the test case sequence that has the steps "change to Alberta"

and further down this we can see the effected metaphor is "Province Combo" which is changed to Alberta and also the cost textbox, GST Label as well as total label are related to this test case sequence. This means that when the province Alberta is selected the metaphors that appear on the screen are Cost of the repair, GST, and the total cost. For Alberta the Provincial tax labeled as QST is not relevant. The test case further documents the input data that should be used; in this case it is 0.9 to be used as the cost as the "i" for input data appears right under the cost textbox (in the set of metaphor). The output set, if we continue vertically in the same column as the "i" in the set "test data", has "oe" for the value or set member 0.9. The reader of this CMTC can easily see that there is no discrepancy between the test data and actual output and this is also shown by the notation "oe" (representing expected output). Similarly, it can be read easily that other calculated values for GST and total cost are also as expected. However, within the same set "actual output", if the reader sees the "ox" in upward direction keeping an eye on the same column and matching the members of various sets, the test case related to invoice items where the invoice item is deleted has failed. This is because to the invoiced items should not get deleted; instead, a message box with warning that invoice items cannot be deleted should appear. By virtue of the CMTC and the presence of the related information it can be elicited that to confirm this aberration Customer "Eric Goldberg" is to be contacted and Programmer "Halim Najmi" is to rectify the problem.

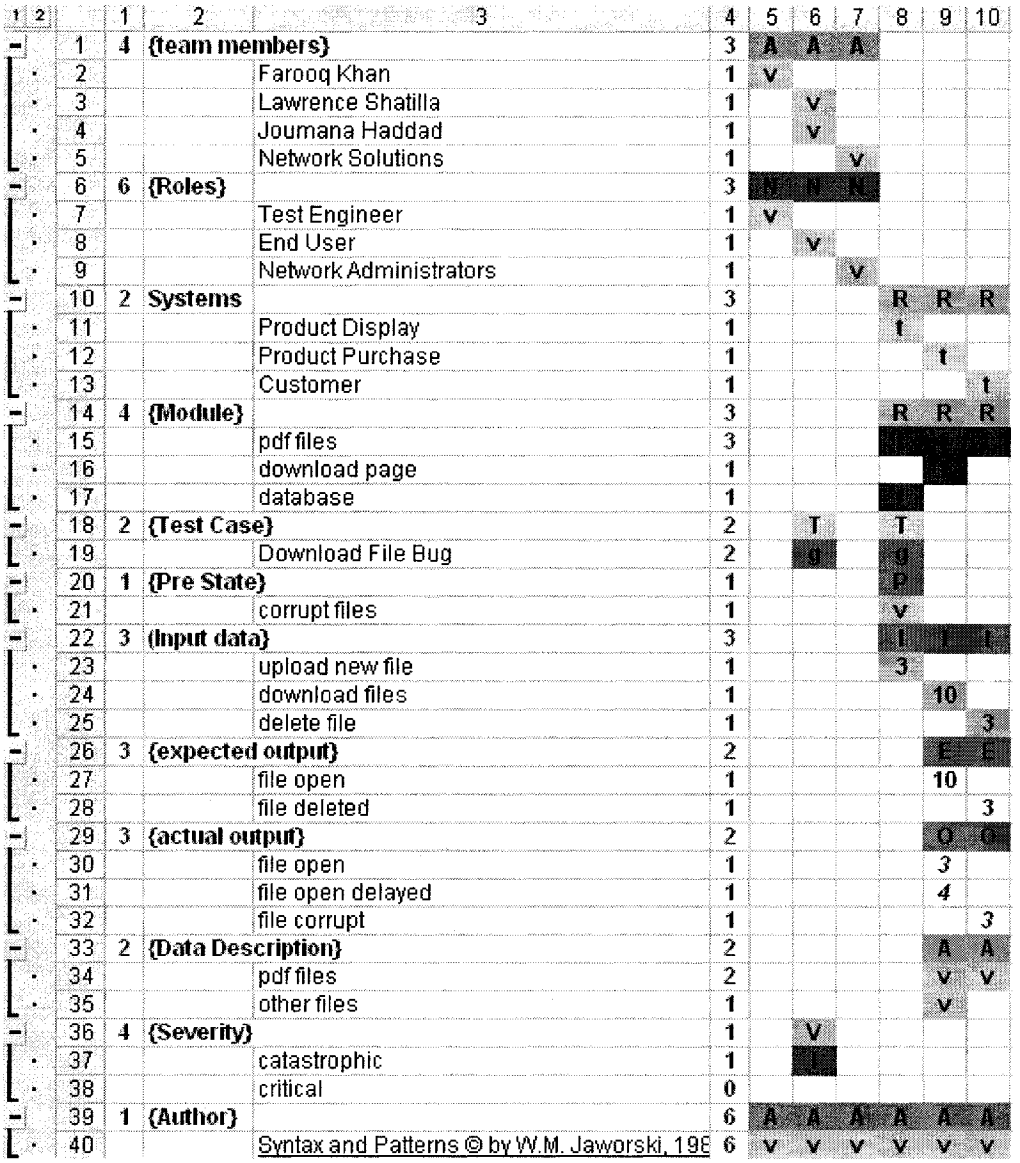


Figure 5.7 Context Maps used during debugging of web based product display system

Figure 5.7 illustrates how CMTC and the notation set of the CMTC can be used to represent iterations involved in a task. Digits are used as set member roles to emphasis on the fact that the task is recursive or iterative. By using digits as set member roles the number of times the particular task is performed during the process is shown.

The maps were used to debug the problems surfaced by virtue of limited download space and involve a third party besides the software vendor and the customer. With the help of Context Maps it was easy to identify the problem and capture the context which revealed a problem that would have been hard to detect by other methods.

It was reported by the customer that his clients were reporting very long delays in opening files and sometimes after the wait only corrupt files would appear. Creation of Context Maps showed that the system deteriorated its performance after a certain number of hits or downloads, especially of the pdf files. This leads the debugging effort to the hosting services and so the problem was easily traced and solved. Figure 5.7 shows that the problem was reported by the roles of the end users as "Download File Bug" and assigned the severity level of catastrophic as a main operation of the business was blocked. As we read vertically, we can see the system affected is the "product purchase", module being the "download page", and input data being "10" which suggests that ten times the page was downloaded during the test execution and was expected to get downloaded all ten times. The actual output of "file open" and "file open delayed" is 3 and 4 respectively and there are 3 "file corrupt" outputs. The difference in the expected and actual output suggests that there is a problem in the file download.

5.3 Performance based analysis of CMTC as a testing tool

Following graph shows how the Context Maps after a certain maturity in their usage help expedite the debugging and error handling process, the long term benefits are even more as the maps are reusable and reportable. We do realize, unfortunately, that a detailed research and mathematical analysis of the following results was not possible due to lack of time and resources.

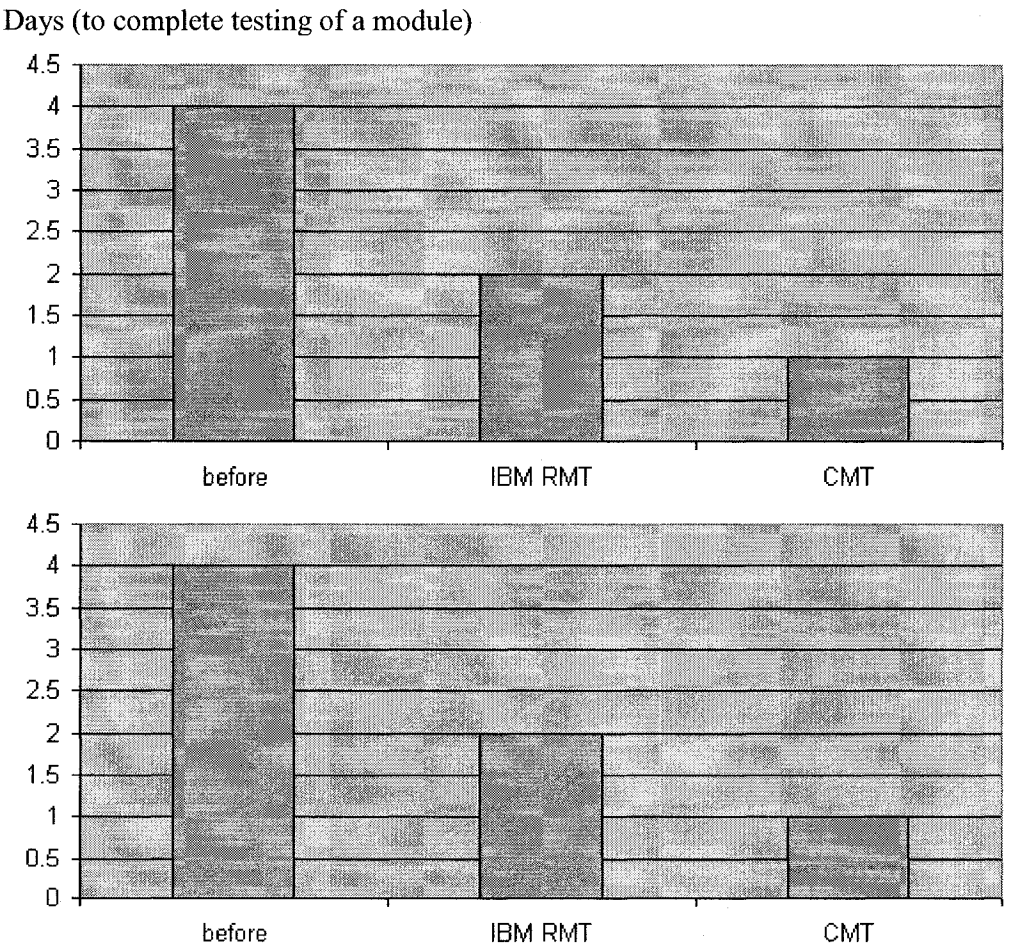


Figure 5.8 Time for catching bugs using different techniques (none, IBM RMT, CMT))

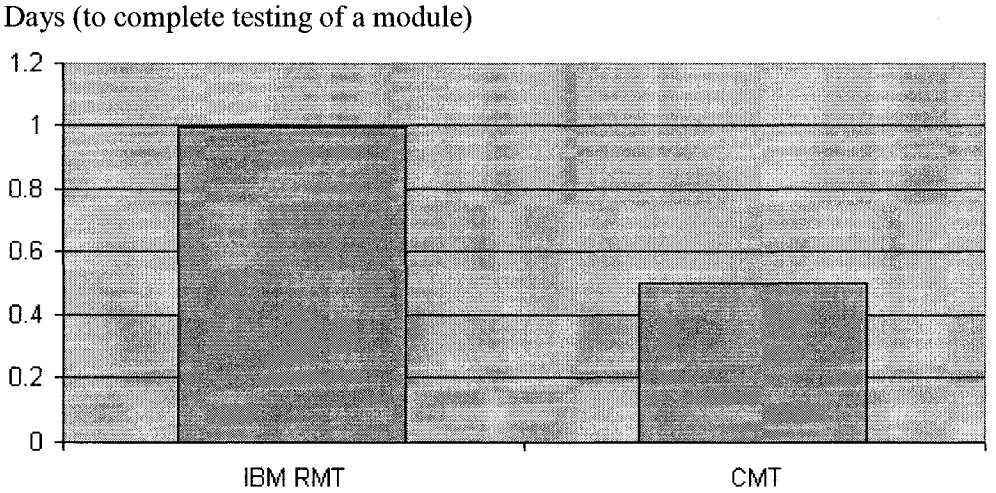


Figure 5.9 Time for catching bugs using IBM RMT, CMT

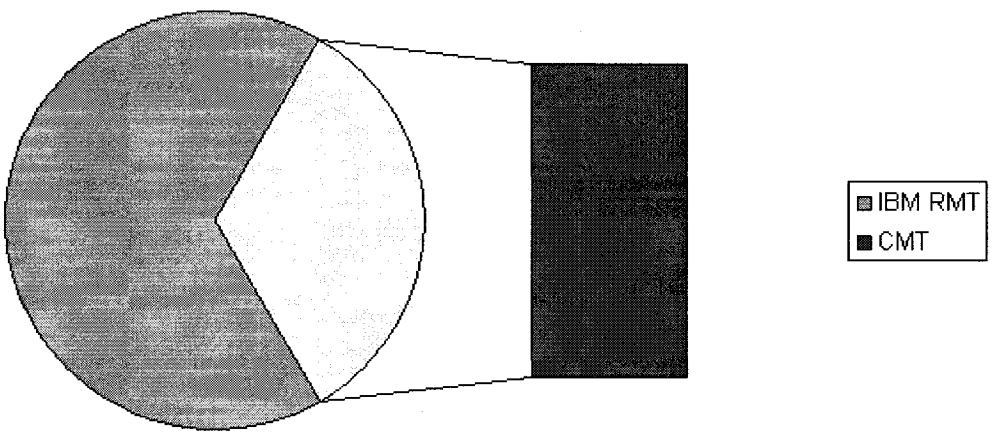


Figure 5.10 comparing IBM RMT and CMT

The following figures show how long it takes to extract some basic answers using CMTC and different defect tracking tools that we used. Questions includes 'which modules are effected by the defect', 'who was the original programmer of the

module', 'what is the input data used', 'which data resulted in the defect', 'what is the expected behavior and how is it different from actual behavior of the system', and 'what are the related test cases that needs to be tested during regression testing'.

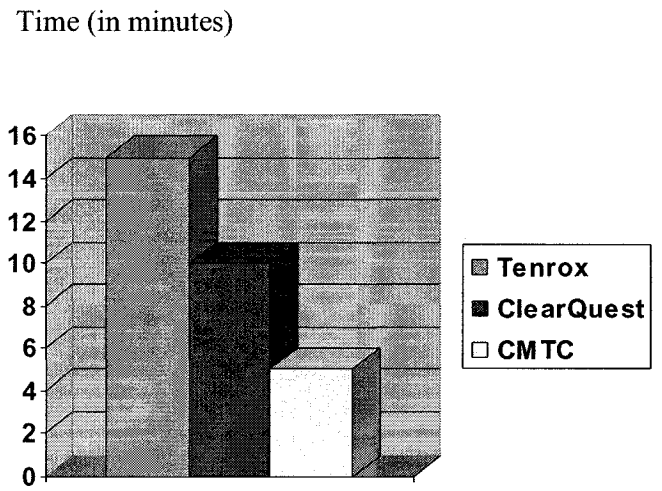


Figure 5.11 Comparing Tenrox, IBM ClearQuest, and CMTC

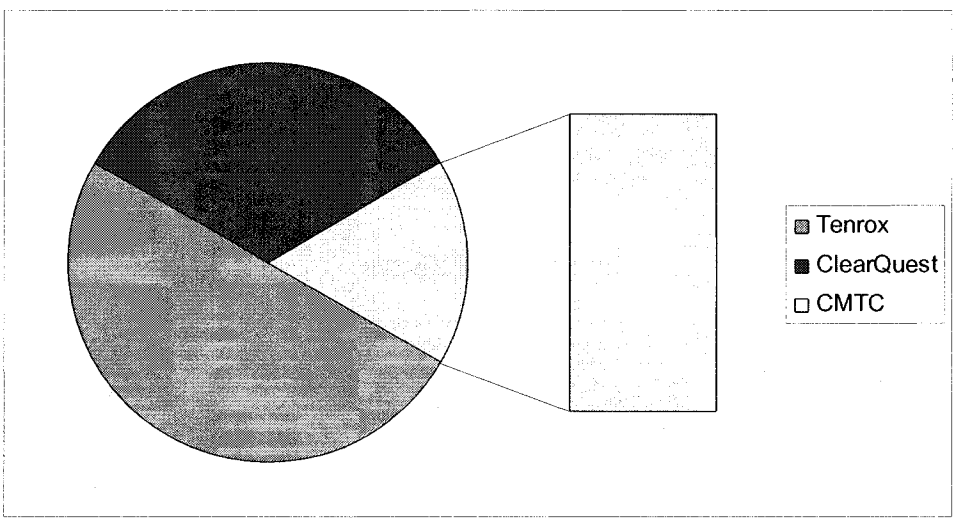


Figure 5.12 Comparing Tenrox, IBM ClearQuest, and CMTC

Most of the CMTC used in this chapter are final CMTC. Throughout the development of these CMTC documents the same structure and notation set is used which makes it easy for everyone participating (no matter how remotely or less often) to be able to play a vital role. However, it must be noted that the success of any software engineering concept depends very much on the dedication of resources and commitment of organization, nowhere more so than in QA or software testing. The result of programming is a product but the result of testing is not a product. It is rather an assurance, and testing software is successful when it finds hidden bugs, not when it passes the software as bug free. Therefore testing is a destructive process and organizational commitment is needed for any testing tool to be effective. The people involved in the implementation of the CMTC in various projects that we have presented in this chapter, considered CMTC as an improvement in software testing and defect tracking as well as software test management, especially for small customization projects for defect and enhancement testing.

Chapter 6

6.1 Conclusions

Our lives are increasingly dependent on computer software and their proper operations. It is difficult to imagine any department or company that does not use computer software. However, further sustained growth in Information technology (IT) and its integration into businesses mandates significant improvements in ensuring that the business requirements have been met through the delivered software products. User Acceptance Testing is the activity in the Software Development Life Cycle (SDLC) that ensures software is fulfilling the requirements and is the right product for the customers' needs.

Our work involves User Acceptance Testing in small customization projects for defects and enhancements. We studied software testing processes and applied software tools in various projects, as discussed in Chapter 3. We identified issues pertinent to the effectiveness of software testing and explore avenues to integrate user acceptance test case planning and documentation in the software process.

Our contribution through this research involves an approach which improved the creation and management of user acceptance testing in small customization projects.

We propose Context Maps Test Case (CMTC) as a tool to achieve our goals. While many approaches and toolsets which we discussed in Chapter 3 focuses on separate tools for test case authoring, test management and defect tracking, our proposed approach uses CMTC as a global tool that covers all aspects of User Acceptance testing as we have shown in Chapter 4. To the best of our knowledge, no other tool supports software testing, defect tracking and software test management in small customization projects as efficiently and cost effectively as CMTC. We compare the results attained from our application of CMTC and other tools in Chapter 5.

CMTC documents contain information related to test management such as the names of the participants involved in the development and testing, alongside their roles. This information about the responsibilities and roles of participants along with the module they have worked on makes it easy for management to assign resources. For example, assigning a programmer to fix a defect is straightforward for managers, as the CMTC document has the information of the programmers for the module. CMTC documents also contain test case related information such as test data including input data, expected and actual output behavior, and test case sequence. Furthermore, the CMTC document contains information for defect tracking such as severity and priority of the defect, as well as module and version where the defect is identified. Such an application of a single tool for user acceptance testing, without segregation of information across various tools and tabs, resolves many problems in small customization projects (for defect and enhancement testing).

The problems that we resolve through our application of CMTC include providing a consistent and simple set based interface for software test related documentation. The

resulting test cases or CMTC documents are easy to understand, expandable to cover related information, and saves time for both testers and managers alike, as shown in Chapter 4. These benefits of CMTC are because of the presentation of all the information grouped in sets, within one simple screen. The information is presented through Maps in such a way that the relation among the various aspects of testing can be read easily. Moreover, the simple format of documenting test cases foster communication and participation of members from different phases of SDLC to help review and add information in CMTC documents as discussed in Chapter 4. The effectiveness of CMTC as a tool for testing of small customization projects for defects and enhancements was recognized by those involved.

6.2 Future Work

Similar to the User Acceptance testing, requirement specification is also an umbrella activity and has a tremendous effect on the success of the project. Research similar to our work could be carried out for requirement specification. Comparisons can be made with requirement definition and management tools such as IBM Rational Requisite Pro, and Borland Caliber Analyst. Requirements specification involves people from different background working together. Moreover, requirements need to be understood by almost every member and team involved in the Software Project. Therefore, it is easy to envisage a role for Context Maps and a tool similar to CMTC, to document requirements and manage the process by facilitating the communication among the members from various teams and phases.

It is necessary to understand contexts while testing any product, hence, another issue for future work is the research that is needed to establish the significance of Context Maps in testing other engineering products.

Context Maps provides an opportunity to improve measurement and estimation of software projects and project planning. There are many related aspects such as available resources, time constraints, software and hardware constraints, which need to be looked at while planning a software project. Context Maps can help in capturing and comparing these aspects. Similarly for estimation, Context Maps can present itself as a tool to compare historic data and relate expert opinions.

References

[1] D. Gelperin, B. Hetzel. The growth of software testing. In *Communications of the ACM*, Volume 31 Issue 6, ACM Press, June 1988.

[2] Muthu Ramachandran. Requirements-Driven Software Test: A Process-Oriented Approach. In *ACM SIGSOFT Software Engineering Notes*, Volume 21, Issue 4, ACM Press, 1996. 66-70

[3] Cem Kaner, *testing computer software*, second edition, Wiley, 1999

[4] Rex Black, "Investing in software Testing: The Cost of Software Quality", 2000; http://www.compaid.com/caiinternet/ezine/cost_of_quality_1.pdf

[5] Borland White Paper, "Eliminate the Testing Bottleneck with Requirements-Based Testing", August 2006; http://uk.builder.com/whitepapers/0,39026692,60256641p-39001054q,00.htm

[6] Agitar White Papers, "Software testing and Quality Assurance", Jul 2006; http://www.agitar.com/solutions/resources/webinars.html

[7] Gregg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakuri, Brian Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *24th International Conference on Software Engineering*, 2002, pp. 130 - 140.

[8] Jussi Koskinen, "Software Maintenance Cost", April 2007; http://www.cs.jyu.fi/~koskinen/smcosts.htm, University of Jyväskylä, Finland

[9] Ozgur Turetken, David Schuff, Ramesh Sharda, Terence T. Ow. Supporting systems analysis and design through fisheye views. In *Communications of the ACM archive,* Volume 47 , Issue 9, ACM Press, 2004.

[10] Visual Studio Team System Business Value Whitepaper, "Driving Productivity and Adaptive Businesses with Intuitive Integrated Application Life-Cycle Management", Nov 2005; http://msdn.microsoft.com/vstudio/why/vsts_whitepaper/default.aspx

[11] Master's thesis by Sanaz Rahmati, *Converting Legacy Relational Database to XML Database through Context Map.* Montreal QC : Concordia University , 2006.

[12] Master's thesis by Kang Zhou , *CONTEXT+: Development Environment for 3P-able Context Maps.* Montreal QC : Concordia University , 2001

[13] Agitar White Papers, "Software testing and Quality Assurance", Jul 2006; http://www.agitar.com/solutions/resources/webinars.html

[14] Roger S. Pressman, *Software Engineering: A Practitioner's Approach,* Fourth Edition, McGraw-Hill, 1997

[15] Rex Black, *Managing the testing process*, second edition, Wiley, 2002

[16] Ilene Burnstein, *Practical Software Testing, a process oriented approach*, Springer, 2003

[17] Ian Sommerville, *Software Engineering,* Fifth Edition, Addison Wesley, 1995

[18] Debra Richardson, Paola Inverardi. ROSATEA: International Workshop on the Role of Software Architecture in Analysis E(and) Testing. In *ACM SIGSOFT Software Engineering Notes archive*, Volume 24 , Issue 4, ACM Press, 1999

[19] IBM Rational Manual Tester, "trial download and documentation", 12 March 2007; http://www-306.ibm.com/software/awdtools/tester/manual/

[20] Roger S. Pressman, "Chapter 3", in *Software Engineering: A Practitioner's Approach*, Fourth Edition, McGraw-Hill, 1997

[21] Master's thesis by Yaozhong Chen, *Application of the Context Maps Modelling the Rational Unified Process and Rational ClearQuest.* Montreal QC: Concordia University, 2000

[22] W. Richards Adrion, Martha A. Branstad, John C. Cherniavsky. Validation, Verification, and Testing of Computer Software. In *ACM Computing Surveys*, Volume 14, Issue 2, ACM Press, 1982.

[23] M. D. Smith and D. J. Robson. A Framework for Testing Object-oriented Programs. *Journal of Object-Oriented Programming*, v 5, n 3, June 1992. 45-53.

[24] Amit Paradkar. Inter-Class Testing of O-O Software in the Presence of Polymorphism. In *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research,* IBM Centre for Advanced Studies Conference archive, IBM Press, 1996.

[25] Rex Black, *Critical Testing Processes*, Addison Wesley, 2004

[26] Roger Ferguson, Bogdan Korel. Illinois Institute of technology The Chaining Approach for Software Test Data Generation. In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM Press, Volume 5 , Issue 1, 1996. 63-86

[27] Qian Yang, J. Jenny Li, and David Weiss. A Survey of Coverage Based Testing Tools". In *International Conference on Software Engineering archive, Proceedings of the 2006 international workshop on Automation of software test, Shanghai, China*, ACM Press, 2006. 99-103

[28] Kai-Yuan Cai, T. Y. Chen, Yong-Chao Li, Wei-Yi Ning, Y. T. Yu. Adaptive Testing of Software Components. In *Proceedings of the 2005 ACM symposium on Applied computing,* ACM Press, 2005. Pages: 1463 - 1469

[29] Imran Bashir and Amrit L. Goel, *Testing OO Software Life Cycle Solutions*, Springer, 2000

[30] Hong Zhu, Patrick A. V. Hall, John H. R. May. Software unit test coverage and adequacy. In *ACM Computing Surveys (CSUR)*, ACM Press, Volume 29 , Issue 4, 1997. Pages: 366 - 427

[31] Ilene Burnstein, *Industrial Application of the TMM* in *Practical Software Testing, a process oriented approach*, Springer, 2003

[32] Scott Ambler, "Web services programming tips and tricks: How to organize a software development team", 2 Nov 2000;

http://www-128.ibm.com/developerworks/webservices/library/ws-tip-team.html

[33] IEEE Standard for Software Test Documentation: IEEE Standard 829-1998. ISBN 0-7381-1443-X, 16 Dec 1998

[34] "Tenrox Release 9 User Guide – Administrator", August 2006, http://www.tenrox.com Note: Private Communication

[35] Production Support Testing Projects Liberty Mutual Indianapolis, March 2007,

Note: Private Communication