Presentation
Paper
Bio

Return to Main Menu

PRESENTATION

W3

Wednesday, Dec 6, 2000

Creating Requirements from from Test Specifications

Anne Mette Hass

International Conference On
Software Testing, Analysis & Review
DEC 4-8, 2000 • COPENHAGEN, DENMARK

Creating Requirements from Test Specifications

Anne Mette Jonassen Hass, DELTA

DELTA, Danish Electronics, Light & Acoustics
Venlighedsvej 4
DK-2970 Hørsholm
Denmark

Tel.: (+ 45) 45 86 77 22
Fax : (+45) 45 86 58 98
Email: amj@delta.dk

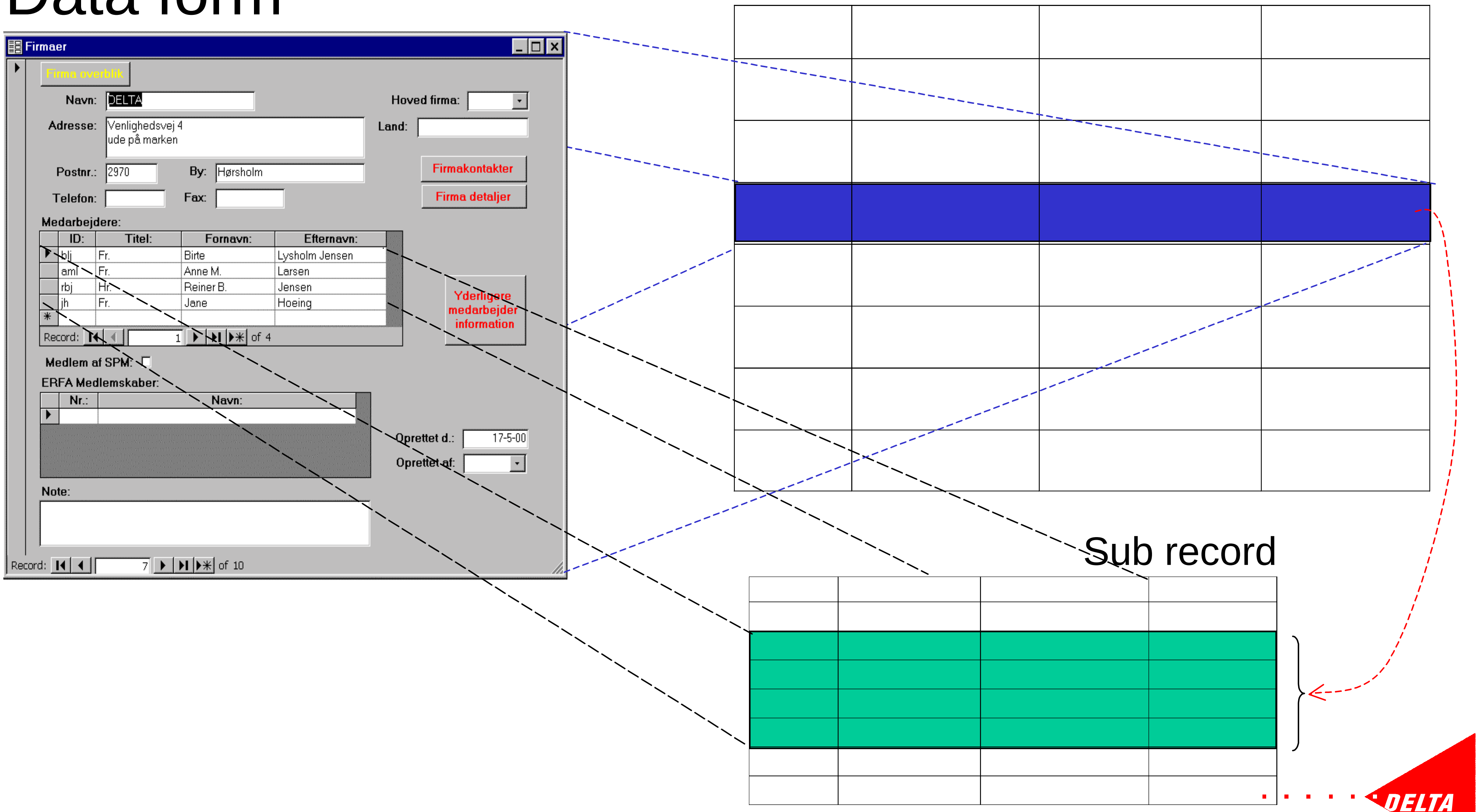
The case

- a third party system test assignment
- an administrative Windows system
- version 3 of a system in production
- just over 100 forms
- **goal:** to ensure credibility
- timeframe: 2 ½ months
- **no written requirements**

Thanks to Mark Fewster and my test colleagues at DELTA.

What the user sees

Data form



Firmaer

Firma overblik

Navn: DELTA

Hoved firma:

Adresse: Venlighedsvej 4
ude på marken

Land:

Postnr.: 2970 By: Hørsholm

Telefon: Fax:

Firmakontakter

Firma detaljer

Medarbejdere:

	ID:	Titel:	Fornavn:	Efternavn:
▶	blj	Fr.	Birte	Lysholm Jensen
	aml	Fr.	Anne M.	Larsen
	rbj	Hr.	Reiner B.	Jensen
	jh	Fr.	Jane	Hoeing
*				

Record: 1 of 4

Yderligere medarbejder information

Medlem af SPM: □

ERFA Medlemskaber:

	Nr.:	Navn:
▶		

Oprettet d.: 17-5-00

Oprettet af:

Note:

Record: 7 of 10

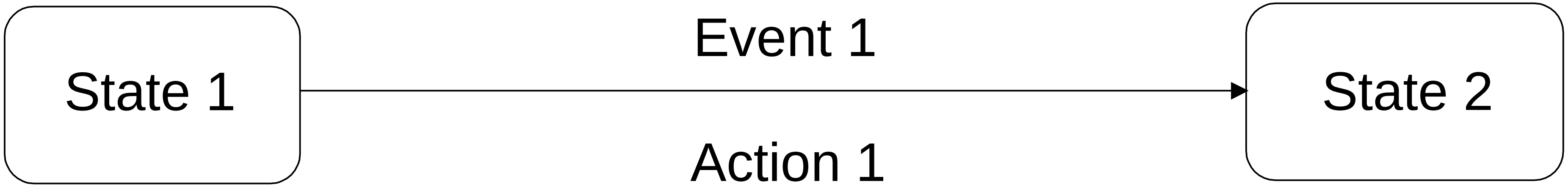
Main record

Sub record

What the tester sees

- a gigantic state machine

creating or updating data
or leaving the field
or leaving the record



State 1

Event 1

Action 1

State 2

the present data
+ the current field
+ the data handling mode

validation of data or
dependent change of data

What to specify

We decided to make generic test scripts for:

- navigation
- general behaviour

and tables for each form:

- data to be handled and how
- fields and their characteristics
- push buttons

Specification overview

Navigation commands

Stationary commands

Command	Reaction
<Crtl>+<Del>	the record is deleted, the next record is shown and the same field is active if the deleted record is the last, the previous record is shown
<F1>	the help form pertaining to the active form becomes active

Field commands

	Reaction for position		
Command	first field	'in between' field	last field
<End>	last field is active		
<Enter>	next field in tab order is active		last field is still active
<Home>	first field is active		
<Shift>+<Tab>	first field is still active	previous field in tab order is active	
<Tab>	next field in tab order is active		last field is still active
Left click in a field	chosen field is active		

Record commands

	Reaction for position		
Command	first record	'in between' record	last record
<Ctrl>+ <End>	last field for last record is active		
<Page down>	next record is active, same field is active		no change – no validation triggers a 'Notification'
<Page up>	no change – no validation triggers a 'Notification'	previous record is active, same field is active	
Right record browser error	next record is active, first field is active		no change – no validation
Left click on [X]	the form closes		

General behavior

Test Case G1.1: Generic test case for test of a data form.

Description: This test case tests the generel behaviour, pertaining to all data forms.

The test case has the following parameters:

F: the field table for the form under test

D: the data table for the form under test

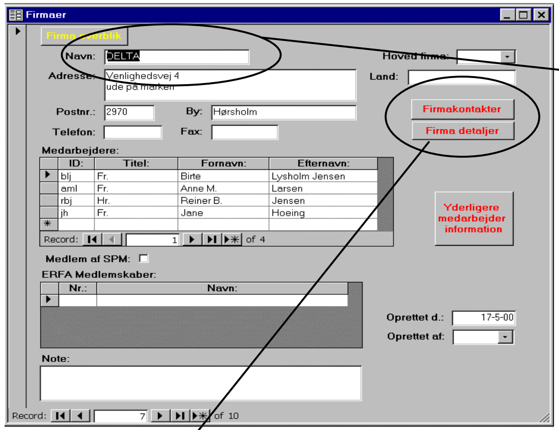
P: the push button table for the form under test

Step	Input	Expected output	Result
a.	Examine the left side of the form.	Record selector is present.	

Test Case G1.4: .. error -form.

Test Case G3.1: .. creation of main record.

Test Case G2.7: .. field type.



Firmaer

Navn: DELTA

Hoved firma:

Adresse: Venlighedsvej 4
ude på marken

Land:

Postnr.: 2970 By: Hørsholm

Telefon: Fax:

Medarbejdere:

ID:	Titel:	Fornavn:	Efternavn:
blj	Fr.	Birte	Lysholm Jensen
aml	Fr.	Anne M.	Larsen
rbj	Hr.	Reiner B.	Jensen
jh	Fr.	Jane	Hoeing

Record: 1 of 4

Medlem af SPM: □

ERFA Medlemskaber:

Nr.:	Navn:

Oprettet d.: 17-5-00

Oprettet af:

Note:

Record: 7 of 10

Push button table

Push buttons		Validation; failure type			Dependent behaviour		
		Left mouse click	Short cut	<Enter>	Left mouse click	Short cut	<Enter>
5	[Firmakontakter]				The form Firmakontakter is current. The only data available is for the current company.		
6	[Yderligere medarbejder information]	An employee must be current in the sub-form Medarbejdere.			The form Medarbejdere is current. The only data available is for the current employee.		

Data table

Main record	1. sub-level	Default	Sorting	Allowed actions
Company		first company sorted by name	acsending by name	CRUD D: entries in MediFirma are deleted entries in FirmaiERFA are deleted
	MediFirma	all employees for company	acsending by last name	CRUD
	FirmaiERFA	all ERFAgroups assigned to company	ascending by number	CRUD

Field table

Fields		Tab order F;B	Default	Type	Validation; failure type	Dependent behaviour
		<Tab> <Shift><Tab> <Enter>	*	C U	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒
1	Navn	2;1	*blank*	A50		
2	Hovedfirma	3;1	*blank*	S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E C I N U I N	
3	Adresse	4;2	*blank*	A250		
4	Postnummer	5;3	*blank*	A10		If a valid Danish postnummer, and the By field is blank, the By will be filled in with the corresponding name. C I N U I N
Sub-form: Medarbejdere						
L1-1	ID	L1-2;8	*blank*	A4		If the creation starts in another field in the list, the ID is filled in with the next running number prefixed by and M.

Navigation commands, classification

Command type	Focus change	Validation triggered
Stationary commands	the focus does not change	none
Field commands	the focus change from one field to another	the field, that is being left
Record commands	the focus change from one record to another	all fields in the record being left

Navigation may be handled by menus, keyboard, mouse, etc..

no initial consensus =>

not very user friendly interface =>

low confidence

Navigation commands, examples

Command	Reaction for position		
	first field	‘in between’ field	last field
<End>	last field is active		
<Enter>	next field in tab order is active		last field is still active
<Home>	first field is active		
<Shift>+<Tab>	first field is still active	previous field in tab order is active	
<Tab>	next field in tab order is active		last field is still active
Left click in a field	chosen field is active		

General behaviour

- the general layout and behaviour of each type of form
    - data form, message boxes
- the handling of different types of fields
    - string, integer, real, date, selection list, etc…..
- the handling of different data actions
    - create, update, delete
- the handling of failed validations
    - error, warning, notification

Generic test script, example

Test Case G1.1: Generic test case for test of a data form.

Description: This test case tests the general behaviour, pertaining to all data forms.

The test case has the following parameters:

F: the field table for the form under test

D: the data table for the form under test

P: the push button table for the form under test

Step	Input	Expected output	Result
a.	Examine the left side of the form.	Record selector is present.	
b.	Examine the top of the form.	Form name is presented in white to the right of the icon in the top left corner.	
c.	Examine the fields – also in terms of sub-forms and fields in sub-forms.	The specified fields and sub-forms are present.	See marks in **F**.
d.	Examine the tab order, using different means of navigation.	The tab order is as specified in **F**.	See marks in **F**.
e.	Click the left mouse button in various fields.	Fields become active when clicked on. Fields without tab indicator cannot be activated.	
f.	Examine the contents of the fields.	Data when first opened is as specified in **D**.	
g.	Browse through the records.	Sorting is as specified in **D**.	
h.	Examine the push buttons.	The push buttons specified in **P** are present.	See marks in **P**.

Data table, example

Main record	1. sub-level	Default	Sorting	Allowed actions
Company		first company sorted by name	ascending by name	CRUD D: entries in MediFirma are deleted entries in FirmaiERFA are deleted
	MediFirma	all employees for company	ascending by last name	CRUD
	FirmaiERFA	all ERFAgroups assigned to company	ascending by number	CRUD

Field table - first step, example

Fields		**Tab order F;B**	**Default**	**Type**	**Validation; failure type**	**Dependent behaviour**
1	Navn	2;1	*blank*	A50		
2	Hovedfirma	3;1	*blank*	S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E	
3	Adresse	4;2	*blank*	A250		
4	Postnummer	5;3	*blank*	A10		If a valid Danish postnummer and the By field is blank, the By will be filled in with the corresponding name.
Sub-form: Medarbejdere						
L1-1	ID	L1-2;8	*blank*	A4		If the creation starts in anothe field in the list, the ID is filled with the next running number prefixed by an M.
End of sub-form						

Field table - second step, example

Fields		Tab order F;B				Default			Type	Validation; failure type						Dependent behaviour					
		<Tab>	<Shift><Tab>	<Enter>		*			C	C	<Tab>		I	☒		C	<Tab>		I	☒	
													N	☒					N	☒	
									U	U	<Tab>		I	☒		U	<Tab>		I	☒	
													N	☒					N	☒	
1	Navn	2;1				*blank*			A50												
2	Hovedfirma	3;1				*blank*			S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E											
										C			I								
													N								
										U			I								
													N								

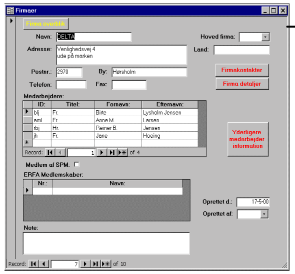
Push button table, example

Push buttons		Validation; failure type			Dependent behaviour		
		Left mouse click	Short cut	<Enter>	Left mouse click	Short cut	<Enter>
5	[Firmakontakter]				The form Firmakontakter is current. The only data available is for the current company.		
6	[Yderligere medarbejder information]	An employee must be current in the sub-form Medarbejdere.			The form Medarbejdere is current. The only data available is for the current employee.		

Performing the test

- specific test script to bind the tables and the generic test scripts together
- actual test assembled from the building blocks according to decided scope
- registration of results directly in tables and generic test scripts
- consider test data and register data used and obtained

Effort is deceptive!



Navn: DELTA

Hoved firma:

Adresse: Venlighedsvej 4
ude på marken

Land:

Postnr.: 2970 By: Hørsholm

Telefon: Fax:

Medarbejdere:

ID:	Titel:	Fornavn:	Efternavn:
blj	Fr.	Birte	Lysholm Jensen
aml	Fr.	Anne M.	Larsen
rbj	Hr.	Reiner B.	Jensen
jh	Fr.	Jane	Hoeing

Record: 1 of 4

Medlem af SPM: □

ERFA Medlemskaber:

Nr.:	Navn:

Oprettet d.: 17-5-00

Oprettet af:

Note:

Record: 7 of 10

Testgroup: 5.1 Firmaer		
Testcase: 5.1.1 Handling companies		
Description: This test case tests the form Firmaer, i.e. its appearance, data presentation and navigation. The defaults are tested for creation. The field types, validations, and dependent behavior are tested for creation and edit. The handling of deletion of records is tested as well.		
Preconditions: The system must be started with the test database in the initial state.		
Tracing: Firmaer data table (**D**), Firmaer field table (**F**), Firmaer push button table (**P**)		
Performed by:	**Date:**	**Accepted:** Yes / No

Step	Input	Expected output	Result
1.	Select Firmaer on the main menu.	The form Firmaer is the active form.	
2.	Perform G1.1, the generic testcase for data forms for the form Firmaer.	As stated in testcase G1.1. Don't forget to log data!	
3.	Perform G1.2, the generic testcase for sub-forms for the sub-form Medarbejdere.	As stated in testcase G1.2.	
4.	Perform G3.1, the generic testcase for creation of main records for main record Company .	As stated in testcase G3.1.	
5.	Perform G3.3, the generic testcase for edit of main records for main record Company .	As stated in testcase G3.3.	
6.	Perform G3.5, the generic testcase for deletion of main records for main record	As stated in testcase G3.5.	

1

Test Case G1.1: Generic test case for test of a data form.

Description: This test case tests the generel behaviour, pertaining to all data forms.

The test case has the following parameters:

F: the field table for the form under test

D: the data table for the form under test

P: the push button table for the form under test

Step	Input	Expected output	Result
a.	Examine the left side of the form.	Record selector is present.	
b.	Examine the top of the form.	Form name is presented in white to the right of the icon in the top left corner.	
c.	Examine the fields – also in terms of sub-forms and fields in sub-forms.	The specified fields and sub-forms are present.	See marks in **F**.
d.	Examine the tab order, both forwards and backwards using different means of navigation.	The tab order is as specified in **F**.	See marks in **F**.
e.	Click the left mouse button in various fields.	Fields become active when clicked on. Fields without tab indicator cannot be activated.	

1.1

1.2

Fields		Tab order F;B	Default	Type	Validation; failure type	Dependent behaviour
		<Tab> <Shift><Tab> <Enter>	*	C U	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒
1	Navn	2;1	*blank*	A50		
2	Hovedfirma	3;1	*blank*	S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E	
					C I N U I N	
3	Adresse	4;2	*blank*	A250		
4	Postnummer	5;3	*blank*	A10		If a valid Danish postnummer, and the By field is blank, the By will be filled in with the corresponding name.
						C I N U I N

Sub-form: Medarbejdere

Test Case G1.2: .. sub-form.

Test Case G1.4: .. error -form.

3

Test Case G3.1: .. creation of main record.

3.2.x

Test Case G2.7: .. field type.

We made it - just

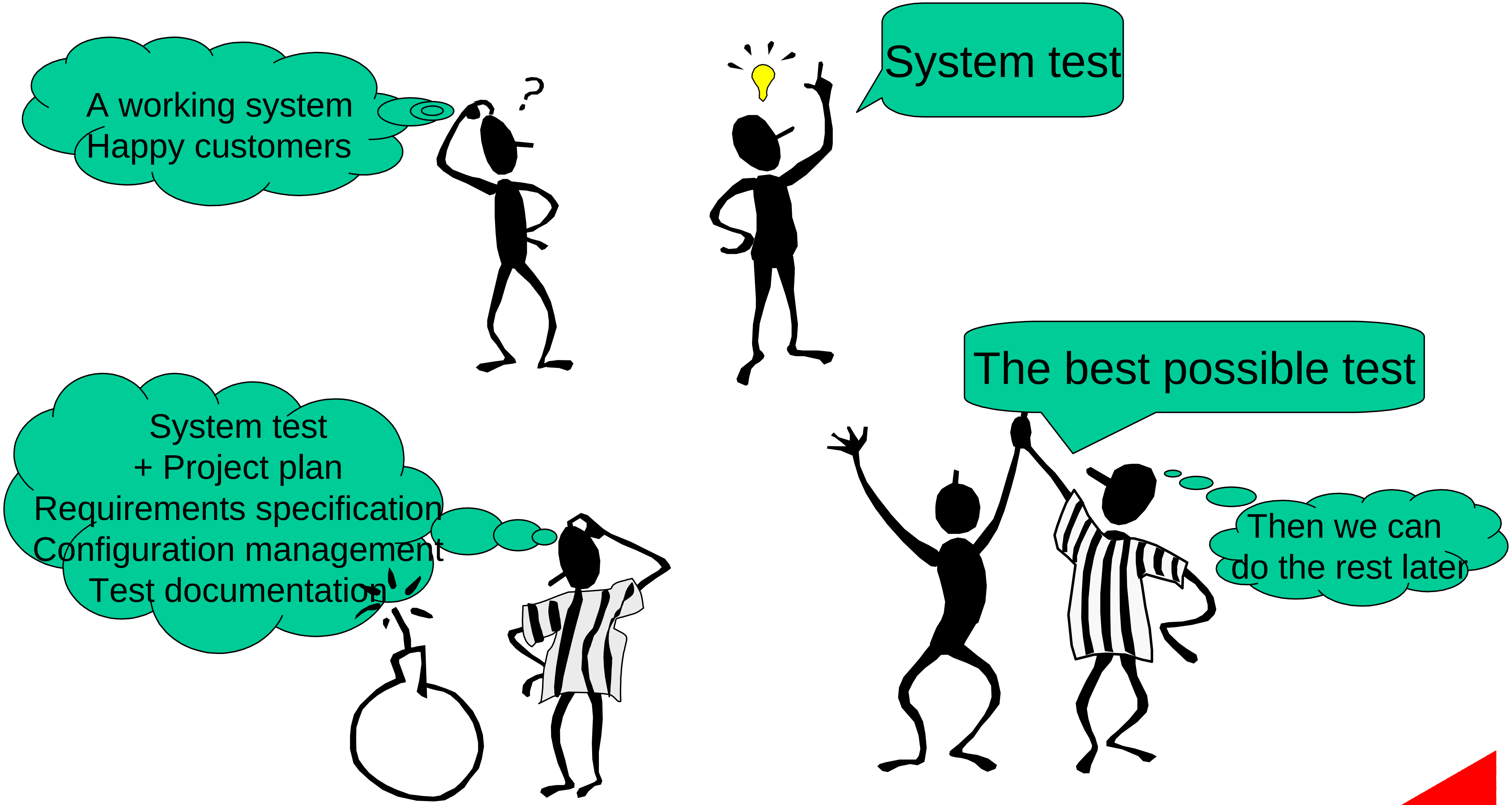
- 1 developer, 2 support people, 2 professional testers, 2 testers (secretaries)
- 2 weeks planning
- 7 weeks testing - debugging - re-testing
- 80% of the forms planned for test - in the prioritised order
- over 200 error reports

Using test tables as specifications

- tables hold a lot of information in compact form
- may be harder to understand initially
- same tables used by designers, developers, testers, and support people
- one update => 4 purposes
- tables also serve as test reporting tool

what ever you do - you have to do it

Third party testing



Testing is fun

Working with a group of people
to deliver a system
that is better than it would have
been without you
is a great source of personal and
professional fulfilment!

Creating Requirements from Test Specifications

Anne Mette Jonassen Hass

DELTA, Danish Electronics, Light & Acoustics
Venlighedsvej 4
DK-2970 Hørsholm
Denmark

Tel.: (+ 45) 45 86 77 22
Fax : (+45) 45 86 58 98
Email: amj@delta.dk

Abstract

This paper describes how we developed a way to create test specifications for a Windows application in a very fast way using a number of tables. In these tables the test requirements were captured. The tables were designed to be used in connection with small test scripts and serve as test reports at the same time. This enabled the testers to define the test strategy and get an overview of the test coverage on the fly.

Testing a Windows system can be a huge task. You stand a fair chance if you have sufficient time, know the system in general, and know the details about what the system is supposed to do – but what if you don't? This paper describes the challenges we faced when asked, as an independent testing company, to do third party testing of a Windows system with over 100 forms in 7 weeks. We had no prior knowledge of the system, and were soon to discover that no requirement specification existed.

The paper describes the nature of testing Windows systems in general, the solution and experiences gained in this particular case, and some hints and ideas for organisations involved in third-party testing.

Acknowledgement: Many thanks to Mark Fewster for supporting the idea and giving expert test advise, even though all the documentation was in Danish at the time.

Introduction

The phone rings one fine morning in the office and a gentleman asks me, if I can test a new version of his system before it goes out to the customers. “Sure”, is the immediate answer, “Let’s just get a few things straight.” I ask him when the system is going out; what type of system is it, what it does, and how big is it. I also ask him what changes have been made and why, and what resources he has got. Finally I ask him what he expects from me.

The answers are discouraging, to say the least. The system is going out in about 2 ½ months. It is the third version of an administrative Windows system, with just over 100 forms.

It is not entirely clear what changes have been made – in fact not all of them have been made yet – and some are based on problem reports gathered from the support centre while some are enhancements of various origins.

There are a number of people involved in the development, but they are mostly tied up doing the last changes; there are also some people in the support centre, but they are already overworked taking calls from the users of version 2. Some of the users are accessible. There is some money set aside for this test activity.

He wants the system to be tested! It needs to go out in such a state, that the users’ confidence in the system will not disappear completely. Can we do the job? It is a high-risk project – But … yes we’ll do it!

Planning

It was very tempting to run in and start testing from day one. We did not do that. We set aside a couple of weeks to get an overview and do some planning. The planning was kept at a minimum, but non-the-less according to IEEE standard for test planning.

The description of this planning is out of scope of this paper. But it was worth the time to establish an overview of the assignment, and of the resources, including the time, the money, the people and their skills, and the test environment.

According to the V-model test takes place in 3 phases: Module Test, Integration Test, and System Test, corresponding to the specification phases. In this case the scope of the test was system test; i.e. the test against the users’ expectations or requirements. There were however no written requirements, and therefore we had to establish some test requirements.

What the User Sees

The users basically manipulate data in forms. The users navigate in forms and between forms, and create, read, update, and delete data in fields in the forms. A form is a view into the data storage, usually a database. Data of identical type are stored in records in a table, for example a table for customers that holds a record for each customer. In a form one record is presented at the time and the user may browse in the records in the underlying tabel, moving up or down through the records in the table. More records pertaining to the main record may be presented in sub-forms within a form.

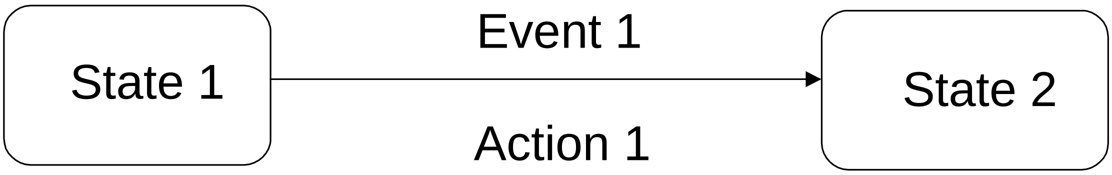
Data is subject to validation according to applicable rules, and data can have complicated links to data in the same form or in other forms.

It is not unusual for an administrative system to have hundreds of forms with data. This system had just over 100 data forms, and the forms where listed, categorised according to an estimated complexity, and prioritised in order of importance to the user.

The users' confidence in the system depends on the system's consistent respons to navigation and correct handling of data, i.e. correct validation and correct reaction to data.

What the tester sees

A Windows system may be regarded as a gigantic state machine. In its simplest form a state machine may look like this:



At any given time during the test we are in a starting state defined by:

- the present data
- the current field
- the data handling mode in terms of creation or update (reading is not interesting in this context and deletion is considered an event in its own right)

The events are the ways the state may be changed, i.e.

- either entering, updating, or deleting data in the field, or
- moving from one field to another or
- moving from one record of data in the underlying table to another

An action is the possible action following an event, for example:

- a validation of the data according to the type of the field
- a validation being cause by leaving a field
- some changes happening to the data dependent on the new content of the field

An event and its pertaining action(s) cause the system to transit to a new state.

If we consider the number of different data handled by even a small Windows systems and the number of fields we can see that the number of states becomes extremely large. Likewise the number of ways to navigate between fields and records, i.e. the state transitions becomes extremely large. And we have to test that for each possible start state all events cause the correct action(s) and lead to the correct end state. Not to mention invalid transactions, transitions pairs, triplets etc., etc…

We needed to be able to establish an overview of all the possible states in the system and all the possible state transitions, and also to establish a way to determine the test coverage we obtained during the test, since it is obviously impossible to obtain a complete coverage.

What we have to test

In this case the plan was to test each form in the prioritised order, tailoring the coverage according to the errors encountered. But when testing a system without any requirement specification we need to be able to establish the test requirements, and in this case we had to provide the test specification rather fast.

We decided to provide the test specifications as form specific tables for each form and generic test scripts for general behaviour. In the following it will be described how we designed these tables and generic test scripts. Following this it will be described how the test was actually performed on the basis of this material.

We started by defining the ways to navigate in the system, i.e. the ways to leave a field or a record. This determines the way a transition between states takes place and provides general guidelines as to what the next state may be.

The actual state at any given point is determined by the presented data, the current form, and the current field. For each form we defined tables to describe what data it could handled and how, and what fields and push buttons it contained.

For each field we described the expected action(s) for each possible state transition, for example the type check to be performed when data is entered or updated and the validation to expect when the field is left.

Push buttons present another way to cause a state transition and some specific action(s); this was also described for each push button.

As mentioned above one of the issues that affect the users' confidence in the system is consistent behaviour. We therefore developed generic test scripts to cater for all general behavior, i.e. the appearance of different types of forms, the validation of each type of field, and the general handling of data creation, updating, and deletion.

The collection of tables and generic test scripts is illustrated in the figure below. The figure contains extracts of the actual tables and generic test scripts. For the time being this is just to give an idea of the amount of work. Each type of table and some of the generic test scripts are described in details following the figure.

Navigation commands

Stationary commands

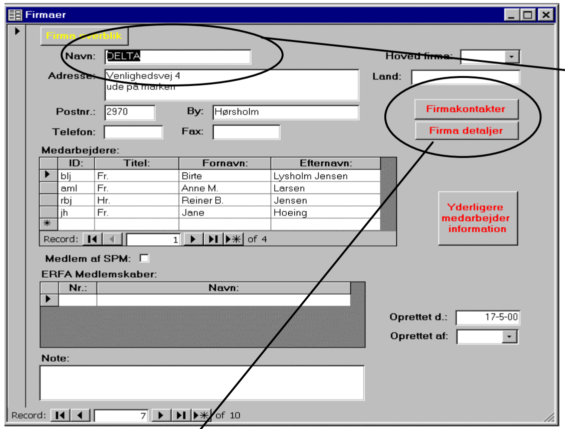
Command	Reaction
<Crtl>+<Del>	the record is deleted, the next record is shown and the same field is active if the deleted record is the last, the previous record is shown
<F1>	the help form pertaining to the active form becomes active

Field commands

	Reaction for position		
Command	first field	'in between' field	last field
<End>	last field is active		
<Enter>	next field in tab order is active		last field is still active
<Home>	first field is active		
<Shift>+<Tab>	first field is still active	previous field in tab order is active	
<Tab>	next field in tab order is active		last field is still active
Left click in a field	chosen field is active		

Record commands

	Reaction for position		
Command	first record	'in between' record	last record
<Ctrl>+ <End>	last field for last record is active		
<Page down>	next record is active, same field is active		no change – no validation triggers a 'Notification'
<Page up>	no change – no validation triggers a 'Notification'	previous record is active, same field is active	
Right record browser error	next record is active, first field is active		no change – no validation
Left click on [X]	the form closes		



Firmaer

Navn: DELTA

Adresse: Venlighedsvej 4
ude på marken

Postnr.: 2970 By: Hørsholm

Telefon: Fax:

Hoved firma:

Land:

Medarbejdere:

ID:	Titel:	Fornavn:	Efternavn:
blj	Fr.	Birte	Lysholm Jensen
aml	Fr.	Anne M.	Larsen
rbj	Hr.	Reiner B.	Jensen
jh	Fr.	Jane	Hoeing

Record: 1 of 4

Medlem af SPM: ☐

ERFA Medlemskaber:

Nr.:	Navn:

Oprettet d.: 17-5-00

Oprettet af:

Note:

Record: 7 of 10

Data table

Main record	1. sub-level	Default	Sorting	Allowed actions
Company		first company sorted by name	acsending by name	CRUD D: entries in MediFirma are deleted entries in FirmaiERFA are deleted
	MediFirma	all employees for company	acsending by last name	CRUD
	FirmaiERFA	all ERFAgroups assigned to company	ascending by number	CRUD

Field table

Fields		Tab order F;B	Default	Type	Validation; failure type	Dependent behaviour
		<Tab> <Shift><Tab> <Enter>	*	C U	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒	C <Tab> I ☒ N ☒ U <Tab> I ☒ N ☒
1	Navn	2;1	*blank*	A50		
2	Hovedfirma	3;1	*blank*	S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E C I N U I N	
3	Adresse	4;2	*blank*	A250		
4	Postnummer	5;3	*blank*	A10		If a valid Danish postnummer, and the By field is blank, the By will be filled in with the corresponding name. C I N U I N
Sub-form: Medarbejdere						
L1-1	ID	L1-2;8	*blank*	A4		If the creation starts in another field in the list, the ID is filled in with the next running number prefixed by and M.

General behaviour

Test Case G1.1: Generic test case for test of a data form.

Description: This test case tests the generel behaviour, pertaining to all data forms.

The test case has the following parameters:

F: the field table for the form under test

D: the data table for the form under test

P: the push button table for the form under test

Step	Input	Expected output	Result
a.	Examine the left side of the form.	Record selector is present.	

Test Case G1.4: .. error -form.

Test Case G3.1: .. creation of main record.

Test Case G2.7: .. field type.

Push button table

Push buttons		Validation; failure type			Dependent behaviour		
		Left mouse click	Short cut	<Enter>	Left mouse click	Short cut	<Enter>
5	[Firmakontakter]				The form Firmakontakter is current. The only data available is for the current company.		
6	[Yderligere medarbejder information]	An employee must be current in the sub-form Medarbejdere.			The form Medarbejdere is current. The only data available is for the current employee.		

Navigation

When testing a system we need to know how the navigation works; i.e. what state transition to expect when we as testers move around in the forms and between forms. In this particular system (as in many others I have seen in my carrier), navigation was not consistent, when we first looked at the system.

We organised a workshop, where developers, support people, and users sat down in the same room and decided on and specified the system navigation. The scope of the test was individual forms, so the navigation between forms was not considered, and therefore the menu structure was not examined either.

Navigation causes actions to occur, i.e. validation of created or updated data and execution of dependent behaviour. Within forms it is possible to navigate by commands (here keyboard-clicks and mouse-clicks), and form-specific push buttons. For the general navigation we split the commands into 3 groups according to what general state transition they would cause and type of validation they would trigger.

Command descriptions

Commands fall in 3 categories:

Command type	Focus change	Validation triggered
Stationary commands	the focus does not change	none
Field commands	the focus change from one field to another	the field, that is being left
Record commands	the focus change from one record to another	all fields in the record being left

In the following a few examples of the descriptions of the commands are shown. Each command will appear in only one place. The tables provide the specification of navigation rules, i.e. the reaction from the system to expect when a command is issued, and what type of validation to expect in connection with this.

Stationary commands

Command	Reaction
<Crtl>+<Del>	the record is deleted, the next record is shown and the same field is active if the deleted record is the last, the previous record is shown
<F1>	the help form pertaining to the active form becomes active

Field commands

Command	Reaction for position		
	first field	'in between' field	last field
<End>	last field is active		
<Enter>	next field in tab order is active		last field is still active
<Home>	first field is active		
<Shift>+<Tab>	first field is still active	previous field in tab order is active	
<Tab>	next field in tab order is active		last field is still active
Left click in a field	chosen field is active		

Record commands

Command	Reaction for position		
	first record	'in between' record	last record
<Ctrl>+ <End>	last field for last record is active		
<Page down>	next record is active, same field is active		no change – no validation triggers a 'Notification'
<Page up>	no change – no validation triggers a 'Notification'	previous record is active, same field is active	
Right record browser error	next record is active, first field is active		no change – no validation
Left click on [X]	the form closes		

It may look trivial, but when you get everybody involved – developers, support people, and users – to sit down and go through all possible ways of navigation, it is not simple at all. But it is very, very useful to get a description of what the testers should expect from the system when they are testing it.

Some commands react differently when issued inside a sub-form. Tables like the above therefore had to be provided for these commands, specifying the expected reaction when issued inside sub-forms.

General behaviour

We have considered part of the general behaviour in the specification of the commands. But there is more to it; we also have to test

- the general lay out and behaviour of each type of form
- the handling of the different types of fields
- the handling of different data actions
- the handling of failed validations

For this we produced generic test scripts to be used each time the tester had to test one of these aspects. The test scripts were produced directly on the basis of behaviours agreed to at the workshop mentioned above.

The generic test scripts, as indeed the test tables, may be written in any way that is suitable for the project. They can be created using a test automation tool, if the test is to be automated, but that is not necessary. A word processor, a database, or a spreadsheet may be used. In this case we used Word.

Forms

A windows system consists of a number of different types of forms, such as data forms, sub-forms, and various kinds of message boxes. We therefore produced a generic test script for each type of form. An example of part of the generic test script for a data form is shown below.

Test Case G1.1: Generic test case for test of a data form.			
Description: This test case tests the general behaviour, pertaining to all data forms. The test case has the following parameters: F: the field table for the form under test D: the data table for the form under test P: the push button table for the form under test			
Step	**Input**	**Expected output**	**Result**
a.	Examine the left side of the form.	Record selector is present.	
b.	Examine the top of the form.	Form name is presented in white to the right of the icon in the top left corner.	
c.	Examine the fields – also in terms of sub-forms and fields in sub-forms.	The specified fields and sub-forms are present.	See marks in **F**.
d.	Examine the tab order, using different means of navigation.	The tab order is as specified in **F**.	See marks in **F**.
e.	Click the left mouse button in various fields.	Fields become active when clicked on. Fields without tab indicator cannot be activated.	
f.	Examine the contents of the fields.	Data when first opened is as specified in **D**.	
g.	Browse through the records.	Sorting is as specified in **D**.	
h.	Examine the push buttons.	The push buttons specified in **P** are present.	See marks in **P**.

This test script refers to form specific information. This is specified in tables described below.

Field types

A number of different types of data items to be handled in fields were defined and their characteristics specified. This is also sometime refered to as controls or control types. Some of these types are:

- alpha-numeric string (A)
- integer (I)
- real (R)
- date (D)
- selection list (S)

The expected behaviour for each type was specified in generic test scripts. An example of an extract of one of these generic field type test scripts is:

<table>
<tr><td colspan="4"><b>Test Case G2.7:</b> Generic test case for test of a decimal field type.</td></tr>
<tr><td colspan="4"><b>Description:</b> This test case tests the general behaviour, pertaining to all decimal fields.
The test case has the following parameters:
H: Number of ciphers before the decimal point
D: Number of ciphers after the decimal point</td></tr>
<tr><th>Step</th><th>Input</th><th>Expected output</th><th>Result</th></tr>
<tr><td><b>a.</b></td><td>Try to enter a character, which is not a cipher, neither a ‘+’ (plus) nor a ‘-‘ (minus).</td><td>The system ‘bibs’, when an illegal character is entered. The character is not accepted.</td><td></td></tr>
<tr><td><b>b.</b></td><td>Try to enter a number with H ciphers (A), a decimal point, and then more than D ciphers (B+C).</td><td>The system ‘bibs’, when more than D ciphers are entered.</td><td></td></tr>
<tr><td><b>c.</b></td><td>Leave the field.</td><td>Data is accepted as A.B
A mark ‘,’ (comma) appears as a 1000-divider, that is 10 thousand will appear as 10,000.00 etc.</td><td></td></tr>
<tr><td><b>d.</b></td><td>Activate the field again. Try to enter more than H ciphers (A+C).</td><td>The system ‘bibs’, when more than H ciphers are entered.</td><td></td></tr>
<tr><td><b>e.</b></td><td>Leave the field.</td><td>Data is accepted as A</td><td></td></tr>
<tr><td><b>f.</b></td><td>Activate the field again. Enter a negative sign followed by H ciphers (A), a decimal point, and D ciphers (B). Leave the field.</td><td>Data is accepted as –A.B</td><td></td></tr>
<tr><td><b>g.</b></td><td>Activate the field again. Enter a positive sign followed by H ciphers (A), a decimal point, and D ciphers (B). Leave the field.</td><td>Data is accepted as +A.B</td><td></td></tr>
<tr><td><b>h.</b></td><td>Activate the field again. Enter H ciphers (A), a decimal point, and D ciphers (B). Leave the field.</td><td>Data is accepted as A.B</td><td></td></tr>
<tr><td><b>i.</b></td><td>Activate the field again. Enter less than H ciphers (A). Leave the field.</td><td>Data is accepted as A</td><td></td></tr>
</table>

The test script specifies that the tester is to leave the field. The field type behaviour should be tested with all the various way of leaving a field as specified for the commands.

Data actions

The actions you can perform on data, i.e. on main records and sub records are:

- create
- read
- update
- delete

The expected behaviour for each action was specified in generic test scripts both for data forms and for sub-forms. These scripts are not shown here.

Handling of special situations

When a validation fails the system will have to react in some way. Three types of possible reactions were identified and specified. Generic test scripts were also produced for the different type of failure reactions. These scripts are not shown here, but the following table provides an overview of the types and the expected reactions.

Failure type	Type	Reaction
Error	E	a message box appears with a 'reasonable' error message the system is locked when the [OK] button is pushed the system returns to the state it was in before the validation was triggered
Notification	N	a 'resonable' message appears at the bottom left corner of the form the message disappears when a command is issued or a key pushed
Warning	W	a message box appears with a 'reasonable' error message the system is locked when the [OK] button is pushed the system goes on according to the command that triggered the validation

Data form specific information

For each data form we need to have test specifications for:

- data information
- field information
- push button information

We defined a set of tables for capturing this information. The tables were filled out for each of the forms to test by the developers, having support people and users review and contribute to the information. The professional testers were also involved in the reviews, making sure that the descriptions were worded in a way that facilitated the actual testing task. This activity captured a considerable number of errors and uncertainties before the actual testing had even begun.

Data information

Each data form handles a main record, and possibly a number of sub-records in sub-forms. We defined a table for this information, of which an example is shown below.

Based on this table we can test that the right data is handled and for each type of data we can test that:

- the **right record** is presented, when the form is opened
- the **sorting** is correct
- the **data actions** allowed for the data are available and work correctly

Form: Firmaer

Main record	1. sub-level	Default	Sorting	Allowed actions
Company		first company sorted by name	ascending by name	CRUD D: entries in MediFirma are deleted entries in FirmaiERFA are deleted
	MediFirma	all employees for company	ascending by last name	CRUD
	FirmaiERFA	all ERFAgroups assigned to company	ascending by number	CRUD

Fields

For the fields in each form we defined a table, of which an extract based on the example form is shown below. Based on this table we can test that the fields are present with the right labels and for each field we can test that:

- the **tab order** works using the various means of navigation
- the **default value** is correct for the various ways to create a new record
- the **type** is handled correctly both during **C**reation and **U**pdate
- any **validation** is performed both during **C**reation and **U**pdate
- the validation is triggered when the field or the entire record is left; and in the latter case both when the field is the current field (**I**n focus) and when it is not (**N**)
- any **behaviour dependent** on the data entered in the field occurs as for validation

It was the professional testers' task to make sure the specification of the validation was worded in a way that made it clear how to test. This included specification of border values and equivalence partitioning. The aim is to test that the validation fails when it is supposed to, and does not when it is not supposed to.

Dependent behaviour occurs when correct data is entered into a field and this data has effects in other places in the system. It might be that the data appears in another field on another form, that the data causes fields to appear or disappear, or that values in other fields are calculated on the basis of the data. We do not consider data being handled as a main record or a sub-record in another form as dependent behaviour, as this is treated in the relevant form specification.

The specification of dependent behaviour was very large and complicated in some cases. This seemed to frighten the developers, but it was a very usefull exercise, and the specification was necessary for all involved.

Fields		Tab order F;B				Default			Type	Validation; failure type						Dependent behaviour					
		<Tab>	<Shift><Tab>	<Enter>		*			C	C	<Tab>		I	☒		C	<Tab>		I	☒	
													N	☒					N	☒	
									U	U	<Tab>		I	☒		U	<Tab>		I	☒	
													N	☒					N	☒	
1	Navn	2;1				*blank*			A50												
2	Hovedfirma	3;1				*blank*			S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E											
										C			I								
													N								
										U			I								
													N								
3	Adresse	4;2				*blank*			A250												
4	Postnummer	5;3				*blank*			A10							If a valid Danish postnummer, and the By field is blank, the By will be filled in with the corresponding name.					
																C			I		
																			N		
																U			I		
																			N		
Sub-form: Medarbejdere																					
L1-1	ID	L1-2;8				*blank*			A4							If the creation starts in another field in the list, the ID is filled in with the next running number prefixed by an M.					
																C			I		
																			N		
																U			I		
																			N		
End of sub-form																					

The table is designed in such a way that it not only provides the information about what to test, but also a way to register directly in the table what was tested by marking the relevant boxes. Thus it was possible get an overview of the coverage by examining what was marked and what was not; and an idea of how the test went by examining which boxes were marked OK and which were marked with a reference to an incident report.

Push button information

A form usually also have push buttons for various events and their pertaining actions. Some events may be handled by menu options, but this was left out of the present test. For each form we defined a table for the push buttons, of which an extract based on the example form is shown below. Based on this table we can test that the push buttons are present with the right labels and for each push button we can test that:

- any **validation** is performed, and that the validation is triggered when the push button is activated
- any **behaviour dependent** occurs when the push button is activated

For push buttons the dependent behaviour should be as indicated in the text on the push button.

Push buttons		**Validation; failure type**			**Dependent behaviour**		
		Left mouse click	Short cut	<Enter>	Left mouse click	Short cut	<Enter>
5	[Firmakontakter]				The form Firmakontakter is current. The only data available is for the current company.		
6	[Yderligere medarbejder information]	An employee must be current in the sub-form Medarbejdere.			The form Medarbejdere is current. The only data available is for the current employee.		

In this table we can also directly register what we have tested and how it went as described above for the fields.

Push buttons also behave as record commands.

State transitions: Combining events and actions

In principle we have to test that all state transitions work correctly. This means that for all combinations of data we have to test for each field that the correct action(s) occur(s) for each possible event. For example that the validation of each field is performed correctly for all the ways to trigger a validation, i.e. for all field commands with the field in focus and for all record commands both with the field in focus and with all the other fields in focus; both during creation and update.

If we calculate the number of test steps this leads to, the result is astronomical.

It is therefore indicated in the field table, which commands must be tried and some boxes are left open, so that the tester can decide which other commands to try. More mandatory commands to try may off cause be specified – the table can be expanded in size as appropriate to cover for more state transitions.

Performing the test

Traditionally the test specification for a system test is build up as a hierarchy of test groups with test cases consisting of test steps. If a complete test specification for the system test is to be build it will be huge and very difficult to maintain. If all test cases are to be planned and defined in full, the test specification will have a tendency to be either to shallow (because the task is simply too big) or too deep (because the tester wants to cater for everything). In a traditional test specification it will be difficult to change the scope of a test case at a later stage.

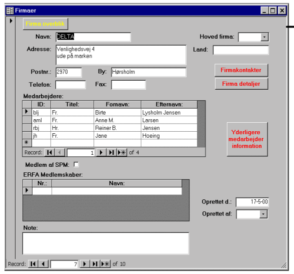
The test is in fact not likely to be just one test run from one end to the other. Depending on the development, parts of the test will be run at various times, and many of the parts of the test will be run several times, possibly with different scopes. The scope of certain areas of the test is also like to change depending on the experiences gained during the test itself. If all this is to be planned before the test specification can be produced, then the production of the test specification is likely to be a bottleneck. We could not afford this.

The test tables and the generic test cases described above were used as the building blocks for each actual test. When a tester was to test a form the full test specification for that particular test case was assembled using the necessary building blocks, depending on what is to be tested and the scope of the actual test.

It was our aim that an experienced tester should be able to perform the test of the forms based on the set of tables and the generic test scripts. The customer did however not have resources to get experienced testers to perform the test. We had a couple of secretaries, familiar with data base systems and Windows in general but not with the system to test; and we had two experienced testers.

We provided a short test script for each of the forms, based on the general data information, to provide the guide for testing using the test tables and the generic test scripts.

The amount of work involved in performing a full test for a form may be deceptive when you first look at the test script for the form and the test tables. The following figure illustrates a small part of the test. The form specific test script refers to generic test scripts, which refer to the test tables. These refer indirectly to other generic test scripts. It is important when planning the actual test performance that this is understood.



Navn: DELTA

Hoved firma:

Adresse: Venlighedsvej 4
ude på marken

Land:

Postnr.: 2970 By: Hørsholm

Telefon: Fax:

Medarbejdere:

ID:	Titel:	Fornavn:	Efternavn:
blj	Fr.	Birte	Lysholm Jensen
aml	Fr.	Anne M.	Larsen
rbj	Hr.	Reiner B.	Jensen
jh	Fr.	Jane	Hoeing

Record: 1 of 4

Medlem af SPM: □

ERFA Medlemskaber:

Nr.:	Navn:

Oprettet d.: 17-5-00

Oprettet af:

Note:

Record: 7 of 10

Testgroup: 5.1 Firmaer

Testcase: 5.1.1 Handling companies

Description: This test case tests the form Firmaer, i.e. its appearance, data presentation and navigation. The defaults are tested for creation. The field types, validations, and dependent behavior are tested for creation and edit. The handling of deletion of records is tested as well.

Preconditions: The system must be started with the test database in the initial state.

Tracing: Firmaer data table (**D**), Firmaer field table (**F**), Firmaer push button table (**P**)

Performed by: **Date:** **Accepted:** Yes / No

Step	Input	Expected output	Result
1.	Select Firmaer on the main menu.	The form Firmaer is the active form.	
2.	Perform G1.1, the generic testcase for data forms for the form Firmaer.	As stated in testcase G1.1. Don't forget to log data!	
3.	Perform G1.2, the generic testcase for sub-forms for the sub-form Medarbejdere.	As stated in testcase G1.2.	
4.	Perform G3.1, the generic testcase for creation of main records for main record Company .	As stated in testcase G3.1.	
5.	Perform G3.3, the generic testcase for edit of main records for main record Company .	As stated in testcase G3.3.	
6.	Perform G3.5, the generic testcase for deletion of main records for main record	As stated in testcase G3.5.	

1

Test Case G1.1: Generic test case for test of a data form.

Description: This test case tests the generel behaviour, pertaining to all data forms.

The test case has the following parameters:

F: the field table for the form under test

D: the data table for the form under test

P: the push button table for the form under test

Step	Input	Expected output	Result
a.	Examine the left side of the form.	Record selector is present.	
b.	Examine the top of the form.	Form name is presented in white to the right of the icon in the top left corner.	
c.	Examine the fields – also in terms of sub-forms and fields in sub-forms.	The specified fields and sub-forms are present.	See marks in **F**.
d.	Examine the tab order, both forwards and backwards using different means of navigation.	The tab order is as specified in **F**.	See marks in **F**.
e.	Click the left mouse button in various fields.	Fields become active when clicked on. Fields without tab indicator cannot be activated.	

1.1

1.2

Fields		**Tab order F;B**	**Default**	**Type**	**Validation; failure type**	**Dependent behaviour**
		<Tab> <Shift><Tab> <Enter>	*	**C** **U**	**C** <Tab> **I** [X] **N** [X] **U** <Tab> **I** [X] **N** [X]	**C** <Tab> **I** [X] **N** [X] **U** <Tab> **I** [X] **N** [X]
1	Navn	2;1	*blank*	A50		
2	Hovedfirma	3;1	*blank*	S – all other companies, sorted by name. 2cm of the name and 2 cm of the address is shown in the list	Must be in the list, if filled in ; E **C** **I** **N** **U** **I** **N**	
3	Adresse	4;2	*blank*	A250		
4	Postnummer	5;3	*blank*	A10		If a valid Danish postnummer, and the By field is blank, the By will be filled in with the corresponding name. **C** **I** **N** **U** **I** **N**
Sub-form: Medarbeidere						

Test Case G1.2: .. sub-form.

Test Case G1.4: .. error -form.

3

Test Case G3.1: .. creation of main record.

3.2.x

Test Case G2.7: .. field type.

An extract of the form specific test script for the form used in this article is shown below. This mainly consists of steps referring to generic test scripts, but for other forms it was necessary to include some more specific steps, when the behaviour for some – well understood and agreed – reason deviated from the standard.

Testgroup: 5.1 Firmaer			
Testcase: 5.1.1 Handling companies			
Description: This test case tests the form Firmaer, i.e. its appearance, data presentation and navigation. The defaults are tested for creation. The field types, validations, and dependent behaviour are tested for creation and edit. The handling of deletion of records is tested as well.			
Preconditions: The system must be started with the test database in the initial state.			
Tracing: Firmaer data table (**D**), Firmaer field table (**F**), Firmaer push button table (**P**)			
Performed by:	**Date:**	**Accepted:** Yes / No	
Step	**Input**	**Expected output**	**Result**
1.	Select Firmaer on the main menu.	The form Firmaer is the active form.	
2.	Perform G1.1, the generic test case for data forms for the form Firmaer.	As stated in test case G1.1. Don't forget to log data!	
3.	Perform G1.2, the generic test case for sub-forms for the sub-form Medarbejdere.	As stated in test case G1.2.	
4.	Perform G3.1, the generic test case for creation of main records for main record Company .	As stated in test case G3.1.	
5.	Perform G3.3, the generic test case for edit of main records for main record Company .	As stated in test case G3.3.	
6.	Perform G3.5, the generic test case for deletion of main records for main record Company .	As stated in test case G3.5.	
7.	Perform G3.2, the generic test case for creation of sub records for sub record Medarbejdere.	As stated in test case G3.2.	

If we wanted a thorough test of a form, the test specification was assembled using all the generic test scripts for all the types of the fields on the form, generic test scripts for all the failure situations to provoke, and specifications of all the ways to leave each field and test the validation and the dependent behaviour.

If the specification or the scope changed, it is easy to re-assemble the test case and run it again.

Test data

No actual data is given in the test specifications as described above. The testers had to make up the test data as they went along. We made sure, that the data in the system – the initial state of the test database – was relatively simple and yet allowed the tester to examine links and dependencies.

It was part of the testing task to log the data used and the data produced, when a test was first performed. This enabled us to gradually extent the specifications with actual test data specifications, making reruns of the tests easier.

The testers need to know a bit about the system to find out where dependencies may be examined. This knowledge may be built up gradually if the test starts with the simple forms and move on to the more complicated. We were however not able to do the test this way – in fact we started with the most complicated form of them all. But it turned out that the testers became sufficiently familiar with the system quite rapidly, and were able to move around in the system to check dependencies as appropriate.

We made it - just

We spend two weeks finding our feet and planning the test. That left us with 7 weeks for the actual specification of the test and performance of the test.

The professional testers produced the generic test scripts, and the developers and support people produced the test tables in the order the test was to be performed. The tester literally took the hot test tables off the printer and performed the tests.

We got through almost 80% of the forms we originally planned to test, leaving out the simplest and most rarely used forms. We found over 200 errors – and keep in mind that this was the third version of the system.

Most of the errors we found were simple to correct for the developers but of major impact on the credibility and user friendliness of the system. Many of the errors found in the beginning were errors that were present in several parts of the system, and they were corrected in these other parts before the testers reached that far, making the system better and the test faster as we moved ahead.

Many of the forms had to be tested several times – changing the scope each time, i.e. only going into problems areas on subsequent runs.

Benefits

There is a lot to gain from handling the specification and test of large administrative system as described. The benefits fall in 3 categories described below.

Using tables as test specifications

Tables provide a lot of information. The tables provide a way to specify the enormous amounts of states and state transitions in a Windows system in a very compact way. Describing each state and each transition seperately would be a very time and paper consuming exercise.

The tables may be a little hard to understand when the testers are first presented with them, but once the testers get the hang of it, the tables reduce the need for very verbose test scripts. The tables are much easier to maintain than long and complicated test scripts.

The tables provide an easy overview both of what to test and of what has been tested.

Using the test scripts as specifications

Ideally the test specifications should have been produced on the basis of user requirement specifications, but there weren’t any in this case. So the developers created the test specifications for the sake of the testers. They did not create requirement specifications – this was made clear from the beginning to motivate them to undertake the work.

When the testing was done, they did however have a set of requirement specifications. The test tables and the generic test scripts were kept up-to-date as the test progressed and errors were found in these or decisions were made that affected the contents of them.

The test tables for each form, i.e. the form specific data table, field table, and push button table, could serve directly as requirement specifications for the form. The support people used them to facilitate the support job. The tables offer an easy way to determine if a reported incident is in fact an error in the system or if it is caused by a misunderstanding or mis-expectation made by the user.

When changes to the system are now considered the developers and support people document the changes directly in the tables. This encourages them to think through all the impacts a change may have on the system from a user’s point of view. The developers use the new tables as the basis for the actual changes to be implemented in the system. By documenting the changes like this the test specification is updated simultanously and ready for testers to use when they perform the test of the changes and the related regression test. The support people use the new tables in their work when the changes are accepted.

Using a word processor to maintain the tables enables tracking of the changes between the existing tables and the new ones.

The same applies for the generic test scripts. It is more difficult for developers and support people to work directly in the generic test scripts, but these test scripts may be reworded into corresponding specification tables for use by the developers and testers. This requires the testers to update the generic test scripts on the basis of the changed tables, but this is a fairly simple task.

Using generic test scripts

The extended use of generic test scripts has a positive impact on the design. The design will be more structured as the designers and the testers thrive to extract and define the generalities in the behaviour of the system, and the user interface is more likely to be uniform through out the system.

The specific test scripts will be fairly small and they are more likely to be uniform even if several people create them. This will ease the execution of the test, and reduce possible misunderstandings and different interpretations.

The test specification is a lot easier to write and to maintain than classic test specifications. Changes are only to be implemented in one place.

Even if the generic test scripts are prepared manually they will provide a solid ground for an easier automisation of the test, if that is wanted or needed at a later stage. We did not have that opportunity in this case – all the testing was performed manually.

Third party testing

Out-sourcing of especially the system testing activity in software development projects is becoming more and more common. Some (usually larger) companies create their own independent test organisations within the company, and some companies offer to execute test of software products produced by other companies. When an organisation, be it an independent company or an independent organisational unit, is used as a third party testing organisation, this organisation faces a number of challenges.

We learned a lot from the case presented in this paper and from other similar assignments. First of all that there often is a difference between what the customer asks for and

- what he really wants
- what he needs, i.e. what you want to give him
- what you are able to give him

In this particular context customers ask for a test to be performed on a system that is under development or near delivery. What many customers want is a sort of rubber stamp approval of their system as it is. This is however rarely what a test will provide. The customer needs to realise that a professional test will bring out some errors and he will have to consider how to handle that.

Testers are often presented with a system to test without the right conditions for performing a professional test. In order for a customer to get a quality software system, he needs documentation of what needs to be tested, resources (time, people, and money), and an overview of the quality of the test and the system.

The requirements may be in any state from non-existant to brilliantly documented, with a pronounced bias towards the first extreme. The testers and therefore the customer need documentation of the system, in terms of at least some specifications of what the system is expected to do and how.

Test requires resources in terms of time and people to perform the test. Time is often a limited resource, and so may the availability of internal resources or money to pay external resources be. The reason for this is often a lack of project management, envolving identification of all development tasks, estimation of the tasks, follow up on actual time used for the tasks, and re-estimation.

When testers plan and perform a test they need to establish an overview of what has to be tested, how the test is progressing, what errors have been found, and what the state of error correction is. These are all activities in the configuration management discipline, and this is often neglected.

It is tempting for testers to try to deliver what the customer really needs. There are however some limitations and threats in this approach. One of the obvious limitations is in terms of time and money. If the tester insists on solving what she sees as the most urgent need first – for example project management - it may not be possible within the constraints of the assignment to solve the most urgent need from the customer’s point of view – the test.

The art is to strike the right balance between what is needed and what is feasible. One of the things to keep in mind is that the tester should keep up the standards, but keep it light. In this case for example we were tempted to skip the planning, but we didn’t for the sake of our own working conditions and for the value of the signals concerning planning that this send to the customer.

A threat in taking on an assignment where some of the pre-conditions are missing is the temptation to provide these yourself. This may lead to co-dependent behaviour, where the tester takes over a responsibility that is really the customer’s, and thereby in the long run preventing the customer from learning how to do things himself. The solution to this is to get the customer as much involved as possible.

This may be done in the form of workshops where the customer gets involved in providing the information necessary for the testing.

Another thing we have learned is that when you get involved in a testing assignment you should take nothing for granted. It is very important to make it clear to youself and to the customer what you need and persue any possible way of getting hard evidence for the existance of material and resources. For example is the customer says that the system is finished – assume it is not, and try to get it delivered to you as it is.

But last and not least – testing is fun, and working with a group of people to deliver a system that is better than it would have been without you is a great source of personal and professional fulfilment!

Wednesday December 6 2000

W3

Creating Requirements from Test Specifications

Anne Mette Jonassen Hass

__Mrs. Anne Mette Jonassen Hass__, M.Sc.C.E., has 20 years experience in IT. She has been involved in all aspects of software development: analysis, design, coding, test, quality assurance, and management. Mrs. Hass has worked in various types of business such as hospitals, the oil industry, telecommunication, hardware producers, and the space industry, in Denmark, Norway, England, and France.

For the last 5 years she has worked as a consultant in Software Process Improvement, assessments, and test. Mrs. Hass has been involved in several test process improvement projects in different companies. She also performes third-party test and validation of software, especially safety critical software. Mrs. Hass is a certified BOOTSTRAP V.3.0 lead assessor, and has performed more than 30 BOOTSTRAP assessments in Denmark for companies of all sizes and in many different branches.

eurostar 2000