# Contents

1 Introduction .................................................. 2
  1.1 Background ............................................. 3
    1.1.1 Business process and business analysts .......... 3
    Business process modeling tools ...................... 3
    1.1.2 Business process and exceptions ............... 4
  1.2 Problem statement ..................................... 4
    1.2.1 Business analyst and exceptions ............... 4
    Good handling is worth its weight in gold ........ 5
    1.2.2 Business analyst and work practices .......... 5
  1.3 Solution ............................................... 6
    1.3.1 Static verification ............................... 6
    1.3.2 Construct ......................................... 6
    1.3.3 Templates ....................................... 7
    1.3.4 Experiment ...................................... 7
    Results and conclusions .............................. 7
  1.4 Last minute notes ................................... 8
  1.5 DELETE ME ............................................. 8
## 1.6 Contribution

1.6.1 On errors ........................................ 8

1.6.2 On features ..................................... 9

## 2 Background

2.1 BPM ................................................. 13

2.2 BPM and SOA ..................................... 13

2.2.1 Business Activity vs. Business Service ............ 13

2.2.2 SOA and Web Services ........................... 14

2.2.3 Semantic Web Services ........................... 15

2.2.4 Prosero ......................................... 16

2.3 Exception Handling ................................ 17

2.3.1 Exceptions ...................................... 17

2.3.2 Exception Handling in Programming Languages .... 17

2.4 Motivation ........................................... 18

2.5 DELETE ME ......................................... 20

## 3 Starting Points

3.1 BPMN .................................................. 21

3.1.1 BPMN; What is it good for? ...................... 21

3.1.2 BPMN for mummies .............................. 23

Basic BPMN ............................................. 23

Core BPMN ............................................. 24

Advanced BPMN ....................................... 26
3.2 Exception Handling Specifications ........................................ 27
  3.2.1 Problems at execution .............................................. 27
  3.2.2 Exception .......................................................... 28
  3.2.3 Exception Handling in BPMN ..................................... 31
  3.2.4 Exception Handling in BPEL .................................... 33
  3.2.5 Business process practices for exception handling ............ 34
    Business methodology .................................................. 34
    Exception handling vs. return values ............................. 36
    Exception Handling Strategies ..................................... 38
    Automatic exception handling ..................................... 40
  3.2.6 Dealing with exceptions in IDEs ................................ 41
    The Java example ..................................................... 41
    Static verification ................................................... 42

3.3 BPMN Tools ............................................................ 44
  3.3.1 Prosero ............................................................ 44
    Architecture .......................................................... 45
    Modeling Support ..................................................... 47
    BPMN to BPEL Translation .......................................... 48

3.4 Summary ............................................................... 48
  ‘Mind the gap’ — BPMN and Prosero ................................ 48
  Exceptions vs. return value ......................................... 49
  Prosero and Activity uniqueness ................................... 49
  IDE, Prosero and static verification ............................... 50
BPMN and exception handling ........................................ 50
Prosero and Sub Processes ......................................... 50
Modeling routine ....................................................... 50

4 Our solution .................................................................. 54

  4.1 Solution overview ................................................... 55
    4.1.1 Classifications and Error Events ......................... 55
    4.1.2 Classifications ——> Error Events ....................... 57
    4.1.3 Additional Error Events by business requirements .... 57
    4.1.4 Higher categorization ....................................... 58
  4.2 Constructs ............................................................. 58
    4.2.1 Internal-Activity ............................................. 59
    4.2.2 Send-Receive ............................................... 60
    4.2.3 Receive-Process-Reply ..................................... 62
  4.3 Templates ............................................................. 65
    4.3.1 Send-Receive ............................................... 66
    4.3.2 Query ........................................................... 69
    4.3.3 Verify-Approve .............................................. 70
    4.3.4 Notification .................................................. 72
    4.3.5 Update-Approve ............................................. 73
  4.4 Afterthoughts ......................................................... 74

5 Implementation ................................................................ 76

  5.1 Extending BPMN ..................................................... 76
7.3 Results ................................................. 97
7.4 Discussion ........................................... 100
  7.4.1 The Good ........................................ 100
  7.4.2 The Bad ......................................... 101
  7.4.3 and the Ugly .................................... 101

8 Conclusions and Future Work 102
  8.1 Our solution ....................................... 103
    8.1.1 Robust BPMN ................................ 104
    8.1.2 Semantic Web Services ..................... 104
    8.1.3 An inconvenient truth ...................... 104
    8.1.4 Can’t see the forest for the trees ........ 105
    8.1.5 Did we matter? ............................... 105
    8.1.6 Between a rock and a hard place .......... 105
  8.2 Future implementation ........................... 106
    8.2.1 Simple enhancements ....................... 106
      Smart exception handling — heuristics ....... 106
      Add business exception icon for Sub Process . 106
      Business process into Template ................ 106
    8.2.2 Difficult enhancements .................... 107
      Complexity according to semantics ............ 107
      Data constraints attribute .................... 107
    8.2.3 Improve Parameters — use infrastructure . 107
    8.2.4 Improve diagram view — perspectives .... 108
8.3  Future research ......................................................... 108
   8.3.1 Improve BPMN — add top level business process .......... 109
   8.3.2 Improve verification — more Events .......................... 110
   8.3.3 Improve Error Events — create hierarchy .................... 110
   8.3.4 Suggesting appropriate handlers .............................. 111
8.4  Immediate papers .................................................. 111
   8.4.1 Viewing business process in the light of data ............... 111
       Business activity classification ............................... 111
       Exception classification ..................................... 112
   8.4.2 Choosing the right static verification for your modeling tool 112
   8.4.3 Exception handling vs. return value in business processes 112
8.5  So long, farewell, auf Wiedersehen, goodbye .................. 113

A  Dictionary .......................................................... 114

B  Preliminary Research ............................................... 117
   B.1 Possible features ............................................... 117
   B.2 Possible static verification methods ........................... 119

C  Activity classification ............................................. 120

D  Experiment complete results ...................................... 126
   D.1 Update return (of goods) ..................................... 128
   D.2 Return goods ................................................... 129
   D.3 Receive returned goods ....................................... 130
List of Figures

1.1 Thesis review business process ......................................... 3

2.1 Deliver book business workflow ......................................... 14
2.2 Unhandled exception notification ....................................... 18

3.1 Handle order ................................................................. 22
3.2 Handle Order with choreography ........................................ 26
3.3 Handle Order with Artifact ............................................... 27
3.4 Request catalogue BP, with Gateway .................................... 35
3.5 Request catalogue BP, with Error Event ................................ 35
3.6 Find customer BP, with Gateway ........................................ 35
3.7 Find customer BP, with Error Event .................................... 36
3.8 Adding a Gateway to a Credit check .................................... 37
3.9 Adding an Error Event to a Credit check ............................. 37
3.10 Return update BP, with Gateway ....................................... 38
3.11 Return update BP, with Error Event .................................. 38
3.12 Handling strategies examples: (a), (b), and (c) ..................... 39
3.13 Handling strategies examples: (d), (e), and (f) ..................... 40
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14</td>
<td>Unhandled exception notification, in Java</td>
<td>42</td>
</tr>
<tr>
<td>3.15</td>
<td>Catching and handling an exception, in Java</td>
<td>42</td>
</tr>
<tr>
<td>3.16</td>
<td>Passing an exception, in Java</td>
<td>42</td>
</tr>
<tr>
<td>3.17</td>
<td>Bad planning: bridge mistake</td>
<td>43</td>
</tr>
<tr>
<td>3.18</td>
<td>Bad planning: unsafe bridge</td>
<td>43</td>
</tr>
<tr>
<td>3.19</td>
<td>Process: Initial product release — happy path</td>
<td>51</td>
</tr>
<tr>
<td>3.20</td>
<td>Process: Initial product release — alternate path</td>
<td>52</td>
</tr>
<tr>
<td>3.21</td>
<td>Process: Initial product release — exception path</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>Customer Initiated Sourcing</td>
<td>61</td>
</tr>
<tr>
<td>4.2</td>
<td>Send-Receive Template with Construct form #1</td>
<td>66</td>
</tr>
<tr>
<td>4.3</td>
<td>Send-Receive Template with Construct form #2</td>
<td>67</td>
</tr>
<tr>
<td>4.4</td>
<td>Send-Receive Template with Construct form #3</td>
<td>67</td>
</tr>
<tr>
<td>4.5</td>
<td>Send-Receive Template actualized</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>Query Template</td>
<td>69</td>
</tr>
<tr>
<td>4.7</td>
<td>Query Template parameters and errors</td>
<td>70</td>
</tr>
<tr>
<td>4.8</td>
<td>Verify Template</td>
<td>70</td>
</tr>
<tr>
<td>4.9</td>
<td>Notification Template</td>
<td>72</td>
</tr>
<tr>
<td>4.10</td>
<td>Update Template</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>BPD: Don’t worry, Be Happy</td>
<td>77</td>
</tr>
<tr>
<td>5.2</td>
<td>Wizard: choose Template</td>
<td>81</td>
</tr>
<tr>
<td>5.3</td>
<td>Wizard: select parameters and errors</td>
<td>82</td>
</tr>
<tr>
<td>5.4</td>
<td>Wizard: choose Task</td>
<td>82</td>
</tr>
</tbody>
</table>
5.5 Wizard: choose Activities ........................................ 83
5.6 Graphical notation: exception boundary .......................... 85
8.1 Calculation process — full view .................................. 108
8.2 Calculation process — Normal flow view ......................... 109
8.3 Calculation process — Exception flow view ...................... 109
D.1 Update return ....................................................... 128
D.2 Return goods ....................................................... 129
D.3 Receive returned goods ........................................... 130
D.4 Perform disposition ............................................... 132
D.5 Issue credit ......................................................... 133
D.6 Administer return request ......................................... 134
D.7 Approve return ...................................................... 135
D.8 Deliver item ........................................................ 136
D.9 HR Hiring ........................................................... 137
D.10 Software Defect Management ..................................... 139
Chapter 1

Introduction

In the beginning, there was the process, a sequence of activities. Then came the businessmen and created the business process. That meant, the activities came from the business world and the scope of the process was a single business goal; for example, purchase a product. And the businessmen raised their hands and brought forth the business analyst to design business processes for their business goals. And for a time it was good.

But dark shadows began to rise, as unforeseen events plagued the execution of business processes. And the people were at a loss, for these exceptions were not charted in the business processes. The cries of the business analyst echoed in the corridors, “It is not my fault. It should not have happened. I cannot think of everything.”

So, it came to us, the good people of BGU, the quest to find the “magic” tool for the business analyst — a tool that will expect the unexpected and identify possible exceptions in the business process. In our journey, we track the following path: the gateway (presenting the background), the challenge (the problem statement), and the triumph (our solution and its assessment).
1.1 Background

1.1.1 Business process and business analysts

Business analysts use Business Process Modeling Notation (BPMN) to model business process diagrams. BPMN [32] has risen in recent years as the popular modeling “language” for business processes. Figure 1.1 is a simple example of a BPMN diagram. BPMN became the popular standard thanks to three main reasons: (i) its simple core elements which allows for clear diagrams and can be extended into complex graphical notation, depicting any business use case. (ii) its interoperability, which allows the business analyst to focus on the proper sequence flow, of the business processes, without concerning himself on the proper implementation of the process; e.g., be more concerned that a ‘Purchase’ process includes delivering the items and not how the items will be delivered. (iii) BPMN suggests a transformation to an executable format of the business process, such as Business Process Execution Language (BPEL).

These strengths of BPMN allow businesses to increase efficiency by automating part of their business processes, using BPEL, and by giving a clear representation and analysis of their business process, using business process diagrams (BPDs).

Business process modeling tools

With the rise of BPMN and BPEL, software-based modeling tools appeared, to help modeling, designing, transforming, and executing business processes. Tools, such as Tibco’s iProcess[40], IBM’s Web-Sphere[23], and the Prosero prototype [13], offer different features that promise to make the work of the business analyst simpler.
1.1.2 Business process and exceptions

What are exceptions? Why don’t you make yourselves a nice cup of tea while we explain. Aha! What if the kettle is malfunctioning? Did you stop to consider that?! Without a kettle, how can you make tea? And without tea how can you engage in reading the explanation you are reading at this very moment?

Just like any real world scenario, executing a business process sequence flow may encounter unexpected, or unlikely, hardship and difficulties — known as exceptions. Exceptions include any type of failure to perform a business task successfully. Its ironic that while exceptions are often defined as “unexpected” events, all instructive languages include methodology to specify possible exceptions and their handling — e.g., in case you fail to deliver the package, call the recipient.

BPMN includes the Intermediate Error Event element, to allow the business analyst describe an exception in the workflow or the handling of an “unexpected” (exception) event. However, BPMN does not require the business analyst to handle exception events. Therefore, none of the modeling tools force the business analyst to insert in their model the handing of a specified or probable exception event. This seems, to us, a major failure to help the business analyst model a safer, more robust, business process.

1.2 Problem statement

1.2.1 Business analyst and exceptions

We are certain the business analyst would like to produce robust business processes. Nevertheless, after reviewing hundreds of business processes [40, 23, 31], we counted only a few cases of exception handling, i.e., Intermediate Error Event. Why doesn’t the business analyst handle exceptions as a common practice?

Silver [35] claims that business analyst don’t deal with exceptions since their tools are unable to execute exception handling, and since they are not equipped with the required training for this task. Our assumption is that business analysts understand
the notion of exception. However, they don’t use them since they usually focus on modeling the “happy path”. That focus, combined with probable deadline pressure causes them to overlook, or forget, pitfalls in the business processes. Even known pitfalls! Meaning, pitfalls in the form of a sub-process with an explicit exception.

Adams [2] suggests it is a common practice that exceptions are dealt with off-line, outside of the system, by a human agent. Therefore business analysts don’t bother with exceptions. Like Adams, we believe it is wrong not to include in your model your off-line business decision regarding the exception. While Adams attacks the problem at execution time, recording the business decision, we would like to solve the problem in modeling time, helping the business analyst to consider and model a business decision.

**Good handling is worth its weight in gold**

There are two clear advantages for the business analyst to specify, in modeling time, how an exception should be handled, in execution time. We will explain by a ‘poem’ example. You could say that adding vowelization to a poem is very helpful for an automatic service to read the poem out loud. It is also beneficial for any human that will read the poem.

It’s a philosophical motivation — the machine aspect is to make life easier (or so they say) and is usually about saving money, as exceptions will be handled automatically. The human aspect is to make something clearer and more coherent, as the business behavior is accurately depicted, by specifying the business rule for handling certain failures.

XXXXXXXX This is why we think business analyst must focus on exception handling. We decided to address and solve the problem of dispersed exception handling in current BPDs.

### 1.2.2 Business analyst and work practices

Our assumption is that for complex business processes, the business analyst is more likely to miss a probable exception, as he is focusing on a long “happy path”.

We hypothesize that by giving the business analyst a small set of possible exceptions, appropriate to the type of the business activity, we will invigorate him to consider possible exceptions. Since it is unlikely the business analyst would like to loose his “happy path” line of thought, we allow him to mark some of the exceptions, so he will be able to return and handle them later.

1.3 Solution

1.3.1 Static verification

As oppose to BPMN, which is an abstract level model, many current programming languages, such as Java, include the restriction that exceptions must be handled. As a result, many software editors that support those languages force the user to handle exceptions; and by that, help the user identify (or avoid missing) possible exceptions. Programming environments employ static verification — verifying the static model — to identify explicit exceptions, explicit references to the scope of the exceptions, and explicit exception handling, and make certain all exception are handled.

Our solution involves extending BPMN with attributes, that will allow modeling tools to use static verification procedure in order to identify unhandled exceptions.

1.3.2 Construct

Construct is an extension of BPMN Activity. It includes the new attributes: Parameters, Errors and Handlers. Errors will hold possible exceptions for the Activity, and Handlers will hold actual exception handlers, attached to the Activity. By comparing the Errors and Handlers attributes, the static verification can identify unhandled exceptions.

Parameters attribute has nothing to do with the static verification. To provide additional support, the tool will recommend, for each Activity, a limited list of business classification, requirements, and exceptions. Selected classification and requirements
will enter the Parameters, while selected exceptions will enter the Errors. Some Parameters are associated with certain Errors, and selecting them will automatically select the matching exceptions.

### 1.3.3 Templates

We further hypothesized that when classifying a business process, the classification can be associated with a workflow pattern; and that business analyst can use the pattern to quickly model a sub-process, saving time and energy.

We refer to these patterns as Templates, which is our secondary feature, as it is not directly related to exceptions. Every Template is associated with a specific classification, and selecting it also selects the appropriate Parameter and its matching Error.

### 1.3.4 Experiment

This study is partly about making someone’s life easier, and partly about making business processes more robust. And that is the main expected contribution — a BPMN modeling tool that is intuitive and easy to use, to produce a robust business processes. In this study, the robustness is measured in how well the process handles possible exceptions.

To assess our modeling tool, we took eleven business processes, of various complexity levels, and modeled them using Prosero’s modeling tool, enhanced with Constructs and Templates. The resulting business process diagrams were compared with an expert opinion, regarding probable exceptions.

**Results and conclusions**

- Using the enhanced tool produces more robust business processes — the business analyst’s choices for exception handling were a close match to that of the expert.
- Using the enhanced tool produces more cluttered business processes — the business analyst handled several unlikely exceptions, which overcrowded the diagram
with redundant exception handling.

1.4 Last minute notes

- Masculine form is used in this paper and refers to both genders (though everyone knows there can be only one).
- Some words will appear capitalized, in certain places, to refer to a BPMN element, and in regular form in other places. For example, “activity” is simply a business operation, while “Activity” is a core element of BPMN specification. The same goes for words such as “Process”, “Sub Process”, “Gateway”, and “Task”.
- Those new to the field, can consult the Dictionary, in Appendix A, for unfamiliar terms.
- Those familiar with business processes, exceptions, and modeling tools, can skip some of the longer chapters, such as Chapter 3 and Chapter 5, though we tried our best to make them as entertaining as possible.

1.5 DELETE ME

1.6 Contribution

To save you the time and effort of reading this comprehensive piece of work, I will tell you its secrets here and now (provided you do not divulge them to unauthorized personnel).

1.6.1 On errors

There are (at least) two distinguishable kinds of errors: syntax errors and exceptions.

The exceptions we fear the business analyst will miss are the exceptions from Sub-Processes. If a Process will call a Process which will call a third Process and there
an error might emerge, then the business analyst would have to remember all these references to identify a possible error. To truly help him identify all possible errors of a Process, one would have to recursively analyze all the Tasks and Sub Processes in it. Though this is the best solution, it entails some problems that must be dealt with, such as presenting inconsistencies to the user.

Instead, we give pattern related errors. We identified high level categories of BPMN Activities — Internal Activity, Send Receive, and Receive Process Reply. To differentiate these Activities from the standard BPMN Activities we call them Constructs, since they are the main elements which build the business process. Each Construct comes with a short list of errors appropriate to its behavior. For the Receive Process Reply Construct we identified some high level patterns of processing. We "molded" these into Templates: Query, Notification, Verify and Update. Each Template comes with a short list of errors appropriate to its pattern.

There will also be a short discussion on a new type of error — Commitment Violation. Commitment concept, though not directly related to Error Handling, is unique enough to deserve its own chapter.

1.6.2 On features

As a programmer, I use tools that give me great assistance in identifying errors in my code and provide code templates. My motivation is clear — to give the business analyst tools that provide similar assistance.

So, the first feature is good static verification in terms of exception handling. The tool will identify where an Error Event was thrown, and was not caught and handled. The tool will highlight such occurrences to the user.

The second feature does not come from the software industry, but more from word-processing: giving suggestions to the user. The tool will suggest to the user what would be an appropriate Error Event in certain cases.

The third feature we will show is offering business process Templates — standard patterns for simple business goals, such as ”Update Document”. This is both useful for
the second feature, by affiliating Templates with Error Events, and in general for fast creation of multiple business processes of similar pattern.

The fourth feature was a happy coincidental result from all the above features. The added implementation data to the business process improves the matching process of business Activity to Web Service. The matching process is discussed later.

To be honest, the last two features are so beneficial that I considered renaming my research to "Enhancing BPMN". But since they were the result of the exception handling research and the fourth feature is applicable only to systems, such as Prosero, that matches Activity to Web Service I decided to leave the name as it is.

Now to an easy part. The related work: . That’s it. None. Many are researching on the transformation from BPMN to BPEL. Even more are working to improve the execution process of BPEL by selecting Web Services on the fly and automatic exception handling. No one is trying to improve the modeling process with exception handling verification that will later, at execution, will prevent process failures. And... with good reason. A major conclusion of this thesis is that while our solution diminishes the amount of unhandled exceptions, it does not solve the more serious problem of identifying unhandled exceptions from Sub-Processes. Although this problem is solvable it requires both an extensive reconstruction of Prosero modeling tool architecture and extending BPMN Process’ meta-data.

To make along story short, the business analyst, while modeling the happy path, is likely to miss possible exceptions.

Now, to make a long story long, let us consider a concrete example. Let’s say we need to add the Activity of “retrieve investors list”. In a standard modeling tool we would add the Activity, name it accordingly, and continue to the next Activity. We would like to force the user to invest an extra second or two to consider: what type of an Activity is that? What exception might occur? It is a Send-Receive Activity, and
so the ‘send’ operation might fail at execution. The business analyst’s experience is extremely valuable when it comes to deciding how probable an exception is.

If the business analyst does not focus on the “happy path” and stops to add the full exception path when one is needed; or if he has the practice to carefully go over the completed business process and consider possible exceptions; then our solution will be of little use to him. But as we have seen, the lack of Error Events in the hundreds of business processes we reviewed suggests this is not the normal case.

Though I like the notion of transforming BPMN to BPEL, this is not the major advantage of the language, in my considered opinion. A language that describes a process sequence flow in a clear and readable way has its own merit, even without automatic execution, that is, even if all tasks are performed by humans. Having said that, I must admit that exception handling is more important for automatic execution than for human execution, for obvious reasons - humans can deal with unexpected events much better than machines (at least for the time being).

Many studies have focused on solving the problem at the execution side, leaving the business analyst with fewer cares. We have two problems with that approach. (1) Like [2] has shown this leads to big discrepancies between the business process model and the actual execution of the process. Meaning, the BPD does not truly represent the business process! (2) There are many business level exceptions. These are the responsibility of the business analyst, who is more capable of assessing and modeling them than the IT professionals, who handle the execution of the business process. (3) Let us consider, for a moment, the case where there are no Web Services to execute the process. It was written for, and will be executed by, the loyal employees of the company (the company). Only humans. XXXXXXX-see above-XXXXXXX
Chapter 2

Background

A short review of the bare necessities (“the simple bare necessities...”, Jungle Book) one must know in order to understand our scope and motivation. Later, in Chapter 3, we elaborate on these topics.

Sections 2.1 and 2.2 discuss how businesses improve their performance by (a) formally modeling their business processes, using BPM; and (b) automating their business processes, using SOA. In Section 2.2.3 we show how the semantics of the business process are used to help the transformation from the BPM (conceptual) level to the SOA (executable) level. Section 2.2.4 introduce our prototype modeling tool, Prosero, which supports semantic-based, BPM to SOA, transformation.

Section 2.3 detail the concept of exceptions. How they harm business processes execution, and by that lowering productivity. By considering exceptions, at modeling time, and preparing proper reactions — i.e., handling the exceptions — the business analyst can minimize failures during the business process execution. Section 2.3.2 shows how programming environments force the (poor) programmer to handle exceptions. Which leads us to our motivation, Section 2.4, to force the business analyst handle exceptions.
2.1 BPM

The principal scope of this study is Business Process Models (BPM\(^1\)). Business processes are like complex recipes that indicate how to handle certain business situations. For instance, a process used in a sales organization would describe the steps involved in taking care of an order for a product by a customer. Steps described in the process can be assigned to staff within the company to be performed manually (e.g., “approve purchase order”), or they can be sent for automatic execution to back-end servers of the company, or to third-party partners and suppliers. The formal documentation of business processes allows companies to improve their efficiency, monitor their operation, and detect bottlenecks or deficiencies.

2.2 BPM and SOA

In recent years, Service Oriented Architecture (SOA) has emerged as a powerful methodology allowing business process analysts to specify business processes and Information Technology (IT) specialists to provide services in a form that is immediately usable at the business strategy level.

2.2.1 Business Activity vs. Business Service

While SOA is an implementation supplied by a specific vendor, BPM is of a higher level of abstraction (“it’s only a model”, Monty Python). Service is a real-world operation. When you execute a service something changes in the world — e.g., your package is delivered or you get a phone call telling you your package is lost. You can’t execute a business activity — it is a concept; a headline with no contents or no instructions of how to perform the activity. For example, many services, from different vendors, may implement the activity *Deliver Package*. Vendors such as Amazon, B&N, and Steimatzky have their own services for delivering a package. They all may even share the

\(^1\)BPM can also stand for the related Business Process Management, but not here.
same BPM (e.g., Figure 2.1), with a sequence of activities *Sell Book*, *Receive Address*, and *Deliver Package*; but have different implementation (service) for each activity.

![Diagram of workflow example: Deliver book](image)

**Figure 2.1: Workflow example: Deliver book**

As another example of the interoperability of a BPM, consider two completely different organizations: restaurants and car rental. If you review restaurant business processes you realize that most will fit the car rental business. The participants and the activities’ implementations are different but the process structure, at the abstract level, is the same. Instead of a kitchen entity you have a garage, instead of a waiter you have a teller, instead of reserving a table you reserve a car, instead of updating the food in your storeroom you update the cars in your parking area, but the model of the reservation process and of the update process is essentially the same.

In Section 2.2.4 we will see how the business analyst can utilize the power of interoperability and save multi-discipline business processes, such as ‘Update inventory’, in a central repository, for re-use in different organizations.

### 2.2.2 SOA and Web Services

SOA organizes IT services as a set of web Services, which are software modules available on the Web through a standard interface. For instance, in order to deliver a package, one can invoke such a web service; once the request is sent through a regular web browser interaction, someone will pick up the order and deliver it to the requested destination. Calling a Web Service is not much different than running a program on a computer in a room full of computers — first, you need to find out which computer to approach. Then, in which folder, on the computer, the program is located. For a Web Service, you need to know which server (a remote computer on the Web) to address and in which folder, on the server, the service is located. For example, www.mail.com/services/deliver can be a web service location.
2.2.3 Semantic Web Services

But what is the purpose of service www.mail.com/services/deliver? Does it deliver packages? Letters? Mail order brides? Anything? How do you know what to expect from a Web Service? This question is important if you want to match a business activity (conceptual) to a web service (actual). Web services describe themselves in a format called Web Service Description Language (WSDL). One must read this description to learn about the web service.

The description includes a formal part and a natural language part. The formal part involves the input and output fields of the Web Service. You can’t put a round peg into a square hole (unless you are on Apollo 13). The input and output structure is specific and cannot be deviated. For example, a “Sum” web service operation can receive exactly two integers and returns a single integer. It will not accept any other type of data and will not return anything but an integer. The name and the description fields, on the other hand, can be written in a natural language. For example: “a very good sum operation”.

As a real world example, here are two operations from one of Amazon’s WSDLs:

<operation name="ItemLookup">
  <input message="tns:ItemLookupRequestMsg"/>
  <output message="tns:ItemLookupResponseMsg"/>
</operation>

<operation name="CartModify">
  <input message="tns:CartModifyRequestMsg"/>
  <output message="tns:CartModifyResponseMsg"/>
</operation>

We can see both operations have a specific message (data object) they can receive and that their name describes their functionality. The schema of the message is defined elsewhere in the WSDL.

The idea behind semantic Web Services and semantic BPM is to identify semantic
relations between a business activity and a web service operation. Examples for strong semantic relations:

- “Ship product” and “Deliver product”
- “Address” and “Home address”
- “Address” which contains city, street, and zip, and “Ku-Ku”, which contains city, street, and zip

The similarity is estimated according Activity’s name, input, and output, and the web service operation name, input, and output. That’s in a nutshell.

A tool capable of identifying the semantic relation can immensely assist the business analyst, and simplify the transformation of business process model (conceptual) to executable business process (actual), by suggesting for each activity a well-suited, matching web service. Our case study tool, Prosero [13], is such a tool.

2.2.4 Prosero

The Prosero project [13] relies on the analysis of gaps between the level of analysis used by business analysts (at the business strategy level) and software developers (at the technical level). To bridge this gap, the Prosero project introduces a set of reference models which are semantically annotated and describe the activities and data objects manipulated by typical business processes. Business analysts rely on these semantic repositories to compose composite business processes that can be safely mapped to executable SOA Web Service orchestrations.

A business analyst defines a process, for a customer, using a prepared reference model repository (RMR). This repository includes predefined tasks and data-objects to be used in building the business process. The defined process is entered into the customer models repository (CMR). The analyst may then customize the business process according to the customer’s specific mode of operation. The Prosero system will then transform the customized business process, in BPMN format, into a SOA process-sequence of web services (implemented in BPEL). The web services are taken from
a Web Service Repository (WSR) and are selected by the Prosero Matcher to fit the functional and non-functional requirement of the overall business process.

2.3 Exception Handling

It is time to introduce the villain of the story — the exception.

2.3.1 Exceptions

A service is an executable, software-based, operation. As such, it has the potential to fail during execution, mostly due to unexpected conditions. This failure is called, in the software industry, an exception.

If a service is part of an orchestrated workflow, then an exception in the service is propagated upwards and the entire workflow might fail. Since the workflow represents a business process, that means the business process execution failed. Failures lead to costly delays and frustrations (and frustrations lead to the dark side).

One would assume that business exceptions vary from one business to the other just like business activities vary. One would be wrong — there are similarities. As we saw, the business activities of restaurants and car rentals are quite similar at the abstract level — “Make reservation” and “Update storage” — and so the business exception might also be similar — “Item missing” (be the item a car or a banana).

2.3.2 Exception Handling in Programming Languages

Exception is a common (and unwelcome) occurrence in the software industry. For that reason, most of the programming languages offer a system to handle exceptions. Not only that, the popular programming languages, such as Java, also offer validation to make sure specified exceptions are handled.

For example, in most Java-supported editors, calling a method which can throw
an exception will force the caller to handle the exception. Otherwise, the editor will highlight the calling statement pointing out that it is a mistake not to handle the exception, as we see in Figure 2.2.

```java
public String getName() throws Exception {
    throw new Exception();
}

public void myMethod() {
    String name = getName();
}
```

Figure 2.2: Unhandled exception notification, in Java.

This validation and its associated warning is the base for our motivation.

2.4 Motivation

We completed our short review of the general scope: BPM, SOA, and the transformation between them. We determined the importance and efficiency for companies to model their business processes and to automate them. Our specific scope is the tools that allow for the modeling, the transformation and the execution — we will focus on Prosero.

Another part of the scope is exception handling. More on exception handling will be detailed in Section 3.2. Still, it is clear that many are motivated to prevent exceptions from occurring. Unhandled exceptions cost time, money, attention, and usually require a human agent to handle them. While most research today focuses on automatic exception handling at execution time, [2, 33, 39, 8, ?], we are determined to attack the problem at the modeling phase. We have several reasons to motivate us:

(i) The earlier you discover a problem in your plan or model, the more problems you avoid later.

(ii) There are many business level exceptions. These are the responsibility of the
business analyst, which is more capable to assess and model them than the IT professionals, who handle the execution of the business process.

(iii) All the solutions for automatic exception handling do not deal with business processes which are not intended for automation. I.e., business processes that were written for, and will be executed by, the loyal employees of the company (the company).

(iv) The way a business handles its exceptions is part of its business logic. It should be depicted in the business process, not only performed at execution.

We acknowledge the major responsibility of the business analyst when writing a business process — to make it robust by avoiding or handling unexpected scenarios (exceptions). This is part of his responsibility to write coherent business processes.

As programmers and software engineers, we have our own tools. As we noted on exception handling, our tools help us to identify where we forgot to handle an exception. We want to give the business analyst the same help we get.

The main problem we want to address is the scenario where a business process contains a call to a second business process (as a sub-process) and that process calls a third business process, which raises an exception. It is unlikely that a business analyst will notice or remember the existence of that exception while he models the normal sequence of activities, and therefore we assume it will not be handled explicitly.

Unfortunately, the architecture of many programming languages is more rigid than business process architecture, which allows for constraints that prevent unhandled exceptions. On top of that, tools such as Prosero do not automatically access referenced sub-processes and so cannot identify exceptions therein. A broader discussion can be found in Appendix B. We concluded that we cannot identify unhandled exception from sub-processes.

Nevertheless, we are motivated to mimic unhandled exception warnings in the Pros-
ero modeling tool. So, we are focused on the secondary problem: how to help the business analyst consider possible exceptions as he models the normal sequence of activities? And that is the problem this research proposes to address, solve, and implement.

2.5 DELETE ME

XXXXXXXXXXXXXXXX

IDEs are our inspiration
XXXXXXXXXXXXXXXX

humans handle exception better than web-services
XXXXXXXXXXXXXXXX
Chapter 3

Starting Points

In this chapter we will present the up-to-date practices of business process modeling. We will focus on the industry leading “language” for writing business processes, BPMN, its syntax — especially for exception handling — and how it was implemented in Prosero (before we came along).

3.1 BPMN

3.1.1 BPMN; What is it good for?

BPMN, Business Process Model Notation [32], is a specification for writing BPMs. The specification was detailed by the OMG. BPMN’s final specifications, for Version 1.0, were set as late as February 2006, just under three years ago. This is the first hint that this standard may have room for improvements. The second hint is that even in Version 1.1, emerging in 2008, all of the enhancements we suggest throughout this study were not added.

Before we begin our review, let’s examine a BPMN example — Figure 3.1.

The first thing we notice about this diagram, prior to all the explanations, is that it is clear. Even people outside the industry can follow the described process of handling
an order. The second thing one might notice is that no data is specified in the process, only activities. More on these two will follow.

Now, how did BPMN come to be? Prior to BPMN there were:

1. Scores of process modeling tools and methodologies; i.e., all sorts of visual business process flow-chart formats. We will not elaborate on these. We will focus on BPMN’s Business Process Diagram (BPD).

2. Web service-based XML execution languages, e.g., BPEL, for Business Process Management systems. These languages are not discussed in this section. Suffice it to say they can be handled very well by a software system, but would be hard to understand by business analysts and managers.

That created some problems:

(A) Business analysts are required to understand multiple representations of business processes.

(B) Business participants that don’t share the same graphical notations might not understand one another.

(C) A technical gap between the format of the business process initial design and the format of the languages that will execute these business processes.

BPMN solution:
(1) BPMN provided a standard graphical notation which follows traditional flowcharting. This standard is now wildly accepted.

(2) To solve the technical gap:

(a) BPMN provides BPD - to be used by people who model and manage business processes.

(b) BPMN provides formal mapping to an execution language of the BPM system - to be used by IT (technical) people who design the process (software) execution.

3.1.2 BPMN for mummies

We now offer a summarized report of BPMN’s technical specification, Version 1.0. [32] — BPMN graphical elements and non-graphical attributes. For further information on BPMN you can read the complete specifications. But for those who prefer to “look and listen,” I highly recommend the [36] online course. It is free, registration is required, and it takes less than two hours. It includes all the information summarized in this background plus some useful tips. However, the first forty minutes are enough to understand the basics.

Basic BPMN

The three most important and basic elements are the Activity — a rectangular shape, the Gateway — a diamond shape, and the Event — a circular shape. As noted, they have different sub-types, graphically depicted by an additional icon inside the basic shape. But, we are not in any hurry to know everything. We’ll touch on some sub-types later. First, understanding the three basic elements is winning half the battle. Let’s look at the BPMN diagram we saw in Figure 3.1:

Activity is a point of decision making. Some data is being prepared or reviewed. Or, some action is being taken by a person or by a machine. For example, “Ship
"Order" is a physical action, while "Send Invoice" can be performed by automated email.

**Gateway** is either a conditional decision, splitting the sequence to several continuations, or a point where several paths join together. The notation of the Gateway and its outgoing flows should make it easy to understand. Here we see that the "Accepted or Rejected?" Gateway is an exclusive decision with one path for each scenario. And the other Gateway joins two parallel paths. Meaning, there is a condition here — continue only after both paths have been completed.

**Events** are one of the innovations of BPMN. They mark a point where the process sends a message, or a signal, to another process. Alternatively, they can mark a point where the process is waiting for a message/signal. Here, the circle on the left is the **Start Event** — a point where the process is initiated (signaled to start); the right circle is the **End Event** — where the process signals its completion. The "Accept Payment" Activity could be replaced by an **Intermediate Event** — waiting for the message 'payment was accepted'.

Now half the battle is won, let us conquer one more outpost by a quick review of the possible sub-types and how Pool and Lane would fit in the above diagram, Figure 3.1.

**Core BPMN**

Here we have BPMN core graphical objects and their relationships, which support the requirement of a simple notation. *(The italic text is a brief description).*

- Flow Objects — *behavior of a Business Process*
  - Events
3.1 BPMN

- Start
- Intermediate (we will focus on Error)
- End

- Activities
  - Process (not a graphical element)
  - Sub-Process
  - Task (atomic activity)
  - Service | Receive | Send | User | Script | Manual | Reference | None

- Gateways (control how Sequence Flow interacts)
  - Exclusive (xor) Decision/Merge
  - Inclusive (or) Decision/Merge
  - Complex (*) Decision/Merge
  - Parallel (and) Fork/Join

- Connecting Objects — connecting the Flow Objects
  - Sequence Flow (order activities)
  - Message Flow (cannot connect two objects within the same Pool)
  - Association (Artifacts with Flow Objects)

- Swim-lanes — grouping
  - Pools (Participant in the Process)
  - Lanes (organize and categorize activities)

- Artifacts — additional information
  - Data Objects (e.g., “Purchase Order [Complete]”, “Purchase Order [Approved]”)
  - Group
  - Annotation

**Pool** is the chief participant body which provides the resources for executing the process, usually, the company which manages the processes. Here we could say, the company is “Big Happy Boxes,” and that would also be the name of the Pool.
Lanes reside inside the Pool. They are categorizations of duties, usually departments in the company. Here we would assign “Ship Order” to the ‘Shipment’ Lane and payment related Activities to the ‘Accounting’ Lane.

Advanced BPMN

Has it occurred to you from where, our process, in Figure 3.1, receives the payment message? Let’s say it’s from another process. The other process is called an external process. We could draw an abstract estimation of that process, since it is not our process, and show the message choreography between the two processes. As we see in Figure 3.2.

![Figure 3.2: Handle Order with choreography.](image)


In this work we deal only with Private (internal) business processes — internal to a specific organization. We will not discuss choreography.

BPMN is intended to be extensible by modelers and modeling tools — meaning: adding attributes to existing elements and adding new element types. To satisfy additional modeling concepts that are not part of the basic set of flow elements, BPMN provides the concept of Artifacts that can be linked to the existing Flow Objects through Associations. Thus, Artifacts do not affect the basic Sequence or Message
3.2 Exception Handling Specifications

Flow, nor should they affect mappings to execution languages. The meaning and the execution of the original ‘Handle Order’ process, in Figure 3.1, and of the process in Figure 3.3, is exactly the same.

Figure 3.3: Handle Order with Artifact.


‘Order’ is an Artifact of type Data-Object, and it is associated (by an Association link) as the output of the ‘Receive Order’ Activity and as the input of the ‘Fill Order’ Activity. Although this information seems very relevant, BPMN does not emphasize Data. BPMN is, rightfully, more concerned with the sequence flow. However, we will see, later on, that for Prosero, Data-Object information is useful for building the execution of the process.

After this crash course in BPMN, you are ready to start modeling diagrams of your own, using the bare necessities: Start Event, Activities, Gateways, and End Event. For more information, I suggest you consult BPMN tutorials [32, 36, 4, 43] or one of the many resources on the Web. We are not quite done with the specification. In Section 3.2.3 we will elaborate on exception handling in BPMN. For that, we must first talk a bit about exceptions.

3.2 Exception Handling Specifications

3.2.1 Problems at execution

To understand exception, we will review three types of execution paths. For any instructive description, be it in a text, diagram or program, there are three paths:
**Happy path** is the sequence of events which can be considered the best scenario to achieve the goal of the instructions. ¹

**Alternative flow** is when everything is not going well, but you can still achieve your goal. ²

**Exception Flow** is a type of an alternate flow. The only difference is that it involves catching an exception, regardless if the goal is reached or not. ³

Consider for a moment the BPMN diagram we saw previously in Figure 3.1: the ‘Accepted’ scenario is the happy path, while the ‘Rejected’ scenario is the alternate path. In both cases, the executor is clear about its state at every moment along the sequence flow. Even if there is a problem with the ‘Receive Order’ Activity, which results in a rejected order, the executor will use the Gateway to follow the alternate flow. In such a case of alternate flow, the Activity has completed and returned control to the process executor with an order response.

But, what if the executor receives control and he is aware that the Activity did not complete? Rather, the Activity aborted as result of a problem, and the executor is in an uncertain state: (a) it received no order response; and (b) he cannot continue with the sequence flow, since the Activity did not complete. That is a different type of a problem — an exception. In such a case of exception flow, exception handling is required.

### 3.2.2 Exception

Adams [2] explains that “an exception is simply an event that is considered to be a deviation from the expected control flow or was unaccounted for in the original process flow.”

Exception at execution time means the system has aborted the current execution, namely, the execution stops at the current scope and the execution control returns to

---

¹Also known as normal path, normal flow, basic flow, and sunny day scenario.
²Also known as alternate path.
³Also known as exception path and exception handling flow.
the calling scope. We say that the system **threw** an exception or **raised** an exception. The system usually informs the calling scope it has aborted — that is the exception message — and might include in the message details of the exception, such as its reason and the particular action that caused it. The message the system passes is also referred to as an exception **event**.

Reasons for an exception can be:

- An unknown instruction. Example: Jump over that ksdghus.
- A limited resource. Example: Jump over that building.
- An instruction, in the process, raising the exception. Example: Raise ‘no-jumping’ exception.
- And many more.

Exception at design/modeling time is an element which refers to either catching or throwing an exception event.

*What’s in a name? That which we call an Exception*  
*By any other name would fail the execution.*  
(ahmm... not quite Shakespeare)  

Russell [33]


We combined these categorizations into two main classifications: Process Exceptions and Data Exceptions. The distinction is Data Exceptions can be specified and raised as a result of business conditions/rules, while Process Exceptions, like system faults, can hardly be predicted by the business analyst.

- Process Exceptions
  1. Failure
     - Generic infrastructure failure stemming from low levels that cannot be anticipated at BPMN level.
– Analogous to “Work Item Failure”.

2. Timeout
– An Activity is taking too long to finish, according to system settings.
– Analogous to “Deadline Expiry”.

• Data Errors

3. Superficial data error
– Can be checked without special processing (e.g., wrong argument type).
– “This is the wrong form. You need the red form!” the administrator said as she looked angrily through her thick glasses.

4. Content level data error
– Requires processing (e.g., incorrect URL).
– “This is the right form. But I can’t understand a word!” the administrator growled, pointing you to the door.

5. Meta-data level error
– Something wrong with meta-data, e.g., unknown sender, authorization, authentication, etc.
– “Who are you and how did you pass security?!” The administrator tears her skirt and turns to combat position.

6. Commitment/precondition violation
– Correct data that violates some previous commitment in the process or requires a missing commitment.
– “I’ve been waiting for over an hour!” The administrator screams at the delivery guy. “I ordered at 3:21, as you can see from the receipt. I am not paying for this Pizza.”

When a task fails, its execution is stopped and the control of the process is handed over to a single handler predefined for that type of exception. The context of the activity is not accounted for. If the handler cannot solve the problem, it propagates the exception up the activity tree; if no handler is found, the entire process (execution) instance aborts.
3.2.3 Exception Handling in BPMN

In BPMN, an exception event is called an *Intermediate Error Event*. We will call it an *Error Event*. As you may guess, an Intermediate Event is a sub-type of Event, and an Intermediate Error Event is a sub-type of an Intermediate Event.

Here we will elaborate only on the elements which are the focus of this paper, related to exception handling in BPMN:
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>An event is something that “happens” during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.</td>
<td><img src="image" alt="Name or Source" /></td>
</tr>
<tr>
<td>Flow Dimension —</td>
<td>Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process but will not start or (directly) terminate the process.</td>
<td><img src="image" alt="Intermediate" /></td>
</tr>
<tr>
<td>Intermediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type Dimension —</td>
<td>Intermediate Events have ‘Triggers’ that define the cause for the event. There are multiple ways in which these events can be triggered. End Events may define a ‘Result’ that is a consequence of a Sequence Flow ending. The ‘Error’ trigger is used for error handling - both to set (throw) and to react to (catch) errors. It sets (throws) an error if the Event is part of a Normal Flow. It reacts to (catches) a named error, or to any error, if a name is not specified, when attached to the boundary of an activity (Task or Sub-Process).</td>
<td><img src="image" alt="Error" /></td>
</tr>
<tr>
<td>Error (a.k.a. Exception)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exception Flow</td>
<td>Exception Flow occurs outside the Normal Flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process.</td>
<td><img src="image" alt="Exception Flow" /></td>
</tr>
</tbody>
</table>

A BPD is not designed to graphically convey all the information required to execute a business process. Thus, BPD elements will include attributes that will hold information about how these elements map to execution languages. Some relevant examples of
3.2 Exception Handling Specifications

attributes:

- ‘ErrorCode’ attribute:
  - For an Intermediate Event within Normal Flow: If the Trigger is an Error, then the error code MUST be entered. This “throws” the error.
  - For an Intermediate Event attached to the boundary of an Activity: If the Trigger is an Error, then the error code MAY be entered. This Event “catches” the error. If there is no error code, then any Error SHALL trigger the Event. If there is an error code, then only an Error that matches the error code SHALL trigger the Event.

- ‘Target’ attribute:
  - A Target MAY be included for the Intermediate Event. The Target MUST be an activity (Sub-Process or Task). This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity.

One or more Intermediate Events MAY be attached directly to the boundary of an Activity.
You can only throw and catch events within a single pool.

3.2.4 Exception Handling in BPEL

The mappings from BPMN to BPEL4WS (Business Process Execution Language for Web Service) are derived by analyzing the BPMN objects and the relationships between these objects. We will focus on errors and we will only cover the mapping to BPEL4WS, even though other execution languages are available.

This part is intended mainly to help understand some of the implementation considerations. It is not relevant for the majority of the study since the execution language is not directly part of our scope.
Table 3.2.5

<table>
<thead>
<tr>
<th>Intermediate Event</th>
<th>Mapping to BPEL4WS</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Intermediate Event" /></td>
<td>This will map to a <strong>throw</strong> element.</td>
</tr>
<tr>
<td><img src="image" alt="Intermediate Event" /></td>
<td>The mappings of the activity (to which the Event is attached) will be placed within a scope. This Event will map to a <strong>catch</strong> element within a scope. If the Error Event does not have an ErrorCode, then a <strong>catchAll</strong> element will be added to the <em>faultHandlers</em> element. If the Error Event does have an ErrorCode, then a catch element will be added to the <em>faultHandlers</em> element with the ErrorCode mapping to the <em>faultName</em> attribute.</td>
</tr>
</tbody>
</table>

### 3.2.5 Business process practices for exception handling

We will first discuss the common business methodologies of the alternate paths (from Section 3.2.1): check your state or return value versus receive an exception and handle it. Then we will argue in favor of exception handling methodology and conclude with a few popular exception handling techniques.

**Business methodology**

We have two pairs of examples to illustrate the choices businesses make in their alternate paths. Each pair represents exactly the same business process, modeled in the two methodologies.

Figure 3.4 represents the concept of first verifying that your action will succeed and only then performing the action. Figure 3.5 suggests attempting the action, and if you believe there is a good chance for a failure, then handle it.

Figure 3.6 is a good example of checking a returned value. In this case the ‘Customer’ Data Object has a special field, and if the customer is not found, then that field is marked with *false*. While Figure 3.7 claims Data Objects should not have special fields
for every type of failure and checking each separately, you should be informed of the failure by an exception.

Figure 3.4: Request catalogue BP, with Gateway.

Figure 3.5: Request catalogue BP, with Error Event.

Figure 3.6: Find customer BP, with Gateway.
Figure 3.7: Find customer BP, with Error Event.

We are not going to sit idly by, saying every business should make his own choices. No. We are not news reporters (who I wish would keep their opinions to themselves). We will argue that, as a rule, cases of failure should be dealt with by using exceptions. But, we must remember the choice is not always ours. In some cases, it depends heavily on whether an existing Activity returns an error statement in a Data Object or by an Error Event.

Exception handling vs. return values

Arguments for exception handling:\footnote{Thanks Guy W.}:

- Clean semantics for reporting errors
- Easier to analyze for exceptional error flows
- Provides a safe aborting method
- Throw the exception where it occurs; Handle it where you can
- Handling scope can cover several activities
- Allows errors and handlers specialization
- Easier to add to an existing diagram

Arguments against exception handling:
3.2 Exception Handling Specifications

- Expensive (time consuming) at execution time
- Breaks the message-passing sequence readability
- Not entirely safe — a missed exception can go a long way
- On occasions, covers up for the language design problems

We will give examples for some of the main Pros.

**Easier to add to an existing diagram** While working on an existing BPD, the business analyst has a very good reason for using Error Events over adding Gateways to check all sorts of conditions. Adding a Gateway in the middle of a sequence flow requires breaking at least one connection and probably moving the Activities to make room for the Gateway and then reconnecting the Activities with the Gateway. Attaching an Error Event, on the other hand, has very little influence on the rest of the diagram. Here is an example: in Figure 3.8, we see the repercussions of adding a Gateway between two Activities. Try to imagine there are numerous elements on both sides.

![Figure 3.8: Adding a Gateway to a Credit check.](image)

![Figure 3.9: Adding an Error Event to a Credit check.](image)

It is quite evident that attaching an Error Event, as in Figure 3.9, is much simpler.

**Handling scope can cover several activities** The combination of Sub Process and Error Event is a big advantage for the visual side and for the working man — cover more cases in less work. Diagram 3.10 shows how each result must be checked separately. In Diagram 3.11, if one Activity fails we exist the Sub Process automatically.
Again, some businesses would choose to follow return values and some would prefer exception handling. Their decision can be based on the business methodology of their current activities return values and on their implementation choices. We highly recommend using exception.

**Exception Handling Strategies**

How to handle an exception is not the focus of this study. We focus on identifying the exception and leaving it to the business analyst to decide what is the proper course of action. Nevertheless, we will make a short review of the default exception handling behavior of systems, followed by popular exception handling techniques.

If no handler is defined, the default behavior for exception handling at run-time is simple: transfer it to your superior. If you are a human executing a business process
3.2 Exception Handling Specifications

(a) Don’t handle

- Strategy: Don’t even attempt to catch a thrown exception.
- Result: Process aborts and control returns to caller.

(b) Ignore and continue

- Strategy: Catch but do nothing about it.
- Result: Usually futile, since part of the process was not completed. Furthermore, the process or its data might be at an illegal state. Even if you complete the process this time, the failure will probably occur next time, as well.

(c) Log

- Strategy: Write the details of the exception in a log.
- Result: Instead of keeping track of a process execution, you can read the log later, for any problems.

Figure 3.12: Handling strategies examples: (a), (b), and (c).
(a) Try again

(b) Recover

(c) Abort

Figure 3.13: Handling strategies examples: (d), (e), and (f).

(d) Try again

- **Strategy:** Try to do the task again.

- **Result:** Depends on the source of the problem. In some cases, you will get the same exception, in other cases, the problem (annoyingly) disappeared.

(e) Recover

- **Strategy:** Try to complete your goal even without completing the failed action. There are various methods of doing this.

- **Result:** The overall goal achieved.

(f) Abort the process

- **Strategy:** Throw your own exception.

- **Result:** This process still fails, but you can make sure the reason is clear, and maybe your supervisor, who will get your exception, will know how to handle it.

   Recover and abort are the two recommended possibilities.

**Automatic exception handling**

Most of the current research on business process exception handling focuses on automatic exception handling [2, 8, 16, 5], at execution time, or on exception handling in an executable language [39, ?, 15], such as BPEL.

---

5Bjarne’s Recovery Recipe.
Prime examples:

Changing a Web Service in mid execution [5]. Instead of having a single Web Service, which is suitable for implementation of a BPMN Activity, there is a set of Web Services. If one fails, then the execution engine attempts to use another one to fulfill the goal.

WSAT [15] is a tool that analyzes Web Services.

Adams’ worklets [2] (my favorite), allows a human agent to handle any alternate flows, including exceptions, the first time they occur. The system records the handling technique and when the exception appears again, attempts to use it. Adams also understands the importance of the business process model representing the actual business process and suggests reverse engineering of the executable business process, with the recorded handling techniques, to the model.

All of these are helpful and important, but don’t assist the business analyst to model the business process correctly in the first place.

### 3.2.6 Dealing with exceptions in IDEs

**The Java example**

We have already noted that existing tools do not help the business analyst to identify whether an Activity throws an Error Event. This is a major difference between BPMN and programming languages, such as Java. In most Java-supported Integrated Development Environments (IDE), referencing (“calling”) a method, which can throw an exception, will force the caller to handle the error. The IDE will highlight the calling statement pointing out that it is an error, as in the screen-shot of Figure 3.14.

To fix this error, the user has two options: catch and handle the exception (Figure 3.15), or pass the exception “upwards” (Figure 3.16).

Identifying thrown exceptions, and by that unhandled exceptions, is part of the static verification procedure.
public String getName() throws Exception {
    throw new Exception();
}

public void myMethod() {
    String name = getName();
}

Figure 3.14: Unhandled exception notification, in Java.

public String getName() throws Exception {
    throw new Exception();
}

public void myMethod() {
    try {
        String name = getName();
    } catch (Exception e) {
        // HANDLE EXCEPTION
    }
}

Figure 3.15: Catching and handling an exception, in Java

Figure 3.16: Passing an exception, in Java

Static verification

It is important to verify, as soon as possible, that known exceptions are being handled. Why as soon as possible? Simple, cost effectiveness. It is always good to discover problems as early as you can — preferably at planning or design time. Consider making a movie: you should make sure that everything is ready — consistent script, reserved locations, lively actors — before you start shooting. Otherwise, people (overpaid people) are going to stand around waiting until everything is ready — a waste of time and money. Verification is much more severe for construction planning, such as building a house or a bridge. The plans must undergo rigid verification processes before they are approved and the construction can begin. This verification is there to save lives. We don’t aim that high in this work. We aim to prevent costly execution failures, such as in Figure 3.17, when to sides of a bridge don’t meet.
In Figure 3.18 we see not only bad structure planning, but also bad exception handling — where a collection of bricks, wood, and what seems to be a rubber duck, prevent the bridge from collapsing. You can’t always blame a design for failing in real life due to an unexpected event. But in this case, you can definitely blame it on improper exception handling.

If you are interested in static verification methodology, you should look at Appendix B for a review of different techniques that can be used to statically verify a BPD.
3.3 BPMN Tools

We are interested in current working routines of the business analyst, how he models a diagram. You can model a diagram on a piece of paper, but you won’t get the assistance that only a quality software has to offer. We reviewed several business process modeling applications, apart from Prosero: eBPMN Designer (by soyatec.com), Business Process Visual ARCHITECT (by visual-paradigm.com), Business Process Modeler for Business Analysts (by eClarus.com), Tibco Business studio (by Tibco.com), and Intalio Designer (by intalio.com).

All the above tools are based on Eclipse\textsuperscript{6} platform, as is Prosero. A partial list of features, supported by each tool, is available in Appendix B.

These modeling tools follow the BPMN specification to the letter. Some may add extensions that fit their users’ needs. They employ static verification to force the user to keep within the BPMN constraints, much like a word processor employs spell checking to warn against mistakes. Since BPMN specifications do not dictate that exceptions must be handled, none of the tools, Prosero included, force its user to handle exceptions.

3.3.1 Prosero

We have already been introduced to Prosero in Section 2.2.4.

The Prosero approach \cite{13} includes a semantic, repository-based architecture and a set of supporting modeling tools. The Prosero semantic repository does not attempt to enable service composition from first principles, but instead to support analysts to produce customized versions of established reference models in an efficient manner. The Prosero methodology exploits the semantic repository to enable smooth collaboration among the customer, analyst, developers, and administrators.

The strength of Prosero comes from the semantic repository: best-practice models stored in the reference model repository, the ability to copy (re-use) a model and customize it into the customer model repository and simplify the transformation to a SOA

\footnote{www.eclipse.com.}
executable process by using the *Terminology* to match each activity in the model to the best-suited web service.

**Architecture**

The *Semantic Repository* within Prosero includes four parts: the terminology, the reference model repository (RMR), the web service repository (WSR), and the customer model repository (CMR). The role of the semantic repository is to hold the various relationships among the repositories and to facilitate the creation of these relationships.

The *Terminology* comprises business-oriented terms, automatically extracted from textual and semi-formal business documents, using computational linguistics term extraction algorithms. In the current state, that part includes terms derived from UBL 2.0 entities [31]. The terminology is used to annotate all the model elements in the repository, according to a context tree.

The *Reference Model Repository (RMR)* stores best-practice models of three types: organizational chart (business units and roles), data models and process models, which consist of activities, participating data, and control flow. These models are expected to be abstract and may be part of standardization efforts. All elements in these models are classified (tagged) with a context.

The *Customer Model Repository (CMR)* stores specific customers’ models. The structure of the CMR is similar to that of the RMR. However, its elements are not related to a context. Instead, each customer has an associated context. Models in the CMR are related to the RMR models from which they are derived. A CMR is actually a modified subset of the RMR, corresponding to the customization requested by the customer. In addition, CMR may add new elements which are not introduced within the RMR.
The Web Service Repository (WSR) stores the meta-data describing web services such as name, input and output, and non-functional properties (such as the Quality of Service (QoS) that is promised by the provider). The WSR includes information on the set of off-the-shelf services that can be deployed for customer processes. These services are “pre-approved” for composition. Each of these services (WSDL and XSD expressions) is tagged by terms from the Prosero terminology.

The Prosero Modeling Tools cover all business process related specifications. The available tools include:

Organizational structure modeling tool, in which the relationships among business units and roles are defined.

Data modeling tool in which the data model (related to business processes) of the customer is defined.

Business process modeling tool, in which the business process is specified using BPMN.

Preference modeling tool, which is used for specifying the customer preferences regarding the nonfunctional requirements.

The tool set is based on the Eclipse framework using the GMF meta-modeling approach.

The Prosero BPEL Generator transforms the specification of the business process modeled by the analyst as a BPMN diagram into an executable BPEL4People orchestration. While this generator cannot create a complete executable BPEL, it certainly serves as a firm basis for the developer to produce an executable orchestration out of the box. The generator transforms the BPMN specification of a CMR into a BPEL script; it matches the abstract activities specified in the CMR to web services of the WSR, based on their semantic similarity and associated non-functional properties; it

---

7Web Service Description Language.
8XML Schema Definition.
creates mediators in case of mismatch among the data types of the various services; and finally it produces GUI forms for human tasks as specified with the BPMN, according to the WS-HumanTask and BPEL4People standards.

The Prosero Portal aims at managing the entire system by controlling the various artifacts created during the system lifecycle. It also maps the roles (analyst, developer, architect, and administrator) to their allowed activities.

Modeling Support

The analyst uses graphical modeling tools for describing the customer model. General-purpose BPMN modeling tools all adopt the palette approach: a palette, with BPMN primitive constructs (Activities, Gateways, Pools, Lanes), appear; the analyst selects icons from the palette, drags and drops them onto the canvas, and constructs a model. In the Prosero modeling tool, in contrast, the analyst starts from a reference model. The key difference is that all elements in the diagram actually refer to entities in the repository: they can be drilled-down (opening a Data Object in the BPMN diagram opens this object in the data modeling tool; opening a pool opens the corresponding unit in the organizational structure modeling tool). The boxes are not “simple box and label” — they include full repository information, such as input and output data types for activities.

The rich semantic information available in the Prosero modeling tool is supported by model checking methods that can assist the analyst in the fine tuning of the reference model into the customized model desired by the customer. For example, the modeling tool supports type checking. When a Data Object is assigned to an activity, the modeling tool verifies that the Data Object can be coerced — through simple data mediation — into the data type expected by the Activity.
BPMN to BPEL Translation

Matching consists of associating activities and data objects in the CMR BPMN diagram to web services in the WSR. We can think of matching as a grounding process, from the abstract description of the RMR, as customized into the CMR, to the concrete elements of the WSR. The matching operation is based on the semantics assigned to elements in the repository (terminology tags) and on the non-functional properties. The result of a matching operation is an ordered (by similarity measure) sequence of matched web services.

We will not detail the Matcher operation. That is for another thesis (in another time, by another student). We will take a closer look at the functional requirements the Matcher considers for each Task. The functional requirements include:

- Task’s input Data-Object
- Task’s output Data-Object
- Task’s type (Service or Human)

This is important information for the Matcher. It helps to filter out nonrelevant Services which do not support the requirement.

3.4 Summary

We will now review all that we have learned by examining the relations between the different topics: BPMN, exception handling, static verification, IDEs, and Prosero.

‘Mind the gap’ — BPMN and Prosero

BPMN supports sequence flow diagrams. It is abstract and deliberately maintains the gap between its business level and the IT level, such as in BPEL. BPMN does not specify how any certain Activity should be implemented nor does it require that an
Activity will have an input or an output. Though BPMN does require Data Objects, it does allow to associate Data Objects, as a sub-type of Artifact, to an Activity. Still, BPMN does not define the schema of the Data Object nor does it enforce that an Activity be associated with the same Data Object. For example, the Activity ‘Send item’ can receive any number of Data Objects: football, basketball, and any good stuff.

In Prosero, on the other hand, Data Objects are vital component. They are crucial to the matching process, as they represent semantic information regarding the Activities with which they are associated. For example, once ‘Sent item’ is saved in the repository, associated with the ‘Item’ Data Object as an input and with ‘Tracking number’ as an output, it cannot be associated with any other Data Object. Not only that, Prosero won’t even allow you to change the input/output data type, or any other shared metadata, for that matter.

We can therefore claim that the gap between the business level and the IT level is smaller for Prosero models than it is for BPMN models.

**Exceptions vs. return value**

A fixed data type for an Activity, in Prosero, is also related to the following discussion: When an Activity is unable to return the required Data Object (e.g., ‘Account document’) and cannot return any other Data Object to describe the outcome (e.g., ‘Document not found’), it must throw an Error Event that will contain that data of the outcome.

**Prosero and Activity uniqueness**

In Prosero, an Activity is the same Activity wherever it is used. One could suggest that the Activity “Negotiate with customer” is implemented differently when placed under the role of ‘customer service representative’ and when placed under ‘customer service supervisor’. In Prosero, you must create two different Activities: “Negotiate with customer (CSR)” and “Negotiate with customer (CSS)”.
IDE, Prosero and static verification

Prosero already employs static verification at modeling time by verifying some BPMN constraints, as well as some additional constraints. BPMN constraints include the connections between BPMN elements (Gateways, Activities, Events). Additional constraints include a single Start Event, Activity is not a valid sequence end, and Data Object can be coerced to the data type expected by the Activity.

Like many IDEs, if any of these constraints are not kept, in Prosero, a warning is logged and the misbehaved graphical element is marked with a red icon. This verification warning is easy, thanks to the Eclipse framework.

BPMN and exception handling

In BPMN, an Error Event, which is part of a sequence flow, signifies throwing an exception and an Error Event, attached to an Activity, signifies catching an exception. And that’s it. There is no demand to catch and handle an explicitly thrown exception at any point in the Process. There is no graphical notation to indicate a Sub Process is explicitly ending with an exception.

Prosero and Sub Processes

The Prosero modeling tool has a relatively slow access to referenced Sub Processes. This is as a result of referencing a Process in a remote database and not on your local machine. This limits the possibilities to analyze a referenced Sub Process’ workflow for information such as thrown exceptions — information that, as we noted, does not appear in the imported meta-data.

Modeling routine

To conclude the starting points we come to the most important stop before the solution. This section describes the working routine of a business analyst using Prosero (or any other modeling tool, for that matter). The routine demonstrates all that we have
learned about BPMN, exception handling and Prosero. Understanding the modeling routine of a business analyst and comparing it to the programming routine of a software analyst will give us a clear view of how to improve the modeling routine.

We will use a simple example: a business process for initial stages of software product release. The business analyst routine is roughly as follows:

1. Focus on the happy path (Figure 3.19)
   (a) Place Pool on the canvas
   (b) Place a Lane in the Pool
   (c) Drag the Start Event and the Activities:
      i. Create product build
      ii. Smoke test
      iii. Copy build to staging area
      iv. Issue release document to QA
   (d) ‘Smoke test’ results might indicate a problem — add a Gateway after ‘Smoke test’
   (e) Add End Event
   (f) Add Sequence Flow connectors between the elements

![Figure 3.19: Process: Initial product release — happy path.](image)

2. Complete the alternate path (Figure 3.20)
(a) Drag 2 Activities:
   i. Allocate defects
   ii. Fix defects — this is a Sub Process, a complex one; meaning, good chance for problems. Make a note to check possible problems
(b) Add Sequence Flow connectors between the elements.

![Figure 3.20: Process: Initial product release — alternate path.](image)

3. Consider exceptions (Figure 3.21)
   (a) See note that ‘Fix defects’ has a decent chance to fail. Consider and determine two possible Error Events:
      i. Takes too long to complete
      ii. Unable to fix some defects
   (b) Add Activity ‘Reconsider release requirements’ as part of the exception handling flow
   (c) Consider the notion that QA may also cause a delay in the release. Make a note
   (d) Add Sequence Flow connectors between the elements
   (e) Return to the ‘Issue release document to QA’ Activity
      i. QA might disagree with the release document and reject it How will this manifest itself?
ii. This Activity includes sending a document. The receiving side (QA) might reject the document; meaning, the sent data was rejected.

iii. Add an Error Event for the case QA disagree with the release document and rejects it — attach the Error Event to the Activity.

iv. Connect its exception to ‘Reconsider release requirements’

Figure 3.21: Process: Initial product release — exception path.

The business analyst’s experience is extremely valuable when it comes to deciding how probable is an exception. Our solution will allow him to utilize his experience in the best possible way to improve his work.
Chapter 4
Our solution

What do we have so far? We have business analysts working on a business process diagram, using a modeling tool. The business process contains BPMN Activities — Tasks and Sub-Processes — referencing Activities in a shared repository, the Customer Model Repository. An Activity might result in an exception, an Error Event. But (a) most business analysts don’t consider the exception path; and (b) the modeling tool cannot indicate what Error Event might occur, if any.

Our objective is that the tool will indicate to the business analyst probable occurrences of exceptions, and as a result, the business analyst will consider the exceptions and handle them. Similar to the behavior of IDEs that warn programmers against unhandled exceptions.

Section 4.1 details how we extended BPMN Activity with three attributes: Parameters, Errors, and Handlers; which allow the tool to identify unhandled exceptions. Section 4.2 characterizes three types of this extension, called a Construct: Internal Activity, Send Receive, and Receive Process Reply, each with its own set of Parameters and Errors. Section 4.3 suggests adding Templates to Constructs to allow the business analyst to quickly produce a Sub Process.
4.1 Solution overview

The first step is to acknowledge that due to Prosero architecture it will be unwise to attempt to identify exceptions in referenced Activities. Appendix B details the limitations of BPMN and Prosero and discusses various, relevant solutions.

Our second step, the solution this study examines, includes extending BPMN by adding two attributes to the local Activity’s meta-data (as opposed to the referenced, shared Activity). These attributes, Errors and Handlers, detail the static verification procedure exceptions and exception handlers, so the tool can warn against unhandled Error Events. Warning is a result of an Error item without a matching Handler item.

The idea is simple — the business analyst, as he works on the happy path, will quickly add possible Error Events to the Errors attribute. Later on, the tool will warn against unhandled Error Events and the business analyst will return to the Activity and add the appropriate Event Handlers to the Handlers’ attribute.

To assist the business analyst even further, we have decided the tool should suggest a limited list of possible Error Events, relevant to the Activity. The business analyst will gain the benefits of the recommendation and of faster Error entry. For that, the business analyst must help by classifying the Activity. Classification of a business activity characterize its general behavior\(^1\). From the classification, one could deduce possible run-time exceptions. We hypothesize it is easier, and faster, to consider Activity’s classification than Activity’s exception. Naturally, the tool should also offer a limited list of standard classifications.

4.1.1 Classifications and Error Events

To determine a valid limited list of standard classifications we reviewed a dozen business processes [40, 23, 31] with about one hundred business activities. Table 4.1 provides a sample review of our findings. The sample details all the activity classes we encountered. The complete table can be found in Appendix C.

\(^1\)Others consider classification as the business objective, or, classify by organizational responsibility.
Table 4.1: Business activity classification sample.

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>Classification</th>
<th>Probable exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Order(s)</td>
<td>Query data</td>
<td>Query failed (order not found)</td>
</tr>
<tr>
<td>Approve return manually</td>
<td>Enter data (&quot;approve&quot;); Update data</td>
<td>Update failed (authorization of CSR)</td>
</tr>
<tr>
<td>Schedule for future release</td>
<td>Enter data (release version); Update data</td>
<td>Update rejected (invalid release)</td>
</tr>
<tr>
<td>Notify customer</td>
<td>Send and Receive data; Produce data (reply)</td>
<td>Send failed (unknown contacts)</td>
</tr>
<tr>
<td>Issue release document to QA</td>
<td>Produce data (document); Send data</td>
<td>Send rejected (QA rejected)</td>
</tr>
<tr>
<td>Receive Report State of Accounts</td>
<td>Receive notification</td>
<td>Receive reject (invalid state)</td>
</tr>
<tr>
<td>Release Form Sign-off (Legal Signatories)</td>
<td>Verify data; Enter data (signature)</td>
<td>Verification rejected (refuse to sign)</td>
</tr>
<tr>
<td>Look for RMA number in received box</td>
<td>Perform action; Produce data (number)</td>
<td>Action failed (number not found)</td>
</tr>
<tr>
<td>Perform System Test</td>
<td>Perform complex action; Produce data (results)</td>
<td>Timeout</td>
</tr>
<tr>
<td>Identify discrepancy between requirements and product</td>
<td>Request data; Analyze data; Produce data</td>
<td>Timeout</td>
</tr>
</tbody>
</table>
4.1.2 Classifications \(\rightarrow\) Error Events

We derive the following classifications and their relations to Error Events:

- **Query** \(\rightarrow\) Query failed (data not found)
- **Update** \(\rightarrow\) Update rejected (unauthorized, data constraints)
- **Send**
  - \(\rightarrow\) Send failed (target could not be reached)
  - \(\rightarrow\) Send rejected (data rejected, sender rejected)
- **Receive** \(\rightarrow\) Receive rejected (data rejected, sender rejected)
- **Notification** \(\rightarrow\) Receive rejected
- **Enter data** \(\rightarrow\) (problem with the form or the tool) — unlikely
- **Analyze data** \(\rightarrow\) analysis rejected (analysis concluded a possible problem)
- **Verify** \(\rightarrow\) Verify rejected (concluded dissimilarities)
- **Perform action** \(\rightarrow\) Action failed (mainly technical failures)
- **Complex action** \(\rightarrow\) Timeout

The user should be able to choose from both of these lists. Furthermore, if he will choose a classification, the matching exception will automatically be chosen, as well.

The authorization Error Event is different than the others and brings in to the game a different type of Error Event. More on that in Section 4.1.3.

4.1.3 Additional Error Events by business requirements

When an activity fails due to authorization, it means the underline action never commenced — it was rejected before having the chance to start — rejected because of the identity of the caller. Although, technically, you could write a business process describing the authorization procedure, it is more common to use communication protocol at the BPEL level.

It is a common practice in business process execution to ensure various requirements on the data transfer protocols between Web Services: Security, Authorization, Authentication, Policy, Trust, Privacy, Non-Repudiation, and more. These business requirements can be defined at the BPEL level, but not at the BPMN level. At the BPMN level, you don’t know if the Web Service that will match your Activity supports
Our solution

the required protocol.

A future version of the matcher could take into account such business requirements so we added to this prototype a few business requirements as a test run:

- Security
- Authorization
- Authentication
- Non-Repudiation

These are added as requirements to the Send and Receive activities. Meaning, a suggested exception is Send/Receive Authorization. If I send data to a third party, then I might be told I am not authorized; when my Activity includes receiving data, the sender might not be authorized.

4.1.4 Higher categorization

Now, we have rather long recommendations lists: classifications and requirements — we will unite under ‘Parameters’ — on one side, and Errors on the other side. Therefore, we will split the suggested Parameters and Errors into three higher-level categorization. Each category will have a shorter list. These higher-level categories’ elements will be known as Constructs.

4.2 Constructs

A Construct is a BPMN Activity with the attributes Parameters, Errors, and Handlers. Ideally, these attributes should be the same for all references of the Activity. But due to technical difficulties — system architecture, mentioned in Appendix B — the references are not related.

The divergence between the Constructs is the lists of suggested Parameters and Errors. We identified three high-level business activities, under which we can group
some of the Parameters’ classifications. We will first give a short review and then elaborate on each.

**A basic activity** Whether the activity is ‘analyze data’, ‘update data’, or a physical action (‘Perform action’), the activity has a basic reason to fail. The activity is internal to the organization and will be executed by one of the organization’s Web Services. We call this Construct: Internal Activity (IA).

**A composite activity** A business process, internal to the organization. The process will most likely include a Start Event that receives data — a sequence of Activities, Gateways, Events, and other Constructs, and an End Event that returns a reply. We call this Construct: Receive Process Reply (RPR). Unlike the Internal Activity, which can only indicate a failure of the activity, Receive Process Reply can fail due to any of its underlying Activities or due to business restrictions of its own that reject data.

In many ways, the differences between IA and RPR are the same as the differences between Task and Sub Process.

**Calling an external activity** An Activity, which is known to call a process outside of the organization, or at the very least, outside of the department. Knowing, at design time, that your Task will send data or is waiting for a response, yields some special Errors. Such cases may include Send/Receive rejections and even communication protocol considerations. We call this Construct: Send Receive (SR).

We will now begin an in-depth examination of these three Constructs. For each, we will present its relevant Parameters, its possible Errors, and the connection between the Parameters and Errors. This connection will allow the tool to automatically choose an Error if the user chose the related Parameter.

### 4.2.1 Internal-Activity

• Parameters:
  – Maximal Duration
    * The longest time the Activity is allowed to take.
  – Failure modes (one by default)
    * General failures that lead to exceptions.
  – Resources 1,...,n
    * Resources the Activity is using, such as databases, etc.
  – Relevant external events 1,...,m
    * Expected events to influence the Activity.

• Errors:
  – Time-out
    * The Activity took too long to complete.
  – Failure-Mode 1,...,Failure-Mode n
    * The Activity failed due to Failure-Mode X.
  – Unavailable resource (1,...,n)
    * The Activity failed due to unavailable resource.
  – External event 1,...,External-event m
    * The Activity failed due to an external event.

• Parameters $\rightarrow$ Errors:
  – Maximal Duration $\rightarrow$ Time-out
  – Failure modes $\rightarrow$ Failure-Mode
  – Resources $\rightarrow$ Unavailable resource
  – Relevant external events $\rightarrow$ External event

4.2.2 Send-Receive

• Behavior: Send a Data Object, and receive a reply (optional). An activity which reached beyond the borders of the organization: contacting a customer or a third-party vendor. As an example, see UBL [31] choreography business process in Figure 4.1.
4.2 Constructs

Figure 4.1: Customer Initiated Sourcing — Send Receive.

- Parameters:
  - Maximal Duration
    * The longest time the Activity is allowed to take.
  - Number of retries (default 1)
    * Number of times the Task will attempt to send the data before giving up. I.e., number of permitted failures.
  - Security
    * Is the Task using a security protocol of some sort.
  - Non-repudiation
    * Is the Task using a protocol to ensure the validity of the data.
  - Synchronous/asynchronous
    * Is the send-receive protocol synchronous or not.

- Errors:
  - Time-out
    * The Activity took too long to complete.
  - Send failed
    * Data was not received by the target.
– Send rejected
  * The target received and rejected the data.
– Incompliance with security/non-repudiation
  * Transmission did not comply with the security/non-repudiation settings.

• Parameters —— Errors:
  – Maximal Duration —— Time-out
  – Security —— Incompliance with security
  – Non-repudiation —— Incompliance with non-repudiation

4.2.3 Receive-Process-Reply

• Behavior: The top-level framework for a business process. Receive a request, execute the process that handles the request and return a reply.
• Parameters:
  – All activity arguments for process
    * Arguments the process will use
  – Maximal duration for the reply
    * The longest time the End Even is allowed to take to send the message.
  – Number of retries for the reply
    * Number of times the End Event will attempt to send the reply message before giving up.
  – Security
    * Do receive and reply use a security protocol of some sort.
  – Non-repudiation
    * Do receive and reply use a protocol to ensure the validity of the data.
  – Synchronous/asynchronous
    * Is the reply protocol synchronous or not.
  – Log update required
    * This process will need to access a Log resource and update its activity.
4.2 Constructs

- Commitment level
  * How committed is the provider of the process to the validity of the data it returns.

- Errors:
  - All activity errors for process
  - Time-out reply
    * The End Event took too long to complete, sending the message.
  - Receive rejected: format
    * Improper syntax of the request. The received data is not of the expected format — missing or unexpected fields. The Error can contain either one or more field names.
  - Receive rejected: authentication
    * The sender of the received data, the caller, could not be authenticated.
  - Receive rejected: authorization
    * Unauthorized request. The sender of the received data, the caller, is not authorized to use this process.
  - Incompliance with security/non-repudiation
    * Reply transmission did not comply with the security/non-repudiation settings.
  - Send failed
    * Data was not received by the Reply target, the caller.
  - Send rejected
    * The Reply target, the caller, received and rejected the returned data.

- Parameters → Errors:
  - Activity arguments → Activity errors
  - Maximal Duration → Time-out reply
  - Security → Incompliance with security
  - Non-repudiation → Incompliance with non-repudiation
In the case of Receive-Process-Reply, the Parameter and Errors refer mainly to the Receive and Reply Activities. If we knew the process part is of certain functionality, maybe we could estimate better the possible errors. The world of business processes is complex and diverse. It would be quite difficult, for example, to compare one “Purchase Order” process with another “Purchase Order”. But if we look at a lower level, we can find a very general outline of short processes — simple handling of the received message. “Purchase Order”, for example, is a complex handling. We divided the simple handling into four types:

- **Query** — search for the data in the received message and return the results.
- **Notification** — update certain entries according to received message.
- **Verify** — compare the received message to your records and approve or disapprove its validity.
- **Update** — update your records with the received message and approve its consistency.

And so we add:

- **Parameters:**
  - Process type: query, notification, verify, update.
- **Errors:**
  - Query failed
  - Verify failed
  - Update failed

- **Parameters** $\rightarrow$ **Errors:**
  - Process type Query $\rightarrow$ Query failed
  - Process type Verify $\rightarrow$ Verify failed
  - Process type Update $\rightarrow$ Update failed

Each Error can contain one or more items of failures. Note that Notification does not have an Error. It is not that the Process cannot fail; it cannot fail on account of business Notification, which does not include any processing.
To sum up: We divided all Activities into three categories of Constructs: Internal Activity, Send-Receive, and Receive-Process-Reply. All of which hold Parameters and Errors to which the referenced Activity is, technically, unaware. We also identified four basic types of business processes, aside from the general business process: Query, Notification, Verify, and Update.

Our innovation is giving the business analyst the ability to assign an Activity its Parameters and Errors according to his knowledge. Although we help by suggesting possible Parameter and Errors and associating items in the two lists, we can’t really verify his knowledge. Meaning, we can’t make sure that a Process marked by the business analyst as a Query is not, in fact, an Update, or even a simple Task. We cannot verify the structure of the referenced Sub-Process or the type of the referenced Task. But there is something we can do — we can allow the business analyst to create a new Sub Process, which behaves in the expected manner and is referenced by the Activity. This leads up to the idea of a Template.

4.3 Templates

The next step in our solution arose with three advantages in mind:

1. The first, that the Construct is only a general definition of the behavior. By giving a more concrete implementation of the behavior, we can give a better assessment of the Errors that might occur.
2. The second, since we cannot statically verify the Parameters and Errors of a Construct to match the referenced Activity, it would be beneficial if instead we created a Sub Process that behaves as the Construct expects it to behave.
3. And the third... those business analysts are only human and seldom choose to add Errors and mess around with the Handlers and everything. There is a need for an added value. A Template is that added value which will make their life easier.
Although the idea came from exception types and the need for static verification, it could also be approached from pattern reusability aspects. For example, if you need to verify a variety of documents, for each type of document, you will need a separate Process. But each Process is based on a very similar sequence flow. Meaning, they have the same pattern. You can represent that pattern by a Template.

Template is a new abstraction layer for BPMN Processes. In Prosero, it is an abstract BPD. By abstract, we mean all of its Activities are referencing nothing and all its Data Objects have not been assigned a type. It is a full length Process, from Start Event to End Event, with a specified sequence flow, but without the knowledge of whom are the Participants, which Activities are part of the flow, and which Data Objects will be passed.

We’ll start with the Send-Receive Template, with a detailed demonstration of the concept of Template, and then review the Templates for Receive Process Reply: Update, Approve, Verify, and Notification.

### 4.3.1 Send-Receive

Although the Construct can reference a Task, it can also fit any Process of the pattern seen in Figure 4.2.

![Send-Receive Template with Construct form #1](image)

Figure 4.2: Send-Receive Template with Construct form #1
The Template still has the Parameters and Errors associated with the Send-Receive Construct, as seen in Figure 4.3.

Figure 4.3: Send-Receive Template with Construct form #2

We do not know what the “Send” Task or the “Receive” Task are. Nor do we know what type of Data Object they handle. But we all agree this is the correct sequence of events to send a message and to receive an answer.

Figure 4.4: Send-Receive Template with Construct form #3

Figure 4.4 shows how the Template’s form helps the business analyst to actualize
the Template into a new Process. In this example, the business analyst chooses to add a new Send-Receive Construct referencing a new Send-Receive Template. The Template diagram appears with a form which forces the user to choose concrete Tasks for the abstract Tasks: ‘Send’ and ‘Receive’.

For example, the user can choose that the ‘Send’ Task is actualized by “Send Bill” Task and the ‘Receive’ is actually “Receive Money”. This choice will automatically assign the associated Data Objects with the types matching the actualized Tasks — e.g., “Bill Form” and “Money Order”. Thus the Template becomes a concrete Process, “Billing” — see Figure 4.5. The Process is then added to the database, and the Construct, with which we started, will reference that Process.

Figure 4.5: Send-Receive Template actualized

Now that we understand what the Template is all about, let’s review the other four behaviors we mentioned: Query, Notification, Verify, and Update. For these, our Construct receives a little bonus. Not only can we describe these behaviors with a Template, we can also associate that Template with a list of Parameters and Errors. When choosing one of these Templates for the Receive-Process-Reply Construct, the Parameters and Errors of the Construct will be checked automatically according to Template information.
4.3.2 Query

Description: A query is received. The query does not require the update of the local state, except possibly for logging the fact that it was received and responded to. See Figure 4.6.

1. Receive query
2. Process query
3. Respond with answer

Parameters: (Figure 4.7)  
- Type = Query
- Security
- Authentication
- A/Synchronous

Errors:  
- Send Failed
- Time-out reply
- Receive rejected: authorization
- Query failed

Possible variations: Binding query, where the response is binding to the agent. May require non-repudiation and more involved logging.

Compensation routines: Undo logging; can also be noop.

Examples of concrete processes covered by this pattern:

- Inventory check
- Delivery time check
• RFQ when item is in catalog

4.3.3 Verify-Approve

Figure 4.7: Query Template parameters and errors

Figure 4.8: Verify Template for Receive-Process-Reply
Description: This pattern covers the case where we receive a document, containing data that we need to approve. We check it against our records and if it matches our records, then we send a confirmation and we update our records, if needed, with the fact that we received this document and approved it (assuming we did). If it does not match, then we send a disapproval message. Ideally, it should explain the problem. See Figure 4.8.

1. Receive Document for verification
2. Compare against records
3. If matches records
   (a) Send approval
   (b) Update approval
Else
   (a) Send disapproval
4. End

Parameters:
- Type = Verify
- Security
- Authentication
- A/Synchronous

Errors:
- Receive rejected: authorization
- Receive rejected: format
- Send failed
- Verify failed (due to technical problems; does not imply document is incorrect)

Possible variants: Verify-Approve-List, where the document is a list of items and we allow for partial approval, i.e., where some items are approved and some are not.

Verify-Approve-Amendment, where the document is an amendment to a previously approved document. It seems like this only affects the specifics of the verify process.

Compensation routines: Undo update
Examples of concrete processes covered by this pattern: Remittance advice, and arguably (see next pattern), receive purchase order, receive supply schedule, purchase order amendment, delivery plan change, receive invoice, self-billing invoice, etc. See Figure 4.9.

4.3.4 Notification

Figure 4.9: Notification Template for Receive-Process-Reply, with Construct form #3

Description: This pattern covers the case where we receive a document that contains data that requires us to update some information, but does not require further processing. This is a notification, and we cannot disapprove.

1. Receive Document for update
2. Perform update

Parameters:  
- Type = Notification

Errors:
- Receive rejected: authorization
- Receive rejected: format
- Update time-out
Compensation routines: Undo update

Examples of concrete processes covered by this pattern: Notification of shipment time; Notification of price change for a catalog.

4.3.5 Update-Approve

Figure 4.10: Update Template for Receive-Process-Reply

Description: Much like Verify-Approve, but where the message requires an update to our state. The main issue is whether the update has additional ramifications. See Figure 4.10.

1. Receive Document for update
2. Evaluate update
3. If acceptable
   (a) Send approval
   (b) Update
Else
Our solution

(a) Send disapproval
(b) Log disapproval if needed

4. End

**Parameters:**
- Type = Update
- Security
- Authentication
- A/Synchronous

**Errors:**
- Receive rejected: authorization
- Receive rejected: format
- Send failed
- Update failed (not the same as disapproval)

**Possible variants:** Again, the case of the document being a list of items such that items may be processed independently, and some may succeed and some may fail (e.g., in/out of stock).

**Compensation routines:** Undo update

**Examples of concrete processes covered by this pattern:** Receive purchase order; cancel purchase order; amend purchase order; receive delivery plan.

Those with keen eye sight might have noticed the “Compensation routines” headline, which we did not mention before. As mentioned in the background, Compensation is a specific business activity to undo the changes the process made until reaching some failure. The usage of BPMN Compensation instead of Error is discussed further in ”Future Work”. However, for the present, when designing a Template of a certain functionality, you can add to the Template a routine to compensate for specific errors.

### 4.4 Afterthoughts

The concept of Construct is to extend BPMN Activity with the attributes Parameters, Errors, and Handlers. The Parameters of an Activity are certain settings that describe
the Activity’s behavior at run-time. From these Parameters, we deduced possible run-
time exceptions. So, Parameters influence the Errors, and Errors are compared with
Handlers to verify exception handling.

To further help the business analyst, we added expert-based recommendations for
the Parameters and Errors. Our solution is based on the premise that it is easier, for
the business analyst, to identify the classification of an Activity than its possible Error
Events. This produced the Constructs Internal-Activity, Send-Receive, and Receive-
Process-Reply, which the business analyst can drag from the pallet instead of a standard
Activity.

As a supplemental feature, which is not directly related to exception handling,
we coupled some of the Parameters’ classification with an abstract level BPD — a
Template. This is significant, especially for RPR, which is very miscellaneous and
usually complex.

The framework we added to the modeling tool is more important than the classifi-
cations we created. While the three Constructs are hard coded and cannot be replaced,
our expert advice for Parameters and Errors can be edited by the business architect.
For instance, if he would like to add a new behavior classification (Template) to RPR
—including new Parameters, Errors and their relations — then it should be quite pos-
sible. The same goes for removing or changing existing Constructs’ Parameters, Errors,
and Templates.
Chapter 5

Implementation

First we will understand the details behind business process diagrams. We will note how a BPD appears when using Activities and when using Constructs. Then, then we will review the Construct’s and Template’s wizard functionality and how their information is retrieved from the repository.

5.1 Extending BPMN

5.1.1 Normal Activity

In Chapter 3 we reviewed Prosero from user-experience perspective. This made sense since the problem we wanted to solve was in the user-experience domain — the modeling tool should help the user consider possible exceptions in the business process. Our solution to the problem, the Constructs and the Templates, will replace the conventional procedure of adding an Activity. The next step is to implement the Construct, the Templates and the static verification procedure that will look for unhandled exceptions. For that, we must review Prosero again. This time from the consistency perspective: How are BPDs saved? How does an Activity, be it Task or Sub-Process, appears in the BPD representation?

There is no restriction on the file format of a BPMN diagram. Usually, the BPMN
structure of the Process is saved in an XML format. Here is an example how Prosero used to save a BPD. We see the “Don’t worry, be happy” diagram, in Figure 5.1, followed by its XML representation.

Figure 5.1: BPD: Don’t worry, Be Happy

```xml
<bpmn:BPD_MODEL xmlns:bpmn="bpmn" name="Dont Worry Be Happy">
    <pools>
        <lanes>
            <elements xsi:type="bpmn:StartEvent"/>
            <elements xsi:type="bpmn:EndEvent"/>
            <elements xsi:type="bpmn:IntermediateEvent" name="UnHappy" Trigger="Error"/>
            <elements xsi:type="bpmn:Task" name="Don’t Worry"/>
            <elements xsi:type="bpmn:IntermediateEvent" name="IsWorried" Trigger="Error" Target="//@pools.0/@lanes.0/@elements.3"/>
            <elements xsi:type="bpmn:SubProcess" name="Be Happy" activityType="Subprocess"/>
            <elements xsi:type="bpmn:SequenceFlow" source="//@pools.0/@lanes.0/@elements.0" target="//@pools.0/@lanes.0/@elements.3"/>
            <elements xsi:type="bpmn:SequenceFlow" source="//@pools.0/@lanes.0/@elements.3" target="//@pools.0/@lanes.0/@elements.4"/>
            <elements xsi:type="bpmn:SequenceFlow" source="//@pools.0/@lanes.0/@elements.4" target="//@pools.0/@lanes.0/@elements.1"/>
            <elements xsi:type="bpmn:SequenceFlow" source="//@pools.0/@lanes.0/@elements.3/@eventHandler.0" target="//@pools.0/@lanes.0/@elements.2" name="catch and throw"/>
        </lanes>
    </pools>
</bpmn:BPD_MODEL>
```
5.1.2 Construct

Our first step is the Construct by which we extend a BPMN Activity, assigning it Parameters and Errors fields. Remember this assignment is only local. So, we add two sub-elements to the Activity XML-schema: Parameters and Errors. Technically, for the purpose of exception handling verification, these two fields could be enough. Nevertheless, for our own comfort — to ease our work on the prototype — we also added the eventHandler sub-element. This will make the verification procedure much simpler, as it only needs to compare the Errors field with the eventHandlers field. Here is an example for the new XML format representing the BPD in Figure 5.1:

```xml
<bpmn:BPD_MODEL xmlns:bpmn="bpmn" name="Dont Worry Be Happy">
    <pools>
        <lanes>
            <elements xsi:type="bpmn:StartEvent"/>
            <elements xsi:type="bpmn:EndEvent"/>
            <elements xsi:type="bpmn:IntermediateEvent" name="UnHappy" Trigger="Error"/>
            <elements xsi:type="bpmn:Task" name="Don’t Worry">
                <eventHandler Trigger="Error" Target="/@pools.0/@lanes.0/@elements.3" name="IsWorried"/>
                <errors name="IsWorried"/>
                <errors name="Timeout"/>
            </elements>
            <elements xsi:type="bpmn:SubProcess" activityType="Subprocess" name="Be Happy"/>
            <elements xsi:type="bpmn:SequenceFlow" source="/@pools.0/@lanes.0/@elements.0" target="/@pools.0/@lanes.0/@elements.3"/>
            <elements xsi:type="bpmn:SequenceFlow" source="/@pools.0/@lanes.0/@elements.3" target="/@pools.0/@lanes.0/@elements.4"/>
            <elements xsi:type="bpmn:SequenceFlow" source="/@pools.0/@lanes.0/@elements.4" target="/@pools.0/@lanes.0/@elements.1"/>
        </lanes>
    </pools>
</bpmn:BPD_MODEL>
```
The main difference from what you would expect from BPMN XML format is the fact that the exception handlers, e.g., ‘UnHappy’ Intermediate Event, are no longer BPMN elements in the main context; they are now eventHandler elements, under the relevant Activity element. The positive in our choice to break the BPMN element pattern is the fact that it is now very easy to see which are the handlers — both if you are the kind of person who likes to read XML and for writing a much simpler XPath. Simple XPath is beneficial for the BPMN to BPEL transformation that must distinguish between throwing Error Events and catching Error Events.

The observant reader should note that the Errors attribute is not relevant for the transformation, but only to the static verification procedure. (I trust the reader also noted that according to the new XML there is an unhandled Error Event — ‘Timeout’. ‘IsWorried’ Error has it’s matching handler).

To understand how much simpler are the XPaths, you can compare them to the XPaths, referring the normal XML representation, in Appendix B. The XPath statements to analyze this Process:

- To get all Handlers for an Activity, $activity:
  
  XPath handlersXpath = ./elements/eventHandler

- To get all Error Events for a Activity, $activity:
  
  XPath activityErrorsXpath = ./elements/error
The pseudo-code procedure to get unhandled Error Events is this:

\[
\text{unhandledErrors} \leftarrow \emptyset;
\]

\[
\text{FOREACH activity IN process DO}
\]

\[
\text{activityErrors} \leftarrow \text{read Errors}; \quad // \text{see XPath}
\]

\[
\text{activityHandlers} \leftarrow \text{read Handlers}; \quad // \text{see XPath}
\]

\[
\text{unhandledActivityErrors} \leftarrow \text{activityErrors} - \text{activityHandlers};
\]

\[
\text{unhandledErrors} \leftarrow \text{unhandledErrors} \cup \text{unhandledActivityErrors};
\]

\[
\text{return unhandledErrors};
\]

This entire pseudo-code can be written as a very long XPath query that will return unhandled Intermediate Error Events elements. The query will not include a call to any external information (i.e., use ‘document()’), which greatly simplifies it.

Since Prosero’s modeling tool is Eclipse based, it comes with a build-in OCL checker. OCL stands for Object Constraint Language. A language in which you can define relationship constraints between objects. In our case, BPMN elements. Therefore, instead of XPath query, we will use OCL rules for our verification. And if the OCL checker will find an Activity that doesn’t pass the verification, i.e., has an unhandled exception, it will highlight it and raise an error message.

That is how we fulfilled our requirement to give immediate verification message on unhandled exceptions. Which leaves the problem of populating the fields. We saw, through the Section 4.3, the forms in which the business analyst fills out the information. We will now elaborate on the wizard which produces these forms.
5.2 Wizards and Constructs and Templates, oh my

The wizard is initiated when a Construct is added to the diagram. There are three, hard coded, Construct possibilities: Internal Activity, Send Receive, Receive Process Reply. The wizard opens the first form, choosing a Template.

![Figure 5.2: Wizard: choose Template](image)

Each Construct can be added as a Task, with no Template, or as a Sub-Process, with the Template representing the referenced Process. For this to work, the wizard connects to the server and requests a list of templates. It receives a complete list of Constructs and their associated Templates. Although the list of Constructs is set, the list of Templates is taken from the RMR and will change according to the data in the repository. The default association is the one reviewed in the Section 4.3; and the
business architect can modify it.

Due to the prototype deadline we were not able to implement the ability to modify the Parameters and Errors recommendation. Currently the architect can only modify the association of Templates with the existing Parameters and Errors.

According to chosen Construct, the wizard displays in the form the list of associated Templates. The business analyst can choose one or continue as a regular Construct.

If he didn’t pick a Template, the wizard will eventually add a Task to the diagram. In the next steps the wizard will produce a Parameters and Errors selection form, Figure 5.3, and a Choose Task form, Figure 5.4.

Figure 5.3: Wizard: select parameters and errors

Figure 5.4: Wizard: choose Task

The last form, Figure 5.4, for choosing the Task the Construct extends, is directly
related to the standard form of importing a Task\(^1\).

If the business analyst picked a Template, in the first form, then although the forms seem similar, the background behavior is different. The modeling tool will contact the repository and requests the Template file. If the business analyst will look at the editor panel, behind the form’s window, he will notice a new BPD loaded. This is the Template’s abstract business process — all the Activities and Data Objects are empty. Still, the wizard presents the second form, of selecting Parameters and Errors in the same manner. A mild difference is that some of the checkboxes are already checked, according to the Template meta-data.

The last form, e.g., Figure 5.5, is quite different, since we are creating a Sub Process and not a Task.

![Figure 5.5: Wizard: choose Activities](image)

The wizard prepare a field for naming this Sub Process, and populated the rest of this form with the empty Activities from the Template file. For the user to actualize an Activity, which is a Sub Process, the wizard will call the “Import Sub Process” form.

\(^1\)This is a bug, since it prevents adding a new Task
To *actualize* an Activity, which is a Task, the wizard will call the “Import Task” form. The existing import forms’ logic will make sure that consistency of selected Activities is kept. Mainly, that the Data Objects, automatically assigned by a selected Activity, are consistent with the other Activities Input/Output requirements. Again, if while selecting Activities you look at the editor panel you will see the empty Activities and Data Object being populated according to your selections.

When clicking the Finish button, the wizard will first save the once Template, now fully grounded/actualized Process, as a BPD file with its given name, send this file to the CMR as a new Process, get the ID of that Process, and add to the Process, with which we started, a Sub Process referencing the Process we just created, by its ID.

**Example:**  In an ‘Order’ Process, we want to add a Send-Receive for a ‘Credit Check’.

1. We choose the Send Receive Construct; the wizard comes up.
2. We select the Send-Receive Template
3. Leave the Errors as is
4. Give the name “Credit Check” and select the two required Tasks.
5. A new Process, called “Credit Check”, is added to the repository, and in the ‘Order’ Process a Sub Process referencing ‘Credit Check’ is added.
5.3 Who handles the Handlers

The wizard populates the Parameters and Errors attributes, but how is the Handlers attribute populated? To provide a ‘proof of concept’, as early as possible, we left it to the user to populate the Handlers. By clicking the attribute, the user has a form in which appear all the Error Events in the Process. The user can select which of those are Handlers for the Activity’s exceptions.

Ideally, the Handlers should have been automatically populated according to the graphical location of the Error Events in relation to the Activity, as seen in Figure 5.6.

![Figure 5.6: Graphical notation: exception boundary.](image)

Maybe in the next version.

5.4 Summary

We saw how modifying the Activity meta-data is expressed in the XML format of the BPD: parameter, error, and eventHandler child elements. We explained how this simplify the transformation of the extended BPMN into BPEL.

Then, we reviewed the operation of the wizard which: (a) offers Parameters and Errors recommendations to the user; and (b) helps the user create a concrete BPD from a Template.

It is now time to take this fully operational Prosero system to the test. In the next chapters we will prepare and execute an experiment to validate the usability of the Constructs and Templates.
Chapter 6

Methodology

Chapters 2 and 3 discussed our place in the world of business processes and the dangers of unhandled exceptions. Chapters 4 and 5 presented our goal, to add to Prosero’s features arsenal the ability to identify unhandled, business level Error Events (exceptions). This new feature comes to help the user of the tool, the business analyst. The question is, in what ways will it help? Will the business analyst work faster? Better? Safer? That is the purpose of the Methodology chapter — to hypothesize the possible benefits of our solution in regards to a variety of use cases.

6.1 Experiment Design

In the experiment, a business analyst receives a business process diagram, modeled using a regular modeling tool. He then re-models the BPD, using the tool enhanced with the new features, the Constructs and the Templates, with their recommendations for Error Events.

Now comes the tricky part, how to evaluate the results. Both original BPD and the newly created BPD will be compared with an expert opinion of the business process. The expert will determine what are the expected Error Events that should be handled. Using the expert’s evaluation we can see if using the enhanced tool yields better results than using a regular tool, which produced the original diagram.
The first and foremost important result is the accuracy of the specified Error Events — are there missing/redundant exception handlers. A secondary result, though still valuable, is the user-experience of the business analyst using the new features. Specifically, how much time does it take to create a business process with and without the added features.

Section 6.2.1 describes our testing group — the type of business processes we will include in our experiment. For each business process input we will get a user’s result and two objective opinions — one for the original and one for the result. Section 6.2.2 demonstrates how to use the objective opinions to evaluate the user’s results. Section 6.3 concludes with our hypothesis: for which type of business processes we expect the best results.

6.2 Experimental Settings

6.2.1 Test-group parameters

We are interested in what type of business processes we should include as input for our experiment.

The major concept that leads us through modeling time, with respect to exception handling, is the likelihood of exceptions at execution time. Although we cannot determine a likely exception without prior knowledge, the business analyst has that knowledge, which means, in order to evaluate our solution, we are interested in the likelihood of the business analyst missing a probable exception. A lesser interest is the likelihood of the business analyst to add redundant/irrelevant exception handling. To put it more formally:

- Likelihood of exceptions
  
  - Complex Activity: A complex Activity (Task or Sub-Process) has higher chances of a failure at execution time
  
  (You don’t need Murphy for that)
– Complex Data Object: An Activity which deals with complex/large Data-Objects, as input or output, has higher chances of failing at execution time

– Business level exceptions: a Sub-Process that explicitly throws an Error Event in one of its alternate paths

• Likelihood of missing an exception

1. Likely exception: the Process must have a likely exception for the business analyst to miss

2. Complex Process: while working on a long/complex Process, the business analyst has higher chances of missing an exception

• Repeating diagram patterns

So, our testing-group will include complex business processes with the three types of causes for a likely exception. The third item, Repeating pattern, is less concerned with exception handling; rather, it deals with the coincidental benefit that our solution includes Templates.

The next step is to define a business process level of complexity and Data Object level of complexity.

In search of complexity

Borysowich [6] categorizes BPs into four levels: Simple, Medium, Complex, and Very Complex. However, he declares there is no “rule of thumb” for such estimation, and indeed, he offers a rather ambiguous definition. [17] gives a thorough review of different complexity metrics for general BPMs: size of the model (activities and gateways), number of possible control flows, nesting depth, comprehensiveness (cognitive weight), anti-patterns, and modularization. The review concludes that “due to the number of factors that contribute to the complexity of a BPM, we cannot identify a single metric that measures all aspects of a model’s complexity”; and suggests various metrics associations. Although I find it a comprehensive study, the use of metrics can only tell
if one model is more complex than another (and so more prone to exceptions). It does not state when a given model is complex in the eyes of a business analyst (a human).

The ‘size of the model’ metric is sufficient to assess business processes that will “overweight” the business analyst, and is also appropriate for Data Object complexity.

[7] suggests the ‘size of the model’ should include not only Activities, but also Gateways (“join and split connectors”). We will use this variant of the metric to give statistically-based definitions for three levels of complexity: Simple, Medium, and Complex.

**Complex** — Number of elements is higher than the average.

**Medium** — Number of elements is equal to the average.

**Simple** — Number of elements is less than the average.

The definition is rather rough, but adequate for this experiment, as our hypothesis will refer mainly to complex business processes.

**BP complexity**

We reviewed hundreds of business process workflows, some BPMN and some of other notation, to measure the expressive characterization. The best domain-specific examples were from IBM [23] (Web-Sphere), Defense Department [1] (OV-6c), and Tibco [40] (BPMN).

- Max. number of Activities: 19
- Max. number of Sub-Processes: 9
- Max. number of Gateways: 9
- Max. number of Lanes: 3
• Avg. number of Activities: 5
• Avg. number of Sub-Processes: 1
• Avg. number of Gateways: 2
• Avg. number of Lanes: 1

DO Complexity

We counted 113 Data-Objects in UBL [31] with:

• Max. number of fields in an object: 35
• Avg. number of fields in an object: 9

6.2.2 Test outcome parameters

There are two categories by which we will evaluate our (smashing) success or (frelling) failure: the robustness of the resulting BPDs and the amount of work the business analyst must invest to complete the BPDs.

Diagram robustness

• Missed exceptions
• Irrelevant exceptions

In other words, we are interested in the recall (REC), accuracy (ACC), precision (PRE), and F-Measure (F-M) of the originals and the of outcomes, compared to an expert’s advice. Using expert’s advice we can calculate the number of true-positive (TP), true-negative (TN), false-positive (FP), and false-negative (FN) exception handlers in the outcome diagrams.

• Recall = (TP) / (TP + FN)
• Accuracy = (TP + TN) / (TP + TN + FP + FN)
• Precision = (TP) / (TP + FP)
• F-measure = (2 * Precision * Recall) / (Precision + Recall)
User experience

- Time to complete
- Ease of modeling

The work characterization, like many user experience evaluations, is difficult to measure. While we will focus on the objective number of minutes it took to complete the BPD and on the number of checkboxes the user had to check and un-check, we will also ask the user to rate the experience of using the wizards compared to the normal drag-and-drop modeling, without assigning an objective value.

6.2.3 Method parameters

These parameters will actually be fixed in our experiment. We mention them since there are multiple ways to perform static verification to identify unhandled exceptions:

- The verification can take place on the local machine, by the tool, or on the remote repository, by a server procedure.

- The result, and possible warning, can be immediate or daily.

- Exception handling related fields, list of Error Events, and list of Event Handlers, can be dynamic or static. They can be populated automatically by the procedure or manually by the business analyst.

The focus of this study is to supply the business analyst with immediate warnings on the local machine. The architecture of the tool and time limit forced us to implement static fields which will be manually populated.

For more information on possible static verification implementations, consult Appendix B.
6.3 Research Hypothesis

To consider the variants of our experiment, we must first be reminded of what it is we hope to accomplish. We want to help the business analyst consider exceptions as he models the happy path. This is a main part of our explicit target to produce more robust BPDs. A more general accomplishment will be to ease the work of the business analyst. The Constructs addresses these two challenges, while the Templates should mainly help with minor BPDs of the same pattern.

6.3.1 Primary hypothesis — using Constructs

We will hypothesize that by using our enhancement, suggesting a limited list of Parameters and Errors, the business analyst will pause to consider classification and/or exceptions. But the pause will be momentary, just to mark a reminder on that Activity, and the tool will later force him to handle that remark on a possible exception. And so this method will not consume much time and will ensure that probable exceptions will be handled (i.e., won’t be overlooked). To put it formally:

- The time to model diagram A using the enhanced tool should be less than 150% of the time using the regular tool. Time, in the case of the enhanced tool, is influenced by the wizards and their checkboxes.

- The accuracy (ACC) and recall (REC) of a diagram modeled using the enhanced tool should be greater than those using the regular tool.

On the other hand, our greatest (hypothesized) fear is that a business analyst will use our help too much. Meaning, he will attempt to handle a great number of Error Events. In which case, the business process is both less readable and includes unnecessary exception flows.

- The precision (PRE) of a diagram modeled using the enhanced tool might be lesser than those using the regular tool.
Table 6.1: Hypothesis: expected results — enhanced tool vs. regular tool

<table>
<thead>
<tr>
<th>Complexity level</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REC</td>
</tr>
<tr>
<td>Simple</td>
<td>=</td>
</tr>
<tr>
<td>Medium</td>
<td>&gt;</td>
</tr>
<tr>
<td>Complex</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

We conclude these hypothesis in Table 6.1. If these expected results will prove to be correct, then we will conclude that our enhancement is helpful mainly in the case of complex business processes. On the other hand, if we will discover disappointing results in some of the business process types, we will recommend not to use the enhancement for those types.

6.3.2 Secondary hypothesis — using Templates

It is reasonable to hypothesize that if many of the business processes follow the same pattern, then using a Template, based on the pattern, should speed up the work. This is not part of our main scope and we will check it in an ‘offhand’ manner. It is worth our attention, in the case results showing modeling time, using the enhanced tool, is double than when using the regular tool. If that will be the case, we can give an estimate using the time saved by the Templates.
Chapter 7

Experiment

In Chapter 6 we reviewed the experimental settings and hypothesis. In Section 7.1 we will review the different modeling routines that will be performed in this experiment. Section 7.2 lists the BPDs that fit the hypothesis. These are our test group — the input for the business analyst. Results and outcomes are in Section 7.3 and discussion of the results is in Section 7.4.

7.1 Modeling routines

To consider the experimental process, we must first understand the difference between modeling as we know it, cases (A) and (C), and the new and improved modeling features our solution offers, cases (B) and (D).
(A) Adding an Activity

A1. Drag and drop an Activity from the Pallet to the canvas

A2. Give name to the Activity (can import existing Activity)

(B) Adding a Construct

B1. Drag and drop a Construct from the Pallet to the canvas

B2. Skip Template

B3. Consider Parameters and Errors

B4. Give name to the Activity (can import existing Activity)

(C) Adding a ‘patterned’ Sub Process

C1. Create and export the required Process

C2. Drag and drop a Sub-Process from the Pallet to the canvas

C3. Import the Process you created

(D) Adding a Template

D1. Drag and drop a Construct from the Pallet to the canvas

D2. Select Template

D3. Consider Parameters and Errors

D4. Import existing Activities to concertize the Template

D5. Give name to the Sub Process

In all cases, (A), (B), (C), and (D), once the business analyst is done with the Activity he can either continue with the happy path or consider an exception flow for the Activity. But for cases (B) and (D), if he added an error, then sooner or later he will have to return to the Activity and attach an Error Event.

We know it may cost the business analyst two to five extra seconds to use a Construct/Template instead of a regular Activity. The experiment will determine when it is beneficial to drag-and-drop a Construct or a Template over a regular Activity.
Table 7.1: Experiment input

<table>
<thead>
<tr>
<th>Complexity level</th>
<th>Business process names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Deliver item, Return goods, Issue credit</td>
</tr>
<tr>
<td>Medium</td>
<td>Receive returned goods, Perform disposition, Administer return request</td>
</tr>
<tr>
<td>Complex</td>
<td>Update return, Approve return, HR Hiring, Software defect management, Software release product</td>
</tr>
</tbody>
</table>

7.2 Business processes

The scenario example is as follows:

in the great city of Be’er Sheva, capital of the Negev, at the heart of Israel, stood the WMYPC software and hardware company. WMYPC grew and extended in recent years and now includes large production, testing and CRM teams. To improve the performance of the company, the management has decided to invest in describing their business processes in BPMN.

The experiment will use the business processes detailed in Table 7.1. The business processes are based on TIBCO [40], Smartdraw [37], and IBM [23] modeling tools’ samples.

Some of these business processes include complex Data Objects, such as ‘Defect’ (Date, Owners, Description, Status, ...) and ‘Release Form’ (multiple approval stages).

The complete list of the input business processes, their activities, and their output of exception handling choices can be found in Appendix D. We will now review and discuss the final outcomes.

---

1 Instead of exception flow, it uses forms return value and multiple alternate paths.
7.3 Results

Table 7.2 shows a few activities’ results, as a sample from Appendix D. These activities are from the ‘Defect management’ business process. From the results we can tell that according to the expert’s opinion, the process has likely exceptions.

Table 7.4 reviews the entire outcome presented by business process and Table 7.5 by process category. Finally, in Table 7.6, we compare the results of using the enhanced Prosero modeling with using the regular Prosero.

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triage defects</td>
<td>8 ; 2 ; x</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Assign defect</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provide extra information</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>TO</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Assess acceptability to business</td>
<td>1 ; 9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fix Code and Unit Test</td>
<td>9 ; 2 ; 3 ; x</td>
<td>3</td>
<td>1</td>
<td>AF</td>
<td>AF</td>
<td></td>
</tr>
</tbody>
</table>

Two minor results, which are not mentioned in any of the tables, are:

- The number of checkboxes, the business analyst had to uncheck, is 113. Calculated from 149 Error checkboxes, checked automatically, from the Parameters, and 32 Errors the business analyst selected. Unchecking 113 is partly responsible for the time difference, which is in favor of the regular tool.
- None of the Sub Processes matched any of our Templates’ patterns. This cast a serious doubt over the Templates usability.
Table 7.4: Results per process

<table>
<thead>
<tr>
<th>Business process</th>
<th>Source</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple business processes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliver item</td>
<td>original</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Deliver item</td>
<td>enhanced</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Return goods</td>
<td>original</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Return goods</td>
<td>enhanced</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Issue credit</td>
<td>original</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Issue credit</td>
<td>enhanced</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td><strong>Medium business processes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive returned goods</td>
<td>original</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Receive returned goods</td>
<td>enhanced</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>Perform disposition</td>
<td>original</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Perform disposition</td>
<td>enhanced</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Administer return request</td>
<td>original</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>Administer return request</td>
<td>enhanced</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td><strong>Complex business processes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update return</td>
<td>original</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Update return</td>
<td>enhanced</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>Approve return</td>
<td>original</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>Approve return</td>
<td>enhanced</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>Software defect management</td>
<td>original</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>Software defect management</td>
<td>enhanced</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>44</td>
</tr>
<tr>
<td>Software release product</td>
<td>original</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>Software release product</td>
<td>enhanced</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>HR Hiring</td>
<td>original</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>HR Hiring</td>
<td>enhanced</td>
<td>1</td>
<td>14</td>
<td>1</td>
<td>2</td>
<td>55</td>
</tr>
</tbody>
</table>
### Table 7.5: Results per process category

<table>
<thead>
<tr>
<th>Complexity level</th>
<th>Source</th>
<th>REC %</th>
<th>ACC %</th>
<th>PRE %</th>
<th>F-M %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>original</td>
<td>16.67</td>
<td>55.56</td>
<td>33.33</td>
<td>22.22</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>50.00</td>
<td>58.33</td>
<td>38.89</td>
<td>43.75</td>
</tr>
<tr>
<td>Medium</td>
<td>original</td>
<td>50.00</td>
<td>84.13</td>
<td>66.67</td>
<td>57.14</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>83.33</td>
<td>82.14</td>
<td>66.67</td>
<td>74.07</td>
</tr>
<tr>
<td>Complex</td>
<td>original</td>
<td>16.89</td>
<td>64.48</td>
<td>50.00</td>
<td>25.25</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>54.10</td>
<td>73.07</td>
<td>50.00</td>
<td>51.97</td>
</tr>
<tr>
<td>Total</td>
<td>original</td>
<td>25.86</td>
<td>67.40</td>
<td>50.00</td>
<td>34.09</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>60.95</td>
<td>71.53</td>
<td>60.00</td>
<td>60.47</td>
</tr>
</tbody>
</table>

### Table 7.6: Actual results — enhanced tool vs. regular tool (original)

<table>
<thead>
<tr>
<th>Complexity level</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REC</td>
</tr>
<tr>
<td>Simple</td>
<td>≫</td>
</tr>
<tr>
<td>Medium</td>
<td>≫</td>
</tr>
<tr>
<td>Complex</td>
<td>≫</td>
</tr>
</tbody>
</table>
7.4 Discussion

Unlike the Conclusions chapter, Chapter 8, this section is a concise assessment of the results.

We know it may cost the business analyst two to five extra seconds to use a Construct/Template instead of a regular Activity. The question is, when is it beneficial to drag-and-drop and Construct or a Template over a regular Activity? We were aiming for Complex business processes, but it seems that it is also beneficial in Simple business processes.

Although the recall results are better than we had expected, we expected better for the rest. Our conclusions will discuss the unavoidable fact that while the enhanced tool identify more probable exceptions than the regular tool, it doesn’t hold enough advantage to make the business analyst use it.

7.4.1 The Good

Recall The resulting recall was outstanding — on average, 61% for the enhanced tool versus 26% for the regular. Meaning, the tool’s suggestions help to identify probable exceptions in the business process.

Accuracy The accuracy is very good for the Complex scenarios — 73% for the enhanced tool versus 64% for the regular. Similar to our hypothesis.

Precision Precision is rather good in for the Simple scenarios, with about 5% in favor of the enhanced tool. As for the Medium and Complex scenarios, we expected much worse and were surprised for the better, with similar results on both tools.
7.4 Discussion

7.4.2 The Bad

Precision  The fact, the precision and F-measure are in favor of the enhanced tool, on average, is suspicious. We would have expected many more False-Positive cases, i.e., redundant exception handling, and the lack of them makes us wonder whether the expert was too generous with his exceptions or was the business analyst too careful with his.

Accuracy  While the accuracy was not as good as expected, for the Medium case, 82% for the enhanced and 84% for the regular, it is not much lower either. This may indicate that the business analyst should use the Constructs only for Complex business processes, which received 73%, for the enhanced tool, and 64% for the regular tool.

Not easy  The wizards are helpful but not user-friendly enough. They take too much time to handle, even after learning their behavior. By “too much time” we mean only a few seconds; but that’s a few seconds per Activity.

Repetitive  Every local reference, to a certain Activity, would have to locally declare, again and again, that the Sub Process may throw this Error Event. The fact that the Activity Errors attribute is not saved in a shared location and automatically populated is extremely aggravating, especially since it is difficult to populate. Therefore I conclude that although our solution is the best possible answer to our requirements, it is not user friendly enough to be used often. Still, used infrequently, it is better than not used at all.

7.4.3 and the Ugly

Adding Exception Flow to the Normal Flow and to the Alternate Flow overloads the diagram, making it less readable. This is likely to happen in high level business processes which may receive accumulating Error Events from their Sub Processes.
Chapter 8

Conclusions and Future Work

So, what did we have here? We were some jolly programmers, who wanted to make some business analysts jolly as well. Essentially that was the motivation. To bestow upon others what was bestowed upon us. We started by wanting to give the business analyst the same help that software engineers get from their tools — warnings on unhandled exceptions. We ended up providing a feature of which software engineers could only dream — suggesting suitable exceptions.

We overviewed the current status of BPMN and the modeling tools that employ it. We identified that since BPMN specifications do not dictate that an exception must be handled, none of the tools force its user to handle exceptions. Nevertheless, there is a consensus that exception handling is imperative, for robust business processes — as evident from the many attempts to handle exceptions automatically at execution time [33, 2, 39, 8, ?]. But we saw no real attempt to enforce exception handling in modeling time.

Exception handling enforcement strategies have developed and matured in most programming languages. Development environments provide outstanding support for this enforcement by static verification. We explored the notion of transferring the enforcement strategies to the world of business processes.

Our goal was to create an expert tool tool will be a valuable assistance to the business analyst, and his clients, by identifying, in early stages, possible exceptions, at
the business analysis and characterization level.

Prosero’s modeling tool is based on Eclipse framework, which is more than adequate to support handling enforcement; and Prosero’s data is flexible enough to allow additional characterization of the business activities.

In Prosero, we implemented our solution, the Construct, and a secondary feature, the Template. The Construct is an extension to BPMN Activity, adding the attributes Parameters, Errors and Handlers. Comparing Errors and Handlers enables the modeling tool to identify unhandled exceptions. It is the responsibility of the user to populate the attributes. The modeling tool offers a wizard that attempts to help as much as possible by suggesting short lists of high level Parameters and Errors. The suggested Parameters are activity classifications and requirements, and the Errors are the related (probable) exceptions.

We hypothesized that our enhanced modeling tool will work best for complex business processes; as the business analyst is more likely to overlook a probable exception, when modeling a long sequence flow. We examined this hypothesis by modeling several business processes, of different complexity levels, using the enhanced tool. We then compared our outcome BPDs to the same business processes, modeled by an expert. The purpose of this experiment — to verify that BPDs, modeled using the enhanced tool, are closer to the expert’s BPDs than BPDs, modeled by a regular tool.

And now, the conclusion...

8.1 Our solution

Like many things in our world, the solution has its pluses and minuses. Unlike many theses in our world, it seems that the disadvantages, of user experience, outweigh the advantages, of safer business process.
8.1.1 Robust BPMN

There is no denial that using the enhanced tool produces more accurate business processes, according to an expert’s opinion. That is, most of the exceptions, that the expert considered probable — are handled in the resulting business process. Unlike the case of the regular tool, in which very few exceptions are handled.

If we consider the expert’s model to be robust — i.e., has lesser chance of failing during execution — “covers all its bases” — then, the enhanced tool assists in producing robust business processes.

8.1.2 Semantic Web Services

Our addition of the Parameter and Error attributes could have a most rewarding affect on the matching procedure, between an abstract Activity and an actual Web Service. Until now, the matching procedure used the name of the Activity and its input/output data type, to offer appropriate matches. Now, it can also give preference to a Web Service that actually throws a specified Error Event, over a Web Service that throws an unknown Error Event. The matching procedure could also consider the protocol requirement parameters.

Since the matching process is Prosero’s strongest advantage, the business analyst may take the time to specify Activity’s Parameters. Furthermore, as much as it is annoying, to repeatedly select the same parameters, for different references, of the same Activity, it may actually be beneficial. In some places, the Activity may require a secure protocol, and in some places, not. Meaning, the local meta-data, though is annoying, actually make sense.

8.1.3 An inconvenient truth

As a programmer I have bad sentiments towards a solution of local meta-data — again and again to specify, for the same Activity, in different diagrams, the Parameters and Errors.
Although, we could consider that, at the BPMN abstract level, the same Activity might have different exceptions according to different business process contexts; in Prosero, currently, an Activity will be matched with a single Web Service— i.e., a single set of exceptions.

### 8.1.4 Can’t see the forest for the trees

It is nice to have high accuracy. But, is it nice enough to have low precision?! Low precision means we have many redundant exception handlers. Meaning, the diagram is less readable, for no good reason.

Furthermore, every exception handling at the BPMN level, translates to another scope, at the BPEL level, wrapping the Activity. That makes the BPEL diagram less coherent as well, and is a burden on the BPEL engine, which executes the process.

### 8.1.5 Did we matter?

The psychological benefit of our wizard asking to consider possible exceptions could be replaced with a methodology, by which the business analyst goes over the business process he has just completed and considers where an exception might occur.

Since the wizard also offers possible exceptions, it has a slight advantage, over such a required work methodology. Anyway, such work methodology is not common (unfortunately).

### 8.1.6 Between a rock and a hard place

Our solution does not directly solve the major problem of identifying explicit Error Events, thrown in a Sub Process. In that respect, our solution is a superficial answer.

However, considering the fact that (a) BPMN’s abstract level make it impossible to determine where exception must be handled, and (b) Prosero’s implementation make it difficult to access referenced Sub Processes; our solution is the best (immediate) option.
8.2 Future implementation

8.2.1 Simple enhancements

Smart exception handling — heuristics

Add heuristics rules that will allow the modeling tool to identify potential pitfalls, by analyzing the Process structure, or Data Object structure. This, basically, means that the tool will automatically populate some of Activity’s Parameters, using the same techniques the user, currently, employs.

Pattern: Identify dangerous patterns in a Sub Process.

Sub Process complexity: A complex/long Sub Process has a higher chance of failure at run-time (than a simple Sub Process).

Data Object complexity: An Activity that handles a complex/large Data Object, either as input or as output, has a higher chance of failing at execution (than an Activity handling a simple Data Object).

Add business exception icon for Sub Process

Until we add a “Top level” characteristic to BPMN Process (see in Section 8.3.1), we propose to help the business analyst identify Sub Processes, which explicitly throw Error Events, by adding an Intermediate Error Event icon at one of the corners of the Sub Process’ rectangle element.

Business process into Template

Add a feature to “abstractify” an existing BPD into a Template, and export it to the RMR.
8.2 Future implementation

8.2.2 Difficult enhancements

Complexity according to semantics

If you have a tool, which already analyzes business Activities for their semantics, you might as well enhance it to estimate an Activity’s complexity, according to the Activity’s name and description. This is probably more useful for Tasks than for Sub Processes.

Data constraints attribute

Add two attributes to specify input and output constraints for an Activity — precondition and postcondition. For example, a ‘Charge credit’ activity input must include a credit card number, with at least 9 digits. If the constraint fails, a ‘Constraint violation’ Error Event will be thrown.

This feature had the Commitment Violations exception (Section 3.2.2) in mind.

Although these constraints can be validated using a series of Gateways, such a series is quite bulky and cluttered. Moreover, it is quite acceptable to consider an improper state of a Data Object as an exception — a business exception. As such, it should be handled in an exception path and not an alternate path (from a Gateway).

8.2.3 Improve Parameters — use infrastructure

While understanding the necessity of a distinction between BPMN and BPEL, we are also aware that our choice of implementation affects the behavior of the process, and more specifically, the possible Error Events. For example, Web Services support various protocols, such as:

- WS-SecureConversation
- WS-Security
- WS-Federation
- WS-Authorization
- WS-Policy
• WS-Trust
• WS-Privacy

Failure of one of the protocols will result in a specific Error Event. In our research we only scratched the surface of the distinction between BPMN and BPEL, as far as Parameters are concerned. Further research is needed to examine how far we can go in specifying Activity behavior without breaching the distinction.

8.2.4 Improve diagram view — perspectives

The simple idea is to allow the user, through different editor perspectives, to view only some of the end-to-end sequence flows of a process. We thought about this specifically in order to diminish the clutter of a BPD packed with exception handling. For example, Figure 8.1.

![Figure 8.1: Calculation process — full view](image)

We could regard this process in two perspectives: Normal Flow, Figure 8.2 and Exception Flow, Figure 8.3. One could also add a perspective with or without Data Objects.

8.3 Future research

The following sub-sections mainly deal with extending BPMN specification.
8.3 Future research

8.3.1 Improve BPMN — add top level business process

BPMN does not prohibit circularity, e.g., process A can call process B, which calls process A. As a result, it is undetermined where a raised exception would stop. If it is raised in process A it can propagate to B, and then to A again. Therefore, BPMN cannot enforce exception handling, as it is not clear which process should handle the exception.

We define a top level business process as a business process that cannot be called by any other business process and therefore must handle all received exceptions.

Although BPMN is expendable, and we can easily add an attribute marking a business process as top-level, the bigger questions are: what are the business ramifications of defining such a top level? Is it feasible? Does it clash with the notion of BPMN?
8.3.2 Improve verification — more Events

Chapter 3 discussed several Exception Handling strategies in BPMN, according to [36]. Only two of them focused on Error Events. Intriguing other strategies:

- Time out exception pattern
- Unsolicited External Business Exception Pattern
- Solicited External Business Exception Pattern
- Transaction Compensation

These strategies are based on other types of Intermediate Events: Message, Timer, Cancel and Compensation. Further research is required to see whether we can help the business analyst choose which Event trigger to use. The case of Transaction Compensation is especially interesting, since it involves identifying, or defining, a whole process as a Transaction.

Currently the only support we give is in the form of Templates. The business architect can define a Template for a specific, abstract Business Transaction and include all the appropriate Compensation Events.

8.3.3 Improve Error Events — create hierarchy

I find it interesting that many systems require the business architect to declare business entities, organization Roles, Units, and Data Objects structures, so the business analyst can use them when modeling. However, no one has taken the time to define a group of exceptions.

BPMN identifies an Error Event by its name and error-code, and the modeling tools leave it at that. Nobody checks if the same error-code is thrown under different names and vice versa. Nobody defines relationships between Error Events.

In the software world there is a hierarchy between exceptions, e.g., `FileNotFoundException` extends `IOException`. Should there be such a hierarchy in the business process world?
Regardless, should an organization define a limited group of allowed Error Events from which to choose?

8.3.4 Suggesting appropriate handlers

Some prototypes [2] see how an exception was handled during execution and can later offer that handling technique as a valid handler for similar exceptions.

We are interested in pre-execution assistance. We already offer classifications for an Activity. For each classification we offer relevant exception. Can we suggest an appropriate handler for an exception?

The research should explore ideas such as a limited repository of exceptions, using semantics to compare Activity’s name and Error Event’s name, and finding how the same exception was handled elsewhere.

8.4 Immediate papers

I should take my preliminary research notes and turn them into papers on the following topics.

8.4.1 Viewing business process in the light of data

BPMN is the standard BPM language, de facto. But, it does not emphasis data. Rather, it focuses on the sequence flow between the Activities. We, on the other hand, believe that the states of the data are what defines a business process.

Business activity classification

The behavior of a business activity can be classified according to its data handling: analyze data, update data, verify data, send data, etc.
Exception classification

Business level exceptions are exceptions that rise from data constraint violation, unlike technical exceptions, such as ‘printer out of order’, which may affect the business process, but are not related to the business data.

8.4.2 Choosing the right static verification for your modeling tool

Because, in Prosero, a referenced Sub Process resides on a remote machine, we chose not to analyze that Sub Process and use only its local meta-data. But what if we had faster access? In our preliminary research, we investigated various types of modeling tools architectures, assigning each its suitable static verification methodology.

8.4.3 Exception handling vs. return value in business processes

We saw many discussions of this subject, in the field of software engineering. It should be interesting to review these discussions at the business process level and the graphical notation of BPMN.

Arguments, such as “It is much more difficult to add a Gateway element to an existing process than to add an Error Event”, do not appear in the software world.
8.5 So long, farewell, auf Wiedersehen, goodbye

I had great time working on this research. I developed an interest in graphical notation languages, as they seem more intuitive for humans. I learned much during the implementation, e.g., keep as much of the solution as possible on the server side. I enjoyed debating on the user experience aspects of this work. I felt that I gave honest, even if simple, contributions, to a the rising field of business process models, through this research and its products — the Prosero modeling tool and our website, which holds references to hundreds of business processes from around the Web. I searched for some time for such a collection of business processes, before deciding to build one myself: http://www.cs.bgu.ac.il/~bpmn/BPDs

I hope you enjoyed reading this research as much as I enjoyed writing it. I thank again my supervisors, colleagues, friends, and family, for all the help and patience, and wish everyone a good life, with minimum failures.
Appendix A

Dictionary

Atomic activity : an execution process cannot pause in the middle of such an activity. The activity must end or abort, before control can return to the caller.

BPMN : Business Process Modeling Notation. A language to describe business processes at the business level.

BPEL : Business Process Execution Language. A language to describe business processes at the IT level.

Catch : see ‘throw and catch’

Choreography : the relation/protocol between process and external process. Specifically the signals that pass between them.

Concertize : making an abstract object more concrete — grounding/actualizing the object.

Exception : A problem, during execution, that usually cannot be identified by checking the status of data objects.

Exception handling : actions that are taken in the case an exception occurs.

Happy path : any model of sequence of events that does not include unexpected incidents or error results. A path that assumes every action in it will produce the
best possible outcome.

**IT** : Information Technology.

**Maps to** : a group, or a diagram, can be translated to another group/diagram. For example, the group \{'one', 'two', 'three'\} can be mapped to the group \{1, 2, 3\}.

**Non-repudiation** : the validity of data transferred in a transmission (between two sides) cannot be repudiated, or refuted.

**Noop** No operation. Doing nothing.

**Orchestration** : managing and coordinating several processes to create a higher process. E.g., Our BPEL engine is an Orchestrator.

**Proof of concept** : a first stage in a project, in which you show your supervisor/investor that your idea (the concept of the project) is valid and applicable.

**Reply** : a message returned by a service provider in response to a request. The message will detail the results of the request.

**Request** : a message sent to a service provider. The message contains data for the service provider to handle.

**Static validation** : see ‘static verification’.

**Static verification** : the process of checking that software meets requirements by doing a physical inspection of it. This in oppose to ‘Dynamic verification’ where mistakes are uncovered by executing the software.

**Throw / Raise** : see ‘throw and catch’

**Throw and catch** : a professional term from the software world regarding events and exceptions. For example: an application attempts to read from a file, but the file does not exist. The application may throw the exception ‘File not found’. All application processes will stop and the exception will go back in the execution hierarchy of the application (A called B called C called . called me) until another
part of the application will catch the exception, handle it and will try to return to normal process.
Appendix B

Preliminary Research

Here we present the preliminary notes we gathered, while trying to rationalize our focus on exception handling the seriousness of the problem and our choice for implementation.

B.1 Possible features

1. Features of existing modeling tools

(a) eBPMN Designer by soyatec.com
- Auto-layout
- Embedded Sub Process design
- Embedded Sub Process refractory by grouping
- Boundary attachment
- Boundary attachment of Start/End Event
- Model explorer to show logical model view

(b) Business Process Visual ARCHITECt by visual-paradigm.com
- Business process diagram
- Data flow diagram
- Entity relationship diagram
- Export business process diagram to BPEL
- Identify candidate business process element by textual analysis
(c) Business Process Modeler for Business Analysts by eClarus.com

- Keep requirements and implementation in sync with BPMN-BPEL Round-Trip Engineering.
- Generate ready-to-deploy BPEL 1.1 or 2.0 codes that can run on any BPEL servers.
- Help validate proposed process changes and identify hard-to-find bottlenecks and potential optimizations with built-in simulation.
- Model validation as BPMN compliant

(d) BPMN Modeling Tool by Prosero project, DT Labs @ BGU

- Support BPMN core elements
- Suggest business process Activities and Data Object from customer repository
- Export business process diagram to BPEL, using a Natural Language Processing procedure to match an appropriate Web Service to each Task

2. Features of editors from other disciplines

(a) Word processor

- Spelling and Grammar validation
- Automatic text (e.g., weekdays, greetings, etc.)
- Creating a document from a template
- Creating new templates

(b) Programming environments

i. Code writing assistance

- Text auto completion — suggesting which reference to use (variable/method/type name)
- Code template - method format, loop format, etc.

ii. Reflection (modify your code without changing its meaning/behavior)

- Renaming a variable and all references to it
- Identifying code duplicities
- Extracting code segment to another file

iii. Static validation (compilation)
• Type matching — e.g., you can’t ask what is the minimum, mathematically, of 4, 21 and Potato Salad
• Unhandled exceptions — e.g., if you are told a function might fail, you must deal with that possibility

(c) Human intellect
• Identifying patterns
• Identifying duplicities — e.g., repeating names
• Identifying the right task for the goal
• Thinking ahead and avoiding exceptions

B.2 Possible static verification methods

1. Static verification procedure
   (a) local vs. remote
   (b) immediate vs. daily

2. Static verification attributes (Parameters, Errors, Handlers)
   (a) dynamic attribute vs. static attributes
   (b) keep consistency vs. allow some inconsistencies

3. Attributes populated:
   (a) automatically (by the tool)
   (b) manually (by the user)
   (c) semi-automatically

We get a total of 216 valid combinations.
Appendix C

Activity classification

We considered some alternate paths as exception paths if the Gateway decision could not be deduced from the return value of the previous Activity.

Classification and Exceptions legend

- Classification
  1. Query data (Example: Find Order(s))
  2. Enter data (Example: Log received items)
  3. Update data (Example: Log received items)
  4. Produce data (Example: Perform Regression Test)
  5. Send notification (Example: Notify customer RMA number is invalid)
  7. Send and Receive data (Example: Notify customer)
  8. Analyze data (Example: Allocate Defects)
  9. Perform action (Example: Negotiate return)

- Sub-Process: g
  - Gateway: s
x Complex activity

- Exceptions

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QF</td>
<td>Query failed (data not found)</td>
</tr>
<tr>
<td>UF</td>
<td>Update failed (data constraints)</td>
</tr>
<tr>
<td>UR</td>
<td>Update rejected (unauthorized)</td>
</tr>
<tr>
<td>SF</td>
<td>Send failed (target could not be reached)</td>
</tr>
<tr>
<td>SR</td>
<td>Send rejected (data rejected by receiver)</td>
</tr>
<tr>
<td>RR</td>
<td>Receive rejected (data rejected)</td>
</tr>
<tr>
<td>NR</td>
<td>Received notification rejected</td>
</tr>
<tr>
<td>EF</td>
<td>Enter data failed (problem with the form or the tool)</td>
</tr>
<tr>
<td>LR</td>
<td>Analysis rejected (analysis concluded a possible problem: not enough resources, refuse to sign)</td>
</tr>
<tr>
<td>VR</td>
<td>Analysis/Verification rejected (concluded dissimilarities)</td>
</tr>
<tr>
<td>AF</td>
<td>Action failed</td>
</tr>
<tr>
<td>TO</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>Classification</th>
<th>Probable exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update return (of goods)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find Order(s)</td>
<td>1</td>
<td>QF</td>
</tr>
<tr>
<td>Find Return</td>
<td>1</td>
<td>QF</td>
</tr>
<tr>
<td>(where goods received?) No/Yes</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Reject return updates</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Notify customer</td>
<td>7 ; 4</td>
<td>SF</td>
</tr>
<tr>
<td>(customer wants to proceed?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Return with prior approval</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>(was credit issued?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>(update return type)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Classification</td>
<td>AC</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>Add order item to return request</td>
<td>2</td>
<td>UF</td>
</tr>
<tr>
<td>Remove order item from return request</td>
<td>2</td>
<td>UF</td>
</tr>
<tr>
<td>Update refund method</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Administer return request</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>Notify customer about RMA</td>
<td>5</td>
<td>SF</td>
</tr>
<tr>
<td>Return goods</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td><strong>Return goods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(are goods required to be returned?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Give customer return instructions</td>
<td>4 ; 5</td>
<td></td>
</tr>
<tr>
<td>Receive returned goods</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>Perform disposition</td>
<td>s</td>
<td>VR</td>
</tr>
<tr>
<td>Issue credit</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td><strong>Receive returned goods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Look for RMA number in received box</td>
<td>9 ; 4</td>
<td>AF</td>
</tr>
<tr>
<td>Find RMA in the system</td>
<td>1</td>
<td>QF</td>
</tr>
<tr>
<td>Notify customer RMA number is invalid</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Resolve RMA number not found</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>List expected items</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Log received items</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Print RMA</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>Perform disposition</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examine goods for disposition</td>
<td>9 ; 4</td>
<td>VR</td>
</tr>
<tr>
<td>(should goods be restocked?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>yes/no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark disposition: good</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Restock goods</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>1 ; 2 ; 3</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Adjust inventory</td>
<td>2 ; 3</td>
<td>UF</td>
</tr>
<tr>
<td>Mark disposition: bad</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Redirect goods</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>Issue credit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update TA spending level</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Credit external account</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Refund using payment policy</td>
<td>1 ; 2 ; 3</td>
<td></td>
</tr>
<tr>
<td><strong>Administer return request</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate refund amount</td>
<td>8 ; 2</td>
<td></td>
</tr>
<tr>
<td>CSR accept refund amount</td>
<td>6</td>
<td>g</td>
</tr>
<tr>
<td>(CSR accept refund amount?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>(Customer accept refund amount?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Adjust refund amount</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Approve return</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><strong>Approve return</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approve automatically</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>(is returned approved?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Notify customer return is denied</td>
<td>7</td>
<td>TO</td>
</tr>
<tr>
<td>(Does customer challenge denial?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Close return request</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Negotiate return</td>
<td>9 ; 4</td>
<td></td>
</tr>
<tr>
<td>(Negotiation results:)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Notify customer refund won't happen</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Modify return request</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Approve return manually</td>
<td>2 ; 3</td>
<td>UR</td>
</tr>
<tr>
<td><strong>Software Defect Management</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Activity classification</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Identify discrepancy between requirements and product</td>
<td>1 ; 8 ; 4 ; x</td>
<td></td>
</tr>
<tr>
<td>Discrepancy is logged in Tracking tool</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Review defects with status New, Open or Reopen (Needs clarification?)</td>
<td>1 ; 8 ; 2</td>
<td></td>
</tr>
<tr>
<td>Triage defects</td>
<td>8 ; 2</td>
<td></td>
</tr>
<tr>
<td>Assign defect</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Provide extra information</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>(Type of defect)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Assess acceptability to business</td>
<td>1 ; 9</td>
<td></td>
</tr>
<tr>
<td>Fix Code and Unit Test</td>
<td>9 ; 2 ; 3</td>
<td></td>
</tr>
<tr>
<td>(Behavior acceptable?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Define or refine requirement</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Schedule for future release</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Close Defect</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>(Required in this release?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Receive Build and Execute Smoke Test</td>
<td>9 ; 4</td>
<td></td>
</tr>
<tr>
<td>(Smoke Test Passed?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>(Retest Passed?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Log Defect(s) [= Identify discrepancy + Discrepancy is lodged]</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Execute Retest of specific defect</td>
<td>9 ; 4</td>
<td></td>
</tr>
<tr>
<td>Perform Regression Test</td>
<td>9 ; 4 ; x</td>
<td></td>
</tr>
<tr>
<td>(New Defects?)</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>Close Defect after fix</td>
<td>2 ; 3</td>
<td></td>
</tr>
<tr>
<td>Create Product Build</td>
<td>9 ; 4 ; x</td>
<td></td>
</tr>
<tr>
<td>Copy build to Staging area</td>
<td>9 ; 4</td>
<td></td>
</tr>
</tbody>
</table>
Analyze is both Perform action and Verification. So we don’t need Analyze.

We conclude that actions classified only as Enter data or Produce data, are not likely to fail. And even if they do fail, the handling of the exception is trivial enough to: (a) have automated response; and (b) not be considered as part of the business methodology.
Appendix D

Experiment complete results

Activity classification legend

1 Query data (Example: Find Order(s))
2 Enter data (Example: Log received items)
3 Update data (Example: Log received items)
4 Produce data (Example: Perform Regression Test)
5 Send notification (Example: Notify customer RMA number is invalid)
6 Receive notification (Example: Receive Report State of Accounts)
7 Send and Receive data (Example: Notify customer)
8 Analyze data (Example: Allocate Defects)
9 Perform action (Example: Negotiate return)

s Sub-Process

g Gateway

x Complex activity
Exceptions legend

QF  Query failed
UF  Update failed
UR  Update rejected
SF  Send failed
SR  Send rejected
RR  Receive rejected (data rejected)
RR-az Receive rejected (authorization)
RR-ac Receive rejected (authentication)
RR-se Receive rejected (security)
NR  Notification rejected
VR  Analysis/Verification rejected
AF  Action failed
TO  Timeout

Exceptions according to:

Org  the original business process diagram, modeled using a regular modeling tool
Enh  a user of Prosero’s enhanced modeling tool (i.e., with suggestions)
Exp  an expert
Additional columns:

\textbf{#sug} Number of suggestions for exceptions, given by the enhanced tool, according to classification

\textbf{#sel} Number of exception the user of the enhanced tool selected from the suggestions

## D.1 Update return (of goods)

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>#sug</th>
<th>#sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Order(s)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>QF</td>
<td>QF</td>
<td>QF</td>
</tr>
<tr>
<td>Find Return</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>QF</td>
<td>QF</td>
<td>QF</td>
</tr>
<tr>
<td>(where goods received?) No/Yes</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject return updates</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notify customer</td>
<td>7 ; 4</td>
<td>2</td>
<td>1</td>
<td>SF</td>
<td>SF</td>
<td></td>
</tr>
<tr>
<td>(customer wants to proceed?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return with prior approval</td>
<td>s</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure D.1: Experiment input: Update return
### D.2 Return goods

<table>
<thead>
<tr>
<th>Action</th>
<th>Type</th>
<th>Y</th>
<th>N</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>(was credit issued?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(update return type)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add order item to return request</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Remove order item from return request</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Update refund method</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>UF</td>
</tr>
<tr>
<td>Administer return request</td>
<td>s</td>
<td>1</td>
<td></td>
<td>AF</td>
</tr>
<tr>
<td>Notify customer about RMA</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>NF</td>
</tr>
<tr>
<td>Return goods</td>
<td>s</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure D.2: Experiment input: Return goods

![Diagram of Return Goods Process]
### Business Activity

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>(are goods required to be returned?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Give customer return instructions</td>
<td>4 ; 5</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive returned goods</td>
<td>s</td>
<td>1</td>
<td></td>
<td>AF</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>Perform disposition</td>
<td>s</td>
<td>1</td>
<td></td>
<td>VR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue credit</td>
<td>s</td>
<td>1</td>
<td></td>
<td>UR</td>
<td>UR</td>
<td></td>
</tr>
</tbody>
</table>

#### D.3 Receive returned goods

![Diagram](image)

Figure D.3: Experiment input: Receive returned goods

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look for RMA number in received box</td>
<td>9 ; 4</td>
<td>2</td>
<td>0</td>
<td></td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>Find RMA in the system</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>QF</td>
<td>QF</td>
<td>QF</td>
</tr>
<tr>
<td>Notify customer RMA number is invalid</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td>SR</td>
<td></td>
</tr>
</tbody>
</table>
### D.3 Receive returned goods

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Task ID</th>
<th>Task Count</th>
<th>Task Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolve RMA number not found</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>List expected items</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Log received items</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Print RMA</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
D.4 Perform disposition

![Experiment input: Perform disposition](image)

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examine goods for disposition</td>
<td>9 ; 4</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(should goods be re-stocked?) yes/no</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark disposition: good</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restock goods</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>AF</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>Adjust inventory</td>
<td>2 ; 3</td>
<td>2</td>
<td>1</td>
<td>UF</td>
<td>UF</td>
<td></td>
</tr>
</tbody>
</table>
Mark disposition: bad | 9 | 1 | 0 |
Redirect goods    | 9 | 1 | 0 |

D.5  Issue credit

Figure D.5: Experiment input: Issue credit

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update TA spending level</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit external account</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td>NF</td>
</tr>
<tr>
<td>Refund using payment policy</td>
<td>1 ; 2 ; 3</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>NF</td>
</tr>
</tbody>
</table>
D.6 Administer return request

![Diagram showing the process of Administer return request]

Figure D.6: Experiment input: Administer return request

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th># sug</th>
<th># sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate refund amount</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSR accept refund amount</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>VF</td>
<td>VF</td>
<td>VF</td>
</tr>
<tr>
<td>(CSR accept refund amount?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer refund amount</td>
<td>g</td>
<td></td>
<td></td>
<td>VF</td>
<td>VF</td>
<td></td>
</tr>
<tr>
<td>Adjust refund amount</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approve return</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>AF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D.7 Approve return

Figure D.7: Experiment input: Approve return

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approve automatically</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>AF</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>(is returned approved?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notify customer return is denied</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>SR</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>(Does customer challenge denial?) No/Yes</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close return request</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negotiate return</td>
<td>9</td>
<td>4</td>
<td>x</td>
<td>3</td>
<td>0</td>
<td>AF</td>
</tr>
<tr>
<td>(Negotiation results:)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notify customer refund won’t happen</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>NR</td>
<td>NR</td>
<td></td>
</tr>
<tr>
<td>Modify return request</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approve return manually</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>UR</td>
<td></td>
</tr>
</tbody>
</table>
D.8 Deliver item

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>#sug</th>
<th>#sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery with invoice</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>AF</td>
<td>AF</td>
<td>AF</td>
</tr>
<tr>
<td>Update invoice</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>UR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate delivery</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(pay on delivery?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delivery</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive payment</td>
<td>9 ; 2</td>
<td>2</td>
<td>0</td>
<td></td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Follow up</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure D.8: Experiment input: Deliver item
## D.9 HR Hiring

![Diagram](image)

**Figure D.9: Experiment input: HR Hiring**

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>#sug</th>
<th>#sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify staffing needs</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define and write up Job description</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send personnel request form to HR</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>SF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine classification</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>AF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(regular hire?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Temporary hire process)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post job internally</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Suitable candidates?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examine existing resumes on file</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Description</td>
<td>Yes</td>
<td>No</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----</td>
<td>----</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop recruitment strategy</td>
<td>8 : 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place ads</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gather resumes</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place ads</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop interview questions</td>
<td>4 : x</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conduct preliminary interview</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conduct interview</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select candidate</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine pay range for the candidate</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Make employment offer</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Candidate acceptance?) Yes/No</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hire candidate</td>
<td>s</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D.10 Software Defect Management

Figure D.10: Experiment input: Software Defect Management

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>sug</th>
<th>sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify discrepancy between requirements and product</td>
<td>1 ; 8 ; 4 ; x</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrepancy is logged in Tracking tool</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Description</td>
<td>Count 1</td>
<td>Count 2</td>
<td>Count 3</td>
<td>Count 4</td>
<td>Acceptability</td>
<td>Fix Code and Unit Test</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Review defects with status New, Open or Reopen</td>
<td>8 ; 2 ; x</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Triage defects</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Assign defect</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Provide extra information</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>(Type of defect)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assess acceptability to business</td>
<td>1 ; 9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fix Code and Unit Test</td>
<td>9 ; 2 ; 3 ; x</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>AF</td>
</tr>
<tr>
<td>(Behavior acceptable?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define or refine requirement</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Schedule for future release</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close Defect</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Required in this release?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Build and Execute Smoke Test</td>
<td>9 ; 4 ; x</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>(Smoke Test Passed?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Retest Passed?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log Defect(s)</td>
<td>2 ; 3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute Retest of specific defect</td>
<td>9 ; 4</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
<tr>
<td>Perform Regression Test</td>
<td>9 ; 4 ; x</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>TO</td>
</tr>
</tbody>
</table>
### D.11 Software Release Product

<table>
<thead>
<tr>
<th>Business Activity</th>
<th>cls</th>
<th>#sug</th>
<th>#sel</th>
<th>Org</th>
<th>Enh</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Product Build</td>
<td>9 ; 4 ; x</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy build to Staging area</td>
<td>9 ; 4</td>
<td>2</td>
<td>0</td>
<td>AF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue release document to QA</td>
<td>4 ; 5</td>
<td>2</td>
<td>1</td>
<td>SR</td>
<td>SR</td>
<td></td>
</tr>
<tr>
<td>Perform System Test</td>
<td>9 ; 4 ; x</td>
<td>3</td>
<td>0</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fix Defects</td>
<td>s ; x</td>
<td>2</td>
<td>0</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(System Test Pass?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate Defects</td>
<td>8 ; 2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smoke Test</td>
<td>9 ; 4 ; x</td>
<td>3</td>
<td>1</td>
<td>TO</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>(Sanity Test Pass?)</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit draft Release form</td>
<td>4 ; 7</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release Form Sign-off (Fulfillment)</td>
<td>8 ; 2</td>
<td>2</td>
<td>1</td>
<td>LR</td>
<td>LR</td>
<td></td>
</tr>
<tr>
<td>Add to Downloads site</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue Product Availability Announcement</td>
<td>4 ; 9</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post Project Review</td>
<td>9 ; 4</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify copyright and licenses of Third Party</td>
<td>1 ; 8 ; 2</td>
<td>3</td>
<td>1</td>
<td>LR</td>
<td>LR</td>
<td>LR</td>
</tr>
<tr>
<td>Issue Reporting Requirements</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experiment complete results

<table>
<thead>
<tr>
<th>Task</th>
<th>4 ; 5</th>
<th>2</th>
<th>0</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify stakeholders of release to QA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release Form Sign-off (Engineering)</td>
<td>8 ; 2</td>
<td>2</td>
<td>1</td>
<td>LR</td>
<td>LR</td>
</tr>
<tr>
<td>Release Form Sign-off (Legal)</td>
<td>8 ; 2</td>
<td>2</td>
<td>1</td>
<td>LR</td>
<td>LR</td>
</tr>
</tbody>
</table>

Figure D.11: Experiment input: Software Release Product
Appendix E

Bibliography by category

Business Processes [20], [18], [19], [11], [31], [17], [7]

Business Processes Diagrams (examples) [1], [23], [40], [31] and more on www.cs.bgu.ac.il/~bpmn/BPDs

Business Transaction [12], [14]

BPMN [32], [36], [4], [41], [45], [44], [43]

BPMN Modeling tools [40]

Semantic BPM [13]

Exceptions and Exception Handling [33], [2]

BPEL [21]

BPEL4WS [29], [42], [24], [10], [30], [9], [25], [26], [27], [22]
Bibliography by category

**Business Processes execution**  [28]

**Web-Services**  [38] , [34] , [5] , [15]

**Patterns in SOA**  [3]


**Exception handling in BPMN**
Bibliography


Conference on Parallel, Distributed, and Network-Based Processing (2006), pp. 39–47.


