THESIS WORK

Attila Donáth

2019
Business Process Management Tools
Developer-Friendly Process Engine vs.
Low-Code BPM Suite

Attila Donáth
Business Information Systems MSc
2019

Thesis Supervisor: Dr. Katalin Ternai
I. számú melléklet
NYILATKOZAT SAJÁT MUNKÁRÓL

Név: Donáth Attila

Szak: Gazdaságinformatikus MSc

A szakadogozat címe magyarul:
Uzleti folyamatmenedzsment eszközök - fejlesztőbarát folyamatmotor vagy low-code BPM csomag

A szakadogozat címe angolul:

Szakszemnáriumi-vezetőkonzulens neve: Dr. Ternai Katalin

Én, Donáth Attila, teljes felelősségem tudatában kijelentem, hogy az általam papír alapon és elektronikusan feltöltött szakadogozat 100%-ban egyezik, és nevezett szakadogozatban szereplő minden szöveg rész, ábra és táblázat - az előírt szabályoknak megfelelően hivatkozott részek kivételével - eredeti és kizárólag a saját munkám eredménye, más dokumentumra vagy közreműködőre nem támaszkodik.

Kelt: Zürich, 2019-11-13

halgató aláírása

Párhuzamos képzés esetén kitöltendő!

Én, Ternai Katalin, teljes felelősségem tudatában kijelentem, hogy a jelen szakadogozatomban és a párhuzamos képzésem leírásának személyes nem haladja meg 10% százalékkal, a Tanulmányi és Vizsgaszabályzat Gazdálkodástudományi kari mellékleteinek II. fejezetében a 10. § (2) alapján. Tudomásul veszem, hogy annakben a szakfelelősök (vagy az általuk megjelölt személyek) 10%-nál nagyobb egyezőséget állapítanak meg, akkor a tanulmányi kötelezettségeimet nem teljesítettem, záróvizsgát nem tehetek.

Kelt: ..........................................

halgató aláírása

TÉMAVEZETŐI NYILATKOZAT

Alulírott, Ternai Katalin konzulens kijelentem, hogy a fent megjelölt hallgató fenti szerinti szakadogatok (egyetemi/ mesterképzésben diplomamunkája) benyújtásra alkalmas és védésre ajánlom.

Budapest, 2019.11.09

konzulens aláírása
II. számú melléklet

NYILATKOZAT A SZAKDOLGOZAT NYILVÁNOSÁGÁRÓL

Név (nyomtatott betűvel): Donáth Attila

Szak neve (amelyen a jelenlegi Szakdolgozatát megírtak): Gazdaságinformatikus MSc

Dolgozatom elektronikus változatának (pdf dokumentum, a megtekintés, a mentés és a nyomtatás engedélyezett, szerkesztés nem) nyilvánosságáról az alábbi lehetőségek közül kiválasztott hozzáférési szabályzat szerint rendelkezem.

TELJES NYILVÁNOSÁGGAL
A könyvtári honlapon keresztül elérhető a Szakdolgozatok TDK adatbázisban (http://szd.lib.uni-corvinus.hu), a világháló bármely pontjáról hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

KORLÁTOZOTT NYILVÁNOSÁGGAL
A könyvtári honlapon keresztül elérhető a Szakdolgozatok TDK adatbázisban (http://szd.lib.uni-corvinus.hu), a kizárólag a Budapesti Corvinus Egyetem területéről hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

NEM NYILVÁNOS
A dolgozat a BCE Központi Könyvtárának nyilvántartásában semmilyen formában (bibliográfiai leírás vagy teljes szöveges változat) nem szerepel.

Zürich. 2019-11-13

Donáth Attila

hallgató (szerző) aláírása
# Table of Contents

1 Overview..................................................................................................................................................................................................................................................................................................................1
  1.1 Introduction..............................................................................................................................................................................................................................................................................................................1
  1.2 Requirements for process modeling tools...........................................................................................................................................................................................................................................................................................................3
    1.2.1 Introduction.................................................................................................................................................................................................................................................................................................................3
    1.2.2 Clarity.........................................................................................................................................................................................................................................................................................................................3
    1.2.3 Comprehensibility........................................................................................................................................................................................................................................................................................................3
    1.2.4 Standardization........................................................................................................................................................................................................................................................................................................3
  1.3 Requirements for process management tools........................................................................................................................................................................................................................................................................................................4
    1.3.1 Introduction.................................................................................................................................................................................................................................................................................................................4
    1.3.2 Ease of use..........................................................................................................................................................................................................................................................................................................5
    1.3.3 Reliability.................................................................................................................................................................................................................................................................................................5
    1.3.4 Speed.................................................................................................................................................................................................................................................................................................6
    1.3.5 Customizability..............................................................................................................................................................................................................................................................................7
    1.3.6 Integrability..............................................................................................................................................................................................................................................................................7
    1.3.7 Change management........................................................................................................................................................................................................................................................................8
    1.3.8 Reporting...............................................................................................................................................................................................................................................................................8
    1.3.9 Security................................................................................................................................................................................................................................................................................9
    1.3.10 Vendor and expert support........................................................................................................................................................................................................................................9
  2 Modeling standards: BPMN, CMMN, DMN....................................................................................................................................................................................................................................................................................................10
    2.1 Introduction....................................................................................................................................................................................................................................................................................10
    2.2 Business Process Modeling Notation (BPMN)....................................................................................................................................................................................................................................................................................................10
      2.2.1 Introduction................................................................................................................................................................................................................................................................................10
      2.2.2 Process modeling................................................................................................................................................................................................................................................................................11
      2.2.3 Sample process..............................................................................................................................................................................................................................................................................19
    2.3 Case Management Model and Notation (CMMN).....................................................................................................................................................................................................................................................................................................19
      2.3.1 Introduction..............................................................................................................................................................................................................................................................................19
      2.3.2 Case modeling..............................................................................................................................................................................................................................................................................20
      2.3.3 Sample case..............................................................................................................................................................................................................................................................................24
    2.4 Decision Model and Notation (DMN).....................................................................................................................................................................................................................................................................................................25
      2.4.1 Introduction..............................................................................................................................................................................................................................................................................25
      2.4.2 Decision modeling..............................................................................................................................................................................................................................................................................26
      2.4.3 Sample decision..............................................................................................................................................................................................................................................................................28
  3 Camunda.................................................................................................................................................................................................................................................................................................................................29
    3.1 Overview....................................................................................................................................................................................................................................................................................29
      3.1.1 Introduction................................................................................................................................................................................................................................................................................29
      3.1.2 The platform...........................................................................................................................................................................................................................................................................30
      3.1.3 Starting a project.......................................................................................................................................................................................................................................................................31
    3.2 Process modeling, data handling..............................................................................................................................................................................................................................................................................................32
      3.2.1 The modeling tool.......................................................................................................................................................................................................................................................................32
      3.2.2 Coverage of the standards...............................................................................................................................................................................................................................................33
      3.2.3 Handling process and business data....................................................................................................................................................................................................................................35
    3.3 User interface, REST API............................................................................................................................................................................................................................................................................................36
      3.3.1 Tasklist, the default user interface.............................................................................................................................................................................................................................................36
      3.3.2 REST API.................................................................................................................................................................................................................................................................................39
      3.3.3 Custom user interface......................................................................................................................................................................................................................................................................40
    3.4 Developer tools............................................................................................................................................................................................................................................................................................42
    3.5 Security............................................................................................................................................................................................................................................................................................42
      3.5.1 Authentication and authorization..................................................................................................................................................................................................................................................42
      3.5.2 Dynamic permission handling..............................................................................................................................................................................................................................................43
      3.5.3 Other security considerations..............................................................................................................................................................................................................................................44
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6 Integration with other systems</td>
<td>44</td>
</tr>
<tr>
<td>3.7 Version control, artifacts, change management</td>
<td>46</td>
</tr>
<tr>
<td>3.7.1 Code versioning</td>
<td>46</td>
</tr>
<tr>
<td>3.7.2 Process versioning</td>
<td>46</td>
</tr>
<tr>
<td>3.8 Process monitoring, troubleshooting</td>
<td>47</td>
</tr>
<tr>
<td>3.9 Reporting and auditing</td>
<td>51</td>
</tr>
<tr>
<td>3.9.1 Runtime and history database, history features</td>
<td>51</td>
</tr>
<tr>
<td>3.9.2 Reporting on history – low level</td>
<td>52</td>
</tr>
<tr>
<td>3.9.3 Reporting on history – Cockpit, Optimize</td>
<td>53</td>
</tr>
<tr>
<td>3.10 Vendor and expert support</td>
<td>54</td>
</tr>
<tr>
<td>4 Pega</td>
<td>56</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>56</td>
</tr>
<tr>
<td>4.1.1 Introduction</td>
<td>56</td>
</tr>
<tr>
<td>4.1.2 The platform</td>
<td>57</td>
</tr>
<tr>
<td>4.1.3 Starting a project</td>
<td>58</td>
</tr>
<tr>
<td>4.2 Process modeling, data handling</td>
<td>60</td>
</tr>
<tr>
<td>4.2.1 Case, stage, flow</td>
<td>60</td>
</tr>
<tr>
<td>4.2.2 Process diagram</td>
<td>63</td>
</tr>
<tr>
<td>4.2.3 Process modeling in practice</td>
<td>63</td>
</tr>
<tr>
<td>4.2.4 Classes and the Enterprise Class Structure</td>
<td>70</td>
</tr>
<tr>
<td>4.2.5 Data storage</td>
<td>71</td>
</tr>
<tr>
<td>4.2.6 Built-in data structures</td>
<td>73</td>
</tr>
<tr>
<td>4.3 User interface</td>
<td>74</td>
</tr>
<tr>
<td>4.3.1 Designing user interfaces</td>
<td>74</td>
</tr>
<tr>
<td>4.3.2 Default end-user portals: Case Worker, Case Manager</td>
<td>78</td>
</tr>
<tr>
<td>4.3.3 GetNextWork</td>
<td>80</td>
</tr>
<tr>
<td>4.4 Developer tools</td>
<td>82</td>
</tr>
<tr>
<td>4.4.1 The environment</td>
<td>82</td>
</tr>
<tr>
<td>4.4.2 Clipboard tool</td>
<td>84</td>
</tr>
<tr>
<td>4.4.3 Tracer</td>
<td>85</td>
</tr>
<tr>
<td>4.4.4 Live UI</td>
<td>87</td>
</tr>
<tr>
<td>4.4.5 Coding in a low-code system</td>
<td>88</td>
</tr>
<tr>
<td>4.5 Security</td>
<td>89</td>
</tr>
<tr>
<td>4.5.1 Authentication and authorization, Role-Based Access Control</td>
<td>89</td>
</tr>
<tr>
<td>4.5.2 Attribute-based Access Control</td>
<td>90</td>
</tr>
<tr>
<td>4.5.3 Other access control features</td>
<td>90</td>
</tr>
<tr>
<td>4.5.4 Other security considerations</td>
<td>91</td>
</tr>
<tr>
<td>4.6 Integration with other systems</td>
<td>92</td>
</tr>
<tr>
<td>4.7 Version control, import-export, change management</td>
<td>94</td>
</tr>
<tr>
<td>4.7.1 Rule versioning and RuleSets, rule check-out</td>
<td>94</td>
</tr>
<tr>
<td>4.7.2 Import-export</td>
<td>97</td>
</tr>
<tr>
<td>4.7.3 Process change management</td>
<td>98</td>
</tr>
<tr>
<td>4.8 Process monitoring, troubleshooting</td>
<td>98</td>
</tr>
<tr>
<td>4.9 Reporting and case auditing</td>
<td>100</td>
</tr>
<tr>
<td>4.9.1 Report Definition</td>
<td>100</td>
</tr>
<tr>
<td>4.9.2 Business Intelligence Exchange</td>
<td>102</td>
</tr>
<tr>
<td>4.9.3 Case auditing</td>
<td>102</td>
</tr>
<tr>
<td>4.10 Vendor and expert support</td>
<td>103</td>
</tr>
<tr>
<td>5 Summary</td>
<td>105</td>
</tr>
<tr>
<td>Bibliography</td>
<td>106</td>
</tr>
<tr>
<td>Illustration Index</td>
<td>112</td>
</tr>
</tbody>
</table>
1 Overview

1.1 Introduction

From the 2000s onwards, the role of information technology has exploded with the advent of digitalization. Regardless of the industry or company size, businesses use different IT tools to manage and support their processes but to varying degrees. Some just try to comply with the regulations – e.g. stores need to have an online cash register – some create a website due to buyer’s pressure, and there are the ones who recognize the potential in the recent changes, and strive to develop their business with the integration of new technologies.

A well-organized, modern company can rely on IT to compete with, or even overtake, long-established companies in many areas – think of Amazon (DePillis and Sherman, 2018) or the emerging FinTech provider, Revolut (Smith, 2018), that puts the banking and insurance industry in competition. In this environment, even non-IT-focused firms are forced to innovate in order to maintain their leadership – or at least a slice of the market. This situation is made even more difficult by the (relative) labor shortage – due to the rapid changes it is a great challenge to keep up with the technology, and thus, it’s often hard to find enough good professionals for a project – which results in missed opportunities. In addition to all these, if a bigger, older company needs to serve existing customers in parallel with ongoing upgrade efforts, the current sources of competitive advantage – broad customer base and big number of stable processes – can become a drawback.

In such a situation, it is strategically important to select the right tools and make conscious use of them – that is, the effective digital transformation of the company. Improvements affect many areas and should be implemented in a coordinated manner, using strategic goals as a compass along the way. Since using computers for work is not new, most of the larger companies have various IT tools that have been implemented and refined over many years, and throughout many projects. The current systems are mostly data-centric (data warehouses, accounting software, human resource management programs), which make it easier to store and retrieve information. However, in most cases, processes only exist in the form of regulations and documentation – but it is also not uncommon that the real, up-to-date versions are only in the employees’ heads. Recognizing this, in recent years businesses have been placing increasing emphasis on discovering, optimizing processes, and developing IT support for them.
The subject of my thesis is to examine this section of the digital revolution from a practical perspective: what kind of software can best support process management for a particular company. As the field is constantly changing, and there are many tools available with significantly different approaches, it is difficult to choose the one that suits a company the best. Out of the many options, I picked two vastly different, but widely used software: one is a smaller but standard-compliant and developer-friendly product that focuses only on processes – Camunda – while the other aims to cover many enterprise functionalities as a single, common platform, and wants to incorporate tech-oriented business users in development efforts as well – Pega.

The idea of this thesis came after a real-world project where there was a debate about which tool would be the best fit. At the time of writing, I have been working with Pega for three years, and took part in multiple projects – but to broaden my view I wanted to get to know a completely different approach, that is, Camunda. This means, that although I know Pega much better, I have done researches on Camunda and tried to learn the platform as good as I could. To have a common starting point for analysis and comparison, I implemented the same process (inspired by a scenario at the company I worked for) on both platforms, the solutions are available at: attiladonath.com/msc-thesis.

1. Illustration: Planned investments of Hungarian companies according to the ITSM research in 2016 (Szabó, 2017, fol.16 translated by the author)
1.2 Requirements for process modeling tools

1.2.1 Introduction
In order to be able to support processes with IT, first, we need to get to know them. While this seems self-evident, it is often a challenge to assemble a model from a variety of documents and conversations with employees – and it usually takes multiple iterations to get a usable result. Many people can be involved in the consultation: technical experts, programmers, external consultants, customer relations professionals, and decision-makers in various fields – so it is important to create a clear, easy-to-understand standard model.

1.2.2 Clarity
Sketched drawing on a blackboard with boxes and arrows can be a good way to convey thoughts quickly, and to those who participate in the discussions from the beginning, it can be transparent and understandable. However, interpreting such a notation requires the context that the participants construct together, without that the figure is incomprehensible – or ambiguous. A box may indicate a manual or automatic process, an arrow may indicate data transfer between two units, or a time dependency. It is important that certain elements always have the same meaning – and that everyone is aware of this.

1.2.3 Comprehensibility
Clarity can be achieved in many ways. A mathematical formula is clear to those skilled in science, and the text of a well-written law is clear to lawyers. However, these methods require a great deal of prior training or can be overly verbose – thus, in an environment where it is required to rapidly exchange information between professionals working in very different fields are not appropriate. For this reason, visual notations are used for process modeling, as usually said, “one image is worth a thousand words” (Romih, 2016). The advantage of this kind of communication is that it’s intuitive, connections and correlations between the elements can quickly be understood without any special training – however, the clarity can suffer due to the compression of the information. In practice, the task will be to find the balance: what is the key information and what can be ignored for faster communication.

1.2.4 Standardization
It is easiest to achieve a balance between clarity and comprehensibility by establishing common context and rules. Of course, there are fields that are not yet covered by commonly accepted tools – typically research and development projects – but business process modeling has been here long enough to find the standards that are right for us for most tasks. Even if a
special component cannot be modeled with a generic notation, it is usually enough to introduce a custom methodology for that specific part only so that the comprehensibility of the whole model is only minimally damaged. The usual argument against standardization is that learning a particular technique, markup language requires additional resources, upfront investment, so the project size must be large enough to be worth it. Fortunately, it is often enough to get to know only the relevant sections from the standard, and this can be accomplished through a few discussions or a few hours of online training. And as we add IT-backed functionality to the models, the use of standardization pays off immediately.

1.3 Requirements for process management tools

1.3.1 Introduction

Process modeling nowadays often is done so that a computer program – a process management tool – can support the execution and automate subtasks, or even the entire process. Most of these software are capable of interpreting a standard (whether proprietary or open standard) visual process model, where each element can be assigned a specific program functions.

In the simplest case, the application can display forms for the users and store the data entered on them. While like that, making decisions and ensuring data quality remains the user’s responsibility, the company already benefits greatly from the fact that the processes currently managed on paper or in many separate applications are moved into a centralized system. The software can guide the user's hands, ensure that tasks that are dependent on one another are executed in the appropriate order and that nothing is missed out. By mapping the organizational hierarchy, we can delegate different tasks to the right person, and thus reduce unnecessary e-mail communication, improve data security, make it easier to guarantee proper handling of personal information, and other legal requirements.

Later, the software can take over some decisions to ensure that, for example, an approval task is automatically assigned to the appropriate person based on the amount of money – thus preventing the possibility of user error or abuse. In clear cases, the lead time of a process can be reduced – for example, if the base criteria for a bank loan are not met, the software may automatically reject the application. Finally, a process management software that integrates with your other systems can automatically perform repetitive tasks: sending emails or postal packages, ordering supplies – or even communicate directly with customers via telephone or chat clients.
In order to delegate tasks to software which are this complex, often management level, and have a great impact on the image of the company, the system requires general acceptance and a high level of trust from the members of the organization. Accordingly, expectations for such software – and their vendors – are high, and as human involvement decreases and automation increases, get even higher.

### 1.3.2 Ease of use

One of the most important criteria is that the software should be clear, transparent, and quick to learn. Employees often see transformations as a threat to their job, they are afraid that they need to give up their usual routines, and face new challenges beyond their current responsibilities – so many will oppose the introduction of a new system. These fears are all real, usually, management starts to implement process management software to optimize work, often in preparation for cost reductions and job cuts. In order to win staff, the software needs to offer something better: improved working conditions, better scheduling, better planning, less unexpected overtime. In addition, the extra work during the introduction phase should be minimized, and the learning period needs to be kept short.

A partitioning strategy can be applied here as well: a company’s operation is comprised of many processes, of these only the relevant ones need to be shown to a particular end-user, and only important information should be displayed on user interfaces – to avoid information overload. Navigation in the application has to be intuitive – should not require the memorization of menu items – and the process needs to be controlled by the program – it must not be expected from the operators to know the order of the steps. Nowadays, as we use many applications in all parts of life, it is also beneficial if the software at the company looks and behaves similarly to other ones that many people know.

### 1.3.3 Reliability

#### 1.3.3.1 Accurate, consistent operation

We accept people to make mistakes, and to a certain extent also incorporate correction and compensation of these mistakes into the business. But although we fail more often than a computer, we can also assess the situation after a failure, take the necessary steps for corrections, and we can also ask colleagues for help. In contrast, software only knows what it is prepared for (let’s ignore advanced artificial intelligence here), and its exact behavior is usually known to only a small group – so for most, it functions as a black box. Therefore, it is
important that the users always get the output they expect for a given input, otherwise, the trust in the system is severely damaged.  

### 1.3.3.2 Errors are easy to detect and handle

However much we strive to make perfect software, errors always occur – and also most of the time it is not economically viable to excessively fine-tune a system. Define a commercially justifiable quality – and communicate this openly with users so that expectations can be managed and program development can progress faster. In the meantime, any unexpected operation must be shown and reported in all cases to provide the ability of correction. We can send error messages to users and keep track of the program run in log files. It is important that the messages and log entries are clear and, if possible, give a suggestion on how to deal with the issue. The software needs to provide error-handling capabilities – it is the best if it has built-in tools with informative user interface, for example, it can list broken processes, and with appropriate privileges, an administrator can fix them to resume normal operation.

### 1.3.4 Speed

#### 1.3.4.1 From the user side

Wages make up a significant part of a company's costs, so it's important to make the most of the operators’ time. On the other hand, also the users want the software to work quickly, as there are few more frustrating things than doing overtime because of idle hours accumulated during the day. We have multiple ways to increase the speed of the user interface, but there are generally two approaches: optimizing the computational operations of the program or performing them in the background, in an asynchronous manner. The important is to find the balance: computing capacity is fundamentally cheap, while program optimization can be an expensive process. But still, while asynchronous operations can work in the background, as they increase in number, coordinating them can become a difficult task.

#### 1.3.4.2 From the developer side

Companies are constantly competing with each other, so rapid reaction is a strategic advantage. Therefore, agility is the dominant trend in software development nowadays, and the tools should support it as much as possible. Some process management systems follow the traditional software development approach: standard tools (code editors, IDEs) can be used, and the application is made available to end-users through code compilation and deployment. Others blur the boundaries between user and developer interfaces and provide a unified,

---

1 A very interesting talk by Cesar A. Hidalgo at Brain Bar 2019, “How Humans Judge Machines?”: [https://youtu.be/NQQ2CEqBe1Y](https://youtu.be/NQQ2CEqBe1Y)
typically web-based platform for both functions. Either way, it is important that new features can be utilized as soon as possible – after proper testing and quality assurance. While with the traditional approach typically the time of the repetitive compilation and deployment is the problem, when the development is done in a unified, web-based platform, the slowness of the system (e.g. because of the firewall-protected Internet connection) and quality assurance pose difficulties, due to the immaturity of new technologies.

1.3.5 Customizability
1.3.5.1 For business users
Development can be accelerated by making the software more flexible and transferring the task of fine-tuning to the business users. This requires that the configuration interface be transparent and easy to use, even without programming knowledge. In process management, the use of low-code and no-code platforms is increasingly common, where development tasks can be performed by a non-programmer, so-called “citizen developer”, with visual models and forms, instead of the traditional text-based coding. However, as some say, like this we ultimately get a domain-specific “programming language” – “Zero-code platforms force you to use their proprietary way of coding.” (Rücker, 2017) – so it's important to draw the line between configuration and development. In my opinion, by narrowing down the available toolset and using intuitively structured user interfaces for business users, we can make a good compromise between the saving on the sparse and expensive coder resources and the quality of the software.

1.3.5.2 Management of complex custom demands
Practice shows that, contrary to the marketing of some companies, skilled software developers will always be needed in a larger project. It is important that besides the most commonly used, simple configuration options, the tool provides proper support for complex tasks, and that those features can be used conveniently, in a standard manner. IT resources are extremely expensive, and this is especially true for the ones working with process management systems – thus it is a key factor in the success of a project, that developers' time is focused on delivering the required functionality rather than fighting the platform.

1.3.6 Integrability
Enterprise software should be able to work along with other IT systems at the company. Moreover, a process management program often plays the explicit role of an integrator between other applications – for example, it generates and sends correspondences about
investment results after the daily closing of the banking core system, or updates the data on the customer self-service portal that runs on the company's web server. Thus, it is expected to support many protocols and formats, and the exchange of information should happen fast and efficient. From a developer perspective, it is important that connection parameters, data mapping, and other general attributes can be set-up easily and quickly, but also, that special needs can be addressed. Some vendors offer ready-made system-specific components e.g. for managing SAP connections or the Microsoft Excel format – these can be of real value if they work out-of-the-box, that is, without customization, in most cases.

1.3.7 Change management

1.3.7.1 Versioning of program elements
Source code versioning is common and expected in software development, and there are several proven tools for this: SVN, GIT, Mercurial, etc. If the process management software is code-based, this expectation can easily be met – one has to choose a standard solution and use it properly. But with the proliferation of web-based “no-code” and “low-code” platforms, the problem, that has been long-solved in the coding world, comes up again – and the systems attempt to solve it by a variety of unique approaches. From the developers' point of view, it is essential to be able to experiment with ease, possibly without interfering with other programmers and to be able to compare different versions and revert to an earlier state if necessary.

1.3.7.2 Transition between process versions
Processes change from time to time – this may happen due to the introduction of a new product or a change in legislation. Such a modification must also be followed in the process management system, which typically results in a new program version. It is important that process instances started on an old version work consistently, even after installing the update. It may be expected that old instances continue according to the previous version – in other situations, fast and efficient migration to the new version is needed, without causing broken processes. Systems can leave the transition to developers, but it is much better if they provide a standard solution, or possibly an administration interface, to handle these changes.

1.3.8 Reporting
The effectiveness and efficiency of the processes need to be measured, and feedback must be given to the members of the organization so that the progress of the corporate goals be trackable and visible. The process management system has to ensure that metrics (i.e. KPIs)
can be assigned to different activities, and those can be handled and evaluated in a structured way. The software can meet this need using internal solutions, or it can cooperate with external tools – e.g. by providing exports for a data warehouse, or by connecting to a BI system. In many cases, the output of a report / analysis is the input of a decision point in the processes, so queries, aggregations, calculations have to run fast.

1.3.9 Security

1.3.9.1 External security

The most valued assets of a business are data and processes – besides human resources – so their protection is very important. Recently, there has been an increasing number of news articles about IT attacks against large companies and government agencies – the main users of process management software, so it is a prime requirement for such a system to provide adequate security against external attacks. It is important that the software is protected by default, that is, it provides a basic level of protection from the moment of installation, without any special settings.

1.3.9.2 Access management

In addition to protection against external attacks, it must also be ensured that users within the company only have access to processes and data that are relevant to their work to prevent abuse and ensure regulatory compliance. The software should be able to integrate with enterprise identity management solutions, such as Microsoft Active Directory, for centralized access control. It is important that changes can be done quickly and easily so that staff transfer, organizational change, or misuse can be managed effectively.

1.3.10 Vendor and expert support

Technology is constantly evolving, bugs can occur, so regular and seamless updates are a priority. In addition, there is often a lack of knowledge within the organization to quickly resolve a problem, so it is important to have the right support network for the software. In a proprietary system, vendors usually operate their own support service, which is either automatically provided through the purchase of the software license or is an optional extra. For open-source programs, enterprise-level technical support is usually only provided by consultancy firms, so in this case, it is important that those partners are available and that they offer appropriate contract terms and guarantees for a longer time period as well.
2 Modeling standards: BPMN, CMMN, DMN

2.1 Introduction

Process modeling, as well as its purpose, has evolved a lot in recent years. In the not so distant past models were primarily for just human readers – business and IT people –, who translated them to regulations and some special parts to computer algorithms. These days, however, it is also important that process engines can parse and execute the modeling output, preferably without further intermediate conversion. At the same time, models still need to have a representation that is easily understandable by humans.

A trio of standards, BPMN, CMMN and DMN, solves this by having well defined visual symbols and structure, and an XML-based technical representation as well. These now emerging – but already quite well established – notations are managed by the Object Management Group (mostly known for the UML) and are also called the “triple crown” (Object Management Group, n.d.). They provide solutions for all the general process and decision management tasks by default, but they are also extensible to support specific needs and to help engine implementation (Object Management Group, 2013, pp.55–59, 2016, pp.15–18, 2019, pp.63–64).

2.2 Business Process Modeling Notation (BPMN)

2.2.1 Introduction

BPMN is the oldest member of the standard trio with its first, 1.0 version released in May 2004. At that time it was developed by the Business Process Management Initiative (BPMI), which merged with the Object Management Group (OMG) a year later, in June 2005. The current latest version is 2.0.2 which was released in December 2013 and published by ISO as ISO/IEC 19510 standard (Trisotech, 2015).

It is used to formalize structured business processes where the sequence of the steps is known and repeatable, and branching points in the process (decisions, parallelizations) are known in advance. Besides the process steps, it also has elements and views to represent actors and interactions between them to provide a full picture of who does what in the organization. The primary goal of the notation is to be well understandable by all kinds of professionals in the organization from managers to analysts and technical people and thus bridge the communication gap between different areas of business (Object Management Group, 2013, p.1). The standard also defines an XML-based formal metamodel and guidelines for process
engines how to execute the models. By now there are many tools – both open and proprietary – which support certain aspects of the BPMN standard – e.g. the visual modeling or the execution.

2.2.2 Process modeling
Simply speaking, a BPMN process model is similar to a flowchart: it consists of circles, rectangles, diamonds, and arrows.

2. Illustration: Simple process
The circles are called events, with a thin border at the beginning is the start event, with thick border at the end is the end event. The rounded rectangle is an activity, more specifically here it is also a task. The arrows are called flows, this one, as it shows the order how elements are traversed, is called sequence flow. Usually, processes are not this simple, they have branching points, e.g. decisions.

3. Illustration: Process with decision
The diamond shape here is an exclusive gateway (see the “X” symbol), which means the process continues with either in the “Yes” or the “No” direction, but never both. Another common usage of gateways is to parallelize the execution.
This process shows that the main course and dessert can be prepared at the same time as they are after a parallel gateway (see the “+” symbol), however, they must be eaten one after another. The diagram also shows an important concept: there are two gateways, the first is a splitting one, while the second is the merging.

It is the easiest to understand how the process is executed using a token model: let’s say there is a car at the start event, and it follows the road defined by the sequence flows. (Freund and Rücker, 2016, chap.2.1.4) The exclusive gateways can be considered intersections, the car can either go left or right. The parallel gateways are trickier in this sense, as there the car would multiply. One instance goes to the left, the other to the right. At the merging parallel gateway these two instances wait for each other, and then reunite, and continue the journey again as one. But what happens if we use an exclusive gateway for merging (which is syntactically perfectly valid)?

The token from the dessert branch would arrive first to the gateway after 10 minutes. As the gateway is not a parallel one, it doesn’t wait for the other token and goes to the next step immediately. But the main course isn’t ready yet (it requires 30 minutes), so at that step, the process would produce an error!

In the above process, the exclusive gateway is not suitable for merging, but sometimes this is the desired solution: let’s say, during an insurance claim handling we request multiple
documents from the client. The documents can arrive one-by-one by post, then they need to be extracted from the envelopes, etc. It can be handled in parallel so that multiple colleagues can work on this part of the process. But there is only one clerk who can enter document data into the central computer system. It does not matter in which order the documents are stored, the only requirement is only that in the end, all the documents are present in the system. Here the nice continuous work is the ideal, instead of the waiting – when the clerk would do nothing for some time and then would get everything at once.

These were the main building blocks that show what needs to be done – but they do not tell anything about who does the work. For this BPMN has the pools and lanes.

6. Illustration: Lanes

This process has two actors: the Clerk and the Manager. If the payment amount is greater than 1000 EUR, manager approval is needed. The outer bounding shape is the pool, the two areas within this are the lanes. Lanes usually represent individual people or groups within the organization – actors whom work can be assigned to. A pool, in turn, represents a higher-level entity, e.g. the whole process – or an organization, when multiple companies are involved in the process.
The communication between pools is done with *messages*, dashed arrows symbolize these. It is important that only messages can be used for this kind of connection as it is not allowed for sequence flows to leave the pool boundaries – which is quite logical if we consider pools as separate entities (e.g. organizations), they cannot directly influence each other, they can just send messages and wait for responses. Pools can be collapsed if we are not interested in or do not know the detailed steps of a process within one entity.

8. Illustration: Collapsed pool

It is also allowed to collapse all of the pools and show just the message flows. However like that little information would remain on the diagram, and interdependencies could not be recognized anymore. To fix this BPMN 2.0 introduced the choreography-diagram (Freund and Rücker, 2016, chap.2.9.4) – not detailed here.
So far activities were depicted with simple rounded rectangles. This is good for a quick overview of the steps, but usually not enough for a complete picture – especially if the diagram needs to be executed by a process engine. To provide this required additional information, BPMN tasks can have types.

The main grouping criteria here is if they are executed by people or automatically by the workflow engine. *User* and *manual tasks* are executed by people – the difference between them is that the former is something done completely outside of the engine (e.g. finding a paper in the drawer) while the latter affects the execution of the process in the engine (i.e. the user needs to press the “Submit” button to continue).

*Service* and *script* tasks are some kind of automated program logic, the service task usually represents a web service call, while a script is something executed directly within the workflow engine. The *business rule task* is for evaluating decisions (see the DMN chapter later). The *send* and *receive tasks* are also technical ones, the engine executes them – they are for handling messages, e.g. writing to and receiving from message queues.

Then there are many tasks, the process can become very big, and if we model all the steps separately, the diagram would be unreadable. In these situations, a group of steps can be encapsulated into a *collapsed subprocess*.

Depending on the modeling tool, these subprocesses can be expanded when viewing the diagram, and thus the detailed steps can be shown. Besides this, as subprocesses create a new execution scope within a model, it is also common to use *(expanded) subprocesses* mainly for

9. Illustration: BPMN task types (in Camunda Modeler)

10. Illustration: Collapsed subprocess
this purpose – not just for hiding details. It is important to note that in BPMN 2.0 subprocesses are embedded intrinsically. This means that a subprocess can only occur within its parent process. It also cannot contain pools or lanes – so its steps should fit into the constraints defined by the parent process, e.g. all tasks must be assigned to the user-defined by the parent’s lane where the subprocess is inserted to. (Freund and Rücker, 2016, chap.2.8.2)

If we would like to model a global, full-fledged process, and reuse it – even multiple times – in a diagram or multiple diagrams, the call activity shape can be utilized to reference it.

The visual difference is that the call activity has a thick border while the subprocess is depicted with a thin border, just like tasks. The connection between the parent and the referenced process is looser than with subprocesses, so during technical implementation, data sharing (i.e. mapping of process variables) needs to be explicitly designed. It is logical, as global processes can occur at multiple places and in different parent processes – which may have a different context, i.e. variable names.

Now with the shapes covered above, a rather complex diagram can be drawn, but still, there is a concept to be introduced. In the real-world, during a process execution various things – either planned or unplanned – can happen. To handle these, BPMN has the events.
The most common ones are the start and end events – already mentioned at the beginning of the chapter. Here the process starts with a message start event and has multiple none end events.

There are also intermediate events – connected to other elements with a sequence flow (“6pm”, “Transfer completed”), and depicted with double borders. They can be catching (i.e. consumer, listener) and throwing (i.e. producer, trigger), which is shown with either an empty (former) or a filled (latter) symbol inside them. Timer events can only be catching events.

Another group is the attached events – placed at the boundary of an element (“30 minutes”, “Technical error”). They can be interrupting and non-interrupting, depicted with solid or dashed double borders, respectively. When an interrupting event occurs, it stops the normal flow of the process, and the execution continues only on the event handling branch. The non-interrupting event behaves differently, it generates a new token for the event handling branch, but in the meantime, lets the process continue on the main flow as well. This means for the process above, that if the “Validate payment data” task is not completed within 30 minutes, a notification is sent to the manager, but later the process also continues in the direction of “6pm” timer event. However, if the “Transfer money” steps fails with an error, the “Transfer completed” signal event is never reached, and the flow will end with “Payment failed” after the “Handle error” task is done. A catching error event is always interrupting by its nature.

12. Illustration: Events
It can happen, that multiple tokens are present in a process – there are multiple active paths –, but after a certain step, the whole process must be terminated.

![Diagram](image)

13. **Illustration: Terminate end event**

In the process above, the goal is to translate a word, but as soon as we have a translation – from any source – the process can end. To speed up the translation, we use two sources in parallel, so after the gateway, there are two active tokens at the two tasks. As soon as the first reaches a *terminate end event*, both tokens are consumed – with a simple none end event only the corresponding branch would end first, and then the task on the other branch would complete as well.

These were the basic elements and concepts of BPMN. There are a lot of other interesting details and solutions in the standard, but these are enough to understand most of the diagrams and to start working with a BPMN tool.
2.2.3 Sample process

This is a real world example of a business trip booking at a consulting company. As a first step, a need arises that a consultant has to go to a client. Then he / she provides the details of the trip: the requester’s name, the destination and the from-to dates. Then a clerk validates the request, making sure that the destination is valid, and that the date interval makes sense (i.e. there is no typo). If everything is fine, the clerk estimates the expenses – plane ticket plus hotel costs – then this estimate gets either approved or rejected. When approved, the final step is the booking of the travel and accommodation – which will be a case, see the CMMN chapter.

2.3 Case Management Model and Notation (CMMN)

2.3.1 Introduction

Usually, it is considered the best, if processes can be modeled in an explicit, precise way, where steps follow each other in the given order at all times – that’s what BPMN is good at. However, it often happens that the workflow cannot be formalized this strictly, the decisions depend on multiple – too many – factors, that are not possible / not worth to cover. It also happens often, that operators who work on the process cannot describe all the steps, they just do what they “have always been doing”. For these unpredictable parts of processes, case management has been developed. While this phrase can mean slightly different approaches, depending on the vendor who provides such a solution, and it is constantly evolving, CMMN
is an accepted standard approach, with the 1.0 version released in May 2014. (CMMN, 2019) The current version, 1.1, is from December 2016. (Object Management Group, n.d.)

2.3.2 Case modeling

CMMN shares some similarities with BPMN regarding its building blocks: tasks and events can be found here as well similarly. It also defines graphical representation, XML-based metamodel and execution semantics. The main difference is that while BPMN is imperative (clearly defines how steps should be executed), CMMN is declarative: it just imposes constraints, which restrict what state changes are allowed and what are not – but it says little about when and how to initiate those changes. (Freund and Rücker, 2016, chap.5.1) Let’s take a simple example for a case: risk assessment at an insurance company.

Let’s take a simple example for a case: risk assessment at an insurance company.

The case itself is symbolized with a folder – and while a BPMN process can be modeled without using a pool as a container, the folder is always necessary for cases. Within this case, there are multiple tasks. The main one, that is started automatically, is the “Assess risk” human task, which is also required – this is shown by the exclamation mark –, so the case cannot be completed unless the risk assessment is done. The other two tasks are manually activated, shown by the play icon – which can be translated as: they can be enabled if needed, but they will not start automatically. Additionally the “Call client on phone” task is also repeatable, shown by the hash symbol, so we can call the customer as many times as it is needed to solve the case – however, we can only run “Get client rating from external provider” once. The decisions, whether to activate the two available tasks or to call a customer just once or multiple times, are entirely up to the operator.
Before going further, an important concept must be introduced: the life cycle in CMMN. While in BPMN the token concept is used to keep track of which task is the next that needs to be executed, in CMMN the life cycle states show what actions can be done by the operator or the engine at a given moment.

First, when a task gets created (e.g. upon starting the case) it becomes available. It only means that it is present, and it will be considered when checking dependencies – but let’s say a user cannot do anything with it yet. Then if its dependencies are met, it can become either enabled (and a user can activate it later) or immediately active – based on the activation setting. An enabled task can then be disabled and re-enabled, while an active task can be completed or terminated. While both completed and terminated states are final, their meanings are different: the former says that the task has been done successfully, while the latter implies, that the task has been interrupted and canceled.

16. Illustration: The life cycle of a task in CMMN (Freund and Rücker, 2016, fig.5.3)
The dependencies and conditions are symbolized with diamond shapes (sentries), which are always attached to another element – these are called entry criterion and exit criterion.

17. Illustration: Sentries and exit criteria

The former – depicted with an empty diamond – “guards” an element from being enabled, while the latter – solid diamond – models when to terminate a scope (e.g. task, case). They both have two types of conditions: one waits for the completion of a previous element (on part), while the other has a data-based definition (if part). In the above case, the “Update client record” task can only be started after the “Assess risk” has completed. Similarly, the operator can only do the “Call client on phone” task if telephone communication is allowed by the client. It is also possible to define conditions of both types for a sentry.

When the entry criteria are met, the state transitions from available to enabled. It is important to note, however, that this change happens only in one direction – meaning that if at a next point of time the entry criteria evaluate to false, the state does not get changed back to available.

The mechanism of exit criteria is similar – changes the state from active to terminated – but it’s probably harder to understand the usage. Let’s imagine a situation, where “Get client rating from external provider” can take multiple days – after all, it’s a complete BPMN process invoked from the case. The operator initiates the process, but little after that gets a phone call from a colleague, that the client is a known customer, and can be trusted – so the “Assess risk” task can be completed without waiting for the external data. Then the “Update client record” gets done as well, and the case could be closed. However, there is still an active task – “Get client rating from external provider” – which no longer has any relevance. The exit criterion says exactly that: as soon as the “Update client record” is done, the case can end. It is worth noting that this kind of end is not a “normal” completion, but a termination. This
can be important, as engines may handle the two statuses differently – e.g. only consider the completion as a successful ending and does not run certain actions upon termination. (Smirnov, 2016)

In a more complex case, it is usually required to group tasks together, and it can also help readability if certain points in the process can be emphasized in the diagram.

For grouping tasks, stages can be used. They behave similarly tasks from multiple aspects: criteria can be attached to them, they can have decorators like manual activation, repetition, etc. However they form a new scope, the tasks contained by a stage are only created (become available) when the stage is activated. In the above example the “Process payment” and “Consult the manager” tasks only get created when the risk assessment completes with a positive result. It is important to note, that stages are not “phases”: there is no defined order between them and tasks can be active in multiple stages at once. Stages can also be nested within one another according to the standard (however engines may have issues handling multiple levels). (Freund and Rücker, 2016, chap.5.2.9)

The oval shapes, “Risk is acceptable” and “Risk is too high” are called milestones, they mark important points – here intermediate results – in the process. They always need a sentry, as otherwise, no evaluation would happen there (for simple notes annotations like “Phone call is allowed” can be used). They are usually not necessary, in the above case they can simply be
deleted after sentry conditions (risk is acceptable or not) are moved, but they can greatly help in understanding the diagram.

Finally, CMMN also has events, just like BPMN – with timer and user types. Here, when a user triggers the “Claim canceled” event, the case gets terminated – similarly to when the risk is too high.

There are other possibilities and elements defined by the standard, e.g. discretionary tasks and a planning phase during case execution (Freund and Rücker, 2016, chap.5.2.16), however, the above parts are the best supported by engines, and for a good basic understanding of most diagrams, this should be enough.

2.3.3 Sample case

Continuing the example from the BPMN chapter, the actual travel and accommodation reservation during the booking process is modeled as a case. The choice was this, instead of a more explicit BPMN model, because in the end, it does not matter if the plane ticket or the hotel is booked first, however, the clerk can have experience which is preferable to start with,
according to a destination. Let’s say, in Spain, there are many free hotel rooms during the winter, but only a few planes go there – while to Germany there are plenty of travel options, but hotel room prices can go up fast in the holiday seasons, so it’s better to secure a room first. Even with this expert knowledge, prices may get too high and accommodation or travel needs to be canceled and re-booked (it can be done multiple times, see the repetition decorator on the stages), or an expense limit raise must be requested from the manager. In the worst case the clerk can also decide to fail the case – this status will be propagated back to the BPMN process and the requester will be informed.

For a comparison, a process with similar (not the same!) state space would be modeled in BPMN as below. Because the two standards have different elements and rules (e.g. in CMMN a user can manually terminate a stage if it is not restricted), an exact rewrite is usually not possible from one to the other, but the main logic can match.

2.4 Decision Model and Notation (DMN)

2.4.1 Introduction

DMN is the third and youngest member of the standard trio with its 1.0 version released in September 2015. The current version is 1.2, released in January 2019. (Object Management Group, n.d.) Its primary goals are to model human decision-making, and to model requirements for and implement automated decision-making. (Object Management Group, 2019, chap.5.2) While it is also possible to represent decisions directly in BPMN or CMMN – i.e. using tasks and gateways or sentries – a logic with multiple conditions could require an overly complex diagram and could become overwhelming and hard to understand for business
users. By using a specific tool, a more concise and more transparent representation can be achieved. Also, like this, the decision logic can be separated from the process models, and released separately, which – as branching conditions (e.g. who is a gold customer) tend to change more often than the process steps – can also result in a faster release cycle thanks to the smaller updates.

2.4.2 Decision modeling

When representing multiple conditions and results, it comes naturally to organize them into some kind of table – this is also a main building block in DMN.

<table>
<thead>
<tr>
<th>Discount</th>
<th>Customer level</th>
<th>Discount percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>“Gold”</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>“Silver”</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>“Bronze”</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>“Regular”</td>
<td>0</td>
</tr>
</tbody>
</table>

21. Illustration: Decision table

These decision tables have at least one input column (here: “Customer level”) and one or more output columns (here: “Discount percent”). Conditions can be as simple as a string comparison, but they can also be more complex expressions using FEEL (Friendly Enough Expression Language) – or another scripting language supported by the decision engine.

Furthermore, rules (rows in the table) have priority, shown by a number in the first column, and every decision table has a hit policy, which is indicated in the top left corner – these determine how rules are evaluated. Here the hit policy is “U”, that translates to unique, which in turn means, that only one rule is allowed to match. Other options include e.g. first (first matching rule determines the result) and priority (the matching result with the highest priority gets returned). There is also a special policy, collect, that can either return multiple results or – if aggregate functions are used – can calculate the summary, minimum, maximum or count of the matching results.

While decision tables can be powerful by themselves, sometimes they cannot express everything, or it is better to modularize a complex logic. For these situations, the standard defines the decision requirement graph (DRG), which is visualized as a decision requirements diagram (DRD).
The above diagram shows the elements and steps that are needed to calculate the discount. First, the customer level needs to be determined by the “Customer level” decision, which in turn requires two input data items, “Summary of ordered value” and “Marketing value”. Then, the discount can be determined using the decision table above.

When a decision table is invoked from a BPMN or CMMN model, the engine checks the corresponding requirements graph and traverses it to provide all the necessary inputs for rule evaluation. So here, when the “Discount” table is invoked, the engine would automatically evaluate “Customer level” first. This kind of automatic graph traversal also means, that as long as the linked table has the same identifier, all other parts of the DMN model can be changed without modifying anything in the calling BPMN or CMMN models – resulting in a loose coupling and thus improved maintainability.
2.4.3 Sample decision

In our booking example, the expense estimate approval is partially automated, using a decision logic.

The “Determine Approval Level” step invokes the decision table below, which returns what action to take, based on the expense limit estimated by the clerk in the previous process step.

<table>
<thead>
<tr>
<th>Booking Approval Action</th>
<th>Approval action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>1 &lt;= 50000</td>
<td>&quot;Automatically Approved&quot;</td>
</tr>
<tr>
<td>2 &gt;= 500000</td>
<td>&quot;Automatically Rejected&quot;</td>
</tr>
<tr>
<td>3 &gt;= 0</td>
<td>&quot;Manager Approval Needed&quot;</td>
</tr>
</tbody>
</table>

According to the table, if the estimated expenses are low (below 50 000 HUF), the limit can automatically be approved. If they are too high (above 500 000 HUF), the logic assumes, that it must be a mistake, e.g. a typo, so the estimation will be rejected. In between those values, a manager needs to verify, whether the estimate is appropriate or not. Note that the table uses the first hit policy, that is why even though a value like 600 000 HUF could match two rows, the result is one action every time.
3 Camunda

3.1 Overview

3.1.1 Introduction

Camunda is a workflow and decision automation software. It has both a – fully functional – open source Community Platform and an Enterprise Platform, extended with some advanced features and contractual support. The initial release was in 2013, after forking Activiti (which is now sponsored by Alfresco, best known by their document management software). Camunda is mainly developed by a likewise named consulting company, founded by Jakob Freund and Bernd Rücker in 2008. The company has its headquarters in Berlin, and it also has offices in San Francisco and Denver. The platform is used by many organizations from small to large ones, including Allianz, AT&T, NASA, T-Mobile, and Universal Music. (Camunda, n.d.; Camunda, n.d.)

Camunda is a Java-based software – just like most of the process management platforms nowadays. But on the contrary to many other players, it tries to be developer-friendly by keeping the usual software development workflows and tools that enterprise developers are used to: standard version control, support of using the best known IDEs and test tools, well written and open documentation. The Camunda developers take this seriously and consider it one of their selling points. They also widely advocate this approach instead of the low-code one – which rather focuses on the business users’ wishes and needs. (Freund, 2013; Kemsley, 2014; Rücker, 2017, sec.2) They clearly differentiate their product from the big suites like Pega and Appian, and target a different segment: lately Jakob Freud, the CEO of Camunda, has also turned down the request to be listed in Gartner’s famous annual Magic Quadrant for Intelligent Business Process Management Suites report. (Freund, 2019)

This is why I choose Camunda as one side of the comparison for this thesis – also representing similar tools like Activiti, Flowable and jBPM to some extent. The latest stable version of the Community Platform is 7.11.0 now, so this will be the basis of the following chapters.
3.1.2 The platform

To get started with Camunda it is the easiest to download the pre-configured Community Platform – the package includes an Apache Tomcat webserver with already deployed web applications. As it is a Java-based system, it requires a Java Runtime Environment or better, a Java Development Kit to be installed. After the start, a welcome screen appears in the default browser, showing the main applications, an example process, and some useful links.

The default end-user interface is called Tasklist, which shows the work items need to be processed, and also presents details about the tasks, such as the form attached to it, a history, a diagram to see where the task is located in the process.

For configuration and monitoring, the Admin and Cockpit applications can be used. The former provides some minimal system configuration abilities, and the user management is done here – unless an external tool, such as Active Directory, is configured for identity management. The Cockpit, on the other hand, provides an overview of the installed and running processes with an exceptional user interface, where steps are displayed on diagrams, process variables can be modified, and execution can be paused in case of an error – just to name a few of the interesting features.

Camunda uses standard BPMN, CMMN, and DMN models – so any compatible tools can be used for creating basic diagrams and decision networks. However, as the standard provides many elements that are not too commonly used, and thus not supported by the engine, care must be taken during modeling. On the other hand, vendors can also decide to extend the models with their own elements and attributes, to improve the practical usability, and simplify the development – Camunda also has such very useful extensions. As a result, it is best to use the Camunda Modeler if the target platform is Camunda. This may sound like a vendor lock-in, and it must be admitted, that to some extent it is for real, mostly on the technical level. However, models can still be opened and viewed by different tools, which helps the cooperation between business people – who want to use their favorite editor, but do not need the fine technical details of the model – and developers. (Camunda also has a tool called

---

3 Camunda Community Platform download page: [https://camunda.com/download/](https://camunda.com/download/)
4 Camunda Modeler download page: [https://camunda.com/download/modeler/](https://camunda.com/download/modeler/)
Cycle for synchronizing diagrams between different software\(^5\), but according to the activity in the GitHub repository, it seems to be neglected nowadays\(^6\).

For code development – which is needed for validation of process steps, integration with external systems, assembling advanced input forms, automated testing and so on – the traditional Java development tools can be used, e.g. Eclipse, IntelliJ, NetBeans.

### 3.1.3 Starting a project

Although, simple processes models can directly be deployed to the engine (using the Cockpit application’s Enterprise version (1 Camunda Basics : Getting Started, 2018) or the REST API), sooner or later we will require some code when working with Camunda. So it is the best to set up a project in the first place, using the Process Application (Servlet, WAR) Maven archetype that can be found in the documentation\(^7\). For the initial setup and introduction of the main features, it is worth watching the excellent video tutorial series of Niall Deehan, linked on the Camunda website\(^8\). As this thesis focuses on the process management features and not on general Java development, I choose to go with the simplest approach, and followed the steps described in the documentation, using the Eclipse IDE.

After initiating the project, the application can be built using Maven install (right-click on the project folder, Run as / Maven install). If everything is fine, at the end a “BUILD SUCCESS” message appears in the console, and a new war file is created under the target directory. Using the pre-configured Community Platform’s default Tomcat server, the war needs to be copied to the webapps directory, in my case it is: [...]/camunda-bpm-tomcat-7.11.0/server/apache-tomcat-9.0.19/webapps – then it will be automatically deployed. To check if the deployment was successful, the Cockpit application can be used. Under the More / Deployments menu item, the date and time of the latest deployed version can be assessed.

---

5 Camunda Cycle webapp in version 7.5: [https://docs.camunda.org/manual/7.5/webapps/cycle/](https://docs.camunda.org/manual/7.5/webapps/cycle/)
6 Camunda Cycle GitHub repository, code frequency: [https://github.com/camunda/camunda-cycle/graphs/code-frequency](https://github.com/camunda/camunda-cycle/graphs/code-frequency)
8 Learn Camunda with Niall: [https://camunda.com/learn/videos/](https://camunda.com/learn/videos/)
The steps above should look rather simple for an experienced Java developer – but most likely business users would have a hard time following them. It is alright, however, and in line with Camunda’s goals: software development part should be done by developers, business users should focus on process and decision modeling using the graphical tools.

3.2 Process modeling, data handling

3.2.1 The modeling tool

For modeling, I used Camunda Modeler as this plays the best together with the Camunda engine.

25. Illustration: Deployments in Camunda Cockpit

26. Illustration: Camunda Modeler
It can handle BPMN, CMN and DMN formats, and has a simple, clean interface. The tool is built using web technologies, the libraries are available on bpmn.io site. There is also an online version, Cawemo, for collaborative editing (like Google Docs).

As the tool follows the standards, high-level modeling is pretty straightforward: new elements can be added from the toolbar on the left or using the mini icons that appear next to a selected shape. The editor helps the user with validations and tries to keep the diagram standard compliant. From the technical point of view, however, the most interesting part is the Properties Panel on the right, that shows the attributes and behavior of an element. These are either standard fields, like Id or Name, or Camunda specific extensions like Listeners.

According to my experiences, the usage of the editor is very easy to learn, even beginners with little technical background could create fairly complex diagrams after a couple of hours. It is also important to note that the Modeler is a standalone application that runs on Windows, Linux and Mac OS, and requires no installation – so it is easy to make it available to users, even in a fairly strict enterprise environment.

### 3.2.2 Coverage of the standards

The Camunda developers’ main focus is BPMN, and as a result, the engine has almost full coverage of the BPMN 2.0 standard. The parts which are not yet covered are either rarely used, or have alternatives (like complex gateways). The most notable unsupported feature that I encountered is the ad-hoc marker, but this can also be substituted with other BPMN elements (like attached events) or using CMMN for unstructured process parts. In the end, it can be said, that for structured processes – which are the most common – the engine has great support, and also works reliably. With DMN and CMMN the picture is not quite this complete – but they are still usable with some restrictions.

Regarding DMN the documentation writes: “Currently the Camunda DMN engine partially supports DMN 1.1, including Decision Tables, Decision Literal Expressions, Decision Requirements Graphs and the Friendly Enough Expression Language (FEEL)” – which covers the most important parts.

The state of the CMMN implementation, on the other hand, is more complicated. The basic elements are covered here as well according to the reference, however, some important parts are either missing, or they are not completely error-free. The most

---

10 Cawemo: [https://cawemo.com/](https://cawemo.com/)
notable missing elements are planning tables, discretionary tasks, and events. And while the planning phase can most of the time be substituted in practice by using manually activated stages and tasks, a timer event would sometimes be required for a real-world case, and there is no nice alternative for that. It is also worth noting, that CMMN tasks are not yet covered by authorization (Shcherbakov, 2018a) (Lindhauer, 2016) – so a custom mechanism needs to be implemented to provide an appropriate level of security if this is a requirement.

To see an error, let’s take the simplified version of the Booking Reservation example case.

During the “Book Accommodation” task the user sets accommodationCost variable. When the task is submitted, the If Part of the milestone is evaluated – to true – and this should terminate the case. However, because of some internal issue, an exception occurs:

An error happened while submitting the task form: Cannot submit task form 2bb8119f-e12a-11e9-9853-0242933ed500: ENGINE-05011 Could not perform transition 'complete on case execution with id '2bb8119c-e12a-11e9-9853-0242933ed500'. Reason: The case execution must be in state 'active' to complete, but it was in state 'terminated'

What I think happens here is the following. The task form submission triggers the task completion processing, during which all the dependencies are evaluated. So the engine notices the milestone condition, evaluates it. Then it terminates the case – which also involves terminating all stages and tasks – thus the “Book Accommodation” task, as well as others, goes into “terminated” state. Then – as all dependencies are handled – the task completion finishes, so the engine wants to set its state to “completed”. But as it has already been set to “terminated” this results in a collision: which is the valid state?
A similar issue has already been reported and fixed (Zilske, 2017) – probably the multiple levels (case - stage – task) makes a difference in the example above. But, unfortunately, it gives me the impression that even for relatively simple models extensive testing needs to be carried out, as the engine is not mature enough yet.

In defense of Camunda: the case management approach, as well as CMMN, is not fully developed yet, and there has also been misunderstanding between vendors lately (e.g. with manual activation play symbol used counter-intuitively (Freund and Rücker, 2016, chap.5.2.4). But most importantly, due to the declarative and permissive nature of CMMN, it is extremely hard to create algorithms that work in all situations.

3.2.3 Handling process and business data

As processes are generally about collecting some data (even if it’s just a true / false value for an approval), it is an important aspect of the design, how this information is stored and retrieved. At gateways, the engine takes a given path based on the gathered or calculated information, and in the end, KPIs are measured, reports are built. Camunda stores the data items, that are used for transition conditions, as process variables.

Variables – just like in modern programming languages – are assigned to scopes, which can be embedded to each other – with parent-child relationships. A variable assigned to a parent scope (e.g. process instance) is available in child scopes (tasks) – unless it is overridden there. However, if it is set directly to the child scope (as a so-called local variable), it will not be accessible by the parent. In other cases, when the variable is not explicitly declared local, it is propagated to the parent scope. Variables can have various primitive types (e.g. boolean, integer, string), and they can as well store files, JSON and XML documents. There is also an object type that represents a custom Java object. (Camunda, 2019ab) So in Camunda, variables are quite versatile, and the concepts here are very much in line with the Java concepts – which is not surprising as the engine is implemented in Java.

There are situations when shared components with different variable sets are embedded in models – e.g. a process called using a call activity shape. In these cases, variables need to be matched between the outer and inner scopes. They can have different names, plus the called component doesn’t need to get all information from the caller in most cases. This is similar to when a function is called in a programming language like Java: it has a definition for the input arguments and also returns specified value(s) – this shows the interface, the caller needs to obey.
The mapping can occur between two variables by their names, or it can use an expression – for example when type conversion is needed.

Another important aspect of data handling is to know, where items are physically stored. Camunda has a nice, normalized way of storing process variables – in a database table (called ACT_RU_VARIABLE). (Camunda, 2019j) This means, on one hand, that it is easy to find the required information, but, on the other hand – as every process instance has its data in one place – maintaining good performance and creating management reports from this source can become hard. That is why it is advised to handle “business data” (like customer information) separate from the process variables – using custom storage settings and logic – and use the variables only for transition conditions required in the process. (Ghanem, 2016) This is different from some other BPM tools, which provide an ability to create custom data objects and database table mappings – but as with other similar questions, Camunda’s choice is to leave the design of such business data schema to the developers.

3.3 User interface, REST API

3.3.1 Tasklist, the default user interface

As a default end-user interface, Camunda has a simple application – called Tasklist – which shows tasks in the system and also provides the ability to process them. The operator can fill a form linked to the task, view the history of a task, check where the task is located in the whole
process on a diagram. One can also set follow-up and due dates, and manage the owner (assignee) of the tasks if proper privileges are granted.

The interface consists of three main parts. On the left, there is a list of predefined filters – saved search criteria for tasks. The conditions can be edited, and the availability of a filter can be restricted to certain users or groups. In the middle, the results for the selected filter are displayed. After choosing a task, the details appear in the right pane. Basic actions, like starting a process, navigating to the user profile or switching between other Camunda webapps (e.g. Admin, Cockpit) are accessible from the header.

Task forms – that are the main interface to interact with user tasks – can be defined in multiple ways. Embedded, custom-built, HTML-based forms can be referenced – this provides the most option for customization, but it also requires the most work – as the markup needs to be created by hand. A simpler, but also less flexible option is to use a generated form and define the fields in the BPMN XML as extension elements, using the Camunda Modeler.

29. Illustration: Camunda Tasklist
Besides the above options, external forms can be linked (which are implemented in another application), and by default, Camunda displays the process variables, if there is no explicit form definition set. (Camunda, 2019ai)

The front-end of the Tasklist is written using AngularJS (Camunda, 2019f), while the back-end is based on the REST API, wrapped around with a thin layer – mostly for authorization handling. Overall, this is a simple application, and it also has little customization possibility. While there is an ability to write custom plugins for the user interface (Camunda, 2019v), it is quite limited, there are only a couple of points where the application can be extended. There is also no possibility to change the layout or to hide some default elements. Closed (completed, terminated) tasks cannot be displayed in the Tasklist – as they are only available using separate REST endpoints for historic instances, and this functionality is not implemented (Shcherbakov, 2018b). Furthermore, while managing a case, it is not possible to see available, but not yet enabled tasks, or to manually complete a stage or the case.

Having said all this – in my opinion – the interface can be used in some simple situations, but it is rather only good for initial business demos. The reason for this most likely is that – as with other things – Camunda favors the software engineering approach instead of providing a big, customizable user interface: they suggest to build a custom front-end if the default is not good enough for the project at hand (Müller, 2018). Fortunately, the engine has a good and well-documented REST API, which can be used as a basis for such user interfaces.
3.3.2 REST API

With the REST API, almost all of the operations can be done in Camunda, from querying tasks and submitting forms to admin operations, such as deploying a new resource or creating a user (Camunda, 2019u). In fact, the admin interfaces use it – besides Tasklist – as well. While there are other options to interact with the engine or to get information from the database (Java API or direct database queries), the REST API is a standard, well supported and loosely coupled way to do that – so in a lot of cases, this is the best choice.

However, when using it, some considerations must be taken. As the endpoints try to be general, they are built for specific tasks – instead of e.g. a database query, where multiple data sets (tables, views) can be joined, and a combined result can be returned at once. When doing complex things, like listing all variables available in all tasks for a specific process type – or creating a friendly user interface – it can result in many separate requests, big amount of data transferred, and serious post-processing needs to get the desired data set in a usable format.

For the Tasklist user interface, it requires 8 requests (just to the REST endpoint, not counting the stylesheets, scripts, images) to build a quite minimalistic UI. The measurement on the picture above was taken on a fresh installation, on localhost, with only 5 tasks in the whole system – but if the machine is farther away and the network latency is bigger, the numbers can get worse. Up to a certain degree, caching can be introduced – but nonetheless, this is something that needs to be monitored during development.

To mitigate these kind of problems, the development of a community extension was started, which added GraphQL 11 to Camunda. This is an abstraction that can be used to limit data returned by REST endpoints, and also to merge different sources. However, it looks like, that

---

11 GraphQL, a query language for your API: [https://graphql.org/](https://graphql.org/)
the development has stopped since the last commit in the GitHub repository is from 8 October 2017.

Besides all the issues – which are rather general, and not specific to Camunda – this is a REST API that a lot of other software would envy.

### 3.3.3 Custom user interface

To assess how complicated it is to build a custom UI – and to experiment with how an interface can be created that suits CMMN better – I created a minimal, proof of concept front-end in Vue.js using Semantic UI components. The application is solely JavaScript-based, utilizes the REST API, and has no authentication implemented. It is similar to the Tasklist, with an even more minimalistic outlook, but also with some additional functionality.

There are three menu items. *Tasks:* which lists the tasks and cases, displays their details and the task form, plus provides an action bar for case handling. *History:* that shows completed process instances with their variables. And *New Booking:* which starts a new Booking process with a generated business key, that links together the main process and the case, plus other processes called from it (Lindhauer, 2018).

With these prior experiences, I could create a similar, new UI in around 3-5 days – but creating a fully functional, business-ready application with authentication, form validation, nicer date formatting, etc. would be measured in weeks. While this means, that the client can customize everything, this is not always a (real) requirement, sometimes it is more important to show a minimum viable product in a short time. So – in my opinion – this is one of the most important considerations when deciding for or against Camunda, in comparison with other tools providing more advanced out of the box user interface or a good UI builder. It is also worth to mention that some projects do not even need a UI – in these cases, Camunda has a clear edge over others with its lightweight engine, great versatility, and good APIs.

### 3.3.3.1 Case management interface

As the custom proof of concept application is quite simple and straightforward, it is easier to try it, then to list all the details here. However, there is one part I would like to highlight: the case management. During development, I got inspired by various sources including CattleCrew Case Management UI and examples by Camunda.

---

12 GraphQL for Camunda BPM: [https://github.com/camunda/camunda-bpm-graphql](https://github.com/camunda/camunda-bpm-graphql)
13 Vue JS: [https://vuejs.org/](https://vuejs.org/)
14 Semantic UI: [https://semantic-ui.com/](https://semantic-ui.com/)
My first observation is that in Tasklist it’s not possible to act with cases that have no open tasks – they aren’t listed anywhere. To solve this, my approach is to create a separate list (Cases tab on the left) which shows active case instances.

Then – even if a case has active tasks – there can also be *manually activated* tasks that need to be displayed to the user so that they can be started if needed. In addition, stages and the whole case can require manual completion, unless they are marked as *auto-complete* or terminated by *exit criteria*. In my opinion, these actions are at least as important in CMMN as task completion, and giving the ability to the user to utilize them can provide a great amount of additional flexibility – which is the main reason for using a case management approach. My solution for these is the action bar at the bottom of the screen, which is displayed when a case – or a task from a case – is selected by the user. The bar itself symbolizes the case instance, inside them the groups with a folder icon are the stages, while the cards with no inner content are the tasks. Play icon means activation, stop icon means completion and the fast forward button is for terminating the case. Active tasks are also clickable (links with blue color), by selecting them the corresponding task form is shown.

32. Illustration: Custom end-user interface for case management in Camunda

17 "Underwriting" with BPMN, DMN, and CMMN: [https://github.com/camunda-consulting/code/tree/master/one-time-examples/underwriting](https://github.com/camunda-consulting/code/tree/master/one-time-examples/underwriting)
3.4 Developer tools

It is a rather simple topic with Camunda: the tools are the same as for any other Java-based application from IDEs to automatization. Camunda also provides integration with the widely used Spring Framework and Spring Boot to simplify custom code development. It supports multiple server environments and has great APIs for reading model files to interact with the engine. (Camunda, 2019ah)

The fact that Java developers can feel themselves at home while using Camunda is a very important point, that is often overlooked: while programmers like learning new things, they are also proud of their profession. If they can excel from the beginning of the project, they will like it – otherwise, they can get frustrated and can pull back the development. If a company has a good Java team, it is worth considering this, even if some other platforms offer more functionality, or look better at first.

In addition to the well-known environment and tooling, being so close to the code level provides another important advantage over low-code platforms: automatic testing can be used extensively (Camunda, 2019ag). Combined with the standard-compliant modeling, this can result in a highly reliable, automatically validated application, which may be a requirement in some business segments.

3.5 Security

3.5.1 Authentication and authorization

It depends on the usage of Camunda whether authorization is required at all. For example, if the engine is embedded into another application, then most likely the outer layer takes care of the necessary security checks, and Camunda can skip the costly process of redundant authentication and authorization. However, if the engine is accessed directly – using the REST API or the webapps (e.g. Tasklist) – authorization needs to be configured.

During the access of a secured resource, multiple steps take place. First, the operator needs to authenticate himself: prove that he is who he claims to be. This usually happens by providing a secret password that the engine can compare to a stored value. Then, if the test passes, the engine checks whether there is an identity for the authenticated entity (i.e. operator), that is authorized to interact with the resource in a certain way, defined by permissions. Camunda distinguishes between two types of identities: users and groups. A user can have multiple groups assigned, and multiple users can belong to one group. A resource can be e.g. a process instance or a task, and permissions include create, read, update, delete, plus some special
ones. The authorization check is the matching between the resource-operation and identity-permission pairs using well-defined rules. (Camunda, 2019a)

Out of the box, Camunda has two types of services implemented for handling identities (i.e. users and groups): the Database Identity Service and the LDAP Identity Service. The former provides a writable repository (so identities can be managed via the Admin webapp), while the default LDAP implementation only supports read-only mode: by using it, users, groups and their relationships need to be managed outside of Camunda. It is important to note that custom identity providers can be added to the system, one good example is the Keycloak Identity Provider Extension 18. (Camunda, 2019r)

### 3.5.2 Dynamic permission handling

In process management, there are often situations when authorization needs to be based on dynamic properties, e.g. a process variable or the assignee of the task. However, in Camunda it is only possible to define permissions using static values, so an expression like “if cost < 100 then UPDATE else NONE” is not supported. This means that, in general, condition-based authentication needs to be implemented using a custom logic – e.g. granting permissions to a user before a task, and revoking them after it. On one hand, this results in more work to come

up with a custom solution, if required, but on the other hand, it also means that the default authorization handling remains clean and easily understandable – which is a great value.

There is also one exception to this, for the most common situation: the user who is somehow related to a task (e.g. by being the assignee) gets access to it by default when the task starts, and this temporary permission gets removed as the task ends. The level of this access right can be controlled with the process engine configuration. (Camunda, 2019a, sec.Default Task Permissions)

### 3.5.3 Other security considerations
Camunda tries to follow security best practices and also offers multiple configuration options and features so that the application can be tailored to the need of the project, environment, and organizational rules. It is possible to define password policies, throttle login attempts, define the allowed pattern for user and group names, and control script execution in processes and decisions. There are also default ways for CSRF (Cross-Site Request Forgery) prevention, and to enhance XML security. Besides these, it is also important that the web and database servers, network interfaces are properly configured and secured (e.g. using encryption in the storage and during data transfer as well). (Camunda, 2019ae)

It is very important to note, that Camunda is a modern software, and most of its parts are open source. It follows standards, has good documentation and community, provides nice APIs for the programmers to use, customize and extend the platform. In my opinion, these are the most important values when it comes to security – bugs can always occur, but if the software is generally well-built, they can be tracked down quickly, and the damage can be minimized.

### 3.6 Integration with other systems
On a BPMN diagram, an integration point is usually symbolized with a service task – which has an implementation in the background. With Camunda, this implementation is usually some kind of Java code that connects to an external web service. There are two special cases though, which can simplify development and make the process model easier to handle: external tasks and connectors.

An external task is much like a user task, regarding its lifecycle: first, it gets created, then it is assigned to a list – here it is called a topic – and then a worker picks and completes it. The difference is that here the worker is an external application that connects to Camunda, using the process engines’ API (Java or REST). This pattern provides the usual advantages of a
loosely coupled producer-consumer relationship: scalability, maintainability and the ability to use different technologies for workers. (Camunda, 2019o)

The other specialized case for integration is the connector. By default, Camunda has two simple implementations for REST and SOAP services (Camunda, 2019g). The main advantage here is that no custom Java code needs to be written and maintained, the connector parameters can simply be added to the BPMN definition (as an extension) in Camunda Modeler.

34. Illustration: REST connector configuration in Camunda Modeler

It is also possible to create custom connectors, one good example is the mail connector available as a community extension 19. (Ossler, 2016)

19 Mail connectors for Camunda BPM: https://github.com/camunda/camunda-bpm-mail
3.7 Version control, artifacts, change management

3.7.1 Code versioning

Regarding the code, a Camunda project can be handled as any other Java projects. The source consists of Java codes, XML files (including BPMN, DMN and CMMN models) and other, mostly textual, resources like script files and stylesheets – while the compiled result of the development is generally a jar archive. This means that the usual tools, such as Git 20 for version control, JFrog 21 for an artifact repository and Jenkins 22 for automation can be used.

3.7.2 Process versioning

Most of the time, business processes have a long lifespan, and during it, they change. It is usually required that different versions can be identified and separated – e.g. an already started process instance has to complete according to the old definition (business rules), while new instances need to adapt to the recent changes and use the latest process version. To handle this, Camunda has automatic mechanisms. When a new application is deployed to the server (e.g. a jar file is copied to Tomcat’s webapps directory), the process models are parsed and saved to the database. These resources, which were deployed together, are assigned to a logical group called deployment, and also each individual instances get a new, auto-incremented version identifier. This guarantees that old versions remain intact and that they can be distinguished from newer ones – while provides the ability to selectively remove resource groups (older deployments) from the system if they are no longer needed. The fact, that all of this happens automatically, greatly reduces the chance of a human error during releases – and if something goes wrong, gives a chance for a quick remedy or rollback.

(Camunda, 2019ac; l)

Still, maintaining versions, and the connection between different processes need to be planned. By default, in Camunda, the running processes finish with the same version as what they were started with, and new instances use the latest version. However it’s not always the way the business wants it – for these situations, it is possible to explicitly define which version we want to start a new instance with, and also to migrate instances between versions.

For migration, Camunda provides a great solution – that starts by creating a migration plan, and follows with applying the plan to a set of process instances. During the procedure, the

20 Git - fast-version-control: https://git-scm.com/
21 JFrog - universal artifact management for DevOps acceleration: https://jfrog.com/
22 Jenkins - the leading open source automation server: https://jenkins.io/
activities are matched in the two versions, and optionally event triggers are updated as well. The execution of the migration can either run synchronously and asynchronously, depending on the situation at hand, e.g. the number of process instances, performance requirements. (Camunda, 2019x) It is worth noting that the Enterprise Platform also provides a graphical migration feature, which streamlines the procedure with real-time validations and visualizations. (Camunda, 2019y)

3.8 Process monitoring, troubleshooting

In my opinion, this is a field where Camunda really excels. The Cockpit webapp provides a great overview of processes, cases, and decisions with multiple troubleshooting abilities, from checking and modifying process variables to the migration feature mentioned above. It is important to note that the Enterprise version of the Cockpit offers much more options than the Community – including better search, management of batch jobs, inspection of audit trails and many additional reports – that can be required for bigger projects, but the Community feature set is still exceptional. (Camunda, n.d.)

Processes are visualized with dynamically generated model diagrams, which also show which steps are the instances at and – in case of any error – the incidents as well.
On the left sidebar, basic information is displayed about the process definition, including name, version, deployment, a quick overview of the number of running instances. On the bottom, various other aspects and linked data – such as other processes called from the current one – are available on tabs. The instances of the process are also listed, the list can be filtered by various criteria (e.g. start date, process variables), and the filter can also be saved and later be re-used. (Camunda, 2019w)

With this view, most of the important details can be seen on one screen, without writing custom database queries, or going through big tables. After drilling down, by selecting a process instance, further, specific data are available, including the activity instance tree view and the current values of the process variables.
This way, active tasks, and variable scopes can be explored, new variables can be added, or existing ones can be modified or deleted on the fly. Process instances can also be deleted in case something went wrong. (Camunda, 2019aa) Furthermore, process definitions, as well as individual instances can also be (temporarily) suspended and re-activated, e.g. when a transient error occurs in a connected web service to prevent the creation of incidents until the external issue is resolved. (Camunda, 2019af) The Enterprise version of the Cockpit also provides visual interfaces to modify, restart and migrate process instances. (Camunda, 2019z, s; y)

Visualizations are also available for decisions to make debugging simpler. The view displays the inputs and outputs of the evaluation and highlights the matching row(s) in decision tables. (Camunda, 2019k)
CMMN case handling – with less features than BPMN, but using similar, informative views – is also available in the Enterprise version (in the Community version it is not supported). (Camunda, 2019d)

The Cockpit can also be customized and extended by plugins. (Camunda, 2019n) Although, I haven’t created a plugin for Cockpit yet, according to the documentation, it looks like that here much more extension points are provided than it Tasklist, and also creation of custom database queries (using MyBatis mappings) is supported by the plugin mechanism – to be able to extract more data from the database or to execute custom engine operations. (Camunda, 2019e; v)

By these features, development, testing and even hotfixing a live error becomes much easier, which results in saved time, money and greater reliability – thus bigger trust in the software from its end-users and administrators.

Besides these visual tools, the usual server logs and third-party monitoring applications can be used, and according to my experience, default error messages and exceptions are informative enough. In addition, the open source community forum 23 is active, and valuable information can be found on multiple other sites as well, including the forum of similar projects (Activiti, Flowable) and also the well-known Stack Overflow – which makes finding a solution for a problem much easier.

23 Camunda BPM Forum: [https://forum.camunda.org/](https://forum.camunda.org/)
3.9 Reporting and auditing

As mentioned at data modeling, Camunda sharply separates process and business data – and while it provides multiple tools for reporting on the former, the business data handling is left for the custom solution of the developers. In my opinion, in the longer run, it can result in a much better-organized database, nicer reports, and easier maintainability, but at the beginning of a project, this puts an additional time, planning and implementation burden to the team. Having said that, I will focus on the features provided by Camunda – however, when evaluating the tool, the required initial custom development effort must be taken into account regarding the reporting on business data.

3.9.1 Runtime and history database, history features

While a process is active, its state is maintained in the so-called runtime database. But after it has completed / terminated, there is no need to keep the related data there – that would just increase the amount of records, and thus slow down the execution of new processes. Instead, if historical information is needed, a separate history database is used by default. It is important to mention, that this is not required, the process engine can work perfectly without maintaining historical states, and the developer can choose to disable this feature entirely – or to replace it with a custom implementation. (Camunda, 2019p)

39. Illustration: Runtime and History Databases, History Event Stream (Camunda, 2019p)
By default, while a process is running, it generates *history events*, e.g. when it starts, when a task starts, when a task’s assignee is changed, etc. The sum of these is the *history event stream*, which is directed to a *history event handler*. The default implementation of the handler writes to the database, and works synchronously – but it can also be replaced with a custom implementation, which can even be asynchronous, e.g. using a JMS queue. (Camunda, 2019p, sec. The Default History Implementation, p, sec. Provide a Custom History Backend)

The granularity of the history events can be controlled by using different levels from *none* – where no events are fired – to *full* – where even changes to process variables are noted. Various elements can be tracked, including processes, decisions, cases, user operations, batch runs, incidents, etc. It is also possible to create a custom history level to fine-tune what information to keep, and what to throw away. It can be important for greater projects, as historical data can grow very big in a short amount of time when not controlled properly. (Camunda, 2019p, sec. Set the History Level, p, sec. Implement a Custom History Level) By using custom a setting, it is possible to comply with the law and the business rules but still, optimize the database usage.

In addition to this, Camunda also offers a history cleanup feature to remove old data – with all the dependencies together. There are multiple configuration options, but the general idea is that a job runs periodically, and inspects what can be deleted, based on the setup of the engine and the individual process definitions. Thus it is possible to keep a history for one process (e.g. insurance contract maturity) for ten years, and for another (telephone inquiry from a customer) just for a week. (Camunda, 2019p, sec. History Cleanup)

### 3.9.2 Reporting on history – low level

When the default database-based history handler is used, the *history service* provides the easiest and most convenient way to query historical data – either via the Java or the REST API. As an example, I created a simple table from historical process instances in the Booking application, and also joined the variables when a process instance is expanded.
For this, I used the REST API’s two endpoints (\texttt{history/process-instance} and \texttt{history/variable-instance}) which was a quite straightforward and simple task – but as mentioned in the REST API section, for displaying the combined data from both endpoints, a custom solution needed to be implemented.

Besides the APIs, it is also possible to use direct database queries for more complex situations, however, care must be taken, as the schema can change in later releases of Camunda.

\textbf{3.9.3 Reporting on history – Cockpit, Optimize}

Camunda also offers user interfaces and tools for viewing and analyzing the history – most of these are only available in the Enterprise Platform.

In Cockpit, one can view the history for a process definition – similar to my custom implementation above (Camunda, 2019q). On the initial screen – the dashboard – important metrics are displayed showing executed activity instances, evaluated decision instances and executed jobs (Camunda, 2019h). There are some predefined reports as well – process instance duration and completed task instance reports – which can be extended by writing custom Cockpit Plugins (Camunda, 2019ad). To follow how users work, Camunda has the Open Task Dashboard, that displays assignments by type and group, plus provides a searchable list of human tasks (Camunda, 2019t).
In addition to all these, there is a separate webapp, Optimize, which is specifically made for advanced process analysis. It offers a nice, interactive user interface with various reporting and dashboard creation capabilities, as well as data mining features. The back-end is an Elastic Search index, that is populated using the default REST API – so the data is (almost) real-time, and still, complex questions can be answered with good performance. (Niall Deehan Live Demo Camunda Optimize Camunda Day Copenhagen, 2018) While a custom reporting solution can also be built over the Camunda APIs (e.g. using Microsoft Excel pivot tables, Tableau or the ELK stack), in my opinion, Optimize has the benefit of being a specialized tool for process analysis, and it is tailored to work with data generated by the Camunda engine.

3.10 Vendor and expert support

Having a good tool is not enough for a successful project most of the times – it is also required to find talented and motivated people, and to be able to get expert support when a complex task or a bug arises. Unfortunately, I haven’t worked with Camunda in an enterprise environment, so I can just provide some general thoughts on this matter here.

41. Illustration: Camunda Optimize, Dashboard (Camunda, 2019i)

---

24 Overview of PivotTables and PivotCharts: https://support.office.com/en-us/article/overview-of-pivotables-and-pivotcharts-527c8fa3-02c0-445a-a2db-7794676bce96
25 Tableau: https://www.tableau.com/
26 What is the ELK Stack?: https://www.elastic.co/what-is/elk-stack
As mentioned in the troubleshooting section, the open source community is active, also core developers of the software visit the official forum often, and answer questions – this shows a very positive attitude from the company towards its customers. The availability of the source code is also a great plus when it comes to understanding a design decision, or solving a bug. It also makes communication easier that developers can simply share small code snippets and patches – instead of having to send big zip / jar packages to each other. As the software is basically a Java framework, plus other standard, well-documented tools, the learning curve for a Java developer should be reasonable. All the above combined results in the fact, that it’s easier to find good resources, and even if there are not enough developers or analysts, one can be trained with smaller effort – which makes the initial investment lower, and keeps the project scalable.
4 Pega

4.1 Overview

4.1.1 Introduction

Pegasystems is a company focusing on digital transformation from various aspects, like customer relationship management, sales automation, marketing, customer service operations. It was founded in 1983 by Alan Trefler, in Cambridge, Massachusetts, with the aim to help business people, and to build software that understands how they want things to work. Currently, the company has offices around the world, in 38 locations across Asia, Europe, and North America. Their top customers are from the health care, insurance, banking, and communication service sectors, but the company is also represented in the life sciences, manufacturing and even in government sectors. As Pega says: “If you’ve driven a car, used a credit card, called a company for service, opened an account, flown on a plane, submitted a claim, or performed countless other everyday tasks, chances are you’ve interacted with Pega.” Though, the company is the best known in the US, still, in recent years it has started a bigger expansion in the EMEA 27 region as well. (Pegasystems, 2018a, 2019e; Pegasystems, 2019a)

The platform has also been recognized by Gartner, in their famous Magic Quadrant analysis for Intelligent Business Process Management Suites, where Pega took the leading position for the 12th time in 2019 as well – followed by Appian and IBM. (Pegasystems, 2019d)

The software’s core layer is the Platform – which provides the main business process management features – and it can be extended with multiple strategic applications, targeted to a specific customer engagement segment (marketing, sales automation, customer service) or industry (health care, insurance, financial services, manufacturing, etc.). (Pegasystems, 2018c, sec.Pega’s application offerings) It is common, that Pega is chosen by companies because of these additional features and solutions, and not just because of the core BPM abilities.

In this thesis, I will focus on the core functionalities – there are plenty of them alone – but when deciding for or against Pega, the strategic applications must be taken into account to have a full picture. Pega is one of the strongest advocates of the nowadays fashionable, business-centric, low-code development approach – in this sense, it’s similar to other suites, like Appian. It also uses many custom – proprietary – solutions, instead of following the

27 Europe, the Middle East and Africa
standards. So for this comparison, Pega can be a great counterpart of code development centric systems, like Camunda. The Platform’s latest version at the time is 8.3. However, as main features have not changed significantly, and the learning and reference materials do not always follow minor changes that fast, the thesis will be based on multiple versions – from 7.3 to 8.2. Where it is important, I will highlight the differences.

4.1.2 The platform
From a developer’s – and user’s – perspective, Pega is a web application, with all its user interfaces available from a web browser. Under the hood it’s Java application, but with a very creative and unique approach. The main entry point – the prweb archive – contains just a minimal logic, mainly for bootstrapping the system of which fundamental elements are stored in the database. After the initialization, everything is read from and written to there: Java source code, compiled classes, images, client-side scripts. If something is modified, the code is re-compiled immediately in the background, and the changes come into effect without any further steps. Though it may seem like a minor technical detail at first, in practice, this abstraction makes the platform futuristic and strange at the same time: on one hand, it contributes to the feeling, that Pega is a “Software That Writes Your Software ™” (Pegasystems, 2018d), on the other hand, it intimidates developers who are used to easily locatable source files and controlled, well-defined build procedures. In my opinion, it is the most important question when it comes to Pega: how much control you want to keep, and how much you trust in the software.

There are generally two groups of user interfaces (portals): the development / administration interfaces, and the manager / user interfaces. The former group comprises of the Dev Studio for advanced development, the App Studio for the less technical, so-called citizen developers, the Admin Studio for managing system resources and configuration, and the Prediction Studio to interact with the decisioning layer. As end-user interfaces, usually the Case Manager (manager) and Case Worker (user) portals are by default – and often they are customized per project. During this thesis, I will focus on the Dev Studio and the default end-user interfaces, as these show the most of the platform’s business process management capabilities.

In Pega, most of the things from process diagrams to user interface definitions, from operator accounts to functions are rules – for a programmer they are the easiest to imagine as specialized classes that can be instantiated as records, with various parameters. Every application is built from these blocks, out-of-the-box and custom ones combined. Each rule
have specific interfaces – forms – for configuration, so development in Pega is mostly the creation of the appropriate rules and connecting, configuring them for the task at hand.

4.1.3 Starting a project

To be able to download the platform – in any form – a registration is needed on Pega’s website. This is free and takes only a couple of steps. When it is complete, after login, general resources – including Pega Academy courses – become available in restricted pages. It is important to note, that registration is only allowed using a “company email address”, free services (like Google Mail) are not accepted. School addresses or ones with a custom domain are fine.

There are multiple ways to try out Pega – for partners or customers a Personal Edition can be downloaded using Digital Delivery Portal or a cloud instance can be requested on Pega Labs. However, without being a partner or customer, the only available option is to enroll in a course in Pega Academy and use the learning environment provided there – which is either a cloud instance or a virtual machine. For this thesis, I used the virtual machine image provided in the Senior System Architect course, with a pre-installed 8.2.1 Platform. The steps to start the software depends on which of the above options is chosen, but usually, each one has clear instructions. After the platform has completely bootstrapped, log in with an administrative user – it also depends on the option chosen above, but it's usually Administrator@pega.com with the password provided in the instructions.

At this point, the initial screen of Dev Studio is shown. As a next step, a new application should be created by starting the New Application Wizard with the Pega Platform menu / New Application item.

42. Illustration: Starting the New Application Wizard

28 Create a new Pega account: https://accounts.pega.com/register
29 Digital Delivery Portal: https://community.pega.com/digital-delivery
30 Pega Labs: https://community.pega.com/pega-labs
During this procedure, many records are created by the wizard, and, unfortunately, these cannot easily be reverted. I suggest to take a snapshot of the virtual machine’s initial state – or create a database backup – before going any further, to prevent frustration. Everything can be fixed later as well, of course – just with a much bigger effort.

On the wizard’s first screen, let’s choose the “Custom” application type. Although it is enough to provide the name of the new application, I suggest to use the Advanced configuration link to give more details – and prevent the creation of an organization with an auto-generated name. Click the Save and Create application buttons to continue.
On the next screen, it is advisable to add at least an admin user, specifically for the newly created application.

As a last step, log in with the newly-created user to start the development (the password might need to be updated during the first login).

4.2 Process modeling, data handling
4.2.1 Case, stage, flow

With process modeling, Pega takes a custom, unique approach. Although there are elements that are similar to standard notations, this does not mean a one-to-one mapping either.

The outermost level, that usually symbolizes a business matter, is the case. It is important to note that, despite the name is the same, it is not like a case defined in the CMMN standard.
A case is divided into multiple main parts, the stages, which represent separate phases during the processing. The primary stages (on the top, with gray heading) comprise the happy path and show what steps need to be taken if everything goes according to the best / simplest scenario. They are traversed sequentially, as it is also shown by the arrows – or so-called chevrons. For handling special situations and extraordinary events, alternate stages (on the bottom, with orange heading) can be introduced – they are not part of the main flow, and they also don’t follow any sequence. From an alternate stage, the case can be directed back to the happy path, or it can be closed there. The terminal points – resolution stages – where cases usually get resolved, are marked with a red line under the header. (Pegasystems, 2018j, sec.Stages)

45. Illustration: Booking case

A case is divided into multiple main parts, the stages, which represent separate phases during the processing. The primary stages (on the top, with gray heading) comprise the happy path and show what steps need to be taken if everything goes according to the best / simplest scenario. They are traversed sequentially, as it is also shown by the arrows – or so-called chevrons. For handling special situations and extraordinary events, alternate stages (on the bottom, with orange heading) can be introduced – they are not part of the main flow, and they also don’t follow any sequence. From an alternate stage, the case can be directed back to the happy path, or it can be closed there. The terminal points – resolution stages – where cases usually get resolved, are marked with a red line under the header. (Pegasystems, 2018j, sec.Stages)
After the stage, the next level is the process/flow, in which the concrete steps reside.

Among other parameters and settings, this is mainly a diagram, similar to a BPMN process model, but with significantly different elements and logic. Note, that the steps of this diagram are also displayed in the previous illustration, in the so-called Case Designer. This is common in Pega, that one thing has multiple views with different focus. Here, the most used properties (e.g. assignee) of the steps are available from the case view as well, but for a more detailed setup, the flow rule needs to be opened.

Besides the general case life cycle, optional actions can also be defined, linked to the case, a stage or a flow. (Pegasystems, 2018j, sec.Adding optional actions) This concept is similar to the manually activated tasks in CMMN or ad-hoc subprocesses in BPMN – it provides flexibility in Pega’s case management approach.

32 As Pega is a software with a long history, it is common, that elements are known by multiple names.
4.2.2 Process diagram

It is worth to briefly go through the elements of a Pega process diagram. This is much simpler than in BPMN, as there are only a few distinct ones and a couple of specialized versions.

![Diagram of Pega process elements]

There are the start and end shapes – with no variations like a message-based start shape in BPMN. The Assignment symbolizes a human step, where an operator needs to do something, and the Utility is an automatic step, where a program logic is called. A Decision is a point where branching can occur, based on a condition. It resembles an exclusive gateway in BPMN, the flow can only continue on one path after it, there is no parallelization. A Subprocess can refer to other flows – which are always global, not embedded. The Run Data Flow shape is for initiating data processing – it’s a different topic, not discussed here. Finally, the Annotation is a simple textual note.

The Automations group contains pre-configured utilities, subprocesses – for common use-cases like PDF creation or getting approval from the manager. The often-used Wait shape is also available from here, which can either act similarly to an intermediate timer event in BPMN, or it can wait until dependent (linked) cases reach a certain status. Among the Advanced shapes, the most notable ones are the Split for Each and Split Join that provide the parallelization ability in Pega flows.

4.2.3 Process modeling in practice

It is important to emphasize that there is no easy way to convert a BPMN diagram to Pega, everything needs to be rethought. Once a company decides to use Pega, it is the best to model there right from the start, as otherwise double work would be needed – and since there is no
formal way to validate the conversion, little benefit comes from using another, even standard, representation. The only exception can be high-level modeling at the beginning of a project when people are more used to other representations – it’s better to focus on the process itself in a meeting than to start by explaining Pega-related details to business people. But still – according to my experiences in a project where we used BPMN and Pega mixed – after some time people should get on the same page, otherwise, detailed planning and acceptance testing of the working software becomes hard.

To show how modeling is done in Pega, and highlight the differences, let’s take the Booking process from the BPMN, CMMN, DMN chapter, and go through its Pega implementation. The case can be seen on the Booking case illustration some pages back.

The first step is to provide the details of the trip (name, destination, date). I choose it to be on a separate stage, *Initiation* – as after the requester submits the form, the case continues with back-office processing. Usually, when work is handed over to a group with different responsibilities – e.g. another business unit within the organization, or in this case, the client-side and administrator-side – it marks a significant change, thus a stage boundary. This initial process consists of just a single assignment, routed to the Requester. The assignee is a case participant – or party for short. This is a special concept in Pega: important members can be distinguished throughout the case lifetime to make it easier to reference them while routing tasks, sending emails, etc. (Pegasystems, 2018j, sec. Configuring a work party)
After the data of the trip is submitted, it needs to be validated and approved – so the next stage (phase) is the Approval. There are two processes / flows here: Validate Request and Validate Expenses. In the former one, the important new concept is the usage of a Smart Shape / Automation, the Approval. This is practically a pre-configured subprocess, which advances the flow to separate directions, based on approval or rejection.

![Illustration: Approval flow configuration](image)

49. Illustration: Approval flow configuration

Here, it simply lets the flow continue with the next step if the trip details are correct and changes to the Rejection stage if they are not. As the Rejection is a resolution stage, the case is also closed there – resolved with a Resolved-Rejected status – if the provided data are not accepted.

Let’s say, the details were approved, so we are at the Validate Expenses flow now (see the illustration a couple of pages above). The first shape in the flow is an assignment, Estimate Expenses. In Pega, assignments need to be inspected together with the outgoing connectors (arrows), as those show the possible actions that can be taken by the operator. With this concept, multiple flow actions can lead out from an assignment – but still, in the end, only one can be chosen. So it’s not parallel execution, but multiple-choice options, which also contributes to the flexibility in Pega’s case management approach.

After the estimate has been submitted, the appropriate approval level needs to be determined. In Pega, the decision shape can be used for this – from a BPMN perspective it is a business rule task and an exclusive gateway combined.
There are multiple options for the basis of the decision, from a simple Boolean expression, through a decision table, to more complex predictive or scorecard models (using Pega’s decisioning features).

Based on the result here, the flow either ends (automatically approved), goes to the Approval shape (manager approval needed), or advances towards the Change to a stage smart shape / automation. The latter is similar to the Approval shape in the sense, that it is also pre-configured – but instead of being a complex subprocess in the background, it just calls a script-like element. So it resembles to a Utility shape, that’s why it is yellow, instead of being blue, which is the color of subprocesses.

If everything is good so far, after the request has been approved, the case moves to Reservation. This flow shows an example, how parallel execution can be achieved in Pega with the Split Join shape (Book Accommodation and Travel).
51. Illustration: Book Trip flow - with parallel Accommodation and Travel booking

The concept is similar to a parallel gateway in BPMN, just less visual, as the details are hidden in the configuration of the shape.

52. Illustration: Split Join with two referenced flows

Although there are also other possibilities here, like joining branches based on a condition, in my opinion, this approach is used, because it is simply simpler to restrict the flow execution to always one branch and hide the parallelization logic, than keeping track of multiple branches – as in BPMN. In the meantime, it also makes understanding the diagram easier for business people, and since parallelization is not too common, it can be considered a good compromise.
After both the accommodation and the travel have been booked, the flow arrives at a decision shape. Here, it is of type Fork, which means, that the connectors after the shape contain the conditions. On the Booking Finished branch, there is a When rule, BookingFinished. A When rule is a reusable condition – in this case, it evaluates to true if accommodation and travel are booked, and the expense limit is met.

The other branch after the decision shape is simply an Else condition, so if BookingFinished evaluates to false – let’s say, the expense limit is not met – the flow goes to the technical assignment, Fix Booking.

This is a custom solution here, as, in Pega, there is a smaller degree of flexibility to create declarative relationships between steps than in CMMN. The idea is this: if everything goes well on the first try, the flow can end. Otherwise, the flow gets stuck at Fix Booking and the Clerk can use the optional actions attached to the Reservation stage to re-book accommodation or travel, change the expense limit or fail booking.
These actions can be started by the operator when the case is at the Reservation, and their start condition – see the right pane on the picture – are met. With a CMMN analog, they are similar to manually activated processes with an entry criterion attached to them. However, here the start condition gets re-evaluated every time, so the “state transition” between *startable* and *not startable* is bidirectional – while in CMMN, once a task is enabled, the entry criterion is no longer relevant.

When something is changed by these actions, the *BookingFinished* condition needs to be re-evaluated. For this, every flow sets a so-called *ticket*, *BookingDetailsChanged* – this is like triggering an event. The ticket is attached to the decision shape in the *BookTrip* flow (see the circle with a triangle on the shape), so each time it gets raised, the flow jumps back there, re-checks the condition, and takes the appropriate outgoing connector, according to the updated state. This mechanism creates the loop needed here, in spite of the fact, that multiple separate flows are used without any direct connection between them.
There are multiple alternative other solutions for the requirement above, I choose this one because it felt the most Pega-like to me. As can be seen, there are some elements similar to CMMN (optional actions, flow start conditions), but in my opinion, a case here is rather like a big bounding BPMN process. Stages are like simple subprocesses, and optional actions are more advanced versions of tasks in ad-hoc subprocesses. That said, I think Pega’s approach is perfectly valid: uses simple building blocks and mainly sequential processing, but also adds the most frequently used declarative features – still keeping the complexity low. Sometimes new shapes could add more clarity – e.g. better visual depiction for parallel flows – but maybe that wouldn’t be worth the price of the increased complexity.

4.2.4 Classes and the Enterprise Class Structure

So far, the case and flow modeling was introduced, and even some other rule types were mentioned, but without a specific structure. To build a good, maintainable application, it is important to align its building blocks properly, and also to create a nice data model. Pega takes an approach that is similar to object-oriented programming: it uses hierarchical classes with special naming conventions, this is the Enterprise Class Structure. Rules are then organized in these classes. Just like in Java, the children inherit from the ancestors, so if a rule is defined on a higher level, it can be reused in a specialized class as well – unless it’s overridden.

56. Illustration: Booking class structure
The top level is the *organization layer*, which is usually named after the company’s stock ticker symbol or some other abbreviation of the name (here: “BIT”, for Business Information Technology). After that, more levels can be added to handle specialization by divisions and unit (*division and unit layers*), or to create generalized frameworks which can be reused across applications (*framework layer*). Here, as the application is simple, the *implementation layer* (Booking) follows the organization one. (Pegasystems, 2018h, sec. Enterprise Class Structure)

Then classes are also distinguished by their functions, the most common ones are the *Work* classes – they are the basis of cases – and the *Data* classes – specifically for creating complex data models, they provide the representation of custom data types (like customer, line item, etc.).

With this structure, Pega has multiple – two-path – inheritance: one defined by the name of the class, called *pattern inheritance*, and one that can be set in addition, the *directed inheritance*.

57. *Illustration: Class inheritance*

The former can be resolved by simply splitting the class name by the dashes, so in the Booking example the *BIT-Booking-Work* class inherits from *BIT-Booking*, which, in turn, inherits from the organization layer, *BIT*. For the directed inheritance, a good example is the *…-Work* classes, which inherit from the *Work-Cover-* class, then *Work-* class, and thus get access to built-in functionalities. (Pegasystems, 2018j, sec. How to reuse rules through inheritance)

### 4.2.5 Data storage

In Pega, both business and process data are generally stored in the main, default database schemas – there are features to manage this, and also to report on the data.

The representation of data items are the *Property rules*, that, just like other rules, also align with the class structure. With a Java analog, they are the attributes in a class, while other rules can be considered as methods / functions (e.g. a flow is a mutator, a when rule is a getter,
Properties can have different types, from simple Text or Integer to complex ones, which have elements called Pages. These pages then have a definition backed by a (data) class and a property of these types can either be single or a collection – page group is like a hash map, page list is like an indexed array.

58. Illustration: Property types

Classes can also be concrete or abstract – the former means that it can be instantiated and its instance can be saved to the database, while for the latter it is not allowed, and thus it usually serves as an element of embedded structures (i.e. page definition), or a basis for inheritance. Concrete classes are connected to database tables by mapping.

The schema, that is being formed like this, is rather denormalized – the same property can be mapped to different tables on different layers: e.g. if parent and child classes have separate mapping. This can be good for performance and some security reasons: table access can be fine-tuned, and detailed access control can be used both on Pega-side and on database-side. On the other hand, this separation means that data-consistency is harder to maintain and the schema is more complex.

Another aspect is, that like this, many separate database columns would need to be created for every property on every layer. This would pollute the database too much and would require many column creation and deletion operations during the development and releases. To
overcome these issues, Pega uses a BLOB column in the database tables, where data are items stored in a special, compressed format – it is the easiest to imagine it as a big XML structure that contains all properties of the saved class instance. By default, new properties are only written to the BLOB, and no separate column is created for them in the database table. This solution simplifies the schema, and it’s optimal enough if a big amount of the stored information needs to be read at once (e.g. all the properties of an instance). However, as the decompression of the BLOB and finding a property inside it is a costly operation, when only a couple of columns needs to be read, but from multiple (many) instances, the additional processing significantly increases the query time. This is usually a situation where multiple instances are filtered by one property, or tables are joined together on a foreign key. In these cases, it is advised that those frequently used properties (but only those) get a separate column in the table, or with Pega terms, they are optimized / exposed. This way, their data is written both to the BLOB and to the separate columns. (Pegasystems, 2018j, sec.Data Storage in Pega applications)

4.2.6 Built-in data structures

In Pega, there are a lot of out-of-the-box data structures that a developer can utilize and extend to make the application leaner and cleaner – one example is the work party, mentioned at process modeling. However, in order to use these default features, one needs to know and understand them well – which is not so self-evident, as it is hard to find complete, up-to-date documentations that mention all the necessary details and connection points between the parts.

There are also many default properties for classes based on default structures, such as Work- or Data-, which are usually distinguished by their prefix. The ones starting with pz are for internal processing – they can change between product releases and they should be considered read-only. The px properties are read-only as well, while py properties can be read and written by custom rules. (Pegasystems, 2018j, sec.How to manage properties)
Many times, the proper usage of these extension points separates a good application design from a bad one. According to my experiences, this is something that can only be learned by practice, and the knowledge of these tricks makes a developer valuable. I think this situation could – and should – be improved by providing better documentation and by modularizing the default features. For example, if parent-child case relations are not used in an application, this feature could be turned off – that would simplify the configuration user interfaces, and remove unused properties (ones that contain “Cover”, see the picture above).

4.3 User interface

4.3.1 Designing user interfaces

This is one of the biggest strengths of Pega – it is easy to create a nice, intuitive UI very fast. Of course, as always, if some feature has a broad out-of-the-box support (like a framework), it means, that creativity is constrained – however as we are talking about business software, it is safe to assume that most of the times very similar functionalities need to be supported, and a well-tested, pragmatic solution is better than a fancy design.

There are multiple ways to put together a section / view – in different situations, one is better than the other, but the end result will be the same. For business users and also for prototyping the simplest approach is to use the Case Designer, and edit the appearance of an assignment form there.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxApplication</td>
<td>Booking</td>
</tr>
<tr>
<td>pxApplicationVersion</td>
<td>01.01.01</td>
</tr>
<tr>
<td>pxCoveredCount</td>
<td>0</td>
</tr>
<tr>
<td>pxCoveredCountOpen</td>
<td>0</td>
</tr>
<tr>
<td>pxCoveredCountUnsatisfied</td>
<td>0</td>
</tr>
<tr>
<td>pxCreatedDateTime</td>
<td>20191030T173715.017 GMT</td>
</tr>
<tr>
<td>pxCreateOperator</td>
<td>Admin@BIT</td>
</tr>
<tr>
<td>pxCreateOpName</td>
<td>Admin</td>
</tr>
<tr>
<td>pxCreateSystemID</td>
<td>pega</td>
</tr>
<tr>
<td>pxCurrentStage</td>
<td>PRIM1</td>
</tr>
<tr>
<td>pxCurrentStageLabel</td>
<td>Initiation</td>
</tr>
<tr>
<td>pxCurrentStageSubscript</td>
<td>PRIM1_1</td>
</tr>
<tr>
<td>pxExternalSystemUpdateCount</td>
<td>0</td>
</tr>
<tr>
<td>primsName</td>
<td>B-8</td>
</tr>
<tr>
<td>pxLastSyncTimestamp</td>
<td>BIT-BOOKING-WORK B-8</td>
</tr>
<tr>
<td>pxLockHandle</td>
<td>BIT-Booking-Work-Booking</td>
</tr>
<tr>
<td>pxMarkedForSuspension</td>
<td></td>
</tr>
<tr>
<td>pxObjClass</td>
<td>BIT-Booking-Work-Booking</td>
</tr>
<tr>
<td>pxSaveDateTime</td>
<td>20191030T173715.974 GMT</td>
</tr>
</tbody>
</table>

59. Illustration: Some default properties of a case
In this editor, selected / promoted fields can be dragged from the left pane, or others added by name. Also, other views can be embedded. However, when more control is needed, it is better to open the rule form of the section in question. There, depending on the complexity, the configuration form can have two appearances. One is a simplified version, which uses Design Templates – this provides the ability to select a predefined layout and display fields using it.

60. Illustration: Provide Details view in Case Designer

61. Illustration: Provide Details section using a Design Template
The other is the old version, having full design capabilities – this resembles to a WYSIWYG editor by showing all the details and element groups.

62. Illustration: Provide Details section with full design capabilities

New elements can be added from the Layout, Basic and Advanced dropdown menus.

63. Illustration: Section elements

Fields can be organized by containers, called *layouts*. There are simple ones such as the Dynamic Layout on the picture above, which simply groups the elements and displays them in
one or more columns. But there are also complex layouts, which iterate over the elements of a collection (page list or group), and create tables (*Table*), accordions, tabs, menus (*Dynamic layout group*), etc. Sections can also be embedded in each other, and like this, modular, reusable structures can be created.

Fields can be displayed using various controls, such as *Checkbox* for boolean values, *TextArea* for long texts, or *DateTime*, which shows a calendar. Data processing can be simplified with advanced widgets from *AutoComplete* fields to *Signature Capture*, and information can be visualized by a *Chart* or *Map* – just to name a few.

It is worth to note, that custom widgets can also be created, even by defining them at the HTML level (Tolnai, 2018). However, this is usually a last resort, as Pega – being a low-code tool – advises against using custom coding to simplify maintainability.

Styling of the elements can be achieved in multiple ways. The traditional approach is to use a *Skin* rule – which is basically the Pega version of a CSS stylesheet.

![Illustration: Skin rule](image)

While this provides a comprehensive and visual view of the styles, in my opinion, it adds too much complexity – the textual representation of CSS is simple enough, even for non-technical people if proper selectors (id, class) are used. Pega also noticed this, and in version 7.3.1 they introduced simple helper classes which can be added to elements – without skin modification (Pegasystems, 2017a). As in recent years, UI development has a great focus at Pega, and a lot of improvements have already been done, I expect simplification here as well.
4.3.2 Default end-user portals: Case Worker, Case Manager

Depending on the role of the users in the organization, they can see different views of the application with different functionalities – the so-called portals. For end-users, it can be the Case Manager or the Case Worker portal, based on the user’s main tasks in the organization – whether they need to manage other people or solve issues for customers. Although these portals are usable out-of-the-box, they usually rather serve as a basis for customization. Thanks to Pega’s modular UI approach, it’s easy to hide / replace a section or insert a completely new widget – e.g. a chart or a report.

From the above two, the simpler one is the Case Worker portal, which is intended for operators, who directly work on the tasks, issues, customer complaints, etc.

After login, the user is presented with a list of work items (assignments) linked to him. Besides this, there are collaborative features accessible from the left-side menu: Pulse is similar to a forum or message board (Pegasystems, 2018g) Spaces are groups around specific topics (Pegasystems, 2018i) and Documents are shared items – rich text snippets or files (Pegasystems, 2018b). The operator can also see the teams he is in, and the cases that he has something to do with (i.e. is a work party within the case). At the bottom of the sidebar, recently opened elements are listed. There is also a search box in the top right corner, that is used extensively. The implementation behind it is based on Elastic Search, and indexing of items can be customized – this makes it a simple, fast, yet versatile solution.

After selecting an assignment – either from the My Work list, from Recents or via search, it is loaded into the main work area, along with the details of the corresponding case.

65. Illustration: Case Worker portal, My Work
The top center area is the section about the assignment itself, while the widgets around it present useful information for processing. Below the assignment form, the stages of the case are shown by the chevrons, with the currently active stage highlighted. It is also clickable, then processes within the stages are displayed in an overlay box. At the bottom, a summary of the case data and the audit log can be viewed by default – but also custom tabs can be added here. The right sidebar displays additional information and connected data – such as creation and update time of the case, the work parties, etc. As this is the part of the application where operators spend the most time, it is almost always tailored to the needs of the users by adding, removing or modifying sections.

The Case Manager portal is generally very similar to the above, but the main screen there is a dashboard with various informative widgets – to assess the progress of work and the load of the operators.

66. Illustration: Assignment in Case Worker portal
There is also a Reports menu item, where commonly used default and custom reports can be accessed – more on the reporting later.

4.3.3 GetNextWork

In Pega, there is an important concept for distributing work, called GetNextWork. This is an approach where operators are given tasks by an algorithm, instead of choosing – or cherry-picking – from a list. This can be beneficial, as assignment urgency can be managed and taken into account by an automation, thus service levels can be enforced. Also like this, work amount across users is evenly distributed – operators cannot always pick the simple tasks – and thus the system can be fairer and better balanced.

The process in nutshell is the following. During case processing, assignments can be routed to work baskets / work queues – instead of directly to users. These work baskets are linked to (multiple) operators, acting as shared task lists.
For an operator, multiple work baskets can be enumerated, from which the user can fetch tasks (on the picture above it is only the Clerks@BIT). There are also options that influence how the algorithm prioritizes tasks (Pegasystems, 2018h, sec.How to manage assignment selection), as well as system-wide settings to fine-tune the procedure, or – if needed – the algorithm itself can also be modified.

From the end-user perspective, the mechanism is usually invoked by a press of a button – e.g. in the header of the Case Worker portal.

68. Illustration: Operator record, Work, Routing

69. Illustration: GetNextWork button added to Case Worker portal
Then the highest-ranked assignment is selected by the \textit{GetNextWork} algorithm – e.g. the one with the highest urgency – and it is opened, similarly when a user clicks on an item in the \textit{My Work} list.

\section*{4.4 Developer tools}

\subsection*{4.4.1 The environment}

As mentioned before, development in Pega is done within the web browser. This is good from one hand – we only need a network connection to the server (Internet / intranet) and a modern browser – but also poses challenges as everything is new, compared to code-based development – the good old tools, methods are not applicable, and the network connection can also be a bottleneck.

There are multiple portals for development (App Studio for citizen developers, targeted solutions for specific frameworks, such as Decision Manager portal, etc.), but I will focus on Dev Studio (Designer Studio before version 8) as this is the general, most commonly used place where development is done.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{70.devstudio.png}
\caption{70. Illustration: Dev Studio}
\end{figure}

It looks similar to some IDEs: the central area shows the rule currently being edited, while around it, menus and panes take place. On the leftmost side, various views can be accessed by clicking on the icons, from top to bottom the following.

The first is \textit{Recents}, it shows a list of recently opened rules. The \textit{Case types} shows a list of case definitions (Work-) in the application, while \textit{Data types} shows data classes (Data-) that
are linked to the application – such as customer, line item, etc. The App icon opens the Application Explorer, which displays a hierarchical class viewer – during development a lot of time is spent here, as this shows the functionalities implemented in a class the best. The Records view shows all the rule types in the system, and after selecting a type, displays a list of the available instances.

Lastly, the Favorites is a place where rule instances are listed, which were previously marked by the developer – it can be used to quickly access often needed rules. In the left bottom corner is the operator menu, after clicking it, user-related functionalities are displayed, such as: viewing profile, changing development-related preferences, opening the Operator and Access Group records, editing favorites and log off.

On the top, there is a menu bar with multiple items. The first one, Application, contains items related to the current application, and also the new application wizard is available from here. The next, Configuration, is kind of a main menu in Dev Studio, its items lead to various wizards and landing pages.
The Launch menu is for starting other portals – e.g. Case Worker – in a development context, to test the application. With Create, new case instances, as well as new rule instances can be created.

On the right side of the header, there is a search box where rule instances can be searched by their name, as well as other textual attributes. As the application gets bigger, it is used more and more often – this is the best way to find a rule without knowing its location in the class hierarchy, or to check if a functionality has already been implemented. This fact shows the importance of naming conventions and proper documentation of rules in Pega – the next developer can search for keywords here. Right next to the search box is an icon with a question mark, that opens the help menu, with links to valuable resources – documentation, guides, support page.

Just like in code-based development environments, there are multiple debugging tools in Pega as well, accessible from the bottom right corner of Dev Studio. I will highlight the three most commonly used ones: Clipboard, Tracer and Live UI.

4.4.2 Clipboard tool

As an application runs, it stores data in the memory, and later uses it for calculations, decisions. In Pega, these data items are kept at a globally accessible place, called Clipboard, and they are organized into pages. These pages, in turn, contain properties, and they can also contain other embedded pages. During development, it is often required to check the values in the memory to assess if the program runs according to the requirements. For this, the Clipboard tool can be used – it is similar to variable inspection during Java debugging. (Pegasystems, 2018j, sec.How to view clipboard data)
Amongst the many pages one is worth to be highlighted, the `pyWorkPage` – this is where the data of the currently opened case can be found. Other pages contain information about the system context, e.g. the `OperatorID` shows the currently logged in operator’s properties, or represent global data items, such as currency exchange rates.

A lot of pages are generated by the underlying layers of the application stack, such as out-of-the-box platform functionalities, which makes finding relevant information harder. It is also important to emphasize, that all of these properties are globally accessible by any part of the Pega application – which, in my opinion, is a legacy issue, that needs to be paid attention to during development, to avoid building “spaghetti code” with tangled cross-references.

### 4.4.3 Tracer

While the Clipboard tool shows a static state of the application at a given time, the `Tracer` gives a great overview of executed steps during application run. By using this tool, it can be followed, how calculations are done, how conditions are evaluated, how user interface elements are built – almost all aspects of the program – and like this, errors can be spotted and located, which otherwise would be very hard to track down. (Pegasystems, 2018j, sec. The Tracer)
75. Illustration: Tracer

Depending on the configuration, various events and different parts of the application stack can be inspected. The visual output is a big table with information about what has happened, at which rule, and what was the so-called *step page*, the rule’s primary context. The actual values of the step page are stored for each step, and so, it can be followed back, where a modification to a property occurred. On the picture above, the page at line 599th was opened for inspection – as indicated by the arrow at the line.

### Properties on Page TraceEvent [pyWorkPage]

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxUpdateSystemID</td>
<td>pegas</td>
</tr>
<tr>
<td>pxUpdateDateTime</td>
<td>20191102T130054.1190 GMT</td>
</tr>
<tr>
<td>pxCaseAssetFormat</td>
<td>Case secondary content</td>
</tr>
<tr>
<td>pxFromDocumentSet</td>
<td>None</td>
</tr>
<tr>
<td>pxBrowseCases</td>
<td>True</td>
</tr>
<tr>
<td>pxFromDocumentSet</td>
<td>0</td>
</tr>
<tr>
<td>pxCurrentStage</td>
<td>PRM2</td>
</tr>
<tr>
<td>pxInternalAssignmentHandle</td>
<td>ASSIGN-INTERNAL-BIT-BOOKING-WORK-10-FINTERNALCASEFLOW</td>
</tr>
<tr>
<td>TotalCost</td>
<td>0</td>
</tr>
<tr>
<td>pxSelectItem</td>
<td>True</td>
</tr>
<tr>
<td>pxConflictTime</td>
<td>20191102T124956.670 GMT</td>
</tr>
<tr>
<td>pxNextItem</td>
<td>Next &gt;&gt;</td>
</tr>
<tr>
<td>pxConfirmationNote</td>
<td>pyStepRoutedForApproval</td>
</tr>
<tr>
<td>pxSubItems</td>
<td>SubItem</td>
</tr>
<tr>
<td>pxCancelLabel</td>
<td>Cancel</td>
</tr>
<tr>
<td>pxPushNotificationsEnabled</td>
<td>False</td>
</tr>
<tr>
<td>pxFromObject</td>
<td>WorkClass</td>
</tr>
<tr>
<td>pxSaveDateTime</td>
<td>20191102T130054.285 GMT</td>
</tr>
<tr>
<td>pxFormButtons</td>
<td>False</td>
</tr>
<tr>
<td>pxApplicationVersion</td>
<td>01.01.01</td>
</tr>
<tr>
<td>pxTemporaryObject</td>
<td>False</td>
</tr>
</tbody>
</table>

76. Illustration: The content of the pyWorkPage at the 599th step
The Tracer can be compared to debuggers in the Java world, however, instead of going through the executed lines of the source code, it displays debugging events. This is an abstraction, that is, on one hand, required – Pega generates the source code, and so, it can be hard to process for a human reader – but on the other hand, provides a more concentrated view of the building blocks, and thus, makes easier to understand the flow of processing. It is a frequently used, invaluable tool which is unique to Pega, and – in my opinion – greatly contributes to the success of the platform.

4.4.4 Live UI

As mentioned before, Pega’s UI building capabilities are exceptional, and they are also complemented by a useful debugging feature, the Live UI. It is similar to a DOM inspector used during website development, but instead of HTML elements, it shows the Pega building blocks of the user interface, such as layouts, sections, fields.

By using the Live UI, it is easy to locate elements, even in deeply embedded structures. Then the containing section can be opened with a click, and the attributes can be inspected and modified. In case one needs to find the wrapper (parent) of a section – let’s say for a repeating layout, where it is easier to locate a piece of the repeated content, but the layout properties can be modified in the wrapper – the complete tree structure is shown and can be traversed in the right sidebar.
4.4.5 Coding in a low-code system

This is something, that – in theory – should be avoided at all times, and also, Pega discourages custom code development a lot. However, as businesses and their tasks are unique, no matter how much a developer tries to avoid it, he will encounter problems that cannot be solved by simple, out-of-the-box means – and this happens rather sooner than later.

In Pega, the Activity rule is provided to bridge this gap – however, it usually falls short. In other words, this is the worst programming interface I have seen so far.

78. Illustration: API Activity that modifies the status of a case (Open, Resolved, etc.)

Some Pega developers may find it unfair to showcase an out-of-the-box (thus “private”) Activity as a negative example, but in my opinion, it clearly presents that even with the knowledge and experience of the core developers, it is impossible to create a nice, readable program logic. Conditions (“When” labels with a border) are not visible by default, and “Jump” (aka. GOTO) transitions need to be used. Also, the parameters of a method become only visible when one expands the row – but then that step takes the whole screen, and the context is lost.

Having said all that, I like Pega’s innovative and unique solutions a lot, but this issue had to be highlighted, as I have seen so many developers disliking Pega because of this. I think, the reason of this clumsiness due to business decisions (emphasis on the low-code, no-code approach), and not the lack of expertise in the platform development team – there are world-class engineers at Pega. In my opinion, this problem could be solved relatively easily by providing a very limited Java code-based API (or even better, by using a simple scripting
language that compiles to Java) which covers the current Activity methods, and by allowing developers to actually write short codes.

4.5 Security

4.5.1 Authentication and authorization, Role-Based Access Control

There are multiple authentication types supported by default including SAML 2.0, Kerberos, OpenID Connect, Basic Credentials (simple user name and password combination). In addition to these, a custom service can also be implemented, and for some commonly used solutions – like LDAP – there are also examples provided, to make starting the implementation easier.

After the user has been authenticated, the corresponding Operator ID record will be read to the memory, the Clipboard. Every user has one or more Access Groups assigned, which determine the roles he assumes, and, in turn, the roles determine which object (class) at what level he has access to. So the chain looks like: Operator ID → Access Group(s) → Role(s) → Object(s). (Pegasystems, 2018h, sec.Access control)

The basis of the setup are the classes. It can be determined, per class, if its instances can be read, written, deleted – and also if rules linked to it, with other words the behavior of the class, can be read, written, deleted. In addition, reports and activities have special importance as the former generates database queries (thus accesses a great amount of data), and the latter provides less-restricted programming capabilities, which poses security issues. The access level can be a value from 0 to 5, which indicates the production level of the application, 1 being experimental, 5 being production and 0 meaning that no access is granted, regardless of
the environment. Instead of a number, so-called Access When rules can also be used – they evaluate a condition and based on that, either grant access (return true) or not (return false). In addition to these, Privileges can be defined on classes – by using them, fine-grained control can be achieved. For example, a Flow Action can require a privilege to run – on top of the class, access type, production level / Access When restrictions. (Pegasystems, 2018h, sec.Access control record types, c, sec.Authorization models)

4.5.2 Attribute-based Access Control
Complementing the widely-used role-based model, Pega also offers another option to secure data – based on attributes of Assignment, Data or Work (case) instances. It is worth to mention that by using Access When rules in the role-based model, properties of instances can also be checked, and read, update, delete operations can be controlled. However, in addition to a different basis (which sometimes matters a lot alone, by simplifying the configuration), attribute-based control also gives other options, restriction on property-level: individual properties can be encrypted or obfuscated. For example, while displaying a bank account number, some digits can be replaced with stars – and this is all done in the background by the engine, so after once being configured, the developer doesn’t need to care about it, it will be enforced everywhere in the application. Role and attribute-based access control coexist – the former is compulsory, always-on, the latter is just optional. If both of them are used, all conditions must be satisfied, for the operator to get access.

4.5.3 Other access control features
On top of the above, general features, there are also some specific ones as well. For example, access to Work Baskets can be secured, based on the operator’s roles, and attachments to cases have a category-based control as well (Pegasystems, 2018h, sec.How to secure workbaskets and attachments). While these perfectly make sense and can simplify setup sometimes, one needs to be careful with using multiple approaches at the same time, as it can
become very hard to manage and remember all conditions. Plus, when the application security needs to be assessed (e.g. during a compliance audit), all the different levels used must be understood, documented and tested – which easily becomes a burdensome task. So even though, there are a lot of features provided by the platform, security must be designed very carefully to keep maintainability.

### 4.5.4 Other security considerations

Besides the explicit security features, it is also important how well the software is built, whether the industry best practiced are followed, and how much effort is required from a developer to keep the program safe while using or extending it. According to my experiences, with Pega, the picture is a little mixed. Being an enterprise software used by the biggest companies, security has always been a central concern, and there have always been many features and great architectural designs supporting this. However, the platform was not secure by default, and it was not emphasized enough in the past.

In version 7, a separate *System Management Application* – that shouldn’t have been available outside of the local network at all – was often forgotten about, and left accessible, without even a minimal protection. There were also multiple sample operators with default password enabled by default. These issues are still present at companies that haven’t been able to upgrade to a newer version yet.

Fortunately, this situation has changed a lot recently – from 7.3.1 an integrated security checklist is shipped (Pegasystems, 2017c), from version 7.4 sample operators are disabled by default (Pegasystems, 2017b), and in version 8 the System Management Application has been removed. Also, the opportunities to learn about proper security configuration are improved, most of the Pega Academy courses are freely accessible after a quick registration, and many new documentation pages have been written with great quality. Still, the hard to read Activities, the many different rule types and the generally complex nature of the system requires great focus from developers in order to avoid mistakes.

But with enough attention, the tool-set for building a secure application is given. On top of the good access control, there are features to prevent cross-site scripting attacks and other external content-related problems. Policies can be defined to require minimum password length and complexity, multi-factor authentication can be enforced, CAPTCHA challenges can be displayed in suspicious situations, and operator accounts can be locked automatically after too many failed login attempts. Various security-related events can be logged, and also custom
events can be added. The so-called Guardrails, which provide suggestions on how to improve rules, also help to increase the quality of the application, and thus security as well.

4.6 Integration with other systems

When it comes to interacting with other systems, Pega offers many out-of-the-box features, templates, and wizards to make it easy. There are a lot of third-party software and protocols supported by pre-configured rule types, and this usually covers most of the requirements of a regular organization – which streamlines development by a great degree.

Based on the relationship between the members of the communication, two groups are differentiated: Connectors and Services. The former means, that Pega reaches out to an external entity, e.g. when it needs to query some data or wants to send a message. The Service, on the contrary, is when Pega is the provider, which offers an interface that a third-party application can connect to, e.g. publishes case data using the REST protocol.

81. Illustration: Out-of-the-box connector and service rules
By using these rules, the complexity of the underlying implementation is hidden, thus even less experienced developers – or business users – can create new integrations (e.g. fetch data from a SOAP endpoint) as soon as they understand the basics, and they are shown an example. In bigger projects, it can lead to a significant improvement in task distribution and in the overall performance of the development – which results in faster delivery.

In case a special requirement cannot be satisfied by the default rule types, first, one can fall back to a more general type (e.g. using Connect SOAP for a specific CMIS requirement instead of the more targeted Connect CMIS) – and as a last resort, Java code can also be used with an Activity rule.

As different software use different data structures, during the processing of the request and response, mapping and transformation are usually needed. In an interaction, the data elements, that are formatted according to the external interface’s requirements, are stored on temporary pages, as instances of a so-called integration class (Int-). This intermediate layer forms an abstraction which makes processing easier and ensures that, if the external source changes, only a small part of the Pega application needs to be modified. For mapping between known formats, there are features which simplify the procedure: e.g. Parse Delimited rule for parsing a CSV and XML-Stream rule to generate a valid XML document.

---

82. Illustration: Connect SOAP rule form

By using these rules, the complexity of the underlying implementation is hidden, thus even less experienced developers – or business users – can create new integrations (e.g. fetch data from a SOAP endpoint) as soon as they understand the basics, and they are shown an example. In bigger projects, it can lead to a significant improvement in task distribution and in the overall performance of the development – which results in faster delivery.

In case a special requirement cannot be satisfied by the default rule types, first, one can fall back to a more general type (e.g. using Connect SOAP for a specific CMIS requirement instead of the more targeted Connect CMIS) – and as a last resort, Java code can also be used with an Activity rule.

As different software use different data structures, during the processing of the request and response, mapping and transformation are usually needed. In an interaction, the data elements, that are formatted according to the external interface’s requirements, are stored on temporary pages, as instances of a so-called integration class (Int-). This intermediate layer forms an abstraction which makes processing easier and ensures that, if the external source changes, only a small part of the Pega application needs to be modified. For mapping between known formats, there are features which simplify the procedure: e.g. Parse Delimited rule for parsing a CSV and XML-Stream rule to generate a valid XML document.

---

33 Content Management Interoperability Services: [https://en.wikipedia.org/wiki/Content_Management_Interoperability_Services](https://en.wikipedia.org/wiki/Content_Management_Interoperability_Services)
4.7 Version control, import-export, change management

4.7.1 Rule versioning and RuleSets, rule check-out

In Pega, all the development effort is centered around rules, these represent the configuration and program logic – so these need to be versioned. As they are stored in the database and have a proprietary representation, the usual tools, such as Git or SVN cannot be used, instead, there is a custom approach for version control.

Rules are organized into so-called RuleSets – similar to Java packages, they usually comprise of related functionalities – and within these, versions are defined. Based on multiple criteria (e.g. logged in operator, selected application), Pega chooses the appropriate rule every time from the proper RuleSet, with the proper version, attached to the proper class. This procedure is called rule resolution, and it provides the unique flexibility of the system. (Pegasystems, 2018h, sec.Rule resolution)

When defining an Application, the RuleSets with specific versions are referenced – this controls, which rules can be used from the system and which must be ignored in that context.
It is important to highlight, that not all rule types are versioned. There are e.g. the *Dynamic System Settings*, which act as global instances across the whole system – and they influence all applications. In the Java-world these would be e.g. configuration files, schema definitions. Because of their special nature, they need special handling – which is most notable during import-export of applications (see in the next chapter).

In addition to rule versioning, there is a little different, but still related concept, mostly to help cooperation between developers: rule *check-out*. When using this, before editing a rule, it must be *checked out* first. That means, a lock is placed on the rule to prevent simultaneous editing, and a copy is created in the developer’s *Private RulesSet* – which is unique to everyone. When the changes are done and tested, the developer has to *check-in* the rule, so that others can also see the modifications. During this, the changed rule is moved from the operator’s Private RuleSet back to the original one, the lock is removed, and a history entry with a comment is added to the rule – similar to a commit message.

---

84. Illustration: Booking application definition with RuleSets
These comments can later be viewed on the History tab, by pressing the View full history button, and as a snapshot is also created at each check-in, previous versions can also be inspected and restored by clicking the pencil icon in front of a row.

On top of these features, one can create branches, review and merge them – similarly to other version control solutions, and it is also possible to set up synchronization between a central node and developer systems.

In general, Pega’s version control works great. However, as it is completely custom-built, it requires developers to learn it from scratch – which takes time and results in strange situations when someone tries to force a familiar kind of thinking, based on different solutions, to Pega. Also, as rules reside in the database – in a very complex structure – they feel intangible, so in a case of an error, beginner developers can easily get lost and make even more mistakes while
trying to fix the issue. I think this mainly comes from the fact that rules represent very small parts of the application – while in Java, a class is one text file, in Pega every property, flow, etc. is a separate rule. On the other hand, this is the strength of the platform that modifications can be done at a granular level, so it is worth to invest the time in learning Pega’s approach.

4.7.2 Import-export

As the program logic is stored in a unique way, the transfer of the elements between systems also has a special format: packages, the so-called product files. These are jar / zip archives that contain the definition of the rules and database schema elements, plus data items that the developer wants to transfer.

To import a package, a wizard can be used (under Configure / Application / Distribution / Import). As a first step, the jar / zip file needs to be provided. Then the developer can choose, if he wants to follow the quick or the advanced path – the former imports all rules immediately, while the latter provides the ability to choose what to import. Database schema changes are calculated automatically by the system, and – depending on the configuration and security requirements – can be either automatically or manually be applied. In case of any issues (duplicate rules, errors) the wizard informs the developer and waits for his decision on how to continue. It is important to note, that as the rules are being imported, they immediately come into effect – this needs to be taken into account if there are active users logged in the system.

To export certain parts, there are multiple options. Applications and RuleSets can simply be exported using the export wizard (under Configure / Application / Distribution / Export), while for more complex situations a so-called Product rule can be created, which provides finer control over what to include in the package. In a Product, rules can be added based on Application and RuleSet as well, but also individual instances can be selected, conditions can be used for filtering class instances, and associated data / interlinked elements can be controlled. As this definition is saved as a rule instance itself, it is usually the preferred way for doing the export: like this everything required for an application can be listed at one place – it also serves as a documentation – and during the release preparation a complete package can be generated by a click of a button – or even by using automation.

Creating a good Product rule that contains everything needed, but no excessive items requires good knowledge of Pega and the particular system due to the many rule types and complex connections between them. It is not uncommon that, by accident, wrong rules are exported, or
configuration elements and security-related data (Operator rules) are transferred from a lower system to a higher (e.g. from development to testing) which can cause serious issues. At first sight, the simple wizards and interfaces look straightforward, but in the meantime, they hide – or do not emphasize enough – necessary information (e.g. about the connection between rules). Because of this, great care must be taken when creating a package, and its content must always be thoroughly examined.

Also, after importing a package, changes are usually hard to undo. Although, a restore point can be created before the import, and some modifications can be rolled back, there are certain elements that do not generate a history record, and thus, their changes cannot be reversed (most notably the SQL operations and some configuration data change) (Pegasystems, 2017d). The solution to this usually is, to use the underlying platform’s restore features, e.g. by creating full database backups before releases – but it is usually a cumbersome and time-consuming task.

4.7.3 Process change management

Before a new version of an application is deployed, changes must be assessed, and their impact must be planned. In a live system, important data is kept and many processes can be active – so any kind of information loss or corruption needs to be avoided. Pega does not provide a specific tool for migration between process versions, so a developer needs to design a custom approach.

There are generally three possible ways. One is to allow the users to use two versions of the application: the previous one for in-flight cases, and the newly deployed one for everything else. Another possibility is to design new flows in a way, that they only extend the previous version – elements are only added to, but not removed from the diagram. This is required, because if a flow is at a point, which is removed from the new version, the system doesn’t know how to continue, and it causes an error. The third option is to create a custom migration logic, e.g. using Activities, that do the mapping between the old and new versions. (Pegasystems, 2018c, sec.How to manage flow changes for cases in flight)

4.8 Process monitoring, troubleshooting

Although there are some out-of-the-box features which can be used for monitoring ongoing processes, e.g. reports and charts on the Case Manager portal’s dashboard (see the user interface chapter), there is no high-level default functionality which provides a complete
overview of all cases / processes in the system. Usually, these are built for every project, based on the specific requirements.

For technical troubleshooting, however, there are more tools provided: from Developer Studio flow and assignment errors can be listed, locks can be checked and there are some more built-in reports. On top of that, the Admin Studio offers a good overview of users, requestors, nodes, background processing functionalities (agents, jobs, queue processors) and some other system components. The system logs are also available from the Developer Studio – and as they are normal text files, they can also be downloaded from the server. Besides these, the debugging tools mentioned in the *Developer tools* chapter are also extensively used for troubleshooting.

Pega also provides an external application for system monitoring (analyzing exceptions, checking background processing, following node states, etc.). Its on-premise version is called Autonomic Event Services (AES) 34, while the nowadays more supported and more up-to-date cloud version is the Predictive Diagnostic Cloud (PDC) 35. These, based on the information gathered from the managed systems, provide a complete overview of issues, and also suggest improvements based on expert knowledge and AI-based evaluations.

---

34 Pega Autonomic Event Services: [https://community.pega.com/knowledgebase/products/autonomic-event-services](https://community.pega.com/knowledgebase/products/autonomic-event-services)


---

87. Illustration: Pega Diagnostic Cloud, Enterprise overview (Pegasystems, 2018e screenshot at 0:48)
4.9 Reporting and case auditing

4.9.1 Report Definition

Pega offers built-in features for querying the database and generating reports out of the data. The most used functionality is the so-called Report Definition rule – which is, with some simplification, an abstraction above an SQL query. As such, it serves both as the basis of user-facing reports and, charts, but it is also used extensively to fetch data for internal operations and calculations. The results can be processed by other rules, like Activities, and often, they are cached via Data Pages – that is a special kind of page which acts similarly to a lazy-loading function: fetches the data only when needed – declaratively – and returns it without running another query operation until it gets stale.

88. Illustration: Report Definition rule form

On the rule form of a Report Definition, the elements of an SQL query can be identified: the SELECT part is the top section, the columns, the WHERE part is the filters on the bottom, while the FROM is the class which the rule is defined on, here the Work- Class (database table) and subreport (subquery) joins can be configured on the Data Access tab, along with result limits and other data-related options.
The report above is a built-in, user-facing one, that is available by default from the Case Manager portal’s Reports menu item.

There are many out-of-the-box reports that can be used as-is – or as a basis for customization. This comes in handy when the new application’s and Pega’s capabilities need to be showcased to the customer who can, like this, have an idea of what is possible to report on right from the start. In addition, not only developers can create or modify reports, but the end-users also have the ability to put together custom queries and visualize them as lists or chars.

89. Illustration: Report results displayed in the Case Manager portal

There are many out-of-the-box reports that can be used as-is – or as a basis for customization. This comes in handy when the new application’s and Pega’s capabilities need to be showcased to the customer who can, like this, have an idea of what is possible to report on right from the start. In addition, not only developers can create or modify reports, but the end-users also have the ability to put together custom queries and visualize them as lists or chars.

90. Illustration: Report editor
The editor interface looks similar to other reporting tools – or Microsoft Excel pivot tables – so business users can feel comfortable while using it. Reports can be kept private or made public, and the result can also be exported to PDF or Excel formats.

4.9.2 Business Intelligence Exchange

Even though there are great built-in reporting features, sometimes they are not enough – or due to performance reasons, it is better to do reporting from an external database, with an external tool. When running a Report Definition, the same computational resources are utilized as for displaying the application user interfaces, running flows and background processes – so when dealing with large amounts of data, this can severely impact the speed of the system. Or it can also be the case, that the company has a data warehouse, and they would like to do centralized reporting. For this, Pega offers the Business Intelligence Exchange (BIX) [36] tool to extract and export data from its database in a controlled manner.

The exports can be run by a platform agent, and they can also be initiated from a CLI tool – which provides the opportunity for creating a completely automated ETL (extract, transform, and load) process chain, involving multiple systems. The output can either be XML or CSV files, and Pega can also insert the output records directly to an external – transfer – database. (Pegasystems, 2019c, sec.Extract rules / File Specification)

One might wonder, why it is all necessary, why cannot we just simply read the data directly from the internal database – without using any intermediary tools. It is also a feasible design choice, however as Pega stores items in a BLOB column by default, this would require the developer to expose all properties that need to be exported (see the data storage chapter). That results in more columns in the tables, greater complexity, and also higher usage of database storage space. What is the best, needs to be decided for the particular project individually, but it’s good that there are more options to choose from.

4.9.3 Case auditing

While working on a case, many important events happen that need to be tracked. Assignments can be transferred to other operators, work parties can change, and errors can occur. To follow all this, Pega logs audit events, and by default, displays them for every case on the information section’s Audit tab.

The list of saved events can be customized for every case to be able to comply with business rules and laws – and to save disk space and processing power by omitting unimportant items. It is also possible to track the change of certain properties in addition to general events, this is called *Field Level Auditing*.

These information are saved to separate *History-* tables (classes), and as these are just like any other classes, reports can be built on them, e.g. to find out how long it usually takes for a case to finish, or at which stage is the most time spent.

### 4.10 Vendor and expert support

As Pega is an enterprise, proprietary software, general support is offered by the vendor itself. According to my experiences, it has a decent quality, operators usually answer support requests (tickets) fast, and if the problem is because of a bug within the platform, they acknowledge it, and engineers start working on the issue. After that, it requires some time for them to come up with a solution (depending on the agreed level of support and the urgency of the problem), but in most cases, they have so far succeeded in correcting the error and providing a fix.

The other leg of the support is the partner companies which work with the software, and offer consulting services. The quality and availability of resources here is a more interesting topic. As Pega is a closed source, expensive application, people usually get to know it only while working at a bigger organization – it is uncommon that someone learns it at home in an autodidactic way, just like it can happen with smaller open-source tools. In the past, it also meant that, although the demand was high for good architects, there were not enough properly
qualified resources available. Noticing this, courses in Pega Academy – that had previously been pricey and one of an important source of income for the company – were made free from Summer 2017 (Pegasystems, 2018f). On top of this, Pega recently gave an opportunity for engineers to do the training and get the highest level system architect certificate for free (Pegasystems, 2019b). There is also a community site 37 that is – by now – widely used, many new articles have been written to present and explain various features of the software in more detail, tech-talks, and other events are held to spread information. In my opinion, these are great steps in the direction to opening up the developer community – and thus improving support options by a great degree – but it’s a transformation in progress.

Thus, it is important to note, that, although the recent improvements are good, the above facts mean, that Pega consultants are currently expensive and sparse resources – that needs to be taken into account, otherwise, a project can run out of budget or stall due to the lack of experts. Some companies choose to train an internal team to mitigate these problems, however, in my experience, it requires approximately a year for a new developer to become productive, and at least two years to be able to work on end-to-end features alone – this is also a huge investment in time and money.

That said, the platform can provide so much advantage over others – if properly used – that it is worth the price for certain projects. Pega provides integrated AI features, has many ready-made strategic applications that accelerate development, and is a market leader in its field, not by accident. The important is, that it must be thoroughly examined if the project at hand really requires all the advanced features of Pega, and if it is the budget is big enough to coop with the high cost of resources – even if multiple iterations are needed to get the proper result.

37 Pega Community: https://community.pega.com/
5 Summary

In this thesis, the nowadays very popular and rapidly changing field of business process management was examined. At the beginning, I defined criteria based on my prior experiences: what is expected from tools, so that they can be used effectively for providing value to clients. To have a common language to discuss modeling ideas, and have a basis to start from, I introduced three commonly used, powerful modeling standards – BPMN, CMMN, DMN – and through these, the sample process for evaluation, Booking. Then, as the main topic, I analyzed two very different process management systems along the same points, with a similar structure, so that they can be compared.

As they follow vastly different approaches, they are good for different tasks. In my opinion Camunda is better for back-end processing and orchestration between other systems – as it’s engine supports more modeling features through the implementation of the BPMN standard, and by remaining closer to the code-level, it is easier with it to react to completely new, unexpected challenges. On the other hand, Pega offers a low-code approach with many unique features that speed up development for common use cases – and that can result in a much reduced time-to-market for applications, which means a huge business advantage.

There are also many other great tools available, and this research can later be extended to cover them as well. To me, the most interesting ones are Appian 38 and IBM 39 for being a close follower of Pega in multiple comparisons, and Bonitasoft 40, for being an open source platform with many features.

---

38 Appian: [https://www.appian.com/](https://www.appian.com/)
40 Bonitasoft: [https://www.bonitasoft.com/](https://www.bonitasoft.com/)
Bibliography


Illustration Index

1. Illustration: Planned investments of Hungarian companies according to the ITSM research in 2016 (Szabó, 2017, fol.16 translated by the author) ................................................................. 2
2. Illustration: Simple process ................................................................................................................. 11
3. Illustration: Process with decision ....................................................................................................... 11
4. Illustration: Parallel execution .............................................................................................................. 12
5. Illustration: Parallel splitting, exclusive merging .................................................................................. 12
6. Illustration: Lanes ................................................................................................................................. 13
7. Illustration: Multiple pools ................................................................................................................... 14
8. Illustration: Collapsed pool .................................................................................................................. 14
9. Illustration: BPMN task types (in Camunda Modeler) ......................................................................... 15
10. Illustration: Collapsed subprocess .................................................................................................... 15
11. Illustration: Call activity ..................................................................................................................... 16
12. Illustration: Events .............................................................................................................................. 17
13. Illustration: Terminate end event ....................................................................................................... 18
14. Illustration: Booking ............................................................................................................................ 19
15. Illustration: Risk Assessment case ..................................................................................................... 20
16. Illustration: The life cycle of a task in CMMN (Freund and Rücker, 2016, fig.5.3) .................. 21
17. Illustration: Sentries and exit criteria ............................................................................................... 22
18. Illustration: Stages, milestones, events ............................................................................................... 23
19. Illustration: Booking reservation ........................................................................................................ 24
20. Illustration: Booking reservation in BPMN ......................................................................................... 24
21. Illustration: Decision table ................................................................................................................ 25
22. Illustration: Decision requirements diagram .................................................................................... 26
23. Illustration: Booking expense approval ............................................................................................. 27
24. Illustration: Booking approval action ................................................................................................ 28
25. Illustration: Deployments in Camunda Cockpit .................................................................................. 28
26. Illustration: Camunda Modeler ......................................................................................................... 32
27. Illustration: Model causing CMMN transition error ....................................................................... 32
28. Illustration: Variable mapping in Camunda Modeler ......................................................................... 34
29. Illustration: Camunda Tasklist .......................................................................................................... 36
30. Illustration: Generated form definition in Camunda Modeler .......................................................... 36
31. Illustration: Camunda Tasklist - REST queries while displaying a task .......................................... 37
32. Illustration: Custom end-user interface for case management in Camunda .................................. 38
33. Illustration: Task Authorizations in Camunda Admin ..................................................................... 38
34. Illustration: REST connector configuration in Camunda Modeler ............................................... 41
35. Illustration: Process migration in Enterprise Cockpit (Camunda, 2019y) ..................................... 41
36. Illustration: Camunda Cockpit - process definition view ............................................................... 42
37. Illustration: Camunda Cockpit - process instance view ................................................................. 42
38. Illustration: Camunda Cockpit – evaluated decision table .............................................................. 43
39. Illustration: Runtime and History Databases, History Event Stream (Camunda, 2019p) ........ 44
40. Illustration: Custom user interface for process instance and variable instance history .............. 45
41. Illustration: Camunda Optimize, Dashboard (Camunda, 2019i) ...................................................... 46
42. Illustration: Starting the New Application Wizard .......................................................................... 46
43. Illustration: New Application Wizard, Advanced configuration .................................................... 47
44. Illustration: New Application Wizard, newly-created admin user .................................................. 48
45. Illustration: Booking case .................................................................................................................. 49
46. Illustration: Validate Expenses flow ................................................................................................ 50
47. Illustration: Flow elements .............................................................................................................. 51

112
48. Illustration: Routing to the Requester case participant........................................64
49. Illustration: Approval flow configuration.................................................................65
50. Illustration: Decision shape properties.................................................................66
51. Illustration: Book Trip flow - with parallel Accommodation and Travel booking....67
52. Illustration: Split Join with two referenced flows.....................................................67
53. Illustration: The transcript of the Split Join shape in BPMN.....................................68
54. Illustration: Booking Finished - When rule...............................................................68
55. Illustration: Optional actions..................................................................................69
56. Illustration: Booking class structure.......................................................................70
57. Illustration: Class inheritance................................................................................71
58. Illustration: Property types.....................................................................................72
59. Illustration: Some default properties of a case.......................................................74
60. Illustration: Provide Details view in Case Designer................................................75
61. Illustration: Provide Details section using a Design Template.................................75
62. Illustration: Provide Details section with full design capabilities............................76
63. Illustration: Section elements...................................................................................76
64. Illustration: Skin rule...............................................................................................77
65. Illustration: Case Worker portal, My Work...............................................................78
66. Illustration: Assignment in Case Worker portal.......................................................79
67. Illustration: Case Manager portal............................................................................80
68. Illustration: Operator record, Work, Routing...........................................................81
69. Illustration: GetNextWork button added to Case Worker portal..............................81
70. Illustration: Dev Studio...........................................................................................82
71. Illustration: Flow records in Records Browser.........................................................83
72. Illustration: Package import from the Configure menu.............................................84
73. Illustration: Debugging tools....................................................................................84
74. Illustration: Clipboard tool, pyWorkPage of a Booking case......................................85
75. Illustration: Tracer....................................................................................................86
76. Illustration: The content of the pyWorkPage at the 599th step................................86
77. Illustration: Live UI, with the Name field of the Provide Details section selected.......87
78. Illustration: API Activity that modifies the status of a case (Open, Resolved, etc.)....88
79. Illustration: Access Role Name with Access of Role to Object records.....................89
80. Illustration: Setting up required privilege for a Flow Action..................................90
81. Illustration: Out-of-the-box connector and service rules.........................................92
82. Illustration: Connect SOAP rule form.....................................................................93
83. Illustration: UI-Kit-7 RuleSet with multiple versions...............................................94
84. Illustration: Booking application definition with RuleSets.....................................95
85. Illustration: Checking in a rule...............................................................................96
86. Illustration: Rule history.........................................................................................96
87. Illustration: Pega Diagnostic Cloud, Enterprise overview (Pegasystems, 2018e screenshot at 0:48)...99
88. Illustration: Report Definition rule form.................................................................100
89. Illustration: Report results displayed in the Case Manager portal...........................101
90. Illustration: Report editor.......................................................................................101
91. Illustration: Case history.........................................................................................103