

Dirk Jan van Lujtelaar

RAINTOOLS SOFTWARE DEVELOPMENT

RAINTOOLS SOFTWARE DEVELOPMENT

Dirk Jan van Lujtelaar
Bachelor's thesis
Spring 2015
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Dirk Jan van Luijtelaar

Title of the bachelor's thesis: RainTools Software Development

Supervisor: Teemu Korpela

Term and year of completion: 2015 Number of pages: 56 + 16 appendices

The aim of this Bachelor's thesis was to develop the RainTools software package for the customer, Stichting RIONED, and learn about the process of software development. The main aim of this Bachelor's thesis was to learn the capabilities and possibilities of C# in combination with WPF and XAML as opposed to regular WinForms.

To achieve this brainstorm began to develop a user interface for the customer to translate input data to XML, feed it to a third party application, read the results from XML, translate it back and display the results in tables and graphs. The key here being that the users do not have to interact with the third party application, the user interface does this for them.

The software was developed using the MVVM architectural pattern. The WPF view was constructed using the XAML language. The model and the view model were constructed using the C# language. Using MVVM allows for easy transitions between different datasets by simply loading existing or empty view models.

The results were astonishing. Together with Stichting RIONED I have created an elegant yet functional user interface. The data input into the third party application has been simplified and informative tables and graphs are presented.

Meanwhile I have learned firsthand the endless possibilities of WPF and XAML but also its caveats. The software development process has been a great learning experience; in particular version control (GitHub), usability, legal issues and product testing.

Keywords: C#, XAML, WPF, WinForms, user interface, XML, tables and graphs, MVVM, version control, usability, legal issues, product testing

PREFACE

The concept of RainTools was created in the summer of 2013, however the development did not start until a year later when the project was accepted as a Bachelor's thesis. In the meantime an Excel spreadsheet was developed to feed the third party application. The work for the Bachelor's thesis was mostly done in Oulu, some of it in the Netherlands. Due to it being an independent software development project the work could be done from home, however regular Skype meetings with the customer's representative took place.

The work was commissioned by Stichting RIONED. All the communication with Stichting RIONED went through the client's representative.

Teemu Korpela supervised the Bachelor's thesis. While his influence on the project was limited to two meetings, he raised several interesting points, for example he was the first to bring up several legal issues that could come into play during the project. We kept in contact through email. He was very responsive and available for assistance whenever needed.

The client's representative was my father Harry van Lujtelaar, a senior project manager at Stichting RIONED. He was pleasant to work with, because we are two minds who do not always think alike but need very little communication to understand each other's point of view. His main job was to communicate the client's wishes to me, make sure that the user's best interests were protected during the development, assist in testing for bugs and coordinate the testing phase.

A lot of communication took place between me, Harry van Lujtelaar and Peter Ganzevles, the creator of the third party application in question (the RainTools Engine). For him the interface was a useful tool to test his application for bugs as it allowed him to communicate with the RainTools Engine a lot easier than creating the required XML files himself. In addition, rather than reading an XML file with thousands of lines the graphical results produced by the interface allowed an easy error checking.

A source of external help worth mentioning is Stack Overflow [33], a digital community where programmers can ask and answer questions. Whenever a seemingly unsolvable problem presented itself someone over at Stack Overflow had the answer. As a last resort this community has been a great help and have taught me a lot of neat programming tricks.

Oulu, 28th September 2015

Dirk Jan van Lujtelaar

CONTENTS

ABSTRACT	3
PREFACE	4
CONTENTS	5
VOCABULARY	7
1 INTRODUCTION	9
2 RAINTOOLS	10
2.1 History	10
3 VERSION CONTROL	13
3.1 GitHub	13
3.2 Redmine	16
3.3 Visual Studio Online	17
4 DATABASE	18
4.1 Microsoft Azure	18
4.2 MSSQL & MySQL	20
4.3 Firewalls	21
5 PROGRAMMING	23
5.1 C#	23
5.2 XAML & WPF	25
5.2.1 WinForms	26
5.3 OOP & MVVM	27
5.4 Styles	27
5.5 File management	28
5.6 Threading	31
5.7 Exporting data	31
5.7.1 XML	32
5.7.2 Excel	33
5.8 Error handling	34
5.9 UserControls	37
6 VISUALIZATION	38
6.1 MahApps Metro	38
6.2 Graphs	39

6.3 Animations	43
7 LEGAL ISSUES	45
7.1 EULA	45
7.2 Privacy policy	45
7.3 Disclaimer	46
8 TESTING	47
8.1 Test users	47
8.2 Distribution	47
8.3 Monitoring	48
9 ENTREPRENEURSHIP	49
10 CONCLUSION	50
REFERENCES	51
APPENDICES	56

VOCABULARY

VOCABULARY	
TERM	DEFINITION
C#	a programming language developed by Microsoft [1].
GUI	<u>G</u> raphical <u>U</u> ser <u>I</u> nterface, the graphical representation of a user interface [2].
HTML	<u>H</u> ypertext <u>M</u> arkup <u>L</u> anguage, a programming language to develop websites [3].
LINQ	<u>L</u> anguage- <u>I</u> ntegrated <u>Q</u> uery, a component that adds data querying capabilities to .NET languages such as C# [4].
MSSQL	<u>M</u> icrosoft <u>S</u> QL Server, a relational database management system developed by Microsoft, used to store and retrieve data [5].
MVVM	<u>M</u> odel, <u>V</u> iew, <u>V</u> iew <u>M</u> odel; an architectural pattern used to structure applications [6].
MySQL	an open source relational database management system now being developed by Oracle [7].
OOP	<u>O</u> bject- <u>O</u> riented <u>P</u> rogramming, a programming structure based on the concept

VOCABULARY

	of using objects; data structures containing attributes and methods [8].
Stichting	Dutch translation of "foundation" [9].
SQL	<u>S</u> tr <u>u</u> ctured <u>Q</u> u <u>e</u> ry <u>L</u> anguage, the language of relational database management systems such as MSSQL and MySQL [10].
VBA	<u>V</u> isual <u>B</u> asic for <u>A</u> pplications, an implementation of Microsoft's Visual Basic 6 in Microsoft Office [11].
WPF	<u>W</u> indows <u>P</u> resentation <u>F</u> oundation (originally called Avalon), a graphical subsystem used to render user interfaces for applications developed in .NET [12].
WinForms	a class library included in the .NET framework used to render GUIs [13].
XAML	<u>E</u> xtensible <u>A</u> pplication <u>M</u> arkup <u>L</u> anguage (originally <u>E</u> xtensible <u>A</u> valon <u>M</u> arkup <u>L</u> anguage), an XML based language used to initialize structured values and objects [14].
XML	<u>E</u> xtensible <u>M</u> arkup <u>L</u> anguage, a markup language using a structure that formats documents in a way that is readable for both human and machine [15].

1 INTRODUCTION

The aim of this thesis was to experience and learn the process of developing software for an organization. When developing for an organization it is important to document everything carefully, as opposed to developing for private use.

The objectives of the thesis were to get accustomed to using version control, code using an object-oriented approach and using the Model View ViewModel architectural pattern. As a result the application developed will be a structurally solid application with an extensive documentation about its previous versions and progress available in the version control system.

My previous software development work was done using the WinForms GUI library. WinForms is an outdated library therefore the main task in this thesis was to learn how to create applications using WPF in combination with XAML and explore their possibilities, allowing for more sophisticated user interfaces.

When the application has been fully developed the testing phase will begin. In this phase the application is tested for usability (user friendliness) and bugs. Bug fixing is one of the most important steps in the software development process. It is important that the application is easy to use as the application is to be used by both inexperienced and advanced users, therefore usability is a high priority.

The company for which the thesis work was done is Stichting RIONED, an umbrella organization for urban water and sewerage in the Netherlands. Within RIONED many professional parties participate; governments (local, state and provincial), companies (suppliers, consultants, inspection companies and contractors) and educational institutions. RIONED's main task is to provide knowledge to the professional world. This is achieved by doing research, bundling existing knowledge and organizing meetings of the mind.

2 RAINTOOLS

RainTools allows users to simulate the functioning of rainwater facilities under load of individual (extreme) rain showers and multi-year precipitation series. The water balance of small scale facilities can be simulated in detail, the functioning of bigger scale systems can be assessed more roughly.

The goal of RainTools is to give a broad scope of users (from beginners to experts) insight in the functioning of rainwater systems. In order to determine the scope of the performance of a system both individual (extreme) rain showers and multi-year precipitation series are used. There are often remarkable differences between the results of rain showers and longer precipitation series.

RainTools is equipped with a set of specific tools to gather the required information on different situations and to display the relevant results. The different tools are tailored for specific user groups each having its own interface. There are several tools for inexperienced users the rest is for advanced use only.

RainTools is available through Stichting RIONED for its participants: more than 400 municipalities in the Netherlands, water boards, consultancies and educational institutes. It is possible that the application is going to be used by plumbers and gardeners who often have to deal with water management on a private property.

2.1 History

In the summer of 2013 an early version of the RainTools Engine already existed, similar to now it was a command line application that required XML files to run. The initial idea was to create a Microsoft Excel spreadsheet with all the required input data and write a Visual Basic routine to translate it to XML. I had a lot of experience working with Excel and Visual Basic so I definitely had the skills to create this application.

After about a month or two of work the first version of the spreadsheet was done. We quickly realized that the amount of data required was enormous as

can be seen in FIGURE 1. On the left a large table with all the required data and on the right a graphical representation of the data in the table. After seeing the result, we decided not to continue with this line of approach. The sheer amount of data was simply overwhelming and from a usability standpoint this was not the application the customer was looking for.

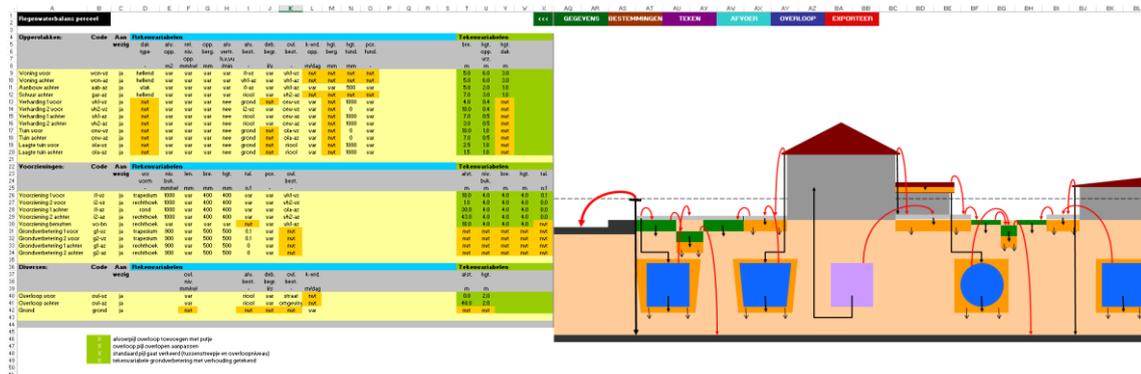


FIGURE 1. First version of the RainTools interface in Excel.

In the meantime I had created my first large WinForms application, this was also an application originally created in Visual Basic but after a lot of compatibility issues with Microsoft Excel I decided to port it to C#. When the first version of the RainTools interface didn't work out I presented the idea of creating the interface in C# instead. This idea was accepted upon seeing the results of my WinForms application.

To be able to continue the work on and working with the RainTools Engine a temporary interface was created in Microsoft Excel, see FIGURE 2 and FIGURE 3. This interface consisted of a spreadsheet with a large table (imported from XML), a button to translate the table to XML and a button to read the results from XML and update the data in the graphs on that same spreadsheet. This spreadsheet is still used to date whenever something new, unsupported by the RainTools Interface, needs to be tested.

FIGURE 2. Data table of current temporary RainTools interface.



FIGURE 3. Results and graphs of current temporary RainTools interface.

The temporary interface has its pros and cons. A big pro is the fact that it allows for very irregular and custom input, allowing the developer of the RainTools Engine to try out very unconventional settings and testing of new features before they are implemented in the RainTools interface. Another advantage is the possibility of easy data interpretation; in the spreadsheet all the raw data is available, combined with Excel's built-in graph library this makes analysing the data rather simple. The big downside is the usability; for anyone other than developers the spreadsheet is unusable because it is too complicated and unintuitive, even the developers sometimes get lost in the vast amount of input data in the spreadsheet and end up making mistakes as there is no data verification of any kind.

After the spreadsheet had been created the work on the actual RainTools interface began.

3 VERSION CONTROL

Version control is the first thing I took into consideration when starting the Rain-Tools software development project. Previously I have had a lot of trouble with version control and keeping it up to date. I have had experience with Redmine and Visual Studio Online but found both inconvenient to use. I was searching for an easy to use version control that allowed me to keep track of updates and bugs [18]. I also searched for something that works well with Visual Studio [16]. This led me to try out GitHub, an online version control that can be connected to Visual Studio [17].

3.1 GitHub

GitHub has several features that made it stand out. GitHub can be connected to Visual Studio, as of version 2015 it is integrated into Visual Studio making it even easier to use [19][23].

Commits and Files

Whenever a file is saved in Visual Studio, it can be committed to the GitHub cloud. GitHub keeps an extensive record of each commit and file. When looking at the details of a commit, the number of changed files and how many code lines have been added and removed in each file can be seen.

GitHub keeps records for every file on the cloud, most important the file's history and changes. From within Visual Studio you can compare the current file to previous iterations of that file making it easy to revert back any incorrect changes that have been made.

Activity

When developing software it is important to keep track of how long you have been working on a certain feature, GitHub helps to accomplish this. GitHub keeps an activity log in several different forms.

The first is a punch card system similar to those used in companies where employees punch in when they come to work and punch out when they leave.

GitHub submits a punch every time a change is committed to the cloud; this helps to keep track of how many hours of work has been done per day.

FIGURE 4 shows the punch card of RainTools. It shows that I have been working on RainTools for 7 days a week and I prefer to work late hours rather than early hours.

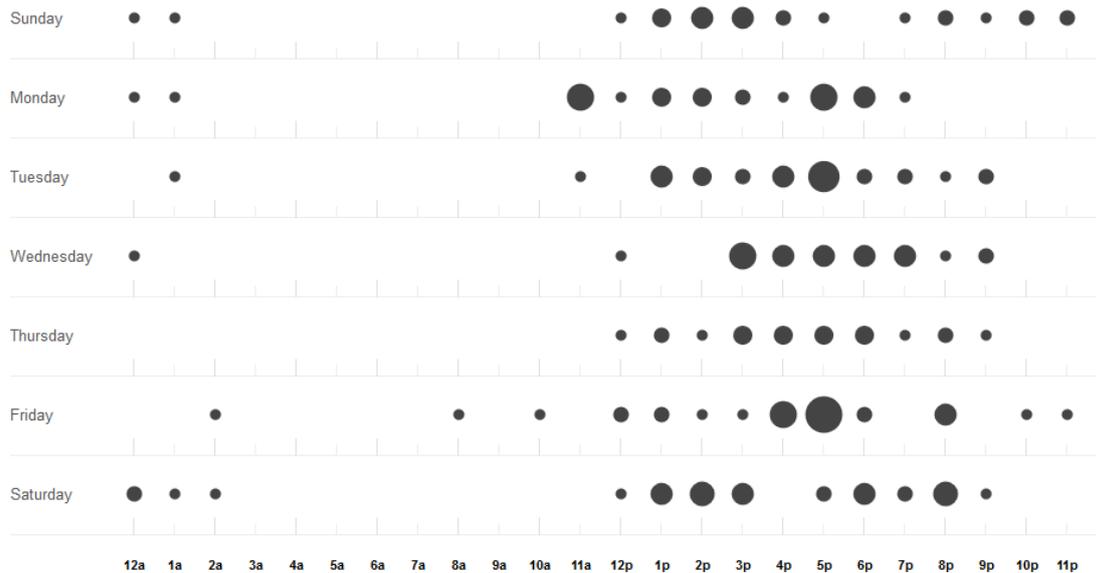


FIGURE 4. Punch card of RainTools. [19]

The second is a graph showing the number of commits per day for a selectable period of time (code frequency). This graph gives an idea of how many and what days I have been working during a specified period of time, using this combined with the punch card the hours spend on a project can be calculated roughly.

FIGURE 5 shows the code frequency of RainTools. It shows the number of added and removed lines of code over the past couple of months. It can be seen that most of the work was finished around April/May. It is interesting to note that towards the end the work mostly consisted of optimizing the existing code. Therefore, there are less lines of code added and removed.

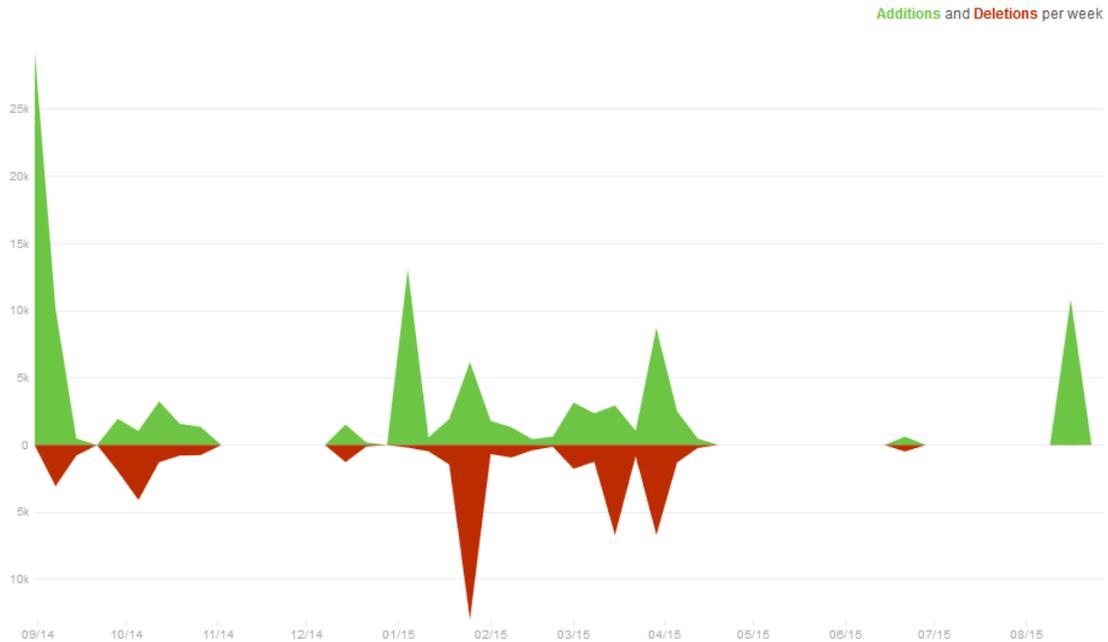


FIGURE 5. Code frequency of RainTools. [19]

Issues

GitHub can keep track of a list of issues. I have used this extensively while developing RainTools. Harry van Lujtelaar’s main role as a liaison between me and the customer was to translate the customer’s wishes to me and be the initial bug tester. Whenever he encountered a bug or a “wish”, he would create an issue on GitHub letting me know a feature needed to be fixed or added.

We turned the GitHub issue list into a combination of a bug list and a to-do list. When closing an issue you can attach a message and a commit to it, letting the customer know that the issue has been fixed or added. As a result the list of closed issues became a change log, with each change being documented in a specific commit.

FIGURE 6 shows the current open “issues” of RainTools, however they are in Dutch. It is my personal to-do list, for example the first item is to build a .chm file system (Microsoft Compiled HTML Help). This is a collection of HTML pages that the user can use when he needs help, a manual of sorts. It is used often in applications and can usually be called by pressing control and “H”.

<input type="checkbox"/>	8 Open	105 Closed	Author	Labels	Milestones	Assignee	Sort
<input type="checkbox"/>		Help bestand in chm format bouwen					1
		#103 opened on Mar 27 by Harrer					
<input type="checkbox"/>		Perceel onderdelen schakelbaar					0
		#93 opened on Mar 17 by Harrer					
<input type="checkbox"/>		Overloop/lediging bestemmingen niet-logische keuzes weglaten					1
		#90 opened on Mar 15 by isokay					
<input type="checkbox"/>		Beginwaarden invoergegevens variabel per project					1
		#89 opened on Mar 14 by Harrer					
<input type="checkbox"/>		Inloop importeren					0
		#82 opened on Feb 28 by isokay					
<input type="checkbox"/>		Omschrijving project					0
		#59 opened on Feb 12 by Harrer					
<input type="checkbox"/>		Hulp schermen per tabblad					0
		#33 opened on Jan 10 by Harrer					
<input type="checkbox"/>		Aantal aangepaste buien variabel					0
		#27 opened on Dec 20, 2014 by isokay					

FIGURE 6. Issues list of RainTools. [19]

Online cloud

The fact that GitHub is online is another advantage. While developing RainTools I worked from several different locations and computers. GitHub made all of this possible effortlessly due to the integration with Visual Studio. Whenever I started the project on a different computer, I could simply pull the latest version from GitHub and continue working where I left off.

3.2 Redmine

Redmine has a lot of features and possibilities, however I disliked using it. I am of the opinion that the version control is something that needs to be automated as much as possible. Redmine allows you to keep track of a lot of things, unfortunately every detail has to be entered in the system manually [21].

To add insult to injury there is no connection possible between Visual Studio and Redmine. This means that to work from multiple locations a third party software like Dropbox or any other cloud storage has to be used. This makes using Redmine a hassle and a job on its own. Therefore, I opted for GitHub over Redmine.

3.3 Visual Studio Online

In my search for the version control I stumbled upon Visual Studio's own version control system called Visual Studio Online. Even though they come from the same software package, I was very disappointed by the connectivity between the two, and as I explained in the Redmine section; I am looking for an intuitive version control [22].

Similar to Redmine, Visual Studio Online requires a lot of details to be filled in before a task can be started. GitHub on the other hand requires nothing in advance and the details can be filled in after committing a change rather than before [20].

An example is while Visual Studio Online asks you how long you have been working on a task by estimate, GitHub attempts to calculate it using the amount of code that has been changed and how much time has passed between each commit. My preference went out to the automated system of GitHub.

4 DATABASE

Early on in the development of RainTools the customer decided to restrict the use of the software to a specific group of people. Thus, it was decided that a user registration database would be required. When the customer decided to also save some usage data, this decision was solidified and I started looking for a database service in the cloud.

Initially, I tried to host the database on my personal website which featured standard MySQL databases. However, I found that when working with C# and Visual Studio, using MSSQL databases is highly preferred over MySQL. Therefore, I looked for a suitable MSSQL host and my eye turned to Microsoft Azure, which I had briefly touched upon during one of my courses [24][25][26].

4.1 Microsoft Azure

Microsoft Azure is a computing platform and infrastructure in the cloud. It is used to host and manage applications. They also provide data services such as cloud storage and databases, which I am using.

The user database of RainTools is hosted at Microsoft Azure, due to both the database and Visual Studio being a Microsoft product connecting the two is very smooth and easy. Without going into too much detail for security reasons the database mainly consists of two tables, one for accounts (FIGURE 7) and one for sessions (FIGURE 8).

[dbo].	
ID	int
Email	varchar(255)
PID	varchar(255)
DatumAanmaak	varchar(255)
AantalSimulaties	int
TOS	int
AantalBekeken	int

+ Add an index.

CID -

FIGURE 7. Accounts table properties (Microsoft Azure), table name is omitted.

To avoid any security liabilities it was decided not to store any personal information other than an email address, which means no passwords. To still be able to control who can access the application, a whitelist was created. This whitelist consists of email addresses and domains, for example anyone with an email address of the domain raintools.nl can access RainTools.

To prevent multiple users using the same email address from accessing RainTools, I came up with the idea of binding the email address to a single computer. This is done by generating a unique computer ID, which is generated when a user first logs in. This ID is a combination of the motherboard ID, hard disk ID and processor ID, providing a unique ID in 99% of the cases.

Whenever a user logs in, the computer ID is generated and compared with the ID stored in the database. If it matches, the user can log in. If it does not match, the user will have to confirm the new location with a security code sent to the specified email address [74][75].

The system is not completely waterproof, two users can still share one account by exchanging the security code constantly. However, this is quite inconvenient and we believe that it is enough trouble to keep people from doing this.

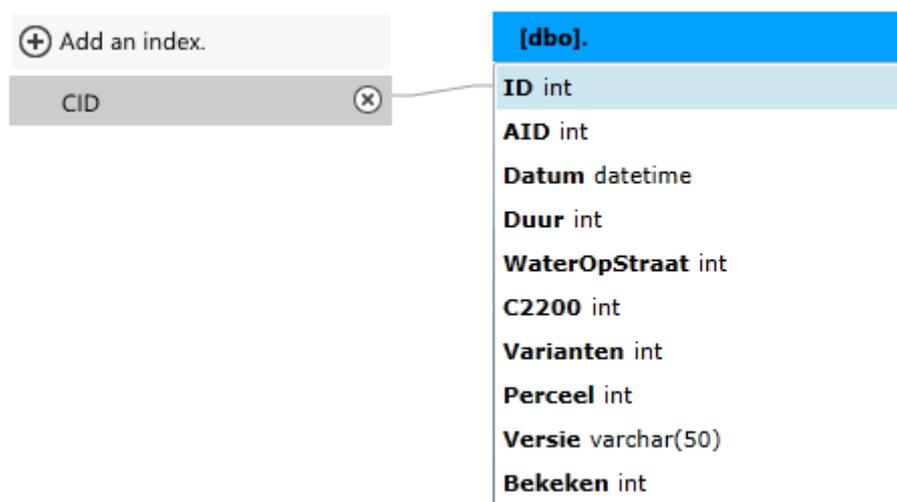


FIGURE 8. Sessions table properties (Microsoft Azure), table name is omitted.

The second table stores session information each time the user logs in. This information is used to monitor the usage of RainTools. The information stored is not harmful to the user, e.g. when the session was logged, how long the session was, how many simulations were done and what version of RainTools was used. This information is useful in the bug testing phase, but also to see what the most used tools are and to help prioritise further development.

4.2 MSSQL & MySQL

As explained earlier, due to connection issues between Visual Studio and MySQL databases I had to switch to MSSQL (Microsoft SQL Server). At first sight they seemed completely similar until I encountered a query that did not function even though I was confident it was correct. I researched the difference between MSSQL and MySQL and found why my query was not working.

EQUATION 1. MySQL query to select a single account.

```
SELECT * FROM AccountsTable WHERE Email = EmailAddress LIMIT 1
```

EQUATION 2. MSSQL query to select a single account.

```
SELECT TOP 1 * FROM AccountsTable WHERE Email = EmailAddress
```

As can be seen in EQUATION 1 and EQUATION 2 there is a slight difference in syntax between MSSQL and MySQL. There are also other differences in the syntax between the two but they are very small. I only encountered one other difference when creating the tables. Where MySQL uses AUTO_INCREMENT to create an index that automatically increases, MSSQL uses IDENTITY.

I was curious about the differences between MSSQL and MySQL other than a few syntax differences. It was not very easy to find and most of it boiled down to preference. The main difference seemed to be that MySQL is available for many platforms whereas MSSQL only runs on Windows platforms. TABLE 1 displays the differences I found between MySQL and MSSQL.

TABLE 1. Difference between MySQL and MSSQL [27][28][29][30].

DIFFERENCE BETWEEN MYSQL AND MSSQL	
MySQL	MSSQL
is open source	is closed source
can be used for free	can be used only after procuring a license, this is included in Microsoft Azure
offers different variations like derived engine based on Sybase, Berkeley DB, Heap, InnoDB and others	is limited for usage within the derived engine based on Sybase
requires little disk space	requires more disk space
does not support all foreign key features and other relational features	supports extensive foreign key features and other relational features
has a limited recovery system	has a very extensive recovery system
can be installed on Windows, Linux and Unix	can only be installed on Windows
does not contain management tools based on GUI	provides management tools based on GUI

4.3 Firewalls

One of the most obnoxious problems I ran into during the development of the database and the testing of the application were firewalls. They turned out to be a problem both inbound and outbound [31]. When the plan of implementing a database was first proposed, I originally planned on hosting it on my own server or on Stichting RIONED's server. However, like most databases both these servers only allowed connections from the localhost. This can be circumvented

by adding IP addresses to the whitelist of the firewall. However, adding each of the users IP address manually was not a plausible solution. Microsoft Azure offered a better service; their firewalls allow an IP address range to be placed on the whitelist. This way I was able to open the database to all incoming connections, rather than just the localhost [32].

The second problem arose when we started testing the application; several test users reported problems connecting to the database. It was not long until I figured out that the firewalls on their network were the problem. The application connects to two servers; the Microsoft Azure server for the database and the SMTP server of my own server to send verification emails. Both these connections were blocked by the firewalls on the client side. The test audience are people in governments and educational institutes. These organisations have a tight security on their network thus several ports had to be opened before the test users could connect to the RainTools servers.

5 PROGRAMMING

When I started this project I had some experience with C#, I had created a few simple WinForms applications prior to RainTools. However, when I read about the possibilities of XAML & WPF I wanted to learn more about the two. The caveat here being that when using C#, the application becomes limited to the Windows platform. However, this was not a problem as the majority of the target users were Windows users.

5.1 C#

I had mastered the basics of C# before starting this project, C# however is very big and I dare say you can never fully master it. Almost every time I code in C# I learn something new or learn a better way to code. The main focus of this project was to learn how to use XAML & WPF but I also wanted to learn how to properly code using the MVVM architecture.

C# in itself is very similar to other coding languages. As with all different languages there are a few syntax differences, but generally speaking if you know one language you will find it easy to transition to another language.

ObservableCollections

One of the main things I learned when it comes to handling data in C# is the benefit of using ObservableCollections [34]. RainTools is an application that handles a lot of data, almost all the data is stored in ObservableCollections. An ObservableCollection is very similar to a List in C#, the important difference being that it implements INotifyCollectionChanged [35]. This means that an event is triggered whenever data in the collection changes, which allows the application to respond when new data is input or data is changed.

ObservableCollections taught me how to manage data efficiently. Ordering data, as well as adding and removing data is made easy by the collection's methods. These collections have a wide variety of possibilities. For example; one of the graphs features a button to sort data ascending and descending, this is made possible by ObservableCollection's sorting method.

LINQ

A second C# feature I had never used before is LINQ. LINQ is used to extract and process data from arrays and collections, LINQ is very similar to SQL. The analogy is simple; if an ObservableCollection is a database table, then a LINQ query is the SQL query used to select data from that table [36][37]. Its syntax is also reminiscent of SQL, as shown in EQUATION 3 and EQUATION 4.

EQUATION 3. Sample MSSQL query to select a single account.

```
SELECT TOP 1 * FROM AccountsTable WHERE Email = EmailAddress
```

EQUATION 4. Sample LINQ query to select a single account.

```
AccountsTable.Where(dbo => dbo.Email.Equals(EmailAddress)).FirstOrDefault()
```

It took me a while to fully understand the possibilities of LINQ, but looking back I am not sure if and how I could have succeeded in handling the data selection and manipulation without LINQ. For example, using LINQ I was able to select specific data matching certain criteria to display different result types [38].

DataBinding

Another C# feature that was used extensively is DataBinding. It does exactly what the name suggests; it binds data to an object. The biggest advantage of using DataBindings is that when data is changed in one place, the change is automatically reflected in all instances of that piece of data in both the View and ViewModel. This is done by setting the binding properties as can be seen in EQUATION 5 [39][40][41].

EQUATION 5. Sample DataBinding of a TextBox.

```
<TextBox.Text>
  <Binding Path="Input.Oppervlakken[10].Parameters[0].Double" ValidatesOn-
DataErrors="True" UpdateSourceTrigger="PropertyChanged" Fallback-
Value="0" Mode="TwoWay" >
  </Binding>
</TextBox.Text>
```

In the example above the Text in the TextBox is bound to the object specified in

the Path. It gives a validation error when the data is not an integer [42][43]. When the text is changed in either the View or the ViewModel, it is updated (because it is a TwoWay binding) and when there is no data known the value returned is 0.

5.2 XAML & WPF

WPF is Microsoft's approach to a graphical user interface framework. It allows for creating a user interface existing of the various user interface elements provided within the .NET framework. WPF is the successor of WinForms.

WPF applications are created using Microsoft's own markup language, similar to how a website is built using HTML. XAML is Microsoft's markup language. It is derived from XML and looks very similar to it [44].

EQUATION 6. Sample XAML window.

```
<Window x:Class="SampleApplication.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="HelloWorld!" Height="300" Width="400">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBox Grid.Row="0" Grid.Column="0" Margin="10"/>
    <Button Grid.Row="0" Grid.Column="1" Margin="10" Content="BUTTON" />
    <GroupBox Grid.Row="1" Grid.Column="0" Margin="10" Header="GROUPBOX" />
    <TabControl Grid.Row="1" Grid.Column="1" Margin="10">
      <TabItem Header="TAB 1"/>
      <TabItem Header="TAB 2"/>
    </TabControl>
  </Grid>
</Window>
```

EQUATION 6 is code for a sample XAML window. Its size is 300px by 400px, it contains a grid with two equally sized columns and rows. The grid contains a few sample WPF elements; a TextBox, a Button, a GroupBox and a TabControl. The result can be seen in FIGURE 9. Creating windows XAML can be a lot of

work compared to using a designer, however in my opinion using a markup language is a lot more precise [45].

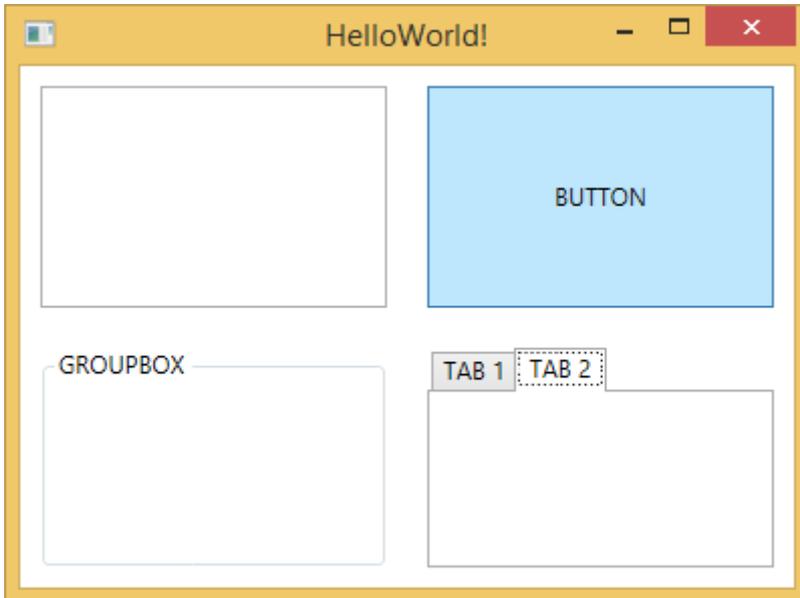


FIGURE 9. Sample WPF window created using XAML.

5.2.1 WinForms

WPF's ancestor WinForms is what I used to create my previous C# applications. WinForms is useful for beginners. It comes with a bunch of pre-set controls and a designer that allows dragging and dropping controls into place. The downside to WinForms is obvious immediately, pre-set controls. For example, if something out of the box is required that is not a pre-set control, like a button with an image and text on it, it is a very difficult task to create it. Whereas using WPF, an Image control and a TextBlock control can be put inside the Button control and the custom control is created.

On the bright side, there are a lot of 3rd party controls made for WinForms to make these custom controls possible. This is because WinForms has been around for so long that all of these controls have been created by the community, however not all of them are free to use.

Because I wanted to have flexibility while creating RainTools and not be limited to free 3rd party controls, I opted to use WPF over WinForms, on top of the flexibility learning more about a new framework is a bonus.

5.3 OOP & MVVM

During my education I learned about object-oriented programming, especially during Java courses. I had applied it a few times during those courses but always shied away from using it when coding in .NET. Prior to this project my mind set regarding object-oriented programming was “why bother”. I was always of the opinion I could achieve the same without object-oriented programming. This held true for the small scale applications that I was creating.

About 2 years ago I created a rather large application. Towards the end I was completely drowning in code and when something had to be edited, I had to edit several different instances. This is when I saw the light and decided that I would apply object-oriented programming to my next project [46].

The architecture I decided to use for RainTools is MVVM. MVVM consists of three components; the Model, the View and the ViewModel. The Model defines all the objects, the blueprint of all the objects. The ViewModel is the instance of all these objects created and given values that are bound to the View. The View is created using XAML and it is the window that displays the data present in the ViewModel [47][48].

Looking back, I do not think that I will ever create another application without MVVM. In the previously mentioned application whenever I wanted to wipe all the data, I had to empty dozens of TextBoxes manually, with MVVM this can be done by simply wiping the ViewModel or creating a new instance of the Model and replacing the current ViewModel. It means hundreds of lines of code compressed into a single line.

5.4 Styles

When designing Views with XAML many visual effects can be applied to any given object, allowing for almost endless graphical possibilities. Often the same effects are added to several objects, in this case it was efficient to create a custom style and apply that style to the objects [49].

A few style examples used in RainTools can be seen in EQUATION 7.

EQUATION 7. Sample styles from RainTools.

```
<Style TargetType="DataGridCell" x:Key="GetalCell" BasedOn="{StaticResource MetroDataGridCell}">
    Setter Property="HorizontalAlignment" Value="Right"/>
</Style>

<Style TargetType="{x:Type Canvas}">
    <Setter Property="UseLayoutRounding" Value="False" />
</Style>

<Style TargetType="{x:Type DataGrid}" BasedOn="{StaticResource MetroDataGrid}">
    <Setter Property="SelectionMode" Value="Single" />
</Style>
```

The first style in EQUATION 7 has the key “GetalCell”. Using this key the style can be applied to the TargetType which is any cell in a DataGrid. The style aligns the content of the cell to the right and it is “BasedOn” another style, which means it inherits all the properties of the “MetroDataGridCell”-style and alters the HorizontalContentAlignment property of that style. This is the equivalent of using inheritance in a C# class.

The second style automatically applies to all objects with the type Canvas because it has no key. The style sets the UseLayoutRounding of the Canvas it is applied to False.

The third style applies to all DataGrids in the application and sets the SelectionMode property of the DataGrids to Single, meaning that a user can only select one row at a time in a DataGrid. The style also inherits another style’s properties.

5.5 File management

An important feature implemented in RainTools is the use of projects and simulations, the so called file management. A user can save his input data in what is called a simulation. These simulations are located in a project. A project can contain many different simulations. Usually, they are quite similar with small differences between them to compare different circumstances.

This was implemented by using Windows folders. Upon first start up the user must select a project folder. This folder will contain a folder for each project. The benefit of this is that the project folder location is flexible and you can copy your project folder over to a different computer if required.

To make it easy for users we wanted to be able to group simulations by the type of engine tool (tile) used (see FIGURE 29 in Attachment 2 for the tiles). The reason for this is that one project can have simulations using different tools. Therefore in the level under the folder of a project we find a folder for each different tool. Within these tool folders are the folders for the simulations. See FIGURE 10, FIGURE 11, FIGURE 12 and FIGURE 13 for a graphical explanation.

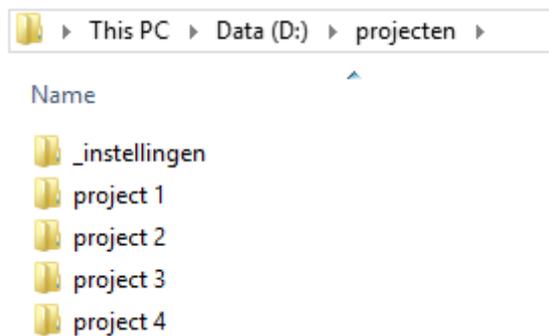


FIGURE 10. Sample project folder, contains 4 projects and a global settings folder.

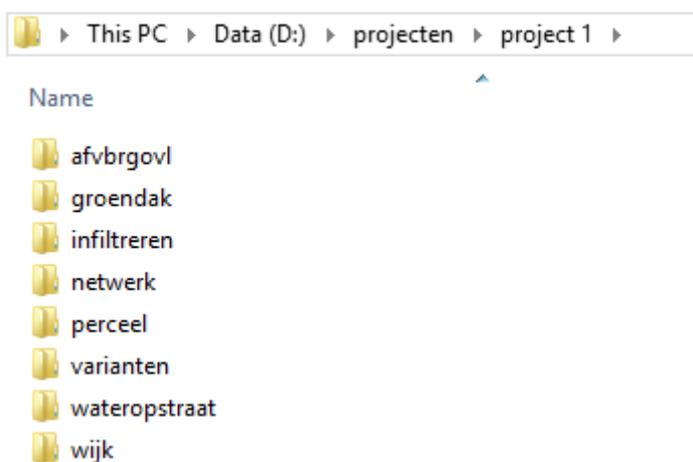


FIGURE 11. Sample folder for a project (each folder represents a different tool).



FIGURE 12. Sample folder containing the simulations of a project using the tool "perceel".

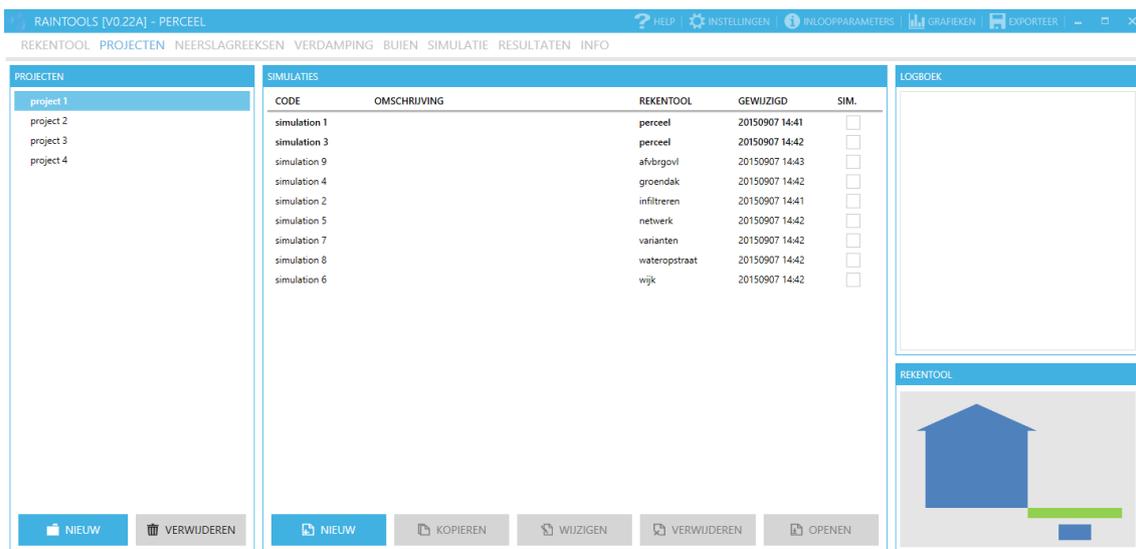


FIGURE 13. Resulting architecture in RainTools, note that "perceel" is the selected tool and therefore "simulation 1" and "simulation 3" are highlighted.

Inside the simulation folder are the simulation's files. These include the result XML files, an XML file that contains information about the simulation (description, date created, edited and whether there are results available), a TXT file with the contents of the logfile and finally an XML file with the input data.

The last file is the most important. This file contains all the data that the user has input for the simulation. This allows users to save and load simulations by opening the files in the simulation folder. A nifty feature is the copy simulation feature. This copies an entire simulation (data and results). This allows users to easily make two simulations that have small differences without having to fill in all the required data a second time.

5.6 Threading

During the development of RainTools I noticed that the more features I added the slower the application became, especially when drawing graphs. This prompted me to look into threading possibilities, and I found that it was relatively easy to implement multi-thread support [50].

RainTools currently operates on three different threads. The first being the interface thread, this always runs in the foreground and is what the user sees right away. The second is the engine thread. When the user starts a simulation the RainTools Engine is started on a background thread. This ensures that while the Engine is running the interface thread is still running while previously the interface would seem “frozen”. The third thread is the graph thread which is also running in the background. This ensures that the graphs can be loaded faster.

EQUATION 8. Starting a background thread in RainTools.

```
BackgroundWorker RekenThread = new BackgroundWorker();
RekenThread.WorkerReportsProgress = false;
RekenThread.DoWork += Rekenen;
RekenThread.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(RekenThreadCompleet);
RekenThread.RunWorkerAsync();
```

EQUATION 8 shows how the engine thread is started. A BackgroundWorker is an object of the .NET framework used to run a method on a background thread. The progress of the method can be reported to display a progress bar, however in this case it is disabled. The method that is performed is “Rekenen” and the event that is triggered when the background work is completed is “RekenThreadCompleet”. Finally the thread is started asynchronously [51][52].

5.7 Exporting data

In order for the interface to communicate with the RainTools Engine an input file is required. The engine takes .xml files as input thus an XML file had to be created with the proper input data. The engine saves the produced results in .xml files that had to be read by the interface. This means I had to both read and write XML files through C#.

A second requirement for the interface was to take the results and export them to Microsoft Excel, a commonly used application to analyse data in the professional world the customer operates in.

5.7.1 XML

Reading and writing XML files is a straightforward task in C# as there is a built in library for this in the .NET framework, it is just a matter of learning to use it. EQUATION 9 is a simple example of how to create an XML file [53][54].

EQUATION 9. Creating a sample XML file.

```
using (XmlWriter XmlFile = XmlWriter.Create(XmlPath))
{
    XmlFile.WriteStartDocument();
    XmlFile.WriteStartElement("Employees");

    XmlFile.WriteStartElement("Employee");
    XmlFile.WriteElementString("ID", 0.ToString());
    XmlFile.WriteElementString("FirstName", "Bob");
    XmlFile.WriteElementString("LastName", "Johnson");
    XmlFile.WriteElementString("Salary", 10000.ToString());
    XmlFile.WriteEndElement();

    XmlFile.WriteStartElement("Employee");
    XmlFile.WriteElementString("ID", 1.ToString());
    XmlFile.WriteElementString("FirstName", "John");
    XmlFile.WriteElementString("LastName", "Smith");
    XmlFile.WriteElementString("Salary", 12000.ToString());
    XmlFile.WriteEndElement();

    XmlFile.WriteEndElement();
    XmlFile.WriteEndDocument();
}
```

The .xml produced by the sample code in EQUATION 9 can be seen in FIGURE 14.

```

<?xml version="1.0" encoding="UTF-8"?>
- <Employees>
  - <Employee>
    <ID>0</ID>
    <FirstName>Bob</FirstName>
    <LastName>Johnson</LastName>
    <Salary>10000</Salary>
  </Employee>
  - <Employee>
    <ID>1</ID>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    <Salary>12000</Salary>
  </Employee>
</Employees>

```

FIGURE 14. XML file created by EQUATION 9.

The problem, which I encountered when reading XML files, is that everything is interpreted as a string. RainTools is an application that mainly deals with numbers thus a lot of parsing from string to double had to be done, often leading to problems with the decimal separator. This was solved by setting the CultureInfo to "CultureInfo.InvariantCulture" during parsing.

5.7.2 Excel

To be able to export data to Microsoft Excel there is one requirement; the user needs to have Microsoft Office installed. Without Microsoft Office installed, C# cannot create .xls files, because the required library is built into Microsoft Office. That said, a user that wants to export data to Excel surely has Excel installed.

Because both Excel and .NET are a Microsoft product cooperation between the two is smooth. Using the .NET library for Microsoft Excel you can start an instance of Excel as a background process and from there it is almost exactly like using VBA from within Excel. EQUATION 10 shows how a simple XLS file can be created using C#. It is important to properly close the instance of Excel otherwise it will keep running in the background and a significant amount of RAM will be used up [55].

EQUATION 10. Creating a sample XLS file.

```
using Excel = Microsoft.Office.Interop.Excel;

Excel.Application ExcelApplicatie = new Excel.Application()
{
    Visible = false,
    DisplayAlerts = false
};

Excel.Workbook Werkboek = ExcelApplicatie.Workbooks.Add();
Excel.Worksheet Sheet = Werkboek.Sheets.get_Item(1);

Sheet.Cells[1, 1] = "A1";
Sheet.Cells[2, 2] = "B2";
Sheet.Cells[3, 3] = "C3";
Sheet.Cells[4, 4] = "D4";

Werkboek.SaveAs(FilePath);

Marshal.ReleaseComObject(Sheet);
Sheet = null;

Werkboek.Close(null, null, null);
Marshal.ReleaseComObject(Werkboek);
Werkboek = null;
```

	A	B	C	D
1	A1			
2		B2		
3			C3	
4				D4

FIGURE 15. XLS file created by EQUATION 10.

5.8 Error handling

The final and perhaps most important lesson learned during this project is handling errors correctly. Error handling in C# is very simple with a try-catch block. However, if done incorrectly, it can do more harm than good as I have experienced first-hand.

When a C# application encounters an exception the application shuts down with very little information about what went wrong (see FIGURE 16). This makes locating bugs as hard as finding needles in a haystack [56][57].

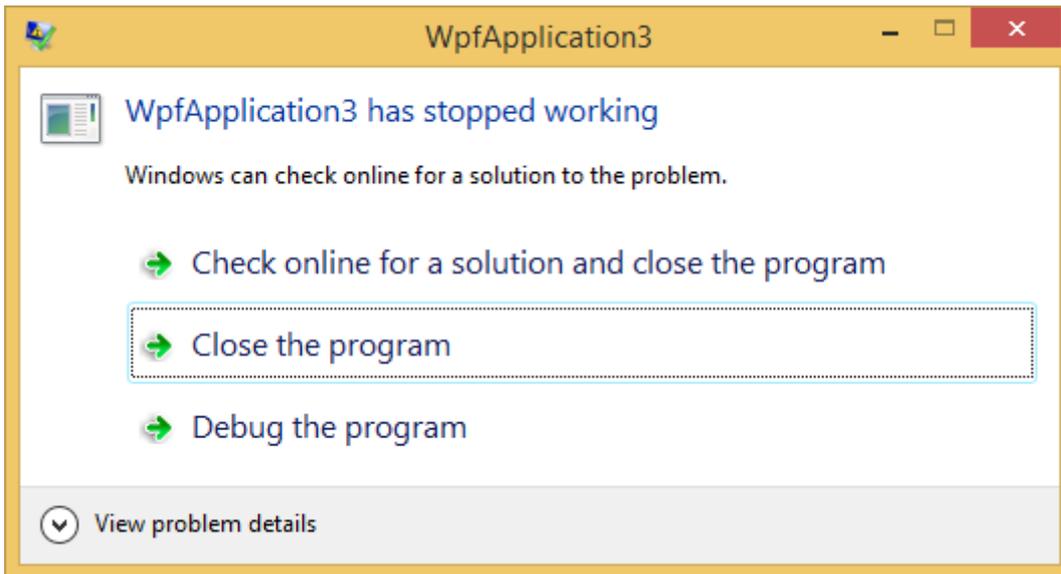


FIGURE 16. Unhandled exception in WPF.

To prevent the error in FIGURE 16 code must be enclosed in a try-catch block. It is good practice to catch several different exceptions rather than grouping them. This makes locating the error in the code easier.

EQUATION 11 shows a proper error handling example from RainTools. When an exception occurs in the code within the try block, it is split into three different categories; when a file is not found, an error message is shown, when a key is not found in the XML file, a different error message is shown and when any different type of exception occurs, a third different error message is shown [58][59].

EQUATION 11. Error handling example from RainTools.

```
try
{
    //code
}
catch (FileNotFoundException)
{
    ShowMessage("RESULTATEN INLEZEN", "Geen resultaten bestand.",
        MessageDialogStyle.Affirmative).ConfigureAwait(false);
}
catch (KeyNotFoundException)
{
    ShowMessage("RESULTATEN INLEZEN", "Oud resultaten bestand, geen neerslagreeks
        gebeurtenissen beschikbaar.",
        MessageDialogStyle.Affirmative).ConfigureAwait(false);
}
catch (Exception)
{
    ShowMessage("RESULTATEN INLEZEN", "Resultaten bestand is ongeldig.",
        MessageDialogStyle.Affirmative).ConfigureAwait(false);
}
```

During the test phase of RainTools we learned what happens when errors are not handled correctly. A test user encountered an exception that was handled, but no error message was attached to the exception. As a result the application kept running and we had no idea what the cause of the bug could be. It took us several days to locate it, which could have been prevented by properly handling the exception in the first place.

EQUATION 12. Incorrect error handling, DO NOT USE THIS!

```
try
{
    //code
}
catch (Exception)
{
    //do nothing
}
```

5.9 UserControls

The RainTools Interface has several repetitive Views, for example the results page. An exact copy of this page is displayed in the interface over 20 times, each time displaying different results for different rain showers or precipitation events. Rather than loading new data on the same page constantly when switching results, I opted to use UserControls instead. As a result, the different pages load significantly faster because data only has to be loaded once.

A UserControl is a WPF View that is defined once and can then be included in any other WPF View, similar to how PHP has the include function. In the above example the result page is defined once and included once for each different result dataset, see FIGURE 17 for the example View [60].

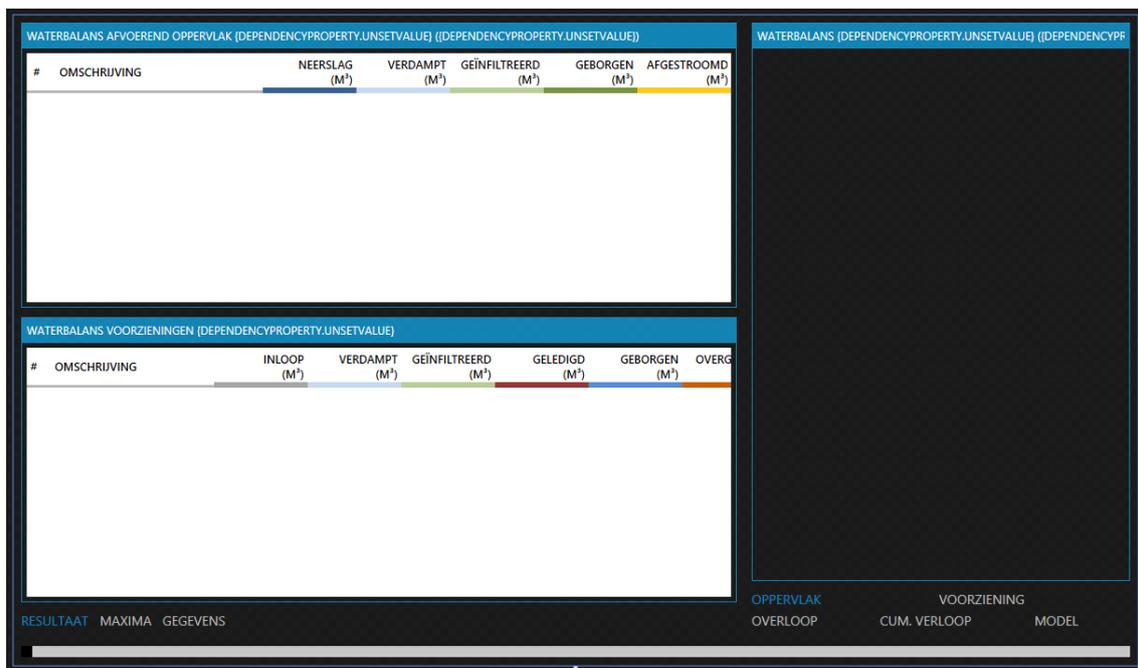


FIGURE 17. Result page UserControl in the Visual Studio Designer.

This View is included in the main RainTools window several times, once for every selected rain shower and precipitation event in the simulation. The result can be seen in FIGURE 24; the result page for extreme rain shower number 5 ("E05") is shown. In the TabControl above the table the user can navigate to other result pages.

6 VISUALIZATION

I have developed several small-scale WinForms applications prior to RainTools. What bothered me about WinForms applications was the “old fashioned” Windows look, similar to how Windows XP applications looked.

Personally, I am an avid Apple user (iPad/iPhone/MacBook) and a big fan of the “minimalistic user interface”-designs. Windows 8 has a similar look when it comes to its Store Apps, however the WinForms applications still have the old fashioned look.

The beauty of WPF applications is that you can change the visualization to whatever you want it to be. This prompted me to look for a style package that resembled the Windows Store Apps and featured the minimalistic design (called metro) that I was looking for. This search lead me to MahApps Metro.

6.1 MahApps Metro

MahApps Metro is an open source style package that transforms your application to a metro-style application. FIGURE 18 shows a very simple metro style application built with MahApps Metro.

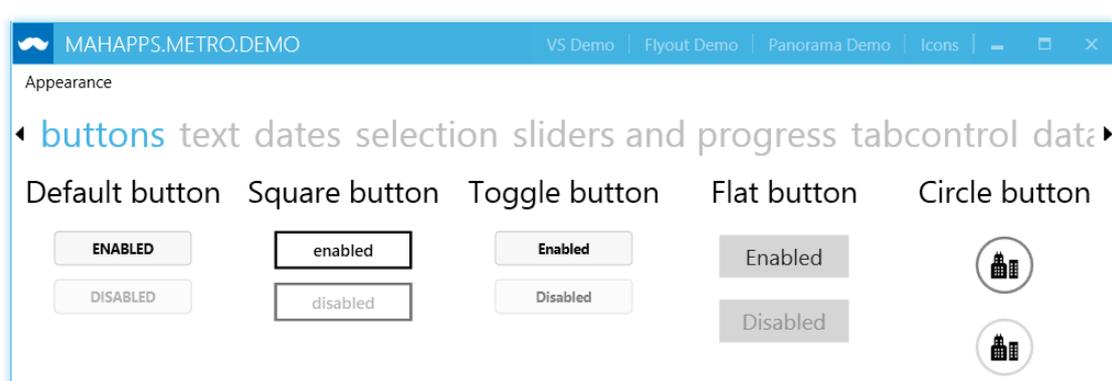


FIGURE 18. MahApps Metro demonstration [61].

MahApps Metro applies the metro style to several WPF controls such as; Button, DataGrid, Dialog, Flyout, TabControl, TextBox, Tile and ToggleSwitch. All of these have been implemented in RainTools.

6.2 Graphs

To display the results produced by the RainTools Engine in a meaningful way the interface needed graphs in addition to the standard tables. The problem I ran into was that XAML does not have graph support built in. There are toolkits readily available, however the downside is that because they are “add-ons”, they do not receive the minimalistic style from MahApps Metro and most are not free to use [62][63][64][65].

To incorporate graphs that fit with the style of RainTools, I decided to create my own graph library. Several types of graphs had to be created; area graphs (FIGURE 19), stacked bar graphs (FIGURE 21 & FIGURE 22), bar graphs and line graphs.

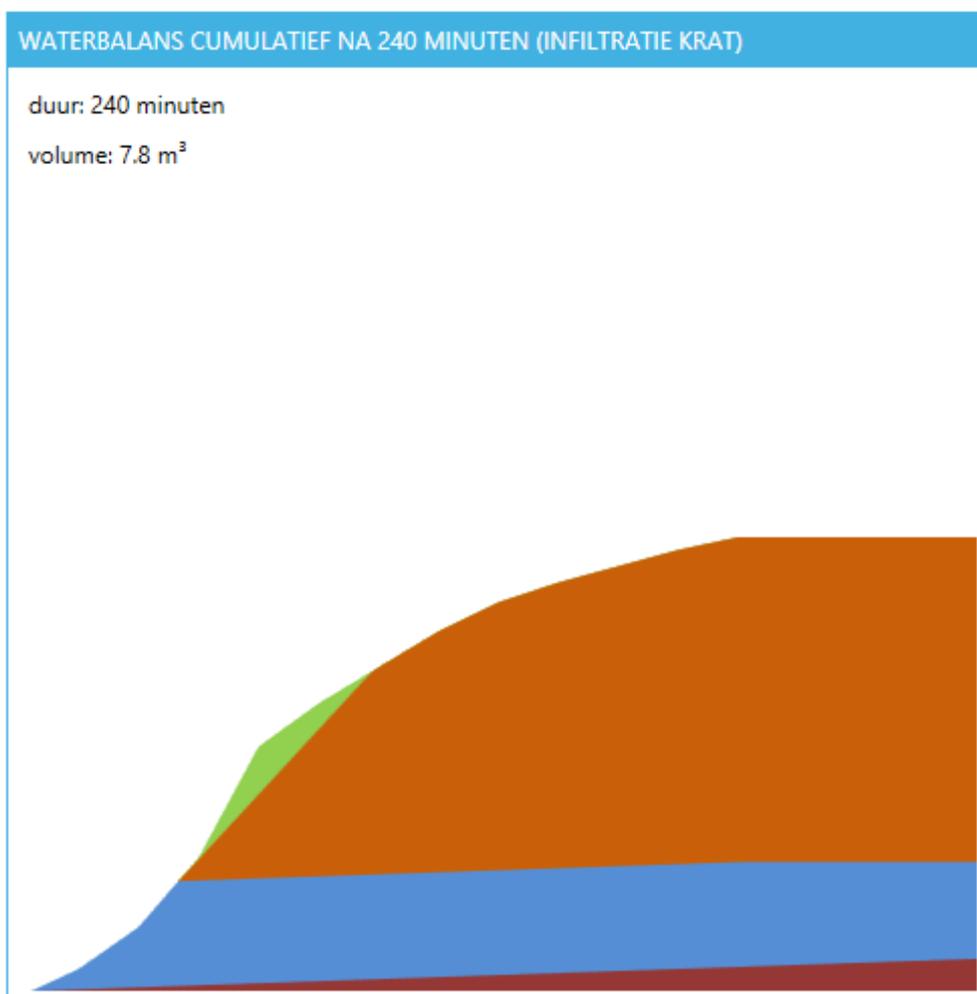


FIGURE 19. Area graph showing distribution of water during 240 minutes.

We chose not to include axes in the graphs in order to reduce the amount of clutter, keep a minimalistic look and reduce confusion because all the axes are scalable. To still give users the exact values an axis normally would, we incorporated tooltips that pop up with all the information when hovering over an area of the graph. The colour legend normally found in a graph is incorporated into the data table as can be seen in FIGURE 24. If more scientific graphs are required, this can be achieved by exporting the data to Excel and creating custom graphs to display whatever information is required.

The area graph was the hardest graph to create. In a perfect world an area graph would be like a stacked bar graph, replacing the rectangles with polygons. The polygons are created by creating an XAML Polygon object, the shape is determined using a C# PointCollection. The problem I ran into is that different polygons would not align perfectly like with stacked bar graphs, the reason for this is pixel snapping.

Pixel snapping is something that I ran into a lot while creating the graphs. Very often the size of one of the areas in the graph will not be an integer. A computer screen can only show complete pixels not halves. Because of this, the size is rounded down to an integer and pixel snapping occurs. When this happens, there are small gaps in between the areas of the graph.

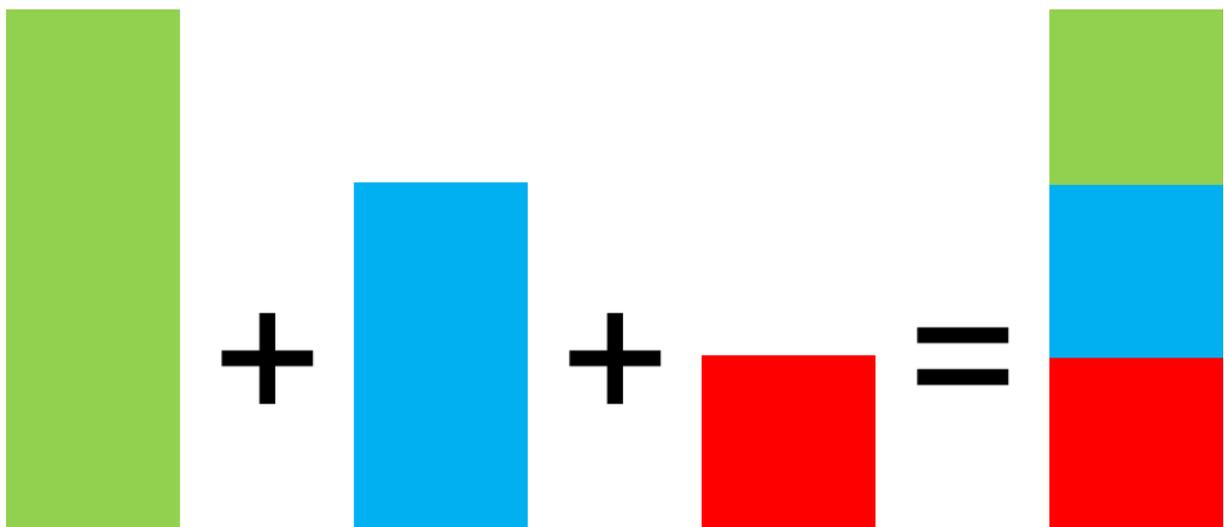


FIGURE 20. Layering rectangles in an area graph.

FIGURE 20 shows how I solved this problem by layering objects rather than stacking them. Using layering I managed to fill up the very thin gaps that often appeared between different areas of the area graphs.

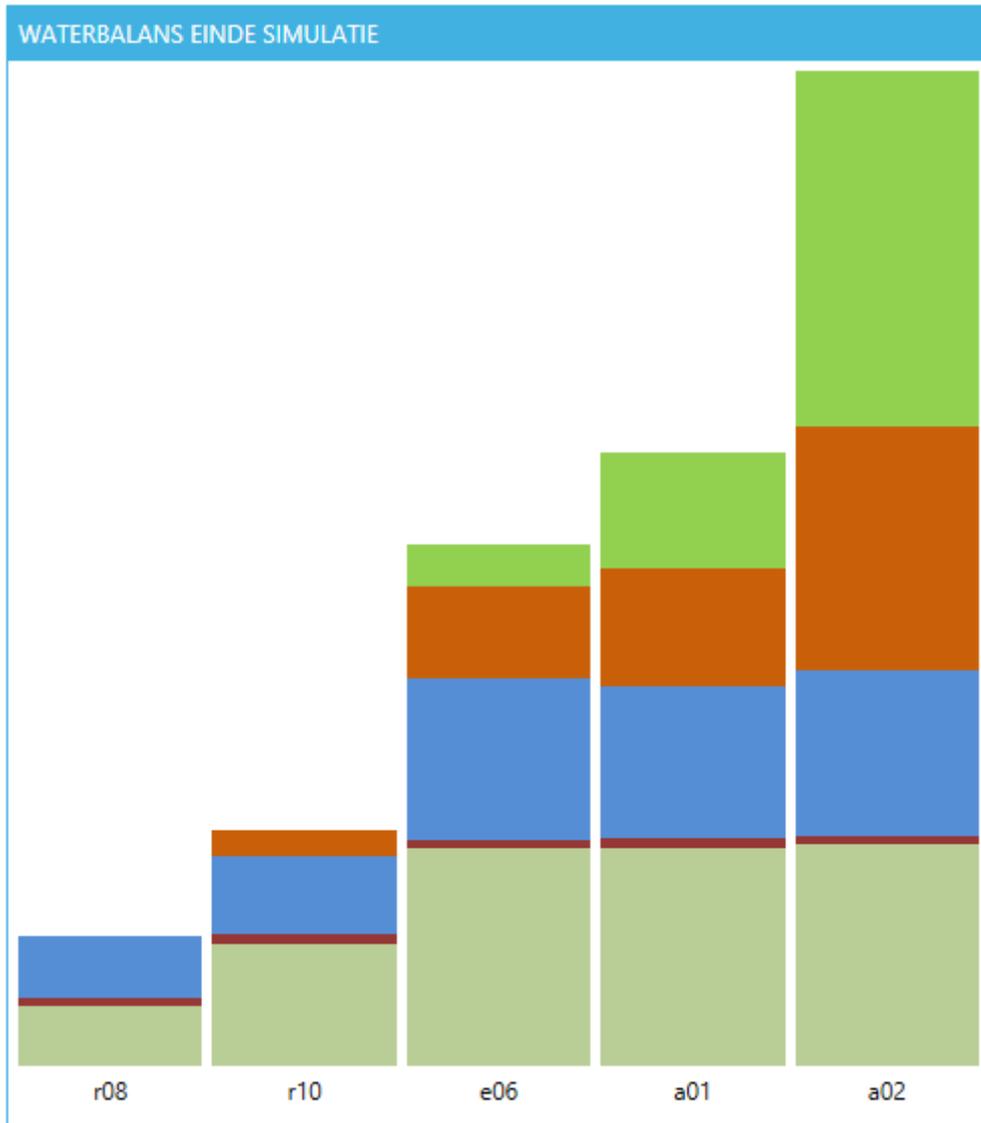


FIGURE 21. Stacked bar graph showing distribution of water for 5 different events in absolute numbers.

Bar graphs were easier to create. They consist of several rectangles drawn onto a canvas. In XAML this is an easy operation as both Canvas and Rectangle are common objects. Stacked bar graphs have several of these rectangles stacked on top of each other using layering.

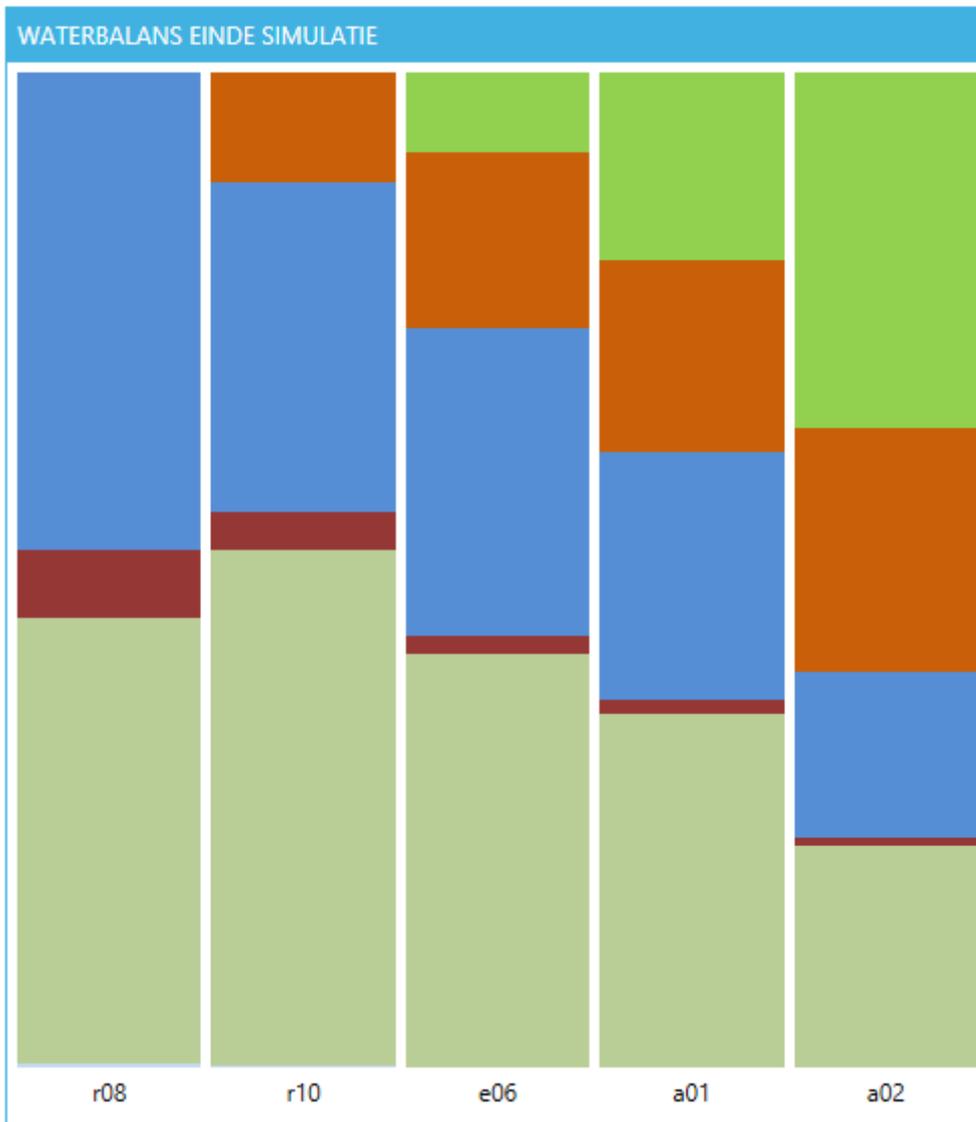


FIGURE 22. Same stacked bar graph as FIGURE 21 but shown percentually.

The stacked bar graph seen in FIGURE 22 has the same data as the graph in FIGURE 21, however the unit of the y-axis is in percentages rather than absolute numbers. To create this graph, the height of the rectangles had to be altered by calculating the percentage and then making the height relative to the canvas height (See EQUATION 13).

EQUATION 13. Setting the height of a rectangle relative to canvas height.

$$Height = \frac{Value}{TotalValue} * CanvasHeight$$

6.3 Animations

Most of the results provided by the RainTools Engine are relative to time. Naturally, the only way to display these results is in a graph where the x-axis is the time like in FIGURE 19.

The problem we encountered with the results is that the graphs and tables only showed the results of the last time iteration. This can be seen in FIGURE 23; the left graph is independent of time, the right is dependent of time. The left graph contains the results of the last time iteration, the marked areas display the same data.

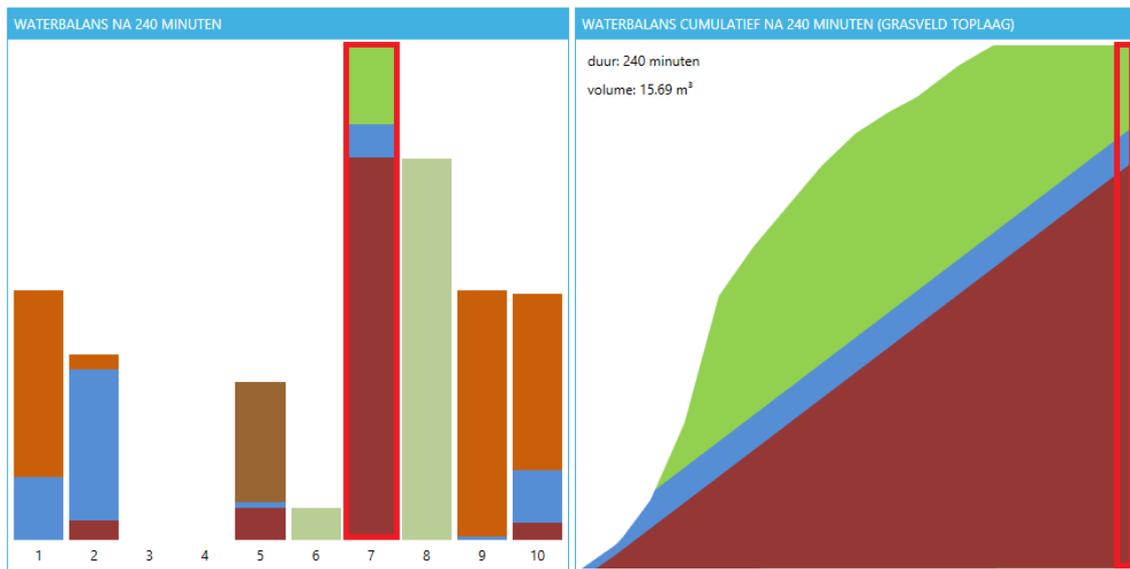


FIGURE 23. Two graphs, left independent of time, right dependent of time.

To solve this problem and to add something unique to RainTools at the same time I decided to make all the results dynamic, dependent of time. I did this by adding a slider to the results page which can be used to navigate through time and show the results at that particular time. Finally a button was added to start the animation, showing the results progress through time.

This was done by using a C# Timer object. This object can be assigned a function each time it ticks and the interval in between each tick can be specified de-

termining the speed of the animation. With each tick of the timer the result tables and graphs are updated. In FIGURE 24 and FIGURE 25 an animated graph can be seen, further examples can be seen in Appendix 3 [66][67].

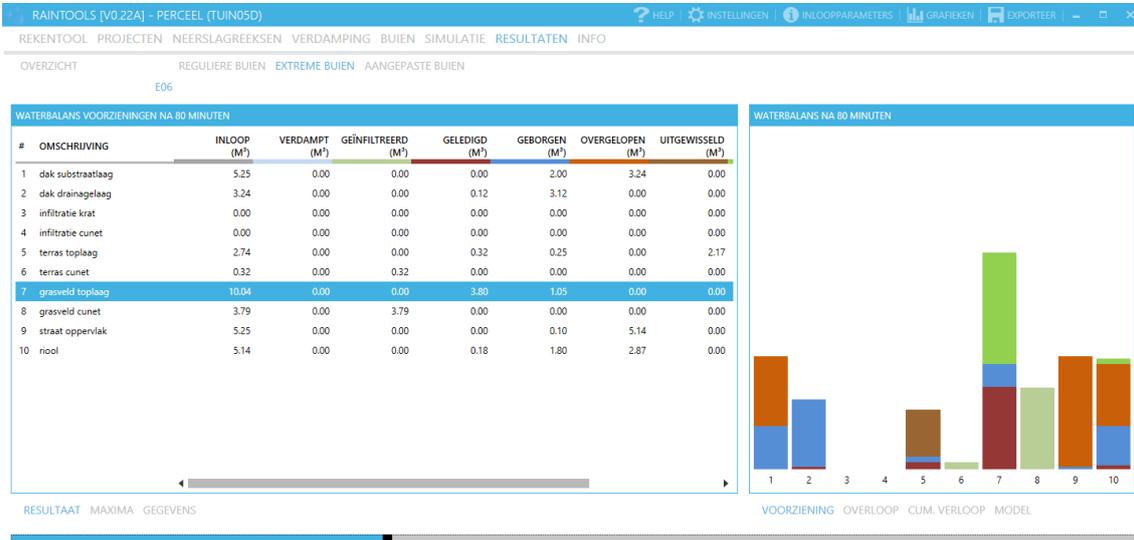


FIGURE 24. Animated results, results after 80 minutes.

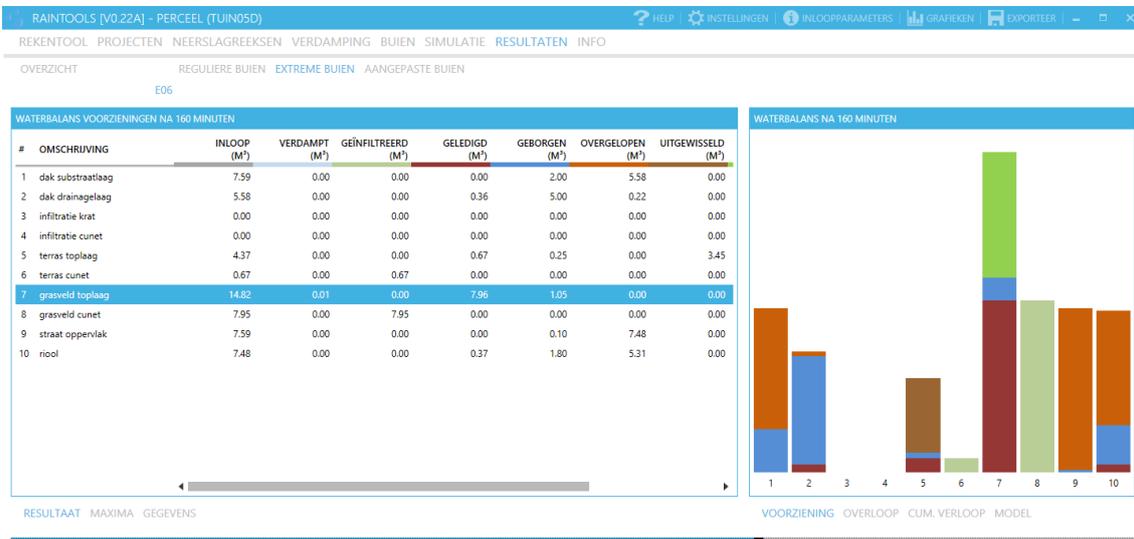


FIGURE 25. Animated results, results after 160 minutes.

7 LEGAL ISSUES

When releasing a large application like RainTools to the public certain legal aspects come into play. It is important that all bases are covered to remove any possible liabilities in the future.

The first step is to make sure that all third party tools used are open source or licensed. In RainTools' case we have only used MahApps Metro and this is an open source package.

The next step is to create an end user license agreement (EULA) and a privacy policy. Upon starting the application for the first time these documents are presented to the user and the user has to accept them, without accepting the user cannot continue using the application.

The final step is to create a disclaimer which is shown in the information tab of the application.

7.1 EULA

The EULA is an agreement that states the “rules” of the application in question. If a user wishes to use the application, he or she must agree and abide with the rules. Failure to do so gives the customer the right to terminate a user's access to the application even if the user has purchased a license. An EULA is important to prevent any misuse of the application [68][69].

7.2 Privacy policy

The privacy policy is a document that states to what extent private information is stored when using the application, why it is stored, what the information is used for and what methods are used to secure the information. The goal of this document is to create transparency. A user will want to know what privacy he or she is forfeiting and whether he or she feels the security is strong enough to give up his information. Similar to the EULA the user has to agree with this policy before being able to use the application [70].

7.3 Disclaimer

The disclaimer is important to remove any liability for mistakes in the application. It states that the application has been extensively tested for bugs, however the customer can never fully guarantee that it is completely free of bugs. Therefore should a bug occur; the customer has no responsibility or liability at all. Simply put, use the application at your own risk [71].

The screenshot shows the RainTools application interface. At the top, there is a navigation bar with the title 'RAINTOOLS [v0.22A]' and several menu items: 'REKENTOOL', 'PROJECTEN', 'NEERSLAGREEKSEN', 'VERDAMPING', 'BUIEN', 'SIMULATIE', 'RESULTATEN', and 'INFO'. Below the navigation bar, there are two main sections: 'Eindgebruikerslicentie RainTools' and 'Privacyverklaring RainTools Stichting RIONED'. The 'Eindgebruikerslicentie' section contains text about the license, its scope, and intellectual property. The 'Privacyverklaring' section contains text about the privacy policy, the use of personal data, and the registration of software. At the bottom of the interface, there is a message: 'Voordat u RainTools kan gebruiken moet u de eindgebruikerslicentie en privacyverklaring accepteren.' Below this message are two buttons: 'ACCEPTEREN' (with a checkmark icon) and 'WEIGEREN' (with an X icon).

FIGURE 26. RainTools EULA and Privacy Policy as shown upon first startup.

8 TESTING

In the final stages of development the application had to be tested extensively for bugs. As a sole developer, it is impossible to find every single bug. When looking for bugs they will not be found, they always find a developer or tester when they are not looking. To speed up the process, we invited several test users to try out RainTools.

8.1 Test users

While in development RainTools was demonstrated to the public several times, each time getting a positive reaction from the audience. Some even expressed interest in participating in the development. It was decided not to take them up on that offer, however some were invited to participate in the test phase of RainTools.

The customer selected several groups of people from different types of institutions in their professional world to participate in the test phase. Currently RainTools is still in the test phase and we have managed to iron out several significant bugs found by our test users. Thus we can conclude that the test phase so far has been a success.

8.2 Distribution

The first thing to look at when starting a test phase is how to distribute the application to the test users, in particular the updates that follow the initial release. Previously I had always distributed my applications by email in a compressed format, however for such a large scale application this was not an option.

While looking for distribution methods I found ClickOnce, a deployment technology by Microsoft built into Visual Studio. Using ClickOnce an application can be published on a website. ClickOnce has two modes; offline and online. We opted for the offline mode, meaning that the application is installed on the user's computer rather than being downloaded on every start up. A setup executable is created by ClickOnce to install the application on the user's computer. Upon

starting the application ClickOnce automatically checks for updates (see FIGURE 27) which is exactly what we were looking for [72][73].

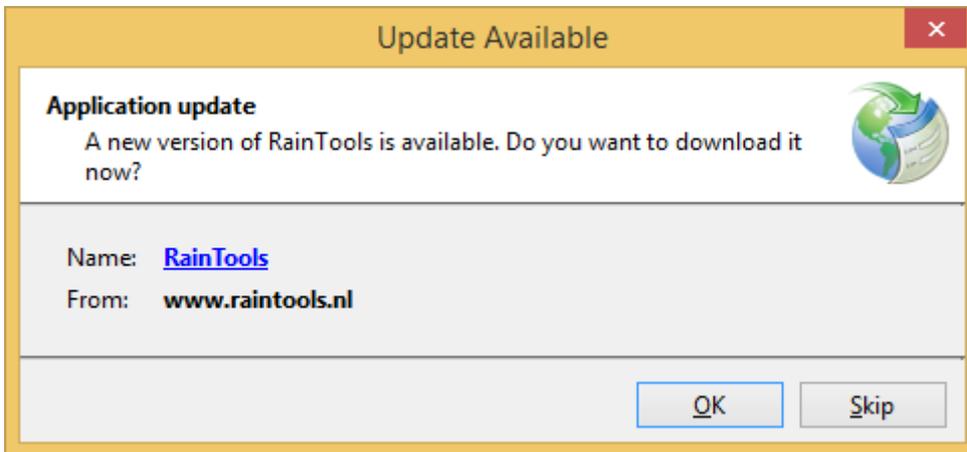


FIGURE 27. RainTools detecting a new version.

8.3 Monitoring

To be able to monitor the test user's data from each session is stored in the RainTools database. This data contains the following information: the number of simulations performed (broken down by type), the number of simulations loaded, the date and duration of the session and most importantly the version of RainTools used. This helps to keep track of the users; what they are doing and whether they are using the latest version of the application.

9 ENTREPRENEURSHIP

As a direct result of this project, I became an entrepreneur. I am now the CEO of ITS-Software, a sole proprietorship. The process of setting up this business has also been a fun learning experience.

First I had to decide on what business entity to set up. As most people know, the sole proprietorship is best fit for a single entrepreneur but it brings several liability risks with it. After weighing the pros and cons, I decided that the risks I take with the development of software is rather insignificant, therefore there was no need for a limited liability company.

The next steps were registering the company with the chamber of commerce, open a business bank account and create a website for the business. We will see what the future holds for my business, but it just shows what an experience this project has been for me personally.

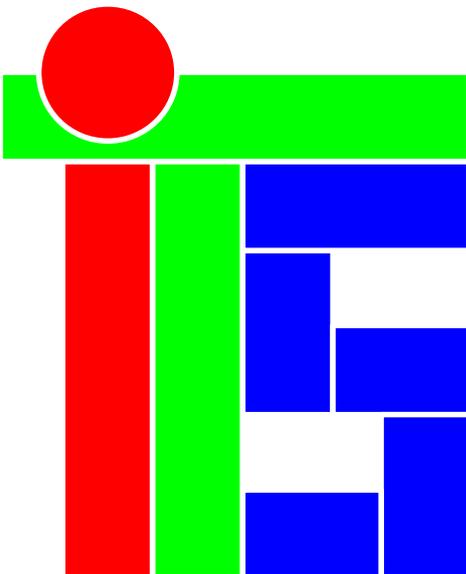


FIGURE 28. ITS-Software logo in the colors well known to developers, RGB.

10 CONCLUSION

RainTools was created, a functional yet stylish interface that does exactly what it is supposed to do with a minimal input from the user. We are confident that even an inexperienced user can use RainTools for its purpose. Even though the development of further tools is still ongoing, the thesis is complete.

When I started this thesis, I set out to learn WPF and XAML and improve my style of coding by using an object oriented approach and a Model-View-View-Model architecture. All of this was and I am very satisfied with the results. This is the first application on which I can look back and be happy with the structural design.

During the development many problems were encountered, most discussed in this thesis. Every problem, however, had a solution, some were easy and some were hard. The biggest problem encountered was the lack of graph support built into WPF. As a result, I had to build my own graph library. This solution had the biggest time investment but was also the biggest learning experience.

Aside from improving my coding style I have also learned many new C# functions and coding tricks courtesy of StackOverflow. I often found myself looking over previous code and editing it because I had figured out a better way. This tells me how much I have learned during this thesis.

The development of RainTools still continues. There are more tools to be added in the future. The bigger the application grows; the more challenging the development becomes, the more I learn from further development.

To conclude, I have learned everything from this thesis I had set out to learn and more, becoming an entrepreneur being the cherry on the top. Where I previously shied away from using version control and object oriented programming, now I cannot imagine doing another project without them.

REFERENCES

1. Wikipedia. 2003. C Sharp (programming language). Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29
2. Wikipedia. 2004. Graphical user interface). Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Graphical_user_interface
3. Wikipedia. 2003. HyperText Markup Language. Date of retrieval 26.08.2015
<https://en.wikipedia.org/wiki/HTML>
4. Wikipedia. 2007. Language Integrated Query. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Language_Integrated_Query
5. Wikipedia. 2004. Microsoft SQL Server. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Microsoft_SQL_Server
6. Wikipedia. 2009. Model View ViewModel. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Model_View_ViewModel
7. Wikipedia. 2003. MySQL. Date of retrieval 26.08.2015
<https://en.wikipedia.org/wiki/MySQL>
8. Wikipedia. 2003. Object-oriented programming. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Object-oriented_programming
9. Wikipedia. 2006. Foundation (non-profit) . Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Foundation_%28nonprofit%29
10. Wikipedia. 2003. Structured Query Language. Date of retrieval 26.08.2015
<https://en.wikipedia.org/wiki/SQL>
11. Wikipedia. 2005. Visual Basic for Applications. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Visual_Basic_for_Applications
12. Wikipedia. 2005. Windows Presentation Foundation. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Windows_Presentation_Foundation
13. Wikipedia. 2007. Windows Forms. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Windows_Forms
14. Wikipedia. 2005. Extensible Application Markup Language. Date of retrieval 26.08.2015
https://en.wikipedia.org/wiki/Extensible_Application_Markup_Language
15. Wikipedia. 2003. Extensible Markup Language. Date of retrieval 26.08.2015
<https://en.wikipedia.org/wiki/XML>
16. Microsoft. Source Control for Visual Studio. Date of retrieval 14.05.2015
<https://msdn.microsoft.com/en-us/library/zxd4dfad%28v=vs.90%29.aspx>
17. Microsoft. Use Visual Studio and Team Foundation Server with Git. Date of retrieval 14.05.2015
<https://msdn.microsoft.com/Library/vs/alm/Code/git/overview>
18. Microsoft. Use Version Control. Date of retrieval 14.05.2015
<https://msdn.microsoft.com/en-us/Library/vs/alm/code/overview>
19. GitHub. GitHub features page. Date of retrieval 15.05.2015
<https://github.com/features>
20. Microsoft. What is Visual Studio Online? . Date of retrieval 15.05.2015
<https://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx>

21. Redmine. Redmine Wiki. Date of retrieval 15.05.2015
<http://www.redmine.org/projects/redmine/wiki>
22. Todd Bishop. 2013. Microsoft challenges GitHub with new Visual Studio Online. Date of retrieval 15.05.2015
<http://www.geekwire.com/2013/microsoft-unveils-visual-studio-online-challenging-github-price/>
23. Microsoft. GitHub Extension for Visual Studio. Date of retrieval 15.05.2015
<https://visualstudio.github.com/>
24. Microsoft. Microsoft Azure: Cloud Computing Platform & Services. Date of retrieval 18.05.2015
<https://azure.microsoft.com/en-us/>
25. Microsoft. Microsoft Azure vs. Amazon Web Services (AWS). Date of retrieval 18.05.2015
<http://azure.microsoft.com/en-us/campaigns/azure-vs-aws/>
26. Motasem Aldiab. 2015. Public Cloud War: AWS vs Azure vs Google. Date of retrieval 18.05.2015
<http://cloudacademy.com/blog/public-cloud-war-aws-vs-azure-vs-google/>
27. MySQL. Why Move to MySQL from Microsoft SQL Server. Date of retrieval 20.05.2015
https://dev.mysql.com/tech-resources/articles/move_from_microsoft_SQL_Server.html
28. ITX Design. MySQL vs MSSQL. Date of retrieval 20.05.2015
<http://itxdesign.com/mysql-vs-mssql/>
29. Microsoft. Microsoft SQL Server 2014 Overview. Date of retrieval 20.05.2015
<http://www.microsoft.com/en-us/server-cloud/products/sql-server/>
30. MySQL. MySQL main page. Date of retrieval 20.05.2015
<http://www.mysql.com/>
31. Microsoft. Configure a Windows Firewall for Database Engine Access. Date of retrieval 20.08.2015
<https://msdn.microsoft.com/en-us/library/ms175043.aspx>
32. Microsoft. Azure SQL Database Firewall. Date of retrieval 20.08.2015
<https://azure.microsoft.com/en-us/documentation/articles/sql-database-firewall-configure/>
33. StackOverflow. StackOverflow main page. Date of retrieval 10.04.2015
<http://stackoverflow.com/>
34. Microsoft. ObservableCollection<T> Class. Date of retrieval 12.04.2015
<https://msdn.microsoft.com/en-us/library/ms668604%28v=vs.110%29.aspx>
35. CodeProject. 2009. List vs ObservableCollection vs INotifyPropertyChanged. Date of retrieval 12.04.2015
www.codeproject.com/Articles/42536/List-vs-ObservableCollection-vs-INotify-PropertyCha
36. Microsoft. LINQ (Language-Integrated Query). Date of retrieval 20.06.2015
<https://msdn.microsoft.com/en-us/library/bb397926.aspx>
37. StackOverflow. 2012. WPF, LINQ and the ObservableCollection). Date of retrieval 20.06.2015
<http://stackoverflow.com/questions/3257457/wpf-linq-and-the-observablecollection>
38. Jovan Popovic. 2012. Using LINQ Queries). Date of retrieval 20.06.2015
<http://www.codeproject.com/Articles/286255/Using-LINQ-Queries>

39. Microsoft. Windows Presentation Foundation Data Binding: Part 1). Date of retrieval 25.06.2015
<https://msdn.microsoft.com/en-us/library/aa480224.aspx>
40. Microsoft. Windows Presentation Foundation Data Binding: Part 2. Date of retrieval 25.06.2015
<https://msdn.microsoft.com/en-us/library/aa480226.aspx>
41. Joel Ivory Johnson. 2008. WPF Data Binding: Part 1. Date of retrieval 25.06.2015
<http://www.codeproject.com/Articles/29054/WPF-Data-Binding-Part>
42. Microsoft. Validation Class. Date of retrieval 05.05.2015
<https://msdn.microsoft.com/en-us/library/system.windows.controls.validation%28v=vs.110%29.aspx>
43. Paul Stovell. 2006. Validation in Windows Presentation Foundation. Date of retrieval 05.05.2015
<http://www.codeproject.com/Articles/15239/Validation-in-Windows-Presentation-Foundation>
44. Microsoft. What is XAML? Date of retrieval 15.03.2015
<https://msdn.microsoft.com/en-us/library/cc295302.aspx>
45. Microsoft. XAML overview. Date of retrieval 15.03.2015
<https://msdn.microsoft.com/en-us/Library/hh700354.aspx>
46. CodeProject. 2015. Introduction to Object Oriented Programming Concepts (OOP) and More. Date of retrieval 17.03.2015
<http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep>
47. Microsoft. The MVVM Pattern. Date of retrieval 18.03.2015
<https://msdn.microsoft.com/en-us/library/hh848246.aspx>
48. Jeremy Likness. 2010. Model-View-ViewModel (MVVM) Explained. Date of retrieval 18.03.2015
<http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained>
49. Josh Smith. A Guided Tour of WPF - Part 5 (Styles). Date of retrieval 20.06.2015
www.codeproject.com/Articles/18388/A-Guided-Tour-of-WPF-Part-Styles
50. Sacha Barber. 2008. Beginners Guide to Threading in .NET. Date of retrieval 23.07.2015
<http://www.codeproject.com/Articles/26148/Beginners-Guide-to-Threading-in-NET-Part-of-n>
51. Microsoft. How to: use a Background Worker. Date of retrieval 23.07.2015
<https://msdn.microsoft.com/en-us/library/cc221403%28v=vs.95%29.aspx>
52. Srivatsa Haridas. 2010. BackgroundWorker Class Sample for Beginners. Date of retrieval 23.07.2015
<http://www.codeproject.com/Articles/99143/BackgroundWorker-Class-Sample-for-Beginners>
53. Microsoft. XmlReader Class. Date of retrieval 28.04.2015
<https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx>
54. Microsoft. XmlWriter Class. Date of retrieval 28.04.2015
<https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx>

55. Microsoft. How to: Access Office Interop Objects by Using Visual C# Features. Date of retrieval 07.07.2015
<https://msdn.microsoft.com/en-us/library/dd264733.aspx>
56. Microsoft. Exceptions and Exception Handling. Date of retrieval 07.05.2015
<https://msdn.microsoft.com/en-us/library/ms173160.aspx>
57. Microsoft. Exception Handling Fundamentals. Date of retrieval 07.05.2015
<https://msdn.microsoft.com/en-us/library/2w8f0bss%28v=vs.110%29.aspx>
58. Microsoft. Exception Handling. Date of retrieval 07.05.2015
<https://msdn.microsoft.com/en-us/library/ms173162.aspx>
59. Daniel Turini. 2005. Exception Handling Best Practices in .NET. Date of retrieval 07.05.2015
www.codeproject.com/Articles/9538/Exception-Handling-Best-Practices-in-.NET
60. Mohammad Dayyan. 2009. How to Create a WPF User Control & Use It in a WPF Application. Date of retrieval 21.05.2015
<http://www.codeproject.com/Articles/32825/How-to-Creating-a-WPF-User-Control-using-it-in-a-W>
61. MahApps. MahApps Metro main page. Date of retrieval 20.03.2015
<http://mahapps.com/>
62. Torsten Mandelkow. 2013. Modern UI (Metro) Charts for WPF. Date of retrieval 03.04.2015
<https://modernuicharts.codeplex.com>
63. CodePlex. 2009. WPF Toolkit download page. Date of retrieval 03.04.2015
<http://wpf.codeplex.com/releases/view/29117>
64. Telerik. Telerik RadChart Overview. Date of retrieval 03.04.2015
<http://docs.telerik.com/devtools/wpf/controls/radchart/overview.html>
65. Mahesh Chand. 2009. Charting in WPF. Date of retrieval 03.04.2015
<http://www.c-sharpcorner.com/UploadFile/mahesh/charting-in-wpf/>
66. Microsoft. Timer Class. Date of retrieval 06.04.2015
<https://msdn.microsoft.com/en-us/library/system.timers.timer%28v=vs.110%29.aspx>
67. CodeProject. 2014. Simple Animation for your Controls using C#.NET. Date of retrieval 06.04.2015
www.codeproject.com/Tips/841159/Simple-Animation-for-your-Windows-Forms-or-Control
68. Wikipedia. 2004. End-user license agreement. Date of retrieval 16.08.2015
https://en.wikipedia.org/wiki/End-user_license_agreement
69. Vangie Beal. EULA – End-User License Agreement. Date of retrieval 16.08.2015
<http://www.webopedia.com/TERM/E/EULA.html>
70. Wikipedia. 2006. Privacy policy. Date of retrieval 16.08.2015
https://en.wikipedia.org/wiki/Privacy_policy
71. Wikipedia. 2006. Disclaimer. Date of retrieval 16.08.2015
<https://en.wikipedia.org/wiki/Disclaimer>
72. Microsoft. ClickOnce Deployment. Date of retrieval 10.08.2015
<https://msdn.microsoft.com/en-us/library/t71a733d%28VS.80%29.aspx>
73. Preeti Baranga. 2007. ClickOnce – Quick steps to Deploy, Install and Update Applications. Date of retrieval 10.08.2015
www.codeproject.com/Articles/17003/ClickOnce-Quick-steps-to-Deploy-Install-and-Update

74. Microsoft. Smtplib Class. Date of retrieval 11.05.2015
<https://msdn.microsoft.com/en-us/library/system.net.mail.smtplibclass%28v=vs.110%29.aspx>
75. CodeProject. 2012. Simple SMTP E-Mail Sender in C#. Date of retrieval 11.05.2015
<http://www.codeproject.com/Tips/301836/Simple-SMTP-E-Mail-Sender-in-Csharp-Console-applic>

APPENDICES

Appendix 1 Memorandum of initial data

Appendix 2 RainTools windows

Appendix 3 Animated results

INITIATION DOCUMENT OF A BACHELOR'S THESIS

Author _____ Dirk Jan van Luijtelaar

Customer _____ Stichting RIONED

Customer's contact person and information _____ Harry van Luijtelaar

_____ Senior Project Manager

Title _____ RainTools

Description _____ The student is to create a user interface RainTools that

_____ communicates with the RainTools Engine. The interface is used

_____ to feed data into the engine and retrieve the results.

_____ The results are to be displayed using tables and graphs

Objectives _____ Using version control, object oriented programming and MVVM

_____ Learning the possibilities of WPF and XAML

Experience the process of software development for an organization

Target schedule _____ Winter 2014: RainTools demo complete

_____ Spring 2015: RainTools base complete

_____ Autumn 2015: Thesis report complete

Date and signatures _____ 26 August 2015

RAINTOOLS WINDOWS

To convey the magnitude of RainTools and the amount of work gone into this project, I included a screen cap of almost all the windows in RainTools. I will not go into the details too much because the content is not relevant to the learning process of this project.

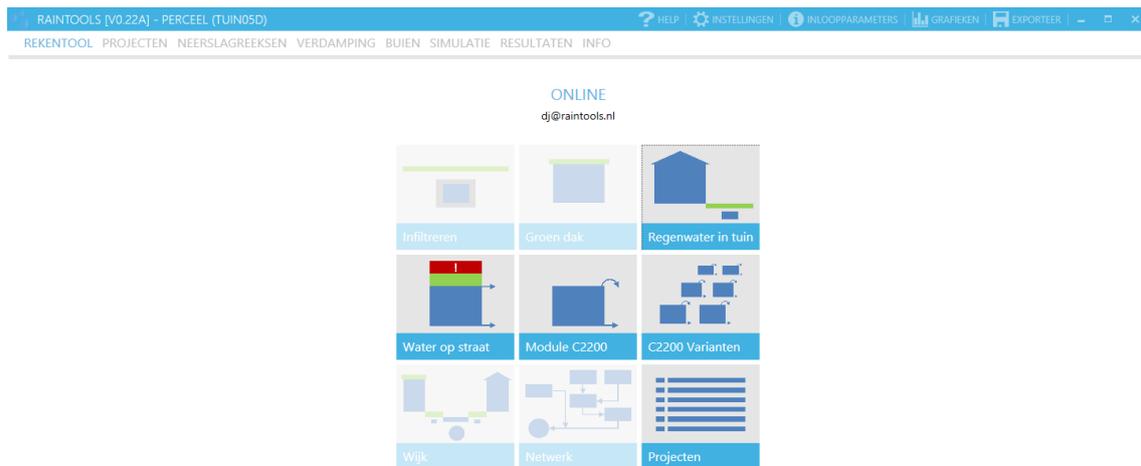


FIGURE 29. Welcome screen with tiles for different tools (some inactive), each tile is tied to a different input screen. On top a menu where several Flyouts can be toggled and under that a bar with a TabControl to navigate to different windows.

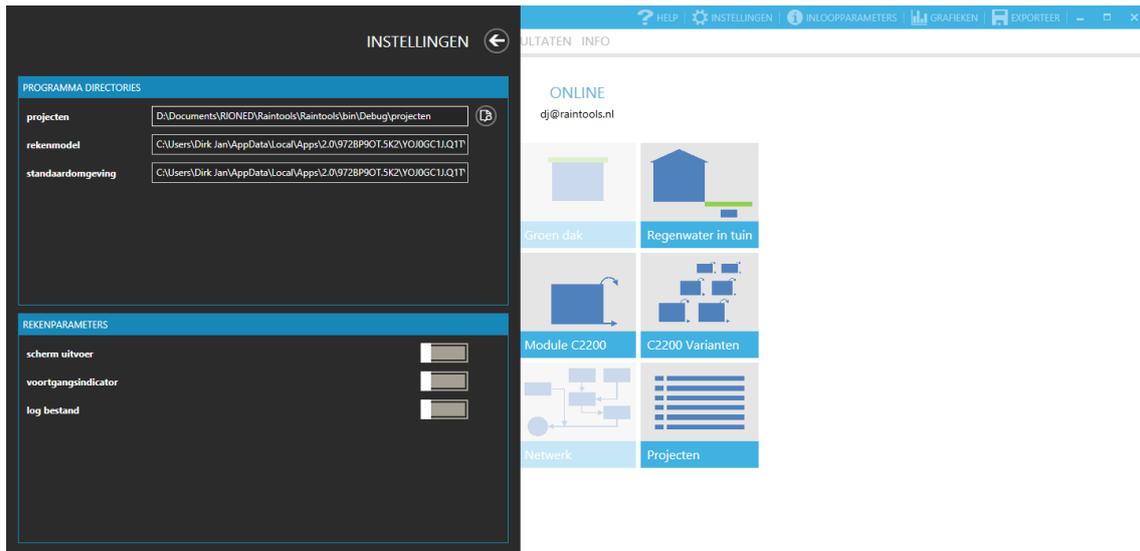


FIGURE 30. Settings Flyout, used to assign project folder path, RainTools Engine path, a path to the default environment (folder which contains default values for constants) and a few general settings.

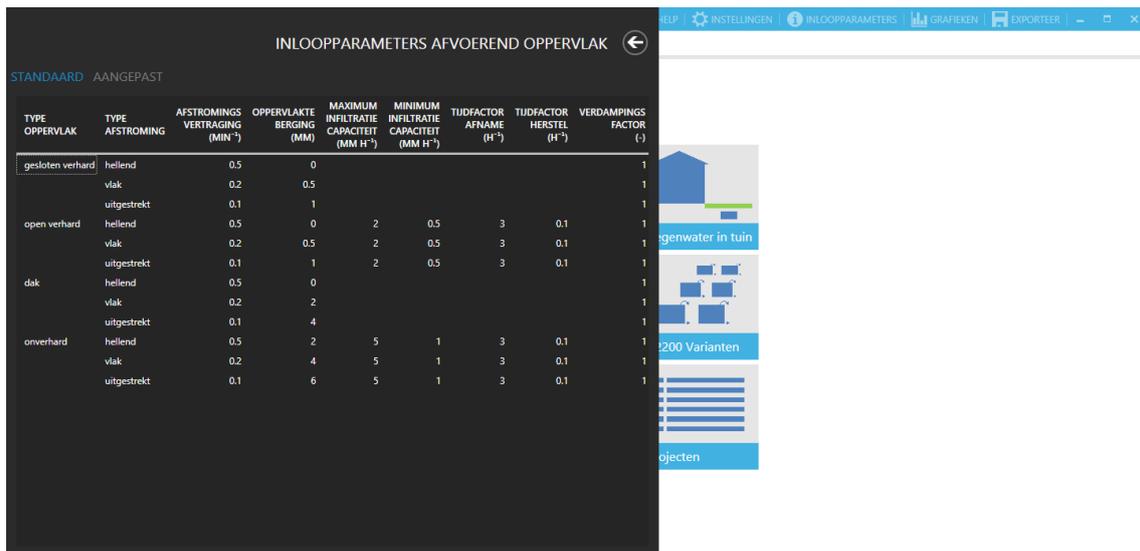


FIGURE 31. Flyout for default parameters for different surfaces. Custom surfaces can be added in the “AANGEPAST” Tab.

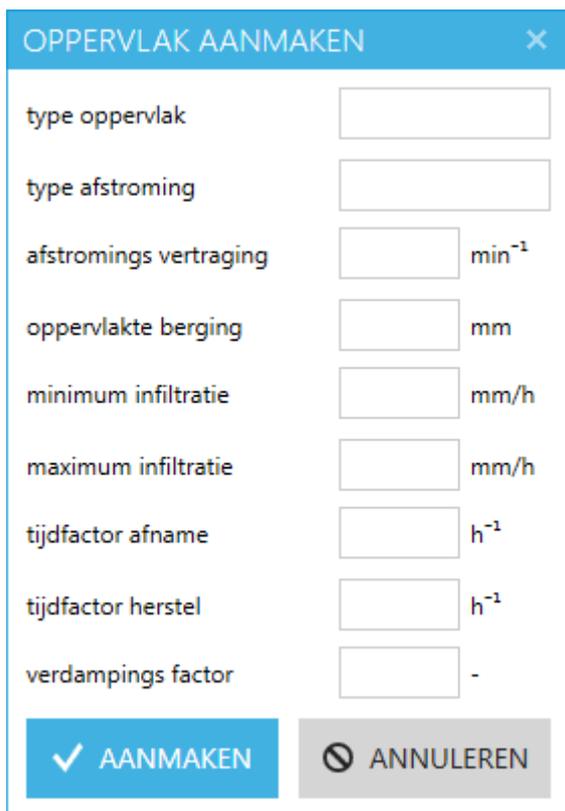


FIGURE 32. Window to add a new surface, similar to the window to edit a surface.

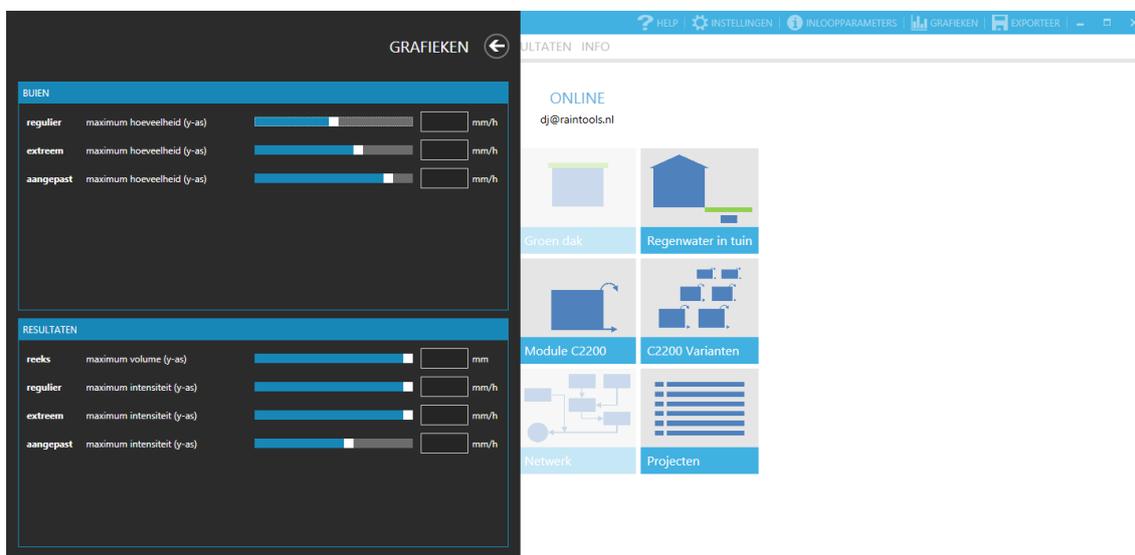


FIGURE 33. Graph settings Flyout, used to set the maximum values of the axes.

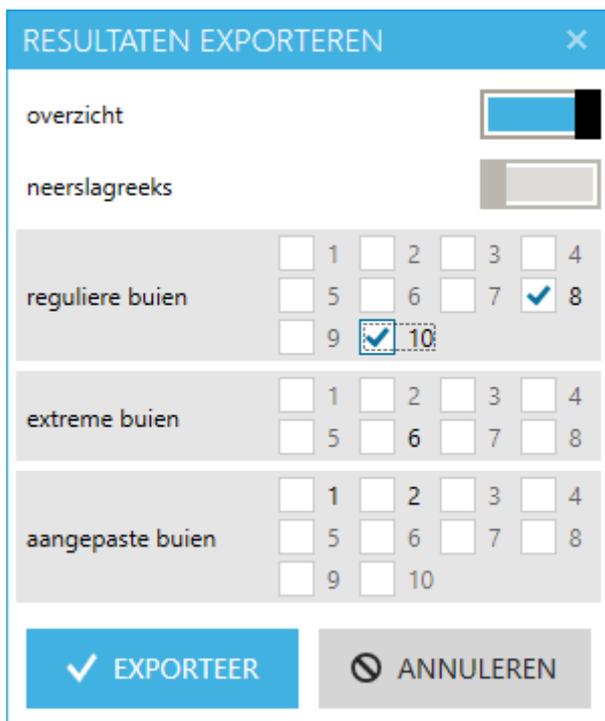


FIGURE 34. Exporting results to Excel, using the checkboxes different results can be exported.

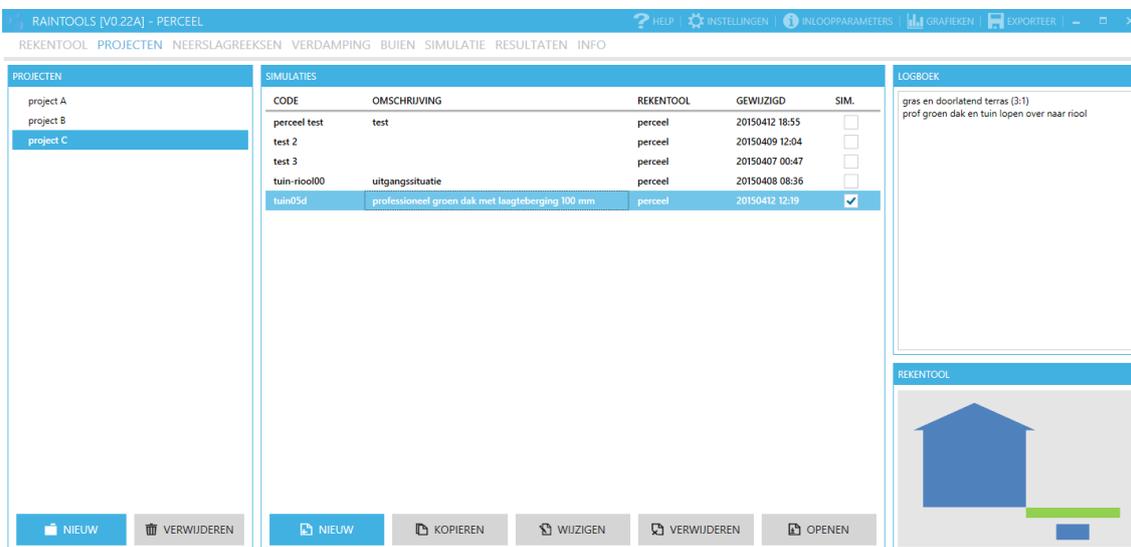


FIGURE 35. Project Tab; on the left the projects found in the project folder, in the middle the simulations of the selected project, on the right a log file of the selected simulation. Buttons are to create and delete projects and to create, copy, edit, delete and open simulations.

PROJECT AANMAKEN

projectnaam

omschrijving

AANMAKEN ANNULEREN

FIGURE 36. Window to add a new project.

SIMULATIE AANMAKEN

simulatiecode

omschrijving

project

project C

rekentool

perceel

AANMAKEN ANNULEREN

FIGURE 37. Window to add a new simulation, similar to the window to edit a simulation.

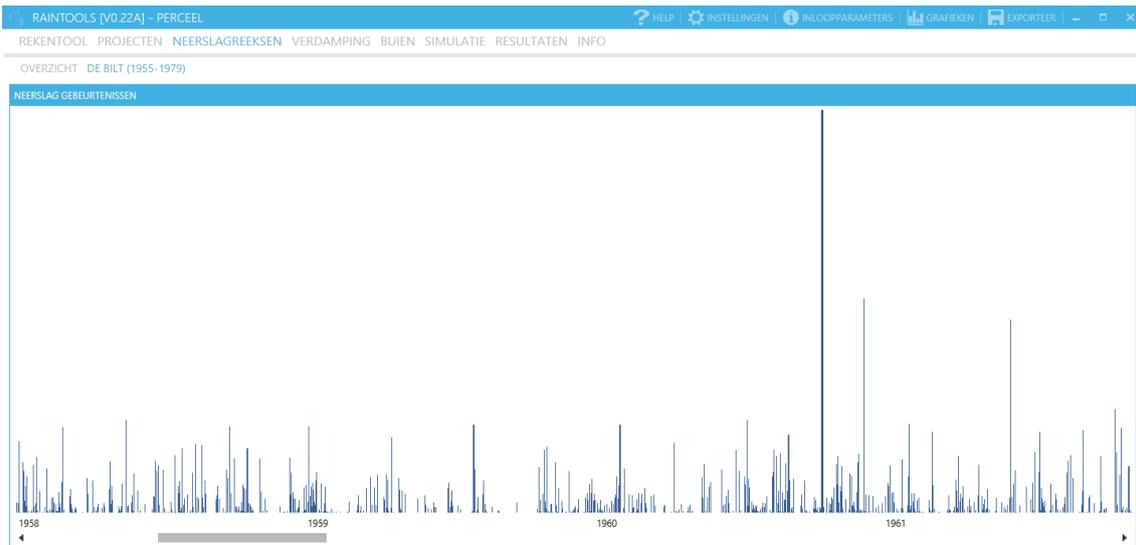


FIGURE 38. Precipitation series data from 1955 to 1979 in a bar graph.

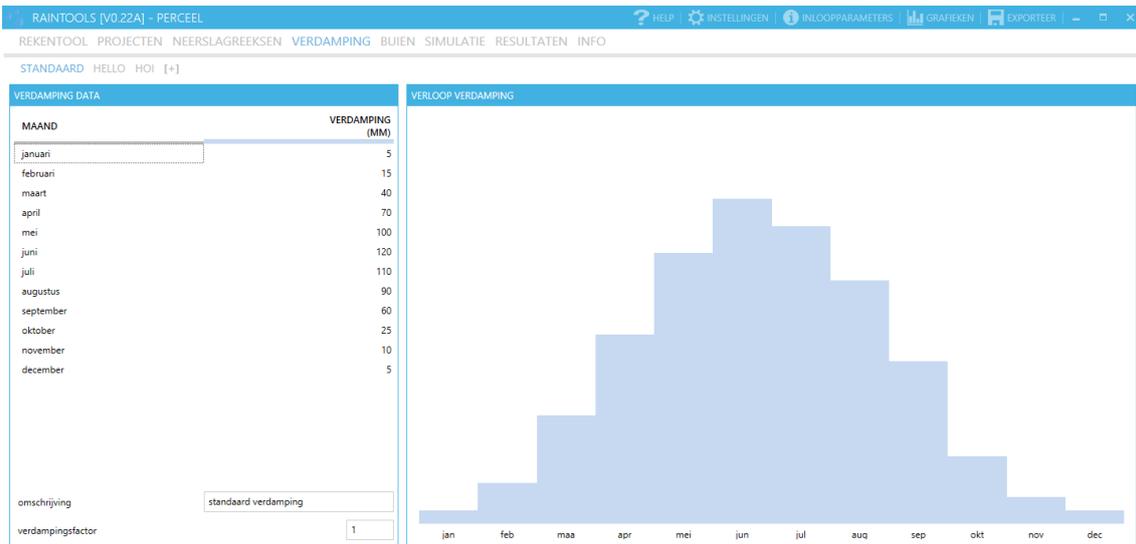


FIGURE 39. Evaporation coefficients, can be edited using the table on the left. When the values are edited the graph is automatically updated. A new evaporation dataset can be added by pressing the “[+]” Tab.

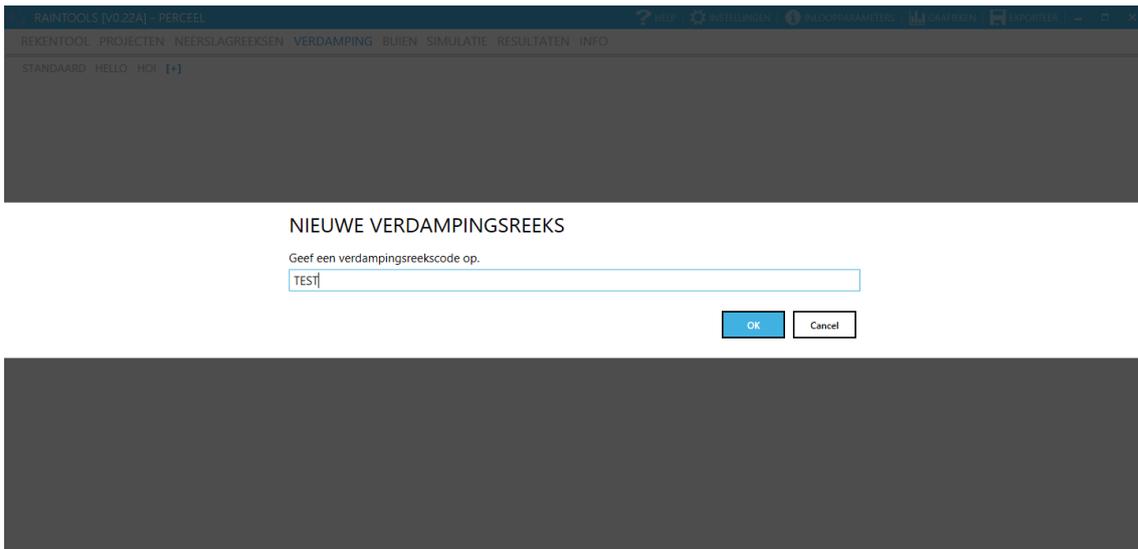


FIGURE 40. Adding a new evaporation dataset.

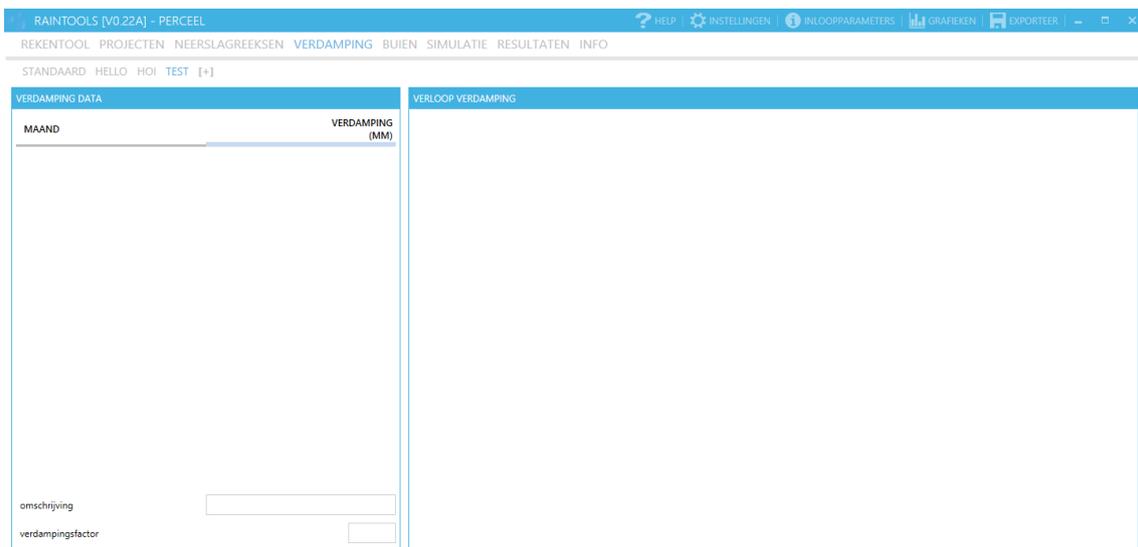


FIGURE 41. After adding the "TEST" evaporation dataset, data will have to be added into the table on the left.

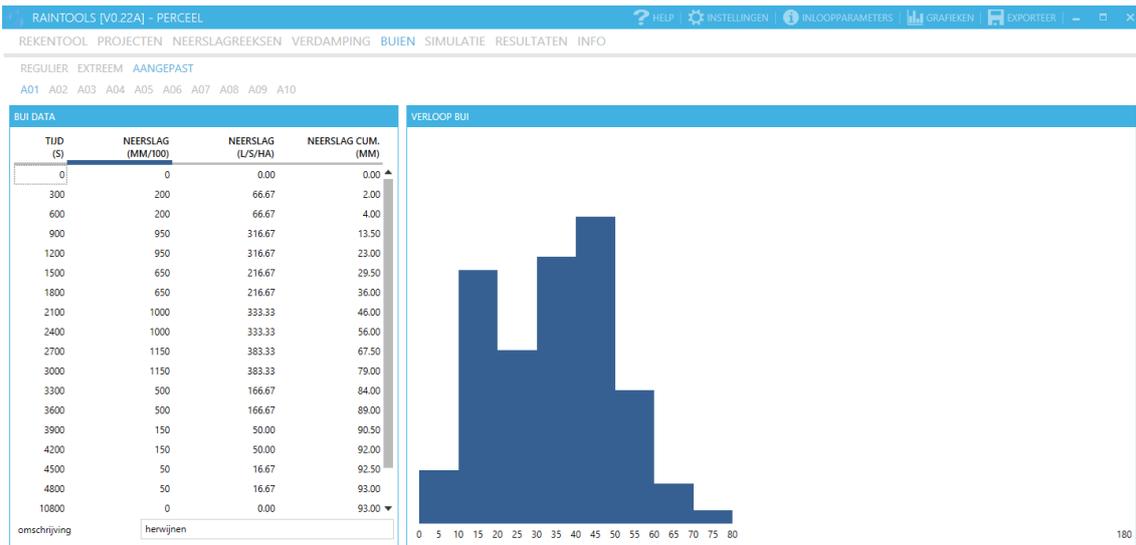


FIGURE 42. Rainfall data, can be edited using the table on the left. When the values are edited, the graph is automatically updated.

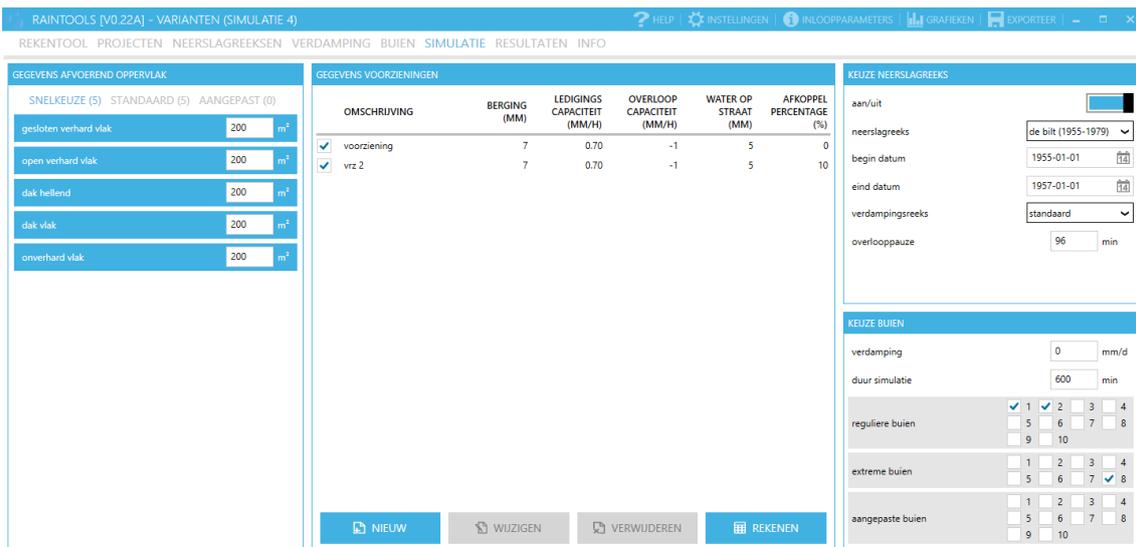


FIGURE 43. Simulation data page for a simulation with pre-defined surfaces. On the left the different surfaces where the sizes have to be input. In the middle the facilities that handle the water coming from the surfaces. On the right the rainfall events can be selected.

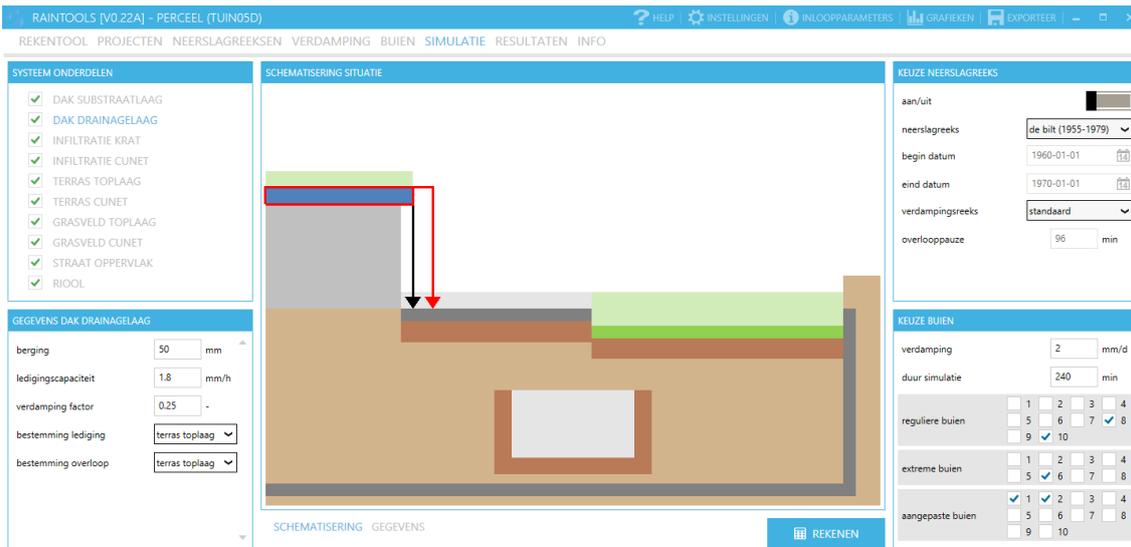


FIGURE 44. Simulation data page for a simulation with default facilities. Image in the center is computer drawn and dynamic to the data that is input. On the left the parameters for the different facilities and on the right the rainfall events can be selected.

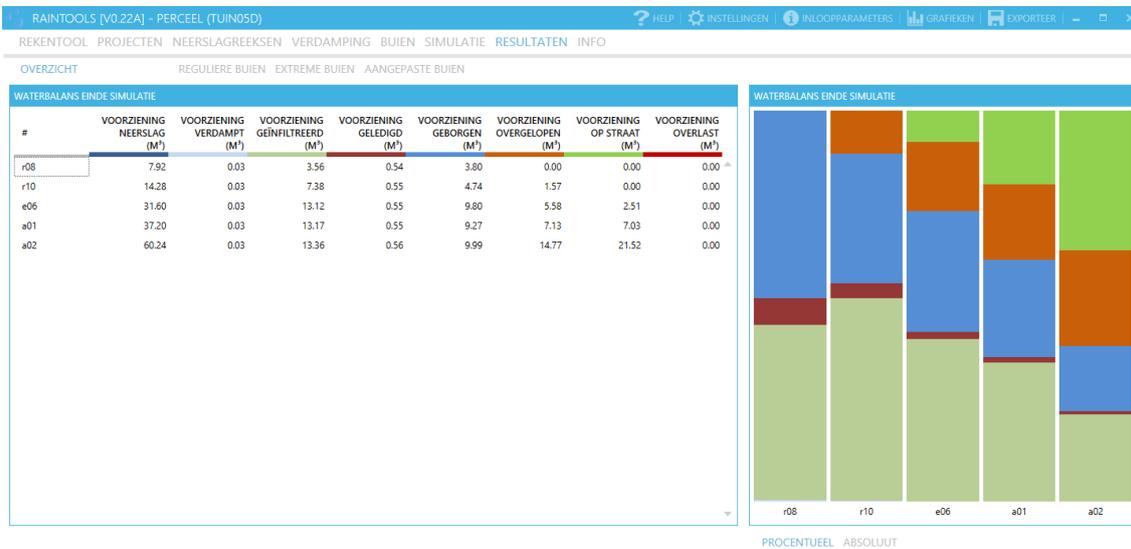


FIGURE 45. Results overview page, different results for different rain events. The results can be viewed by navigating through the different Tabs.

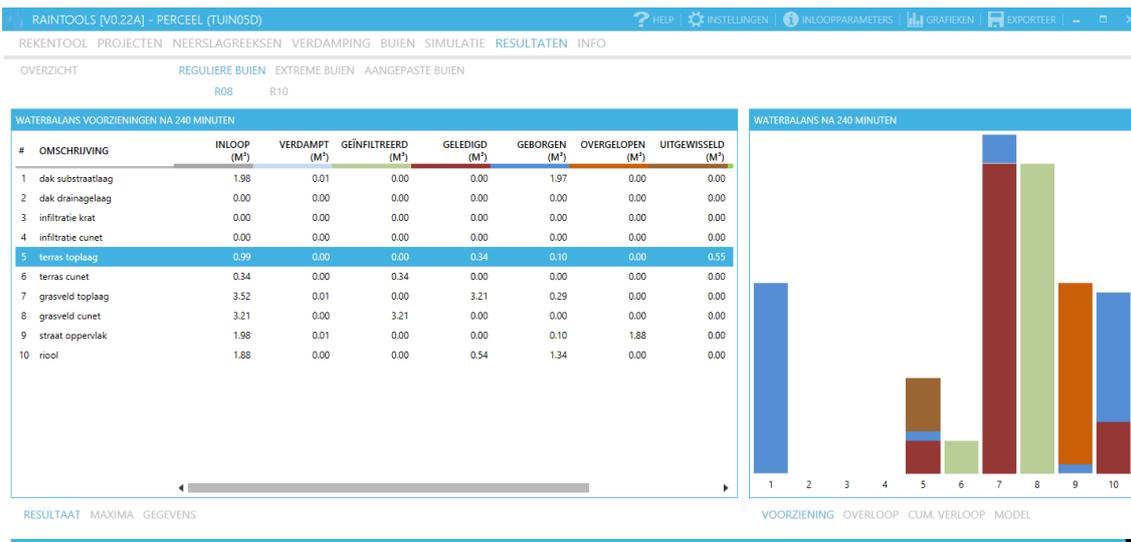


FIGURE 46. Results page with a DataGrid stacked bar graph. Using the slider on the bottom the data can be viewed at different times during the simulation. Each row in the data table represents a rainwater facility that is present in the simulation (see FIGURE 44).

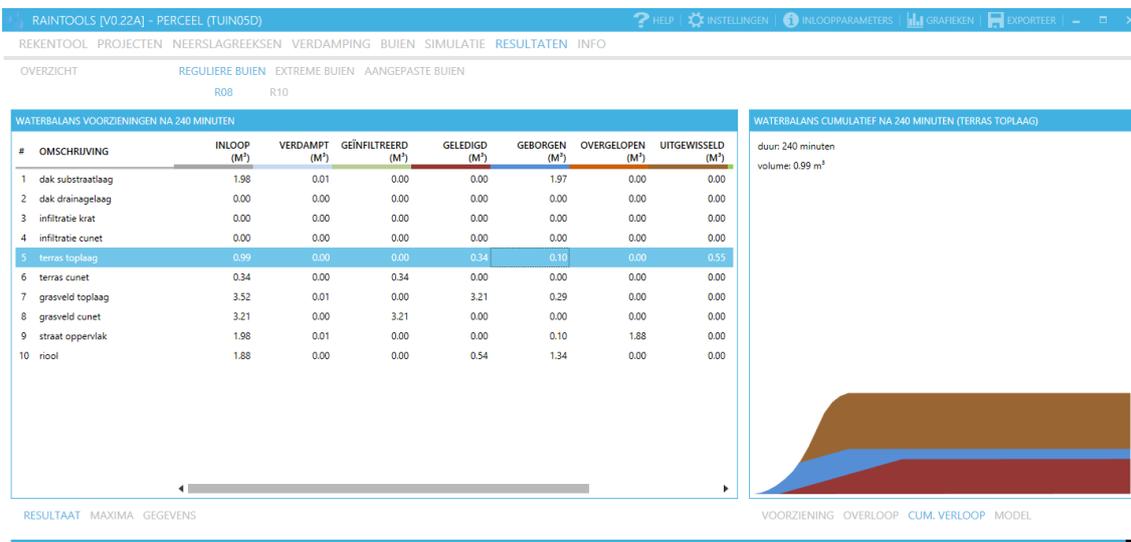


FIGURE 47. Results page with area graph, gives a better overview of time.

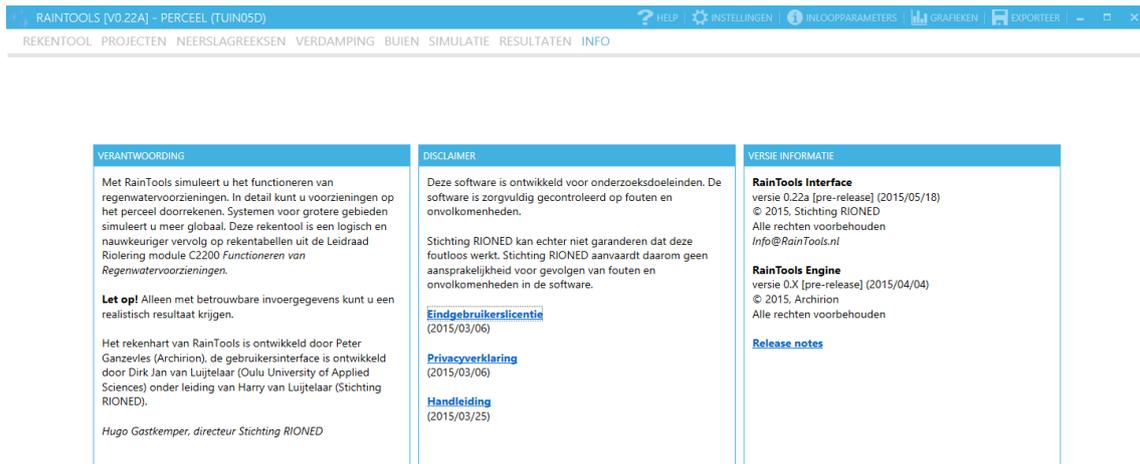


FIGURE 48. Info tab, displays information about RainTools such as disclaimer and links to the privacy policy, manual and release notes.

ANIMATED RESULTS



FIGURE 49. Animation area graph (0 minutes).

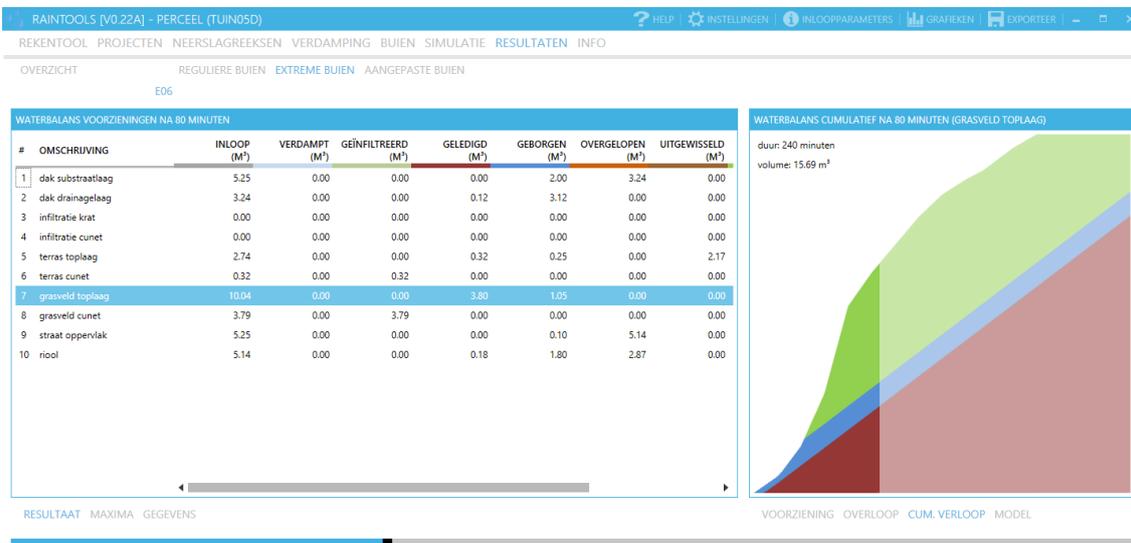


FIGURE 50. Animation area graph (80 minutes).

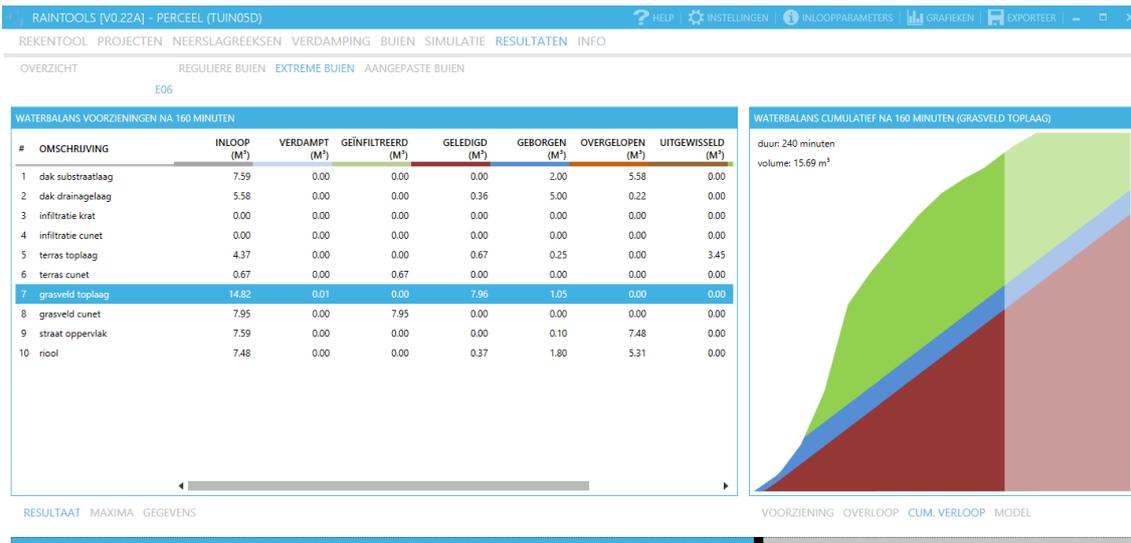


FIGURE 51. Animation area graph (160 minutes).

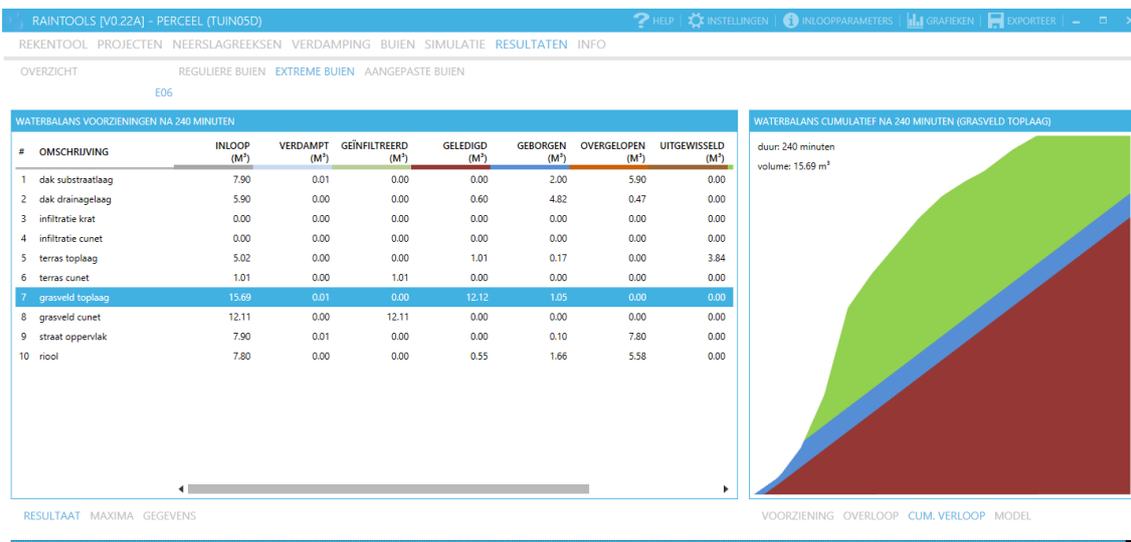


FIGURE 52. Animation area graph (240 minutes).



FIGURE 53. Animation stacked bar graph (80 minutes).



FIGURE 54. Animation stacked bar graph (160 minutes).

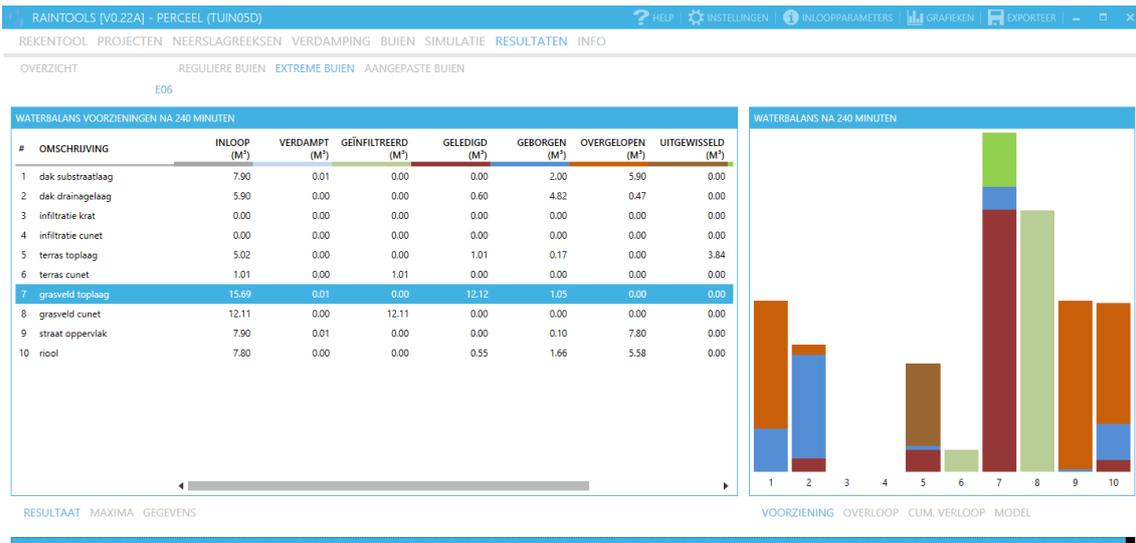


FIGURE 55. Animation stacked bar graph (240 minutes).