Unplagged Final Report

Building the Plagiarism Detection Workbench

Term paper for the master project II
Mentoring Teacher: Prof. Dr. Debora Weber-Wulff

Department Economics II
HTW Berlin – University of Applied Sciences

Elsa Mahari (s0534556) <s0534556@htw-berlin.de>
Dominik Horb (s0534217) <horb@htw-berlin.de>
Tien Nguyen (s0512510) <s0512510@htw-berlin.de>
Benjamin Oertel (s0522720) <contact@benjaminoertel.com>
Heiko Stammel (s0534218) <heiko.stammel@googlemail.com>

Contents

List of Figures

Introduction

Nine months after the start of development, the time of the masters project Unplagged at the HTW has now come to an end for the initial team members. Hopefully though, this isn't the end of the lifecycle of the software Unplagged that was created during that stretch. Some of us will try to continue working on it afterwards and maybe we are even able to attract some other developers to help this open source project in the long run if we are lucky. It was a useful experience after all and the foundation that was layed feels good and strong.

Although there are no direct connections, this present document is at least in some ways a sequel to the Developers Manual, that was written after the first semester of this project. In some instances it assumes prior knowledge of things already described there, so that it doesn't unnecessarily need to be repeated here. All in all, we are going to give a more technical insight into the system than in the report before, which concentrated mostly on the project management and development environment aspects. We will also try to critically analyze the problems that were faced and the mistakes that were made over the course of those two semesters.

For all the team members, the development of Unplagged up to this point was one of the biggest projects that we ever had to start and needed to bring to a state of usable maturity on our own. Most of us came across big or even huge projects at work some time, but the process until the first release was mostly long over and replaced by bugfixing and the eventual feature release. So those team building phases forming, storming, norming and perfoming that Bruce Tuckman famously described(Tuckman, 1965), were something we experienced in this project for the first time in full scale.

During our time of study, it was also one of the first projects, where we couldn't really

envision in the beginning, how the end result would need to look like. It surely happened before at some points, but the implications of wrong decisions in the beginning weren't nearly as big then, as with such a long running development process like here.

With other projects before, there were also often requirements similar to some parts of software systems we already knew and just needed to adapt to the current purpose. This mostly gave us some hints of the feasibility and the best approach to solve a problem. With Unplagged this was somehow different.

First of all, the idea for the project came from outside the team with very short time to research before the team building what it would be about, second we didn't have any experience with plagiarism research at all and additionally the team was bigger then we are normally used to. This means that we couldn't even imagine how the necessary workflow would need to look like, so we started out with one abstract idea of the system from outside of the development team, turned that into a different idea for every team member and eventually had to break it down to one software system.

We discoverd that Unplagged would have some similarities to project management systems like Redmine or Jira and maybe Google Docs after a while, but at the start this wasn't really obvious to us.

Luckily enough, the choices of technologies, architecture or programming languages we made in the beginning, didn't came to haunt us up to this point. Without the experiences we made at internships or work before, this probably would have been a different story though.

Chapter Overview

1. Project Overview

This first chapter will give a very brief overview of the preconditions and project structure, analyzing it by some key figures. It will also describe the "toolbox" of frameworks and libraries that were integrated to have at our disposal for the development.

2. Features

Here we will describe the features and workflow implemented in the system so far.

3. Showtime

The chapter Showtime will explain the preparations that were made for the presentation of the project results at the showtime.

4. Project Analysis and Outlook

In the last chapter the main points are the discussion of the mistakes and problems that were made during the development, with a focus on possible provisions against recurrence in future projects and an outlook on the future of Unplagged.

Conventions

To markup important words in the text, the following typographical conventions are used:

Italic

First used technical terms

`Constant Width`

Programm code, file names, paths

`Bold Constant Width`

Variables that have to be changed by the user

1. Project Overview

The main project description we started to work with was quite short with about 300 words and was also hugely complemented and clarified in meetings with Prof. Weber-Wulff. Nonetheless it contains very important parts that are the foundation of what was built during the project:

> "Right. I'm the plagiarism 'huntress'. I've tested plagiarism detection software since 2004. Mostly, they suck. They either don't work, or are a pain to use, or both. I've spent 10 years trying to educate people about plagiarism, and still the media thinks I have a magic secret for discovering plagiarism, as demonstrated on the GuttenPlagWiki and the VroniPlagWiki. What actually happens there is that quite a number of tools are used for preparing texts, discovering possible sources, comparing them, and documenting them. The last part is done by hand and takes an enormous amount of time.
>
> The idea is to set up a Plagiarism Detection Cockpit that integrates all sorts of bits and pieces, but leaves the teacher in command. It is not to be a general test system that spits out a number for every paper submitted, although the integration of as many such systems as possible will be one of the necessary features of the system. There will be a lot of thought needed for the interface design, as there are massive amounts of data that need to be displayed. How can this be compressed and fit on a screen? For example, a barcode-generation needs to be integrated.
>
> The goal will be to provide a tool that easily produces simple-to-read documentation and deals with all sorts of nastiness that might turn up on the way. The tool must be multi-lingual and open source. It would also be cool

> to integrate some of the little tools such as the Android-based OCR-Scanner that was developed in the SS 2011. I would like for the team to use an agile development methodology so that we can continuiously test with teachers and professors and *Plaggers.
>
> One of the first tasks will be collecting up the computing literature on the topic of plagiarism discovery. A wiki needs to be set up with links to material (online and offline), comments on the papers, and as many navigational indices as possible."(Weber-Wulff, 2011)

In the beginning the main takeaways we used from this text were:

- **no** automated process, "just" a tool that aids the users workflow
- needs an open source license
- multi language support

If you read the text carefully though, you may notice that we overlooked or at least marginalized some important clues, but you will find descriptions of those problems later on in the chapter Project Analysis and Outlook.

One important information that was only stated during the first project proposal was, that it also needed to be web based.

For the frontend this made the choices of languages and technologies pretty easy, because there are some clear web standards, without much alternatives aside from the version numbers. As we wanted to develop in a future-proof manner and chose to support only from Internet Explorer 8 upwards, this lead us to CSS3 for styling, HTML5 for the markup and JavaScript for client-side scripting. With JavaScript being the only one of those three with a real competitior in Googles Dart, but as the browser support is still very weak, this wasn't a possibility.

On the server-side however the possibilities for programming languages and platforms are a little bit broader than this. The most important criterias for us regarding those

choices were, that it had to be flexible and already very widespread, because this would be helpful for an open source project to gather a community later on.

From the twenty most used programming languages according to Tiobe (2012) only PHP, Ruby, JavaScript(with Node.js), Java and Python seemed really suitable for web development. Naturally the familiarity and preferences of the team members also played a role in the final decision. We made bad experiences with Java in this area and felt that structuring JavaScript and Python in a big application could be problematic. From the two languages that were left, the one we had the most experience with and that also has the biggest following was PHP, so it made the final cut.

For the "platform" we only made the decision to use Apache as the web server, because it is the most common(Netcraft, 2012) and it also runs on many different operating systems, so that the real platform choice doesn't need to be made.

As you will see below, the usage of a database was also made in a flexible way. We currently use MySQL, but it should be possible to swap the storage engine easily if necessary.

1.1. Statistics

We gathered some data of the state of Unplagged as of July 7th 2012 with the tool Count Lines of Code(CLOC). We know that by no means this can be used as a quality measurement, but we believe that it can give some structural insights and is a good starting point for an introduction of the technical area.

The table in figure 1.1 shows the complete(At least according to CLOC) count of files, comment lines and lines of code, that are currently part of Unplagged. It contains also all the code that is part of all used libraries or frameworks.

In all the following pie charts the measurements you will see are narrowed down as much as possible[1] to contain only code that was created by members of the team.

[1]CLOC was called with the following arguments: perl /cloc-1.56.pl application tests scripts public

Language	Files	Comments	Code
PHP	3262	275579	429308
XML	194	0	234696
HTML	2645	191	24684
CSS	1220	843	7093
Javascript	59	1932	5239
XSD	108	6	1058
SQL	49	58	102
C	18	16	72
make	4	4	13
Sum:	4013	278629	702265

Figure 1.1.: Complete measurements including library/framework files

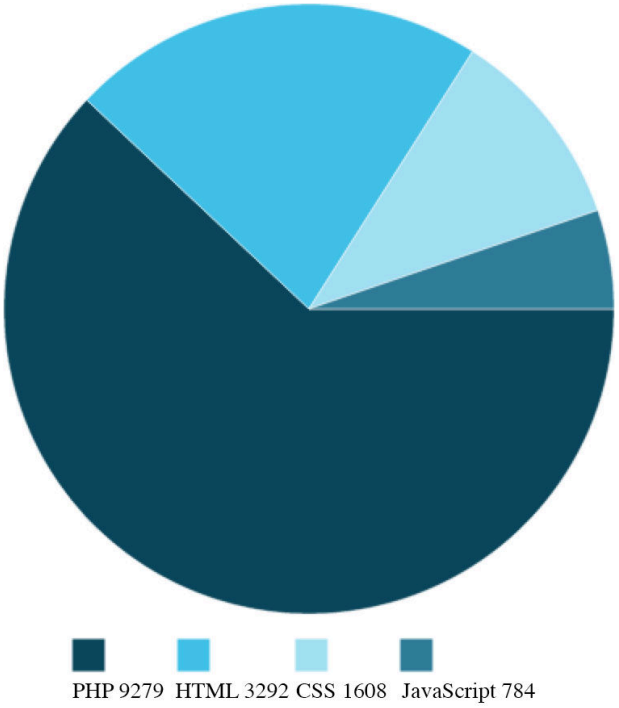


Figure 1.2.: Distribution of created lines of code by language.

As you can see when comparing the numbers in figure 1.2 with the table, we are really

library/Unplagged –force-lang=html,phtml –exclude-dir=libs

standing on the shoulders of giants as is often said, with just about 2.1% of the whole PHP code written by us for example.

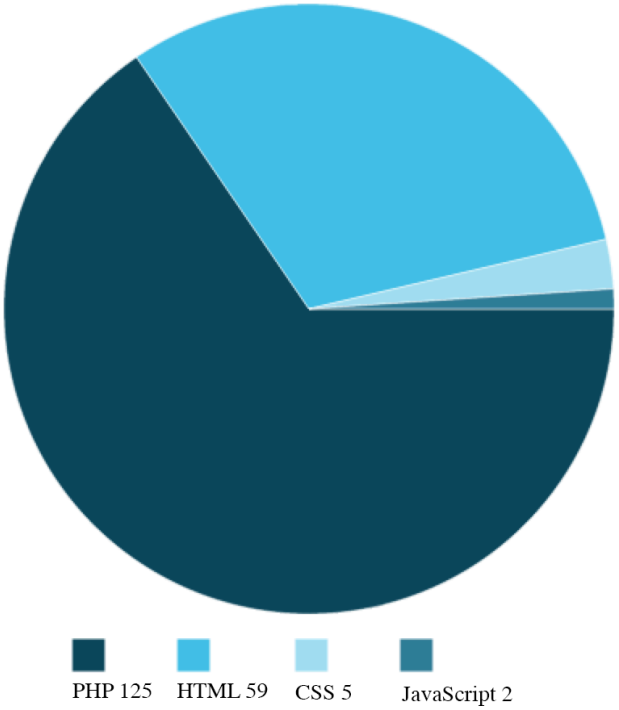


Figure 1.3.: Distribution of self created files in Unplagged.

What could be a bit misleading is the count of HTML files and lines, because they mostly contain some kind of mish-mash of PHP and HTML, including some simple loops to generate reccuring elements.

For the main architecture of the system that lies in the PHP part, we feel that the ratio of 125 files(see figure 1.3) to 9279 lines of code(see figure 1.2) or ca. 75 lines of actual code per file is a pretty good achievement, as it is at least an indicator of a good software design. For JavaScript and CSS the numbers are not that great, but this happened mostly due to performance concerns, because it is really desirable to reduce HTTP requests the client has to make, which can be easily done by putting it all in very big files.

With comments it's even more complicated to judge the quality by pure numbers of lines, because they are just there and have no "understanding ratio" of readers that could be easily measured, but to complete this short excursion into statistical territory: In figure 1.4, you can see the distribution of comments by language, with PHP again leading the

pack by far.

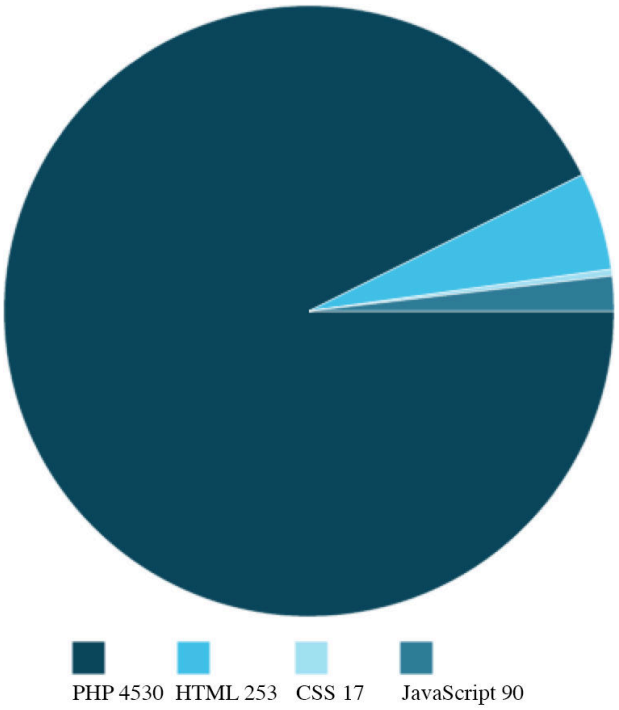


Figure 1.4.: Distribution of comment lines by language.

1.2. Licensing – GPLv3

Choosing an open source license was something that was surprisingly difficult for us, because we never had to do it before and there are so many different possibilites out there with many upsides and downsides.

We eventually decided to use the GNU Public License in version 3(GPLv3). The reasons for this choice are the very widespread usage of it, meaning that many people are already familiar with it, the international validity of the specified terms and the "viral" character of it, so that any additions and changes to the software also need to be provided under the same open source license.

It has therefore the benefit of keeping the possibility to dual-license the system with another proprietary license later on in order to maybe even sell it to organizations who

want to include customizations without making them public, which would be mandatory under the GPLv3, but making it freely available to others that operate under the specified terms.

1.3. Project Structure and Technologies

Over the course of the project we integrated something that could be called “toolbox” of libraries and frameworks, which helped us develop faster, more efficient or according to some kind of “best-practices” in some areas.

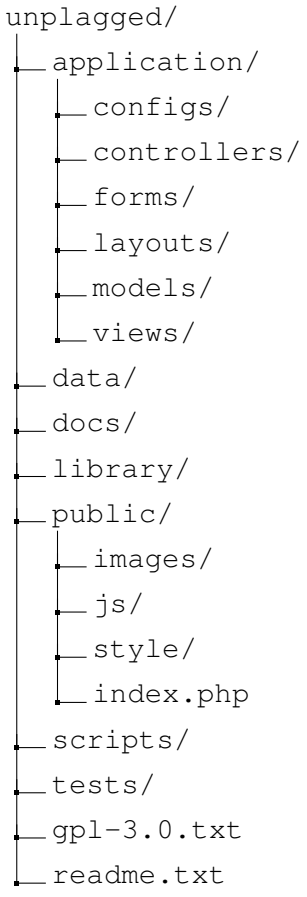


Figure 1.5.: Most important directores of Unplagged

The main directory structure you can see in figure 1.5 is mostly like the one recommended

by the Zend framework documentation[2], which is a major building block of Unplagged and the first thing we integrated.

Most of the custom code that we wrote lies in the `application` directory and `library` contains nearly all the foreign code.

Public resources like CSS, JavaScript and some image files are all served from the `public/` directory, so the web server must be configured correctly to use this directory as the webroot. Every request that is made to the application is routed through the `index.php` file in this directory as well and then dispatched to Zend framework.

1.3.1. Zend Framework

Zend is a collection of PHP classes, that can be used in two different ways, first as a simple set of libraries, of which we for example made use for translating the application into different languages, creating webforms or for the rights management. But second, and more importantly, it also provides a framework for a MVC application, which automatically routes URLs according to a scheme into the correct controller classes.

It has some competitors mainly in CakePHP, but they essentially have the same features, so this choice was again more a matter of taste and experience.

1.3.2. Doctrine ORM

This can as well be used in different ways, but for our purposes, the most important bit was the possibility to write plain old PHP classes and annotate them in a way that maps them to database tables. Through this mechanism it is possible to generate the whole database by calling one simple script with the credentials of the database and have therefore flexibility regarding the used DBMS.

[2] http://framework.zend.com/manual/en/project-structure.project.html

1.3.3. HTML5 Boilerplate

The HTML5 Boilerplate was the starting point for our front end development and is a pretty nice collection of best-practices mostly for CSS and HTML. It contains for example a CSS reset that tries to bring the default styles of all HTML elements for every browser to the same point, some basic CSS classes for image replacements or clearing floats and ways to target older browsers via specific CSS classes.

1.3.4. jQuery

Essentially all the JavaScript we wrote could be written without jQuery. The main benefit it brings though, is that it hides the differences between the JavaScript implementations of different browsers, so that coding gets much more comfortable and robust.

Besides this, jQuery also provides a nice set of animations, that in some cases and if not used excessively, can help make the user interface a little nicer.

1.3.5. Twitter Bootstrap

Twitter Bootstrap is a set of mostly CSS components that was integrated very late, about six or seven month into the project, and replaced the buttons and error messages we already had built. This was primarily done because the usability of those elements had a much nicer flow and the browser compatibility was also more robust out of the box. Later on we also used it also on some more components like dialogs and the paging element.

The problem with Bootstrap is, that webpages that use it tend to look very uniform, because of all the similar looking elements, but we made this choice here, because we felt that for a webapp the usability is more important than a unique design. And hopefully we also kept some uniqueness to it.

1.3.6. Responsive Design

Responsive Design isn't really a tool or framework like all the others described here, but more a paradigm regarding the way a webpage is styled via CSS. It means that CSS media queries are used to target specific viewport widths, so that no matter what device is used to view the application, the page always has an optimized layout.

2. Features

2.1. Notifications and Comments

The following section describes an important part of the application, especially for registered users: the notification system and the opportunity of adding comments basically to any resource in the system.

2.1.1. Recent activity stream

It displays the most recent events in the portal in descending order they happened. Each activity (figure 2.1), one line in the stream, consists of 3 parts: meta information about the activity itself, information about the initiator and comments. The content of each of these parts is being determined automatically, when a new notification is being persisted to the database.

Besides the initiator information and comments, each activity has an own title, description and an icon. These meta information texts can be edited in the portal in the 'Administration' > 'Actions' section. A fixed set of notifcations is provided in the portal, they are editable but not removable.

The following code snippet shows, how to create a new notification when a new automatic plagiarism detection report was created. The static method takes in 3 parameters: a unique name for the notification type, the content object related to the notification and a user object, as the third parameter. A list of all available notification types can either

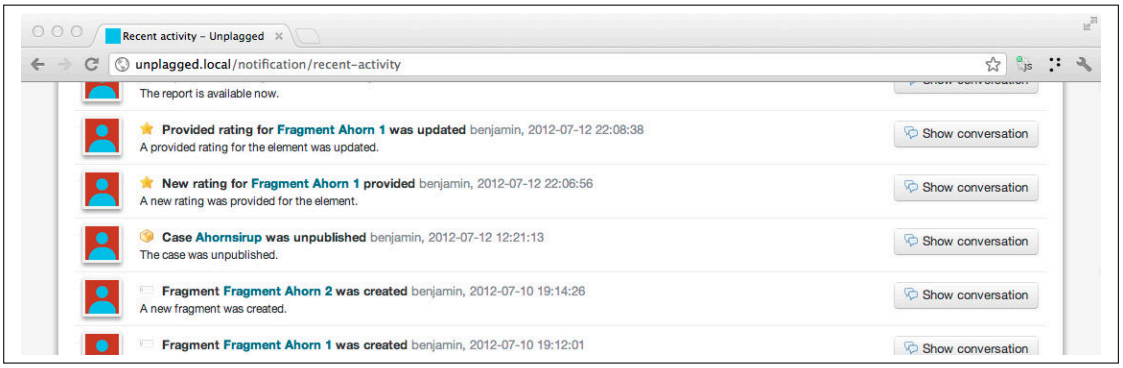


Figure 2.1.: Single activity in the activity stream

be found in the previousely mentioned actions section or in the scripts/build/initdb.php file, where all notifcation types are being declared.

Listing 2.1: Creating a notification for a created report

```
1 Unplagged_Helper::notify("detection_report_created", $report,
      $report->getUser());
```

Unplagged does have an extensive role and permission management. Therefore the grant on each resource is verified, before it is displayed in the activity stream. Usually the resource related to the notification and the resource the permission check is performed on are the same. Although in some cases, e.g. when rating a fragment, the resource will be the rating itself, but the permission check is done on the fragment. In this case the notify-method is called with a fourth, optional parameter, another resource, in this case the fragment. When the user has access on the fragment, all ratings can be accessed as well, automatically and so are the notifcations.

2.1.2. Comments plugin

Comments are simply a small text related to a specific user and a resource. They can be used to share ideas on a content object collaboratively. The most prominent part at Unplagged, where comments are being used, is the activity stream. A comment can be added by any user having access to the notification.

Figure 2.2.: Creating a comment on a resource

For providing a better workflow to the user, the comments can be refreshed and added in place. That means, the position where the user scrolled to in the browser does not get affected. The in-place refreshing is realized through AJAX. The comments container is loaded empty and displaying a small loading image only. Not before the user clicks the 'show conversation' button, the comments are being fetched through a post request to the server. Whenever the result is fetched completely, the spinner graphic is hidden and the comments are appended. The parsing of the comments markup is done in Javascript as well. So the server requests are kept small and the server can return JSON only without any HTML.

Listing 2.2: Refreshing the comments of a resource

```
1   target.show();
2       conversation.hide();
3       loading.slideDown(800, function() {
4         // get the whole conversation
5         $.post('/notification/conversation', {
6           'source': sourceId
7         }, function(data) {
8           if(!data.errorcode) {
9             conversation.html("");
10            $.each(data, function(index, value) {
11              conversation.append(renderConversation(value));
12            });
13            loading.slideUp(800, function() {
14              conversation.slideDown(300);
15            });
16          } else {
17            conversation.html('<div class="comment">' + data.
                  message + '</div>');
```

```
18              loading.slideUp(800, function() {
19                conversation.slideDown(300);
20              });
21            }
22          }, "json");
```

Listing 2.3: Creating the markup of a single comment

```
1  function renderConversation(data, target) {
2      var tpl;
3
4      switch(data.type) {
5        case 'comment':
6          tpl = '<div class="comment">' +
7          '<div class="image"><img class="avatar-small" src="' +
               data.author.avatar + '" /></div>' +
8          '<div class="details">' +
9          '<div class="title"><b>' + data.author.username + '</b
               > ' + data.text +
10         ' <span class="date">' + data.created.humanTiming +
               '</span>' +
11         '</div>' +
12         '</div>' +
13         '</div>';
14         break;
15     }
16     if(!target) {
17       return tpl;
18     } else {
19       target.append(tpl);
20     }
21   }
```

2.2. Fragments

Fragments are the part of the application where found text passages that are plagiairism or potential plagiarism are documented.

A single fragment contains a candidate and a source document. Each of the two documents is saved with a starting position (page number / line number combination) and an ending position. These two positions can be used to determine exactly the text involved in a fragment. To visualize this, the figure 2.3 below shows a sample fragment.

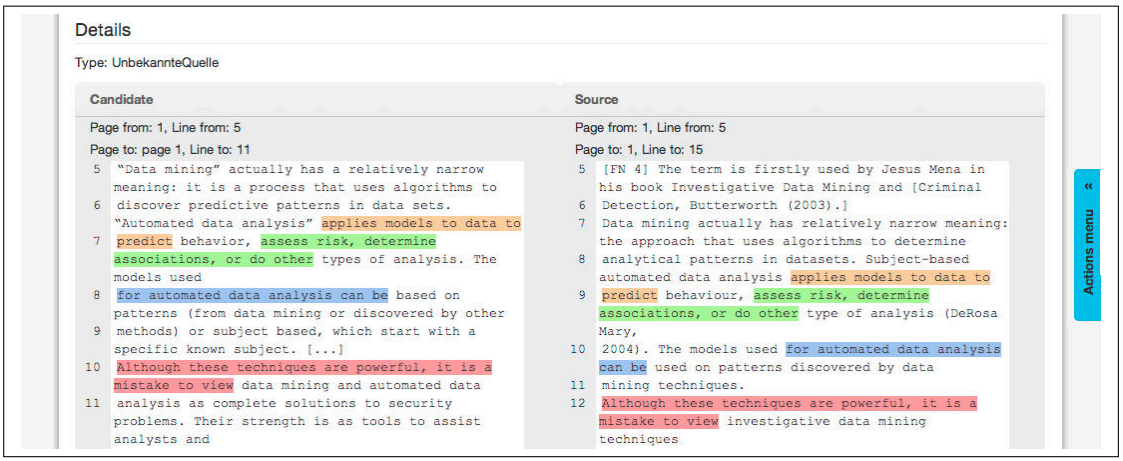


Figure 2.3.: Single fragment with highlighted similarities

2.2.1. Creating a fragment

Such a fragment can be created in two ways, one for people that like using the mouse and another one that can be used with the keyboard only.

The old-fashioned way

The basic way, which can be accessed through the keyboard only, offers a two-column form to the user where the source and potential plagiarism information can be selected by hand. Once a page or line number is changed, the text shown below is updated instantly through AJAX and the similarities are highlighted automatically. Although the

big disadvantage of this method is, that the whole page is never displayed and the user actually has to guess where the starting and ending point of the fragment in the text really area. Therefore the values of the line from and line to fields have to be increased or decreased by hand, until they are adjusted properly.

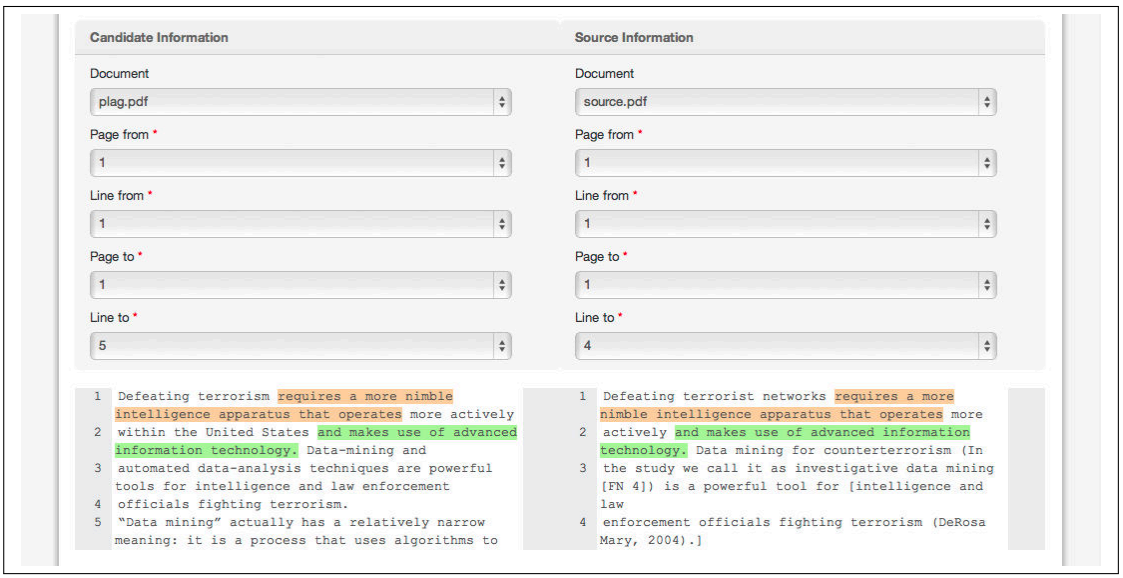


Figure 2.4.: Form for creating a fragment by hand

A more comfortable workflow

Wouldn't it be cool to select text by just marking it with the mouse and having this previousely described form filled out automatically? Well, we though it would be, so we implemented it.

The user has to go to the document inspected in the current case, select a page to start with and then hit the button 'Switch to two-column view for fragment creation'. At this point a second document can be selected on the right side and the similarities in both texts are once again highlighted as shown in figure 2.5. In this two-column view it is also possible to iterate through the pages of the left-side or right-side document to compare page 1 from the left with page 2 from the right and page 1 on the left with page 3 on the right just by one click.

When there are sufficient similarities in an area of the page, a fragment can be created by marking the text, then making a click with the right mouse key to open the context menu

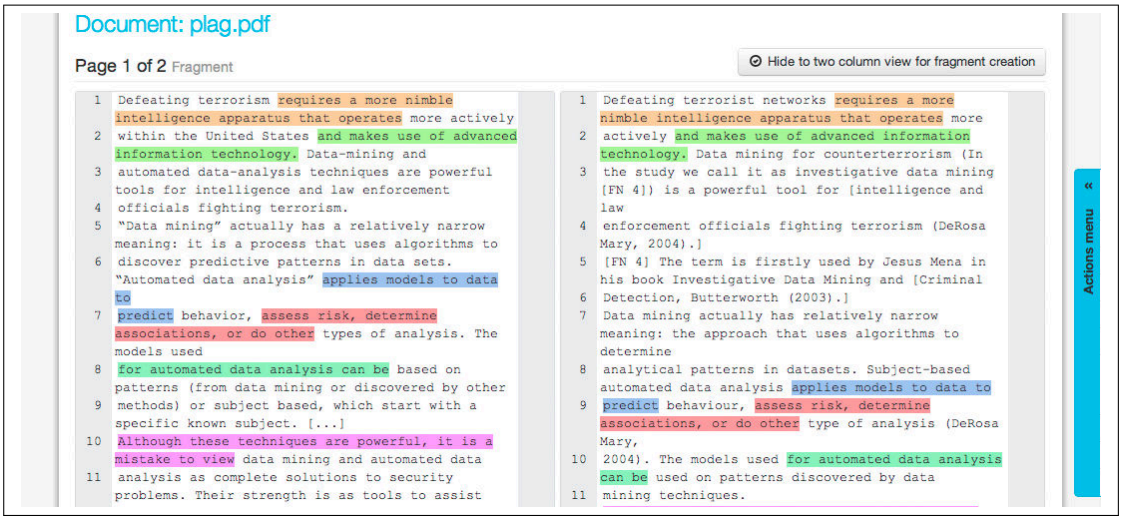


Figure 2.5.: Creating a fragment the modern way - Step 1

and 'set as candidate/source of fragment'. This stores the marked text temporarily until the 'create fragment' button in the context menu is pressed (figure 2.6). The selection of the 'create fragment' button opens the same form as described in the section before, but at this time pre-filled with the start and end values for page and line on both sides automatically.

This techchnique makes it much easier to create a fragment, since the text can be seen in the context of the whole page, before it is added to the fragment form.

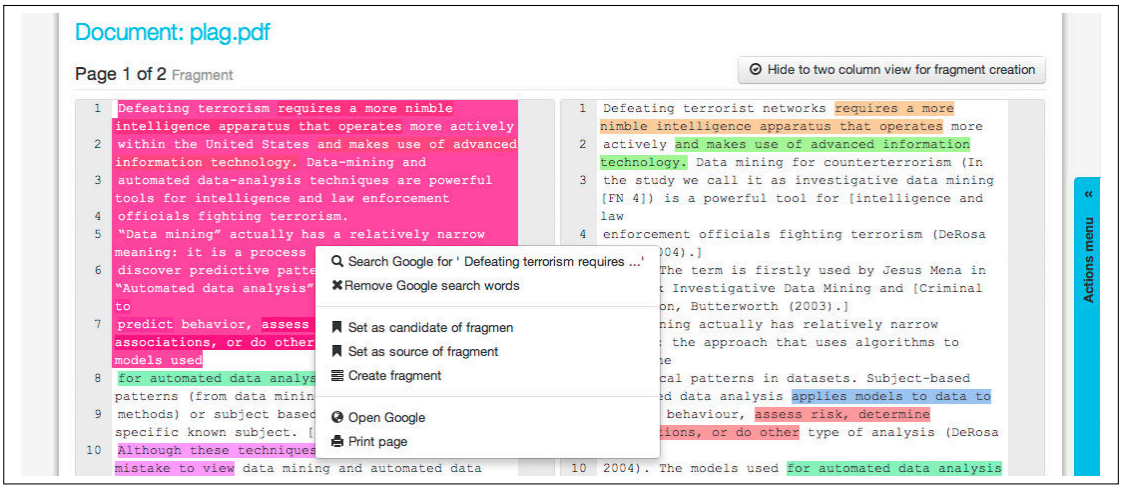


Figure 2.6.: Creating a fragment the modern way - Step 2

2.2.2. Rating a fragment

A created fragment has to be verified by other collaborators of the case in order to be approved for containing plagiarism. This process is described in the current section.

A rating contains information about the user who made the rating, a flag whether it approves or declines the fragment and an optional property that contains a description, why the user gave the rating. Each fragment can be approved by a user only once. However the reason and rating type can be changed by the initiator at any time, until the fragment is approved by a certain amount of people. The amount of ratings to lock a fragment and its ratings for further editing is defined in the case administration form. Whenever this amount is reached, the fragment gets locked automatically and can be unlocked by administrators only.

Figure 2.7.: List of fragment ratings

2.3. User avatar

With the ‘User avatar’ the user gets the possibility to upload a picture to his own profile. Through the function ‘Profil edititieren’ the user gets to the ‘Profil editieren’ form.

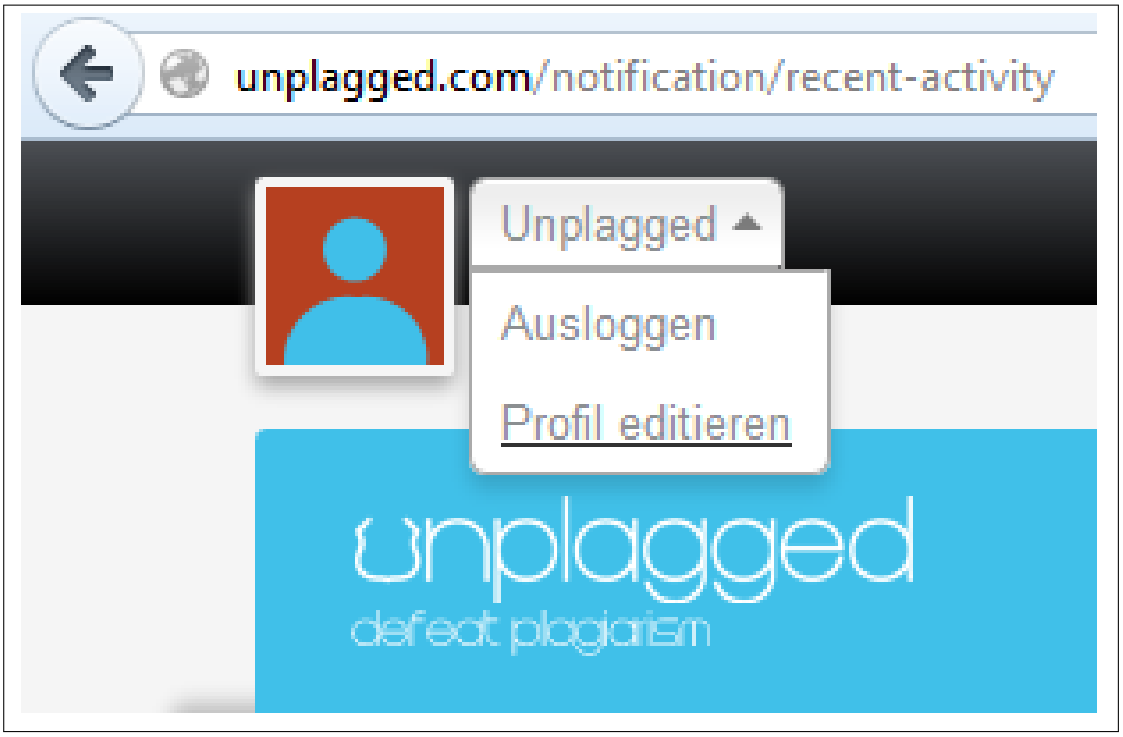


Figure 2.8.: edit profile

The uploader in this form allows the user to upload a picture as his avatar.

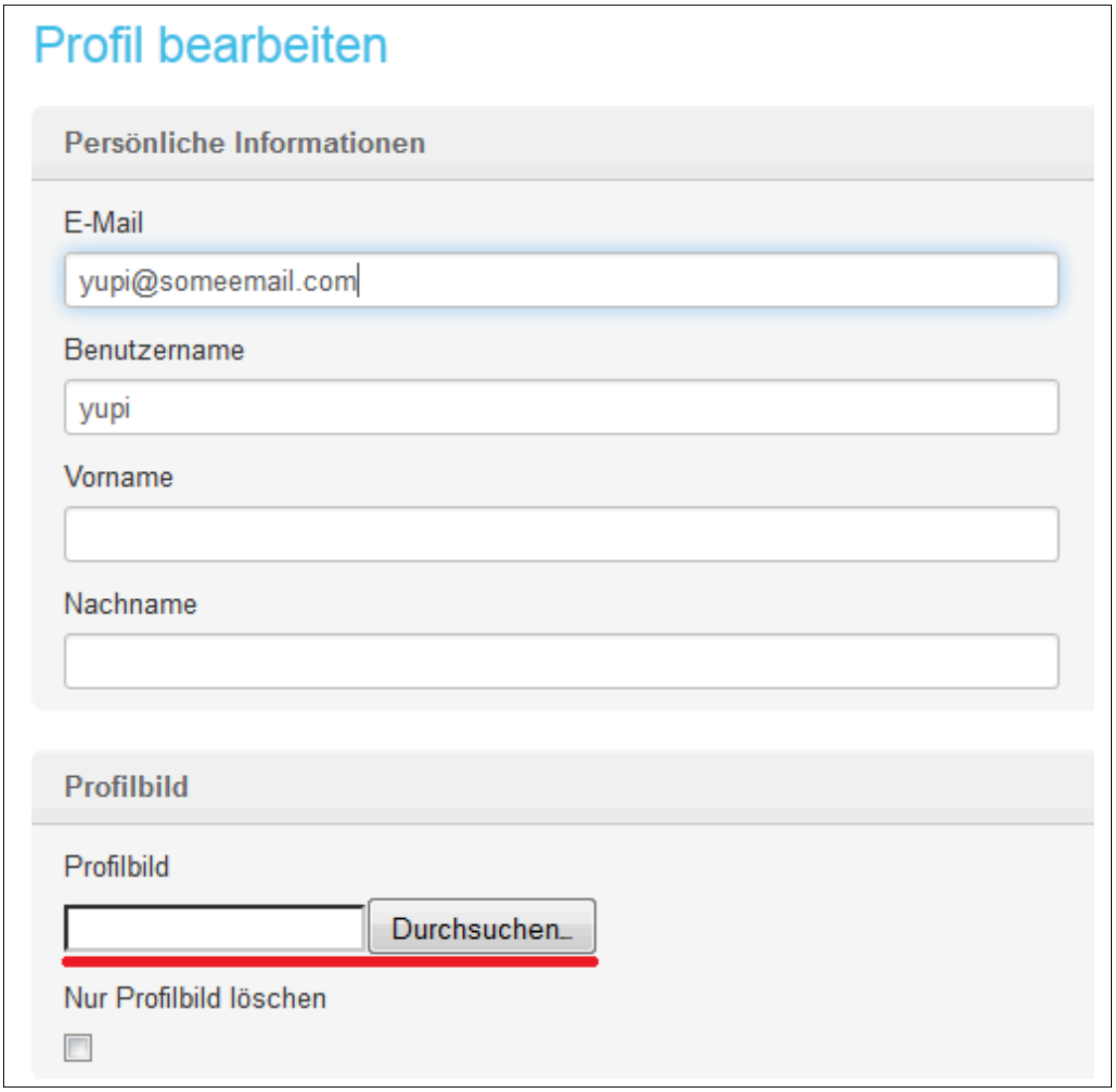


Figure 2.9.: profile avatar uploader

2.3.1. Avatar cropping

While the previous section described how the user avatar can be set, this one explains how the cropping of images that are not in the correct aspect ratio is done. All avatars have to be in a square aspect ratio. However, not all users have the skills to provide an image that meets these requirements. So we decided to crop uploaded images into the appropriate format after they are uploaded.

What we do is taking the shortest of the two rectangle sides and cut a partial from the center of the longer one of the two sides, which has the same length as the shortest side. To make it more visual what that means, figure 2.29 shows an example. The red area is the part that will be cropped.

Figure 2.10.: Cropping avatar

The cropped image then is scaled to the needed width and height, currently 50 x 50 pixels. The algorithm for cropping that has been developed, is shown in the following code snippet.

Listing 2.4: Cropping an image to a square aspect ratio

```
1  public function crop($thumbWidth, $thumbHeight){
2      //getting the image dimensions
3      list($width, $height) = getimagesize($this->file->
           getFullPath());
4
5      //saving the image into memory (for manipulation with GD
           Library)
6      $myImage = imagecreatefromjpeg($this->file->getFullPath())
           ;
7
8      // setting the crop size
9      if($width < $height) {
10       $twidth = $width;
11       $theight = $width;
12       $x = 0;
13       $y = $height / 2. - $width / 2.;
14     } else {
15       $twidth = $height;
```

```
16        $theight = $height;
17        $x = $width / 2. - $height / 2.;
18        $y = 0;
19      }
20
21      // creating the thumbnail
22      $thumb = imagecreatetruecolor($thumbWidth, $thumbHeight);
23      imagecopyresampled($thumb, $myImage, 0, 0, $x, $y,
            $thumbWidth, $thumbHeight, $twidth, $theight);
24
25      imagejpeg($thumb, $this->file->getFullPath());
26
27      imagedestroy($thumb);
28      imagedestroy($myImage);
29
30      return $this->file;
31    }
```

2.4. Automatic Plagiarism Detection Webservices

Unplagged itself is a workbench, where the plagiairism detection is done by hand. Although it provides useful automatic tools that help the user to make the process of plagiairism detection easier. One of these tools are external webservices that check a specific text for plagiairism and try to find sources which can be used for further inspections.

We were talking to 3 companies offering such a webservice: Docoloc, PlagScan and PlagAware. As a proof of concept the PlagAware webservice has been implemented and added to our application, it will be described in the following section.

2.4.1. PlagAware

PlagAware, a website for automatic plagiarism detection is a commercial website which costs money depending on the amount of text to analyze. It figures out possible sources of the text handed in on their website and creates a PDF report with a list of all the sources found, as shown in figure 2.11.

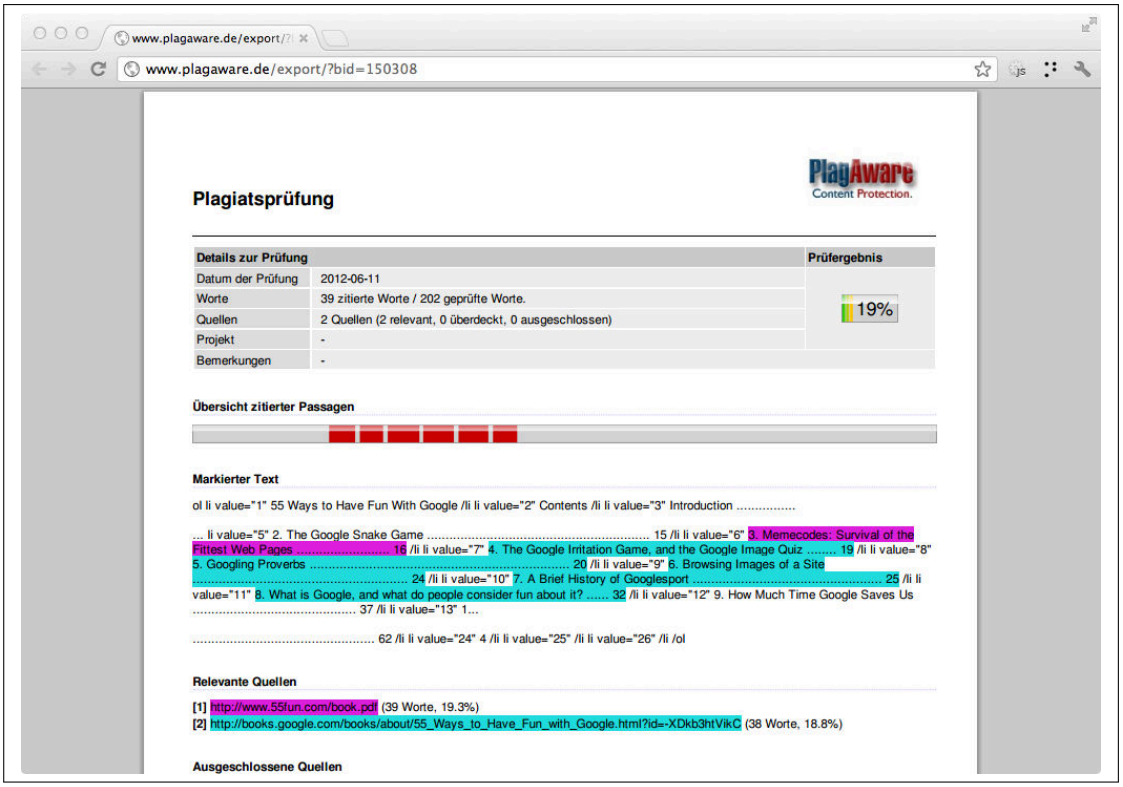


Figure 2.11.: PlagAware result document

The website also offers a webservice which takes text as input and notifies the user when the analyzing is finished. Although it does not provide any information about the sources through the webservice response call, there is only a status code and the percentage of plagiarism responded. Whenever the response has been sent, the user has to go to the PlagAware website and check out the detailed results there.

Since the PlagAware webservice is a simple HTTP-Webservice, the connection is done through an HTTP request with curl.

Listing 2.5: Sending a request through curl to the PlagAware webservice

```
1 public function detect(
      Application_Model_Document_Page_DetectionReport &$report){
2     $url = "http://www.plagaware.de/service/submittext";
3     $fields = array(
4       'UserCode'=>urlencode($this->paUserCode),
5       'ResultUrl'=>urlencode($this->paResultUrl . $report->
            getId()),
6       'TestText'=>urlencode($report->getPage()->getContent()),
7       'DryRun'=>urlencode($this->paDryRun)
8     );
9
10    // url-ify the data for the POST
11    $fields_string = "";
12    foreach($fields as $key=>$value){
13      $fields_string .= $key . '=' . $value . '&';
14    }
15    rtrim($fields_string, '&');
16
17    $ch = curl_init();
18    curl_setopt($ch, CURLOPT_URL, $url);
19    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
20    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
21    curl_setopt($ch, CURLOPT_POST, count($fields));
22    curl_setopt($ch, CURLOPT_POSTFIELDS, $fields_string);
23
24    $output = curl_exec($ch);
25    $info = curl_getinfo($ch);
26    curl_close($ch);
27 }
```

The response is sent through a GET-Request to a pre-defined request URL, in our case '/document/response-plagiarism/report/<report-id>'. The call of this action stores the response of the PlagAware webservice in our database and creates a notifcation for the user, indicating that a new report is available.

2.5. Permission and role management

Unplagged does offer an extensive permission and role management, which controls access to pages and entities within the system. It contains two important parts: roles and permissions. A permission is a actually a single right on a resource within the portal and roles are collections of permissions.

2.5.1. Roles

Roles are being used for assigning pre-defined sets of permissions to a single user in different situations. There exist 4 different types of roles, with different characteristics:

- Global roles
- User roles
- Case default roles
- Case roles

Besides the different role types, there exists a concept of role inheritance in our system. One role can inherit from another one, in this case, a user does have the joined rights of the main role and the inherited role as well.

Global roles

To make this more visual, let's start with the global roles. Global roles include default roles for guests, registered users and admins. A non-registered user does have the guest role by default, a registered user the user role. Since administrative users need some more privilleges, they need the admin role as well. Since the admin role is an inheritable role, it extends the user role and user get's all the rights which are either in the user role or in the admin role.

User roles

When a user registeres on the plattform, a copy of the global user role is created and assigned to the user as it's default role. This role can be modified for each user without influencing other users, so one user can have completely different rights than another one.

Case default roles

All the permissions defined in the roles described above are related to all cases the user is taking part in. Although often it is necessary to add different rights to a user in different cases. For example in one case a user can access all documents, in another one the user can access the files area only. For this case two case default roles are provided: an admin role and a collaborator role, they can be seen as a global role on the case level, used as templates for all cases.

Case roles

When a new case is created, the same process as described for the user roles takes part, a copy of the case default roles is created, they can be assigned to any collaborator of the case (figure 2.12). A case role can be added to a user by accessing the Case > Edit form in the 'case collaborators' section. An autocompletion field let's the user search for collaborators by username and then through a dropdown, the role can be selected, as shown in figure 2.12.

Figure 2.12.: Roles overview

All the different roles used in Unplagged can be managed in the Administration > Roles area (figure 2.13 and figure 2.14).

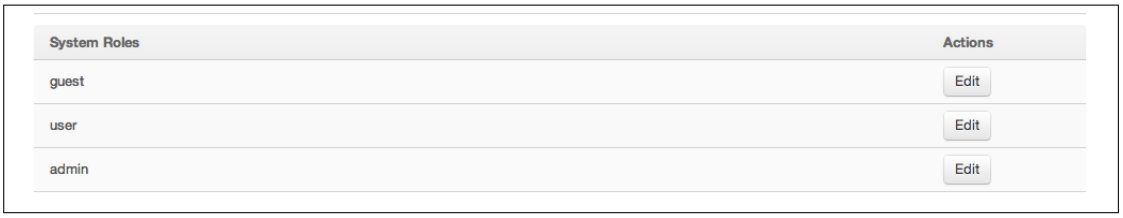


Figure 2.13.: Roles overview

Figure 2.14.: Role form

2.5.2. Permission types

There exist two permission types which are extending from an abstract permission model – page permissions and model permissions. How they differ from each other will be described further on. Usually a permission is related to a specific object, but we also do have global permissions, which define access to all entities of a kind.

A single permission basically contains four properties:

- type – what model / page type is protected (e.g. document, file, case)
- action – what action of the type (e.g. read, list, create)
- base – which entity is protected (e.g. a real entity id or a * for global rights)
- permission type – page permission or model permission

When a user does not have access to a page resource or a model resource, one is redirected to the previously accessed page automatically.

Model permissions

The ModelPermission entity manages access to single objects within the system, this includes for example cases, files, documents, fragments and comments. Each of them currently has a fixed set of permission actions, based on the CRUD design pattern – create, read, update, delete and a fifth one called authorize. The authorize permission allows a user to control the permissions on a specific entity.

If the user does have the authorize permission on an entity, the form to edit them, can accessed through the actions menu in the list view of the object, as shown in figure 2.15.

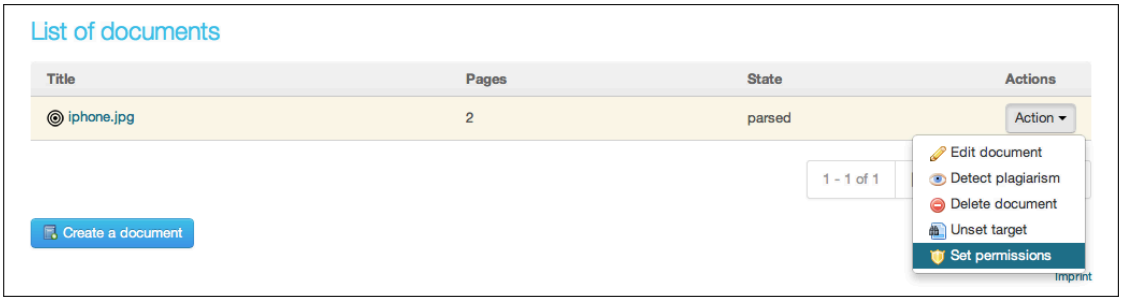


Figure 2.15.: Access form for editing model permissions

The form provided to set the permissions offers an autocompletion textfield at the top, where usernames can be searched and afterwards the permissions can be defined by enabling or disabling the buttons below the username (figure 2.16).

The permission check on models does have to be done by hand in any controller action, where needed. A simple selection of the required permission and then a call of the hasPermission-method on the user role is sufficient. The following example checks the users read permission on a specific document:

Listing 2.6: Checking the user permission on an entity

```
1 $permission = $this->_em->getRepository('
      Application_Model_ModelPermission')->findOneBy(array('type
```

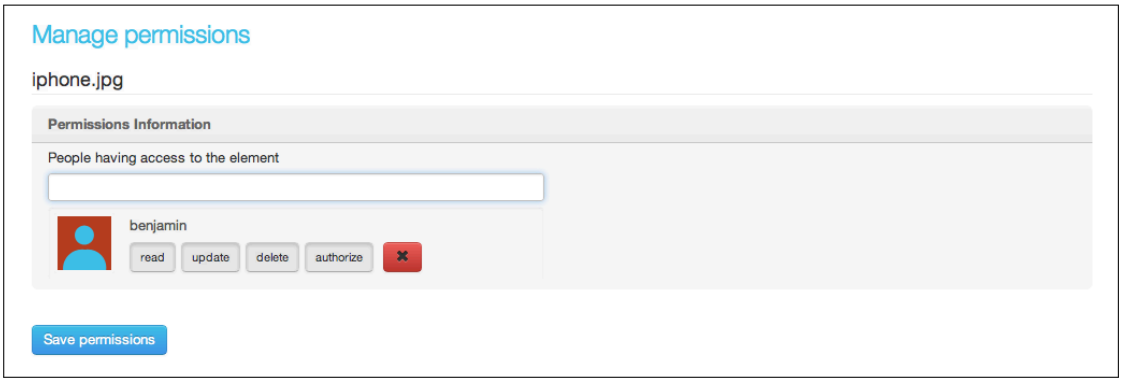


Figure 2.16.: Set permissions on an entity

```
    '=>'document', 'action'=>'read', 'base'=>$document));
2 if(Zend_Registry::getInstance()->user->getRole()->
    hasPermission($permission)){
3   // user has access
4 }
```

Page permissions

The second type of permissions are the page permissions. They control access to controller and action methods. This means, they define whether a user can for example access the page /document/list or not. The available permissions are being generated on each deployment of the site automatically. So when a new action in a controller is created, it will be available as a new permission after the next deployment.

2.6. Barcode

The barcode is a visual representation of the amount of plagiarism in each page of the target document. In our application it is used in two representations, which use the same data source but have another visual layout and details shown within.

Real-time data as source

As the data source we are using real-time data of the document. The data array being generated basically consists of elements with the page number as the key and the percentage of plagiarism as value. That means two resources are included, on the one hand the document pages and on the other hand the fragments on each of these pages.

In the beginning we decided to calculate the percentage of plagiarism for the barcode, whenever the barcode was requested by a user. Although when we had multiple barcodes of large documents on the home page, it turned out to get very slow loading this page. So we limited the data gathering to an update of a fragment or page in the document, the two sources the barcode relies on. This allows us to have the data in real-time, but the calculation of the data does not have to be done on each request and can be cached until the data base changes.

Scalable Vector Graphics (SVG) for representation

Since Unplagged is a web-based project and we are relying on modern techniques as HTML5 and CSS3, we decided to rely on SVG support of the browser as well. This format does have the advantage that it allows building beautiful graphics on the fly that can be changed dynamically and do not require any server resources to be generated, since the generation is being done on the client-side and we can make the graphic dynamically responding to the users browser width. The following code snippet shows the markup for a two bars barcode, each of a 50 percent width and a height of 100 pixels.

Listing 2.7: Barcode SVG markup

```
1 <svg xmlns="http://www.w3.org/2000/svg" version="1.1" style="
      width: 100%; height: 100px;">
2 <rect x="0%" y="0" width="50%" height="100" style="fill
      :#000000;"></rect>
3 <rect x="50%" y="0" width="50%" height="100" style="fill:#
      f80012;"></rect>
4 </svg>
```

The barcode can contain up to 5 different colors, which represent the amount of plagiarism within the page:

- light blue: the page is disabled for the barcode
- white: page not available or no plagiarism
- black: more than 0 percent plagiarized
- dark red: more than 50 percent plagiarized
- light red: more than 75 percent plagiarized

As mentioned above, the barcodes can be ouput in two different representations, these will be described below.

Representation with labels, used on the home page

The first representation can be found at the home page, here are the barcodes of all published cases being displayed. The barcode is shown with page numbers below and the width is dynamic, so the barcode increases when the browser width gets bigger and the barcode gets smaller, when the browser width decreases, automatically.

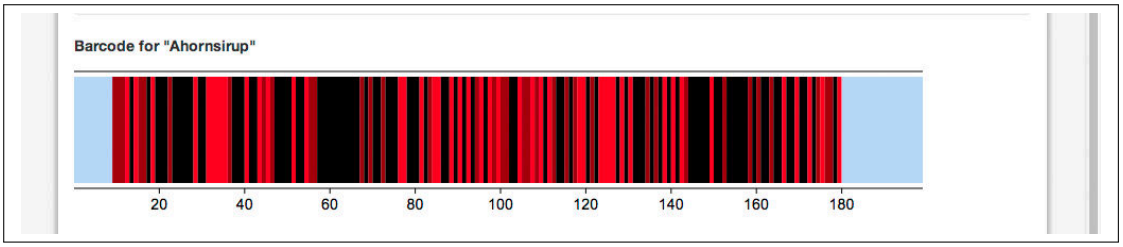


Figure 2.17.: Barcode on home page

Since this mechanism is very dynamic, an algorithm needed to be developed to calculate the x axis values, taking into consideration the page width and the number of pages in the document being visualized. It assumes that a label needs at least 45px of width and the whole barcode has to look well to a minimum width of 500px.

So what we do is, we start with a stepsize of 10 for the page numbers: 10 20 30 40 50, now we check if the amount of pages can be displayed with such a fine scale to stay in the 500px range. For example a page count of 200 leeds to 20 values on the x axis (200

/ 10) and if we assume 45px are necessary to display a value on the axis properly, we would need 900px of width, but we have only 500px available. Otherwise we would have crossovers of the labels when we are getting below 900px. So the stepsize is increased by 10, until we stay in the 500px maximum width. As it turns out, a step size of 20 pages is sufficent to get a width of 450px in the end. The code below shows this calculation.

Listing 2.8: Generating the barcode x axis

```
1 private function generateAxis() {
2         $labelStepsize = 10;
3
4         while (true) {
5             $count = sizeof($this->pages);
6             $labelCount = floor($count / $labelStepsize);
7
8             // we assume a label needs 45px and 500px is the
                  width that needs to be displayable without
                  crossovers
9             if (45 * $labelCount > 500) {
10                $labelStepsize += 10;
11                continue;
12            }
13            break;
14        }
15
16        $label = $labelStepsize;
17        $x = 0;
18        while ($x < ($this->width - ($this->initWidth *
              $labelStepsize))) {
19            $x += ($this->initWidth * $labelStepsize);
20            $this->result .= '<text x="' . $x . $this->
                  widthUnit . '" y="' . $this->y . '" font-family
                  ="Arial" font-size="14" text-anchor="middle">'
                  . $label . '</text>';
21            $this->result .= '<line x1="' . $x . $this->
                  widthUnit . '" y1="' . ($this->y - 20) . '" x2=
                  "' . $x . $this->widthUnit . '" y2="' . ($this
                  ->y - 15) . '" stroke-width="1" stroke="#000000
                  "></line>';
22
```

```
23                 $label += $labelStepsize;
24             }
25         }
```

Representation without labels in the report

The second area where barcodes are used, are the final reports, which will be explained in the next section. Since the space available in the PDF is much less than on the website and the library we are using for generating the PDF report does not support text in scalable vectors graphics, we decided to display the barcodes without labels for now. However the data visualized is exactly the same.

Figure 2.18.: Barcode in the report

2.7. Report

The report is needed to collect and document all the found plagiarized fragments of the reviewed document. It will be used as an evidence proof. The report documents all founded plagiarized parts including the corresponding sources.

The report has also a barcode on the first page to give the reader a quick visuell overview to the plagiarization level of the reviewed document.

Below you can see the generation and the outlook of an example of a report.

2.7.1. Generating of a Report

To generate a report it is neccessary that there exist approved fragments for this case. Without approved fragments in a case it is not possible to generate a report for this case. This is made to have only approved fragments listed in the report. This makes sure that only real plagiarized textparts are listed in the report.

To generate a report the user needs to choose the tab 'Berichte' and than click at the button 'Bericht erstellen'. After the user clicked at the button 'Bericht erstellen' it has the possibility to fill up the report form to give the title of the report and to give some evaluations about the reviewed document.

After the user saved the entries of the form the report will be generated and the report will be listed.

2.7.2. Outlook of a Report

The report has besides his first page, three different parts. The evaluation text, the list of approved fragments and the references to the sources.

The evaluation text is individual text which was given from the person who generated the report through the report form.

After the evaluation text the list of approved fragments will be shown.

At the end all the references to the found textsources is mentioned.

Figure 2.19.: example of a first page of a report

2.8. Simtext

Simtext is a common name about a tool which helps to check for text similarity between two inputs. This tool uses complex algorithm to compare the inputs. The result of the comparison is an output which should contain similar text passages between the two inputs and the positions of the text passages on the documents.

There are many softwares for text similarity tester, but at the beginning of the development for simtext in our workbench, we used the software SIM of Dick Grune. This software is free and the result after the comparison process is reliable. We integrated this software to Unplagged but there were some disadvantages about this software. Then we saw that the Vroniplag community had developed a simtext tool, which was also based on SIM, but cleverly modified in Javascript. This version was quite simplified and optimized for

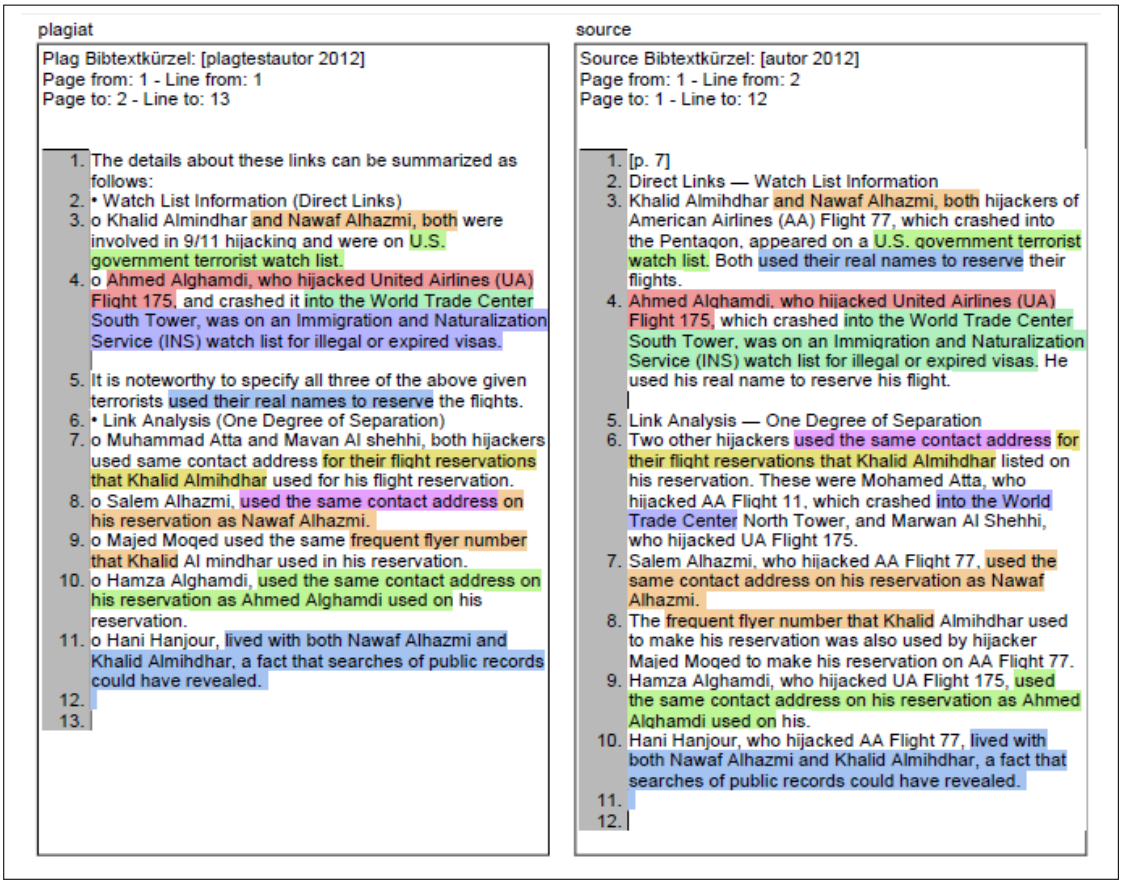


Figure 2.20.: example of a listed approved fragment

web browser. We would like to adapt this Javascript version in our workbench as well, but unfortunately it was not that easy to integrate it. The reason was that some of its functions were not suitable in our environment at all. Therefore we decided to modify this Javascript version in PHP.

Here are some descriptions about the development and improvement of simtext tool for our workbench.

2.8.1. SIM in C - Original version

The program was developed by Dick Grune. It was written in C and runnable in other languages like Java, Pascal... and had a quite complicated structure.

"The general outline of the similarity checker is as follows:

1. the files are read in (pass 1)
2. a forward-reference table is prepared
3. the set of interesting runs is determined
4. the line numbers of the runs are determined (pass 2)
5. the contents of the runs are printed in order (pass 3)"

The result of the comparison process was a plain text file which showed the similar text passages and the positions, where the plagiarized text was similar with the source text.

We integrated this software into our workbench, but because it was not specifically developed for web, there were some disadvantages of this software. They are:

- it was complicated to run the program - It had to be compiled through the C command
- input/output were plain text files
- there was no color for marking the similar text passages
- only similar text passages were shown, not the whole documents
- the result was simple and monotone and not impressive

- In order to compare HTML texts or other file types, it was maybe needed to parse the inputs in plain text files. After comparing, the result as plain text file should be parsed again in HTML so that it can be performed in the browser. Therefore the comparison process was complicated and the speed could be decreased if there were a lot of documents.

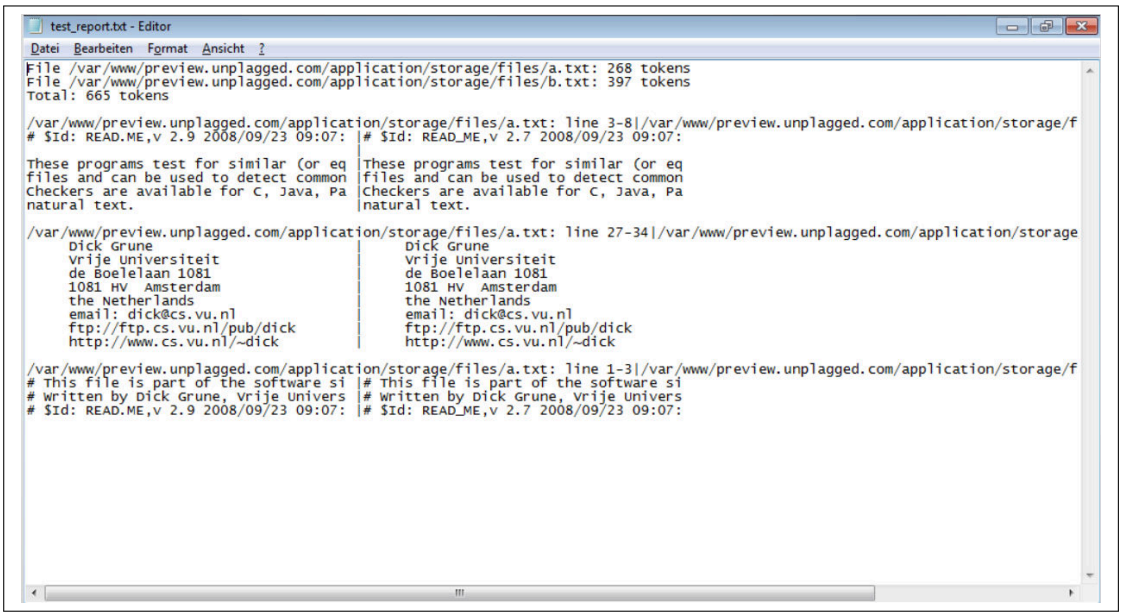


Figure 2.21.: Simtext in C

2.8.2. Vroniplag's Javascript version

We found out that Vroniplag also had its own text comparison tool, which was written in JavaScript. According to the author, this web-based version was also based on the original version of Dick Grune.

The Javascript version could compare two fragments, finds out the similar text passages and marks them with different colors. A text passage is defined by default as a phrase which is more than 4 tokens (words) long.

We thought this version was more flexible. Besides the text comparison function, it was also possible to mark the similar text passages with different colors. This could help the

user get a better overview about the result. Because it was written in Javascript, it could also read the HTML texts and compare them directly without the parsing process as in the SIM version above. In addition, there was a button which could be switched on/off to show/hide the colors.

But when we tried to adapt this version to our workbench, there were some difficulties that we thought the adaptation was too complicated. The difficulties were:

- Because Vroniplag is a Wiki-Community, so the version was written and adjusted to this Wiki's environment. That means, the author has customized the version so that it fit to the Wiki page's attributes. There were a lot of the author's improvements and changes of relative functions only for the purpose that it could work in Wiki. Therefore it was not easy to adapt the version again in our workbench, because we did not have the suitable conditions as by Wiki

- If the user does not have/want Javascript turned-on on the browser, it would be difficult to see the colored result of the text comparison tool

- We wanted to make our own thing

2.8.3. Unplagged's PHP version

After considering advantages and disadvantages of the two versions, we decided to modify the Javascript version in PHP language. Because of the disadvantage of the Wiki's environment, we took only the main part of the Javascript version - the text compare function - and tried to modify it to our workbench.

Text comparison function

We tried to figure out how the text comparison function works. First we just translated every code line from Javascript into PHP. It was not difficult to rewrite the code, but it took much time and effort to find and fix the errors. We had to debug most of the code

Figure 2.22.: Vroniplag Colored Fragments

lines. In some cases we found out that it did not work because some issues seem to be the same thing in both languages, but actually if they were written in PHP, they had a different meaning than if they were written in Javascript.

For example we had to check, if a key named *match_tag* existed in a hash table named *match_table*. If yes, the value for this key will be added. In the JavaScript version, they used *if (match_tag in match_table)*, where it did not matter if *match_tag* was a key or a value. In PHP version, the equivalent function *if(in_array($match_tag, $match_table))*, was used. We thought it would be the same thing like in Javascript, but the result we received was always an empty array. After several times debugging we realized that it was about an associative array in PHP and this function searched only for the value. We had to check for the key and *if(array_key_exists($match_tag, $match_table))* had worked finally.

Another example for the diversity of the same issues in different languages was the

different meaning of NULL. In Javascript NULL means an object with no initial value, in PHP it means 0, empty or false (3). So there were some places in the code declared with NULL, it should be changed to -1 in PHP in order to receive the expected result.

After fixing all the errors, we got a nice text comparison function in PHP. The final result is an array of token lists in the form:

Listing 2.9: Cropping an image to a square aspect ratio

```
1 array{doc_1, start_token_1, doc_2, start_token_2, length}
```

where doc_X are indices into the documents array, and start_token_X are indices into the respective token list of the documents, length is the default length of a phrase.

For example we had two dummy inputs

```
$doc1 = "test1 test2 test3 test4 test5 test6 test7 test8 test9 test10";
$doc2 = "test1 test2 test3 test4 test5a test6 test7 test8 test9";
```

Figure 2.23.: simtext result

Here was the matches

```
                doc1   test1   doc2   test1   4
array   {       0      0       1      0       4       }
                doc1   test6   doc2   test6   4
array   {       0      5       1      5       4       }
```

Figure 2.24.: simtext result

The result was presented on browser

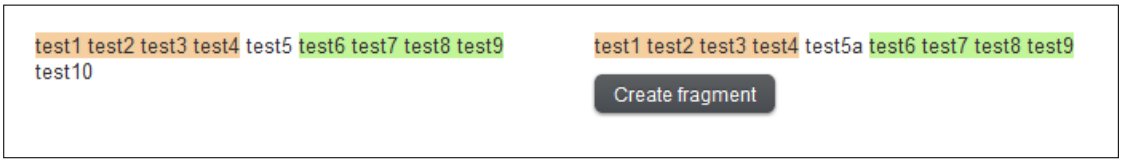


Figure 2.25.: simtext result

Disadvantages of the text comparison function

The first disadvantage of the text comparison function was maybe the restriction of the length of a phrase. By default a phrase was a list of tokens which was more than 4 tokens. That means it could be difficult to find out plagiarized positions if their length was less than 4 tokens.

Another disadvantage was that the strict comparison algorithm could compare only continual phrases. That means if at some positions there was a tiny change such as a break line or a comma, the algorithm could miss them.

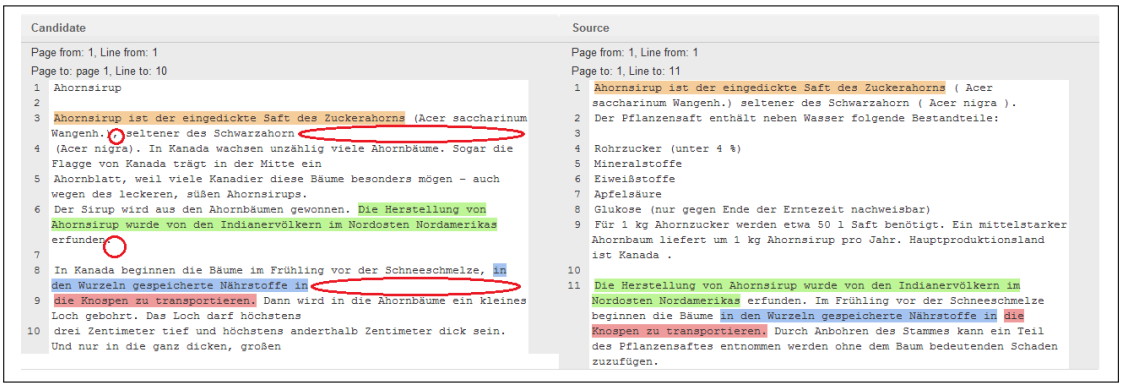


Figure 2.26.: a strict comparison algorithm

Here is the source code of compare text function

Listing 2.10: Cropping an image to a square aspect ratio

```
1 private static function compareText($documents,
      $min_run_length){
2      $documents_len = array();
3
4      $final_match_list = array();
```

```
5     $match_table = array();
6     $docs = sizeof($documents);
7
8     for($doc_idx = 0; $doc_idx < $docs; $doc_idx++){
9       $doc = $documents[$doc_idx];
10      $tokens = sizeof($doc) - $min_run_length + 1;
11      $doc_len = sizeof($doc);
12      $documents_len[$doc_idx] = $doc_len;
13
14      // If the word is smaller than the min_run_length, do
            not analyse it
15      if($tokens <= 0){
16        continue;
17      }
18
19      $min_token_idx = 0;
20
21      for($token_idx = 0; $token_idx < $tokens; $token_idx++){
22        $match = array_slice($doc, $token_idx, $min_run_length
              );
23        $match_loc = array($doc_idx, $token_idx);
24        $match_tag = implode(" ", $match);
25
26        if(array_key_exists($match_tag, $match_table)){
27          if($token_idx >= $min_token_idx){
28            $best_match = array($doc_idx, $token_idx, -1, 0,
                  0);
29            $matches = $match_table[$match_tag];
30            $nr_matches = sizeof($matches);
31            for($idx = 0; $idx < $nr_matches; $idx++){
32              $match_peer = $matches[$idx];
33              $peer_doc_idx = $match_peer[0];
34              $peer_doc = $documents[$peer_doc_idx];
35              $peer_token_idx = $match_peer[1] +
                    $min_run_length;
36              $peer_len = $documents_len[$peer_doc_idx];
37              $our_token_idx = $token_idx + $min_run_length;
38
39              if($peer_doc_idx == $doc_idx){
```

```
40                continue;
41              }
42
43              while($peer_token_idx < $peer_len
44              && $our_token_idx < $doc_len
45              && $peer_doc[$peer_token_idx] == $doc[
                    $our_token_idx]){
46                $peer_token_idx++;
47                $our_token_idx++;
48              }
49
50              $len = $our_token_idx - $token_idx;
51              if($len > $best_match[4]){
52                $best_match[2] = $match_peer[0];
53                $best_match[3] = $match_peer[1];
54                $best_match[4] = $len;
55              }
56            }
57
58            if($best_match[2] != -1){
59              array_push($final_match_list, $best_match);
60              $min_token_idx = $token_idx + $best_match[4];
61            }
62          }
63          array_push($match_table[$match_tag], $match_loc);
64        }else{
65          $match_table[$match_tag] = array($match_loc);
66        }
67      }
68    }
69    return $final_match_list;
70  }
```

Applying the color to the matched texts

For marking the color to the matched text passages in plagiarized and source documents, we used the HTML--tag to define in which text passage the color will be applied. The -tag had a class-attribute named fragmark-x with x from 1-9 performing one of nine different colors. The CSS color attributes will applied to the text passage corresponding to the class name.

This is the function to apply color to the text passage.

Listing 2.11: Cropping an image to a square aspect ratio

```
1 private static function getMarkedTexts($comparedTexts,
      $wordsLeft, $wordsRight, &$listLeft, &$listRight){
2     // Colors have css classes "fragmark1" to "fragmark9"
3     $col = 0;
4     $nr_col = 9;
5
6     for($i = 0; $i < sizeof($comparedTexts); $i++){
7       $res = $comparedTexts[$i];
8       $wordsLeft[$res[3]] = "<span class=\"fragmark-$col\">" .
            $wordsLeft[$res[3]];
9       $wordsLeft[$res[3] + $res[4] - 1] = $wordsLeft[$res[3] +
            $res[4] - 1] . "</span>";
10
11      $wordsRight[$res[1]] = "<span class=\"fragmark-$col\">"
            . $wordsRight[$res[1]];
12      $wordsRight[$res[1] + $res[4] - 1] = $wordsRight[$res[1]
            + $res[4] - 1] . "</span>";
13
14      $col = ($col + 1) % $nr_col;
15    }
16
17    // At this point the text is exploded word by word, now
        the line breaks have to be added again, as in the
        original document
18    $resultLeft = Unplagged_CompareText::addLinebreaks(
        $listLeft, $wordsLeft);
19    $resultRight = Unplagged_CompareText::addLinebreaks(
```

```
            $listRight, $wordsRight);
20
21      return array("left"=>$resultLeft, "right"=>$resultRight);
22    }
```

And the CSS attributes

Listing 2.12: Cropping an image to a square aspect ratio

```
1 /*colours for the simtext operation (compare function) to mark
      the same texts*/
2 .fragmark-0 { background-color: #f5cf9f; }
3 .fragmark-1 { background-color: #c2f598; }
4 .fragmark-2 { background-color: #a7c6f2; }
5 .fragmark-3 { background-color: #f29f9f; }
6 .fragmark-4 { background-color: #aff2be; }
7 .fragmark-5 { background-color: #e8a3ff; }
8 .fragmark-6 { background-color: #e6e181; }
9 .fragmark-7 { background-color: #b8b8ff; }
10 .fragmark-8 { background-color: #f5cf9f; }
11 .fragmark-9 { background-color: #a5e6ed; }
```

2.8.4. Uses of text comparison function in Unplagged

The function text comparison was an important feature which was used much in many different sections in our workbench.

Fragment modify form

In fragment modify form the user could create a new fragment. In addition he could see the compared text passage with colors directly while adjusting the form as well. This feature was enabled by using an ajax function combining with an onchange-event in Javascript.

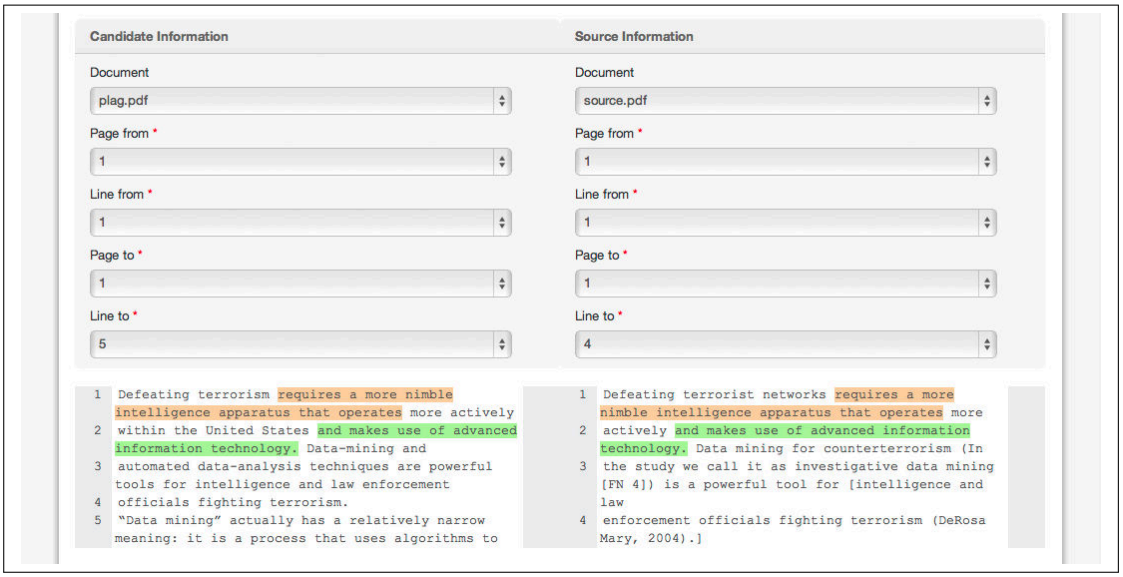


Figure 2.27.: Text comparison function in fragment modify form

Here is the code in Jquery

Listing 2.13: Cropping an image to a square aspect ratio

```
1 // executes a simtext comparison in fragment modify form
2   function compareTexts() {
3     if($("#candidateLineFrom").length != 0) {
4       $.post("/document_page/compare", {
5         candidateLineFrom: $("#candidateLineFrom").val(),
6         candidateLineTo: $("#candidateLineTo").val(),
7         sourceLineFrom: $("#sourceLineFrom").val(),
8         sourceLineTo: $("#sourceLineTo").val(),
9         highlight: true
10      }, function(response) {
11        if($('#candidateText').length == 0) {
12          $('#fieldset-candidateGroup').append('<div id="
                candidateText" class="src-wrapper"/>');
13          $('#fieldset-sourceGroup').append('<div id="
                sourceText" class="src-wrapper"/>');
14        }
15        $('#candidateText').html(response.data.plag);
16        $('#sourceText').html(response.data.source);
17      }, "json");
```

```
18      }
19      return false;
20    }
21    $("#candidateLineFrom, #candidateLineTo, #sourceLineFrom, #
          sourceLineTo").change(function(){
22      compareTexts();
23    });
24    compareTexts();
```

Fragment show view

After a new fragment was created, the user could see it in details by clicking on the link of the fragment. The fragment view page shows on the left side the plagiarized text, and on the right side the source. There was also a button to hide/show the colors. If the user hit the button ”Hide color” the text would be shown with no colors.

There might be a small improvement in comparison with Vroniplag, that the page and the lines of the fragment would be shown here not manually, but automatized.

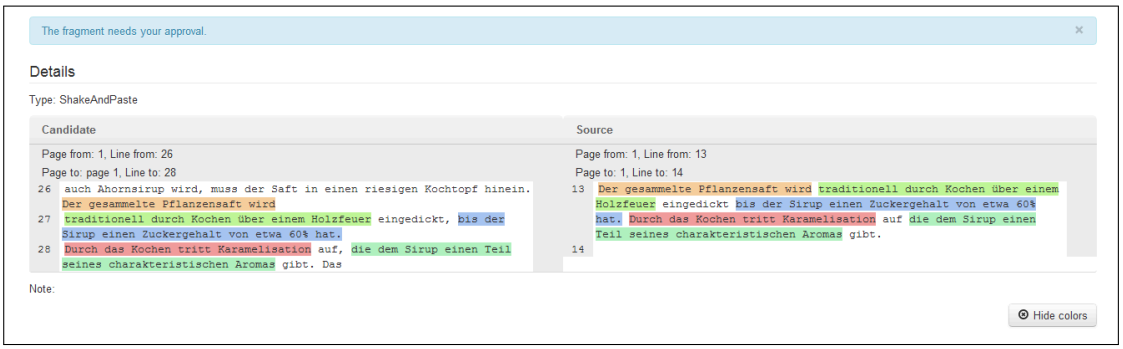


Figure 2.28.: Text comparison function in fragment show view with colorsr

Compare a document with all possible sources from database

The text compare function could be used to compare a document with many other documents from Unplagged’s database. This process could take a lot of time because it

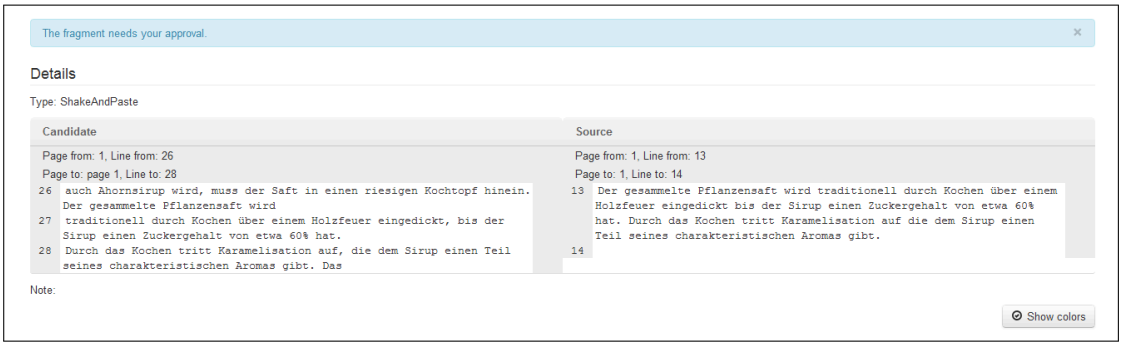


Figure 2.29.: Text comparison function in fragment show view with no colorsr

depended on the document's size and the number of the sources which the document would be compared with. We used a cron job to control when the process was finished. After that the created report could be opened.

The report was an HTML text and could be seen directly on the browser.

Figure 2.30.: Simtext of many documents

2.9. Bibtex Metadata

Bibtex function was another feature in our workbench. Based on VroniPlag, Bibtex was created to provide the user an overview of the document's bibliographic information.

The user had the possibility to add the bibtex information of a document through the document modify form when he created a new document.

In the document modify form there was an extra section for bibtex information. In this section, the bibtex modify form was first shown in its full form. There were over 20 fields in it. All of fields were taken from VroniPlag. The fields *Author* and *Title* must be filled, the field *Year* could be filled as *o.J (ohne Jahrgang)* if it was unknown.

If the user wanted to specify a type for his document, there were also 3 separated subforms available:

- book
- periodical
- essay

After saving bibtex information, the user could see it again by the tab Bibliography in Unplagged. By Bibliography, there was a list of all the documents with their bibtex information are shown. If the user would like to change something, he could choose the action button to come back to the document modify form.

If the user clicked on a bibtex link, he would see the details of the bibtex information of the document. More especially, to give the user an overview, that which fragments had belonged to this bibtex, a list of those fragments with the comparison and colors would also be shown here.

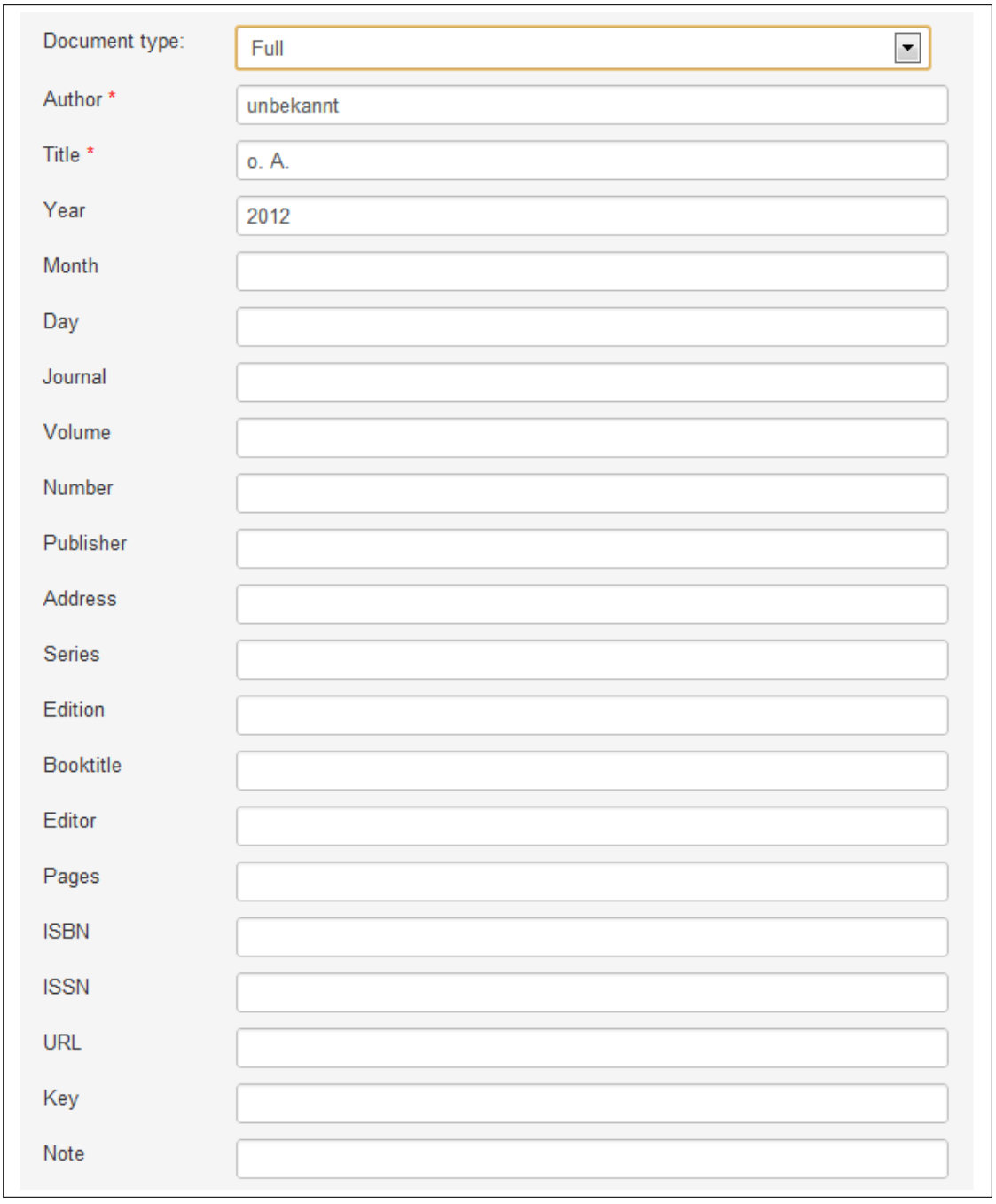


Figure 2.31.: Bibtex full form

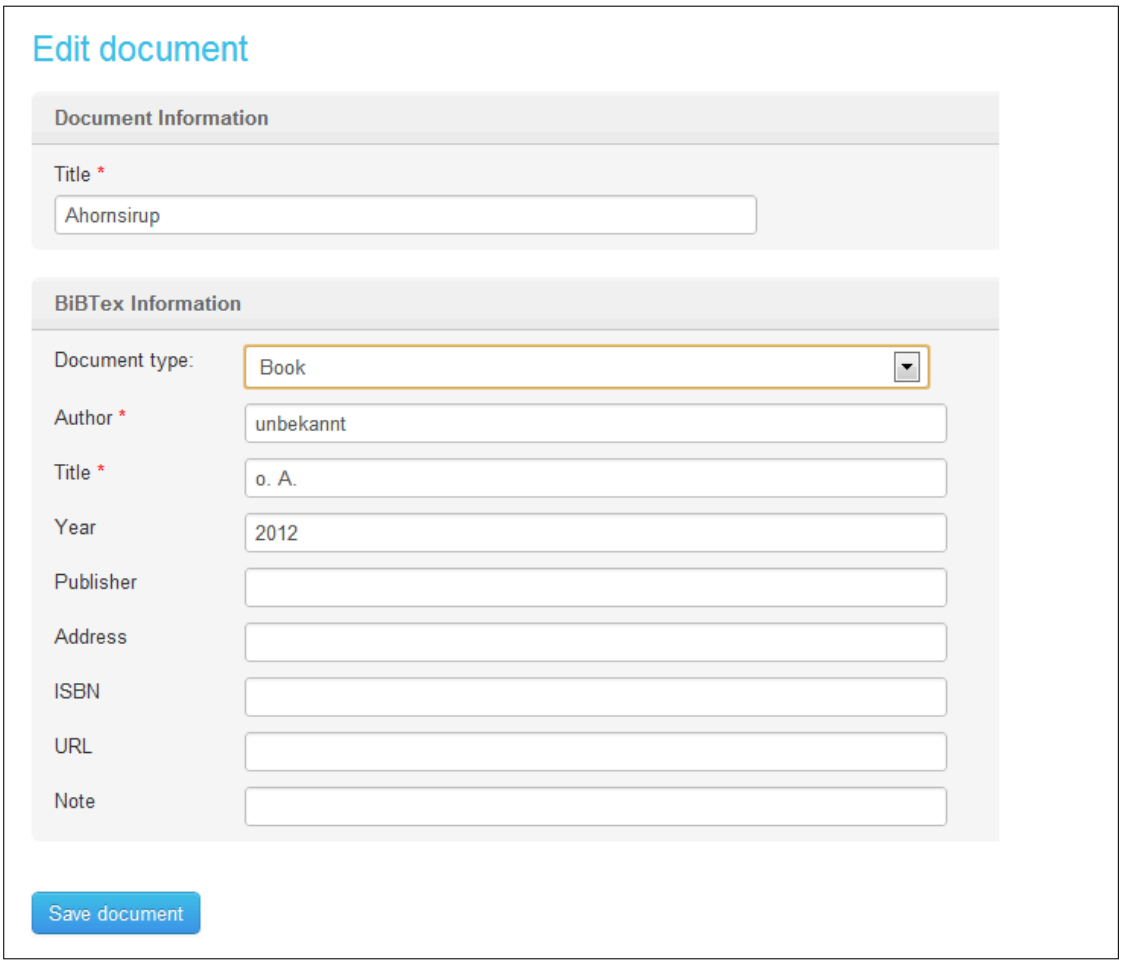


Figure 2.32.: Bibtex book form

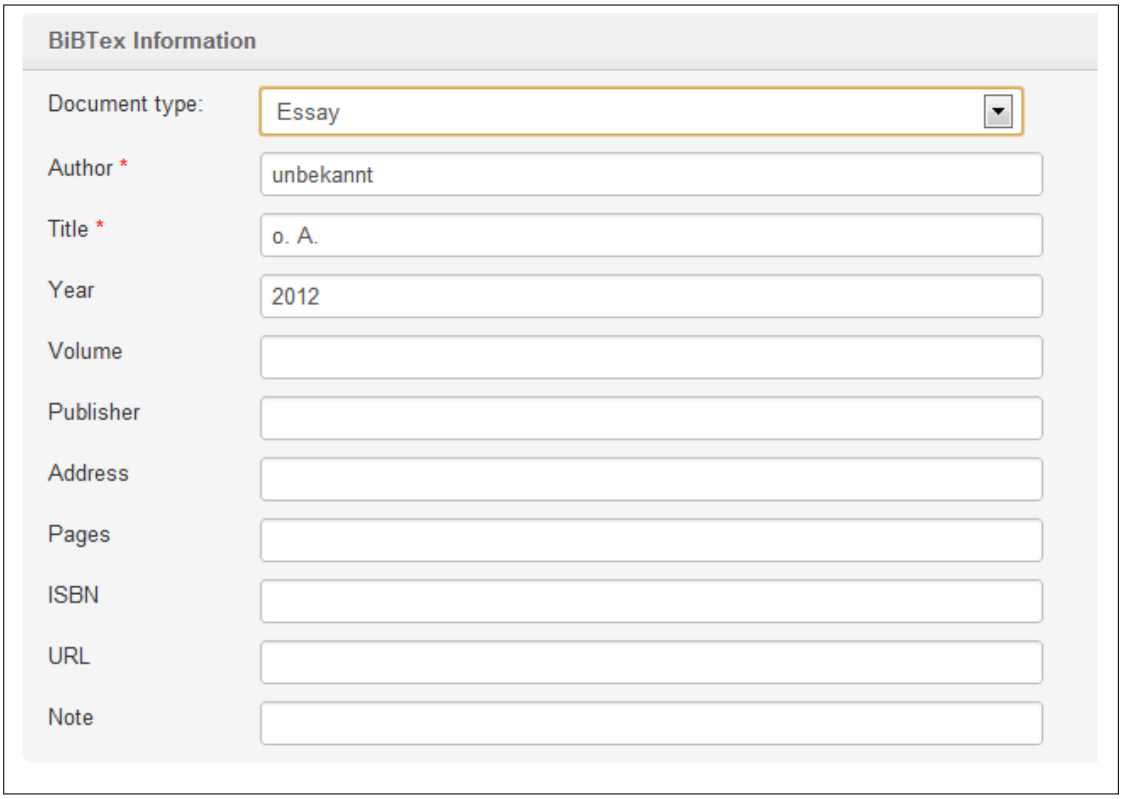


Figure 2.33.: Bibtex essay form

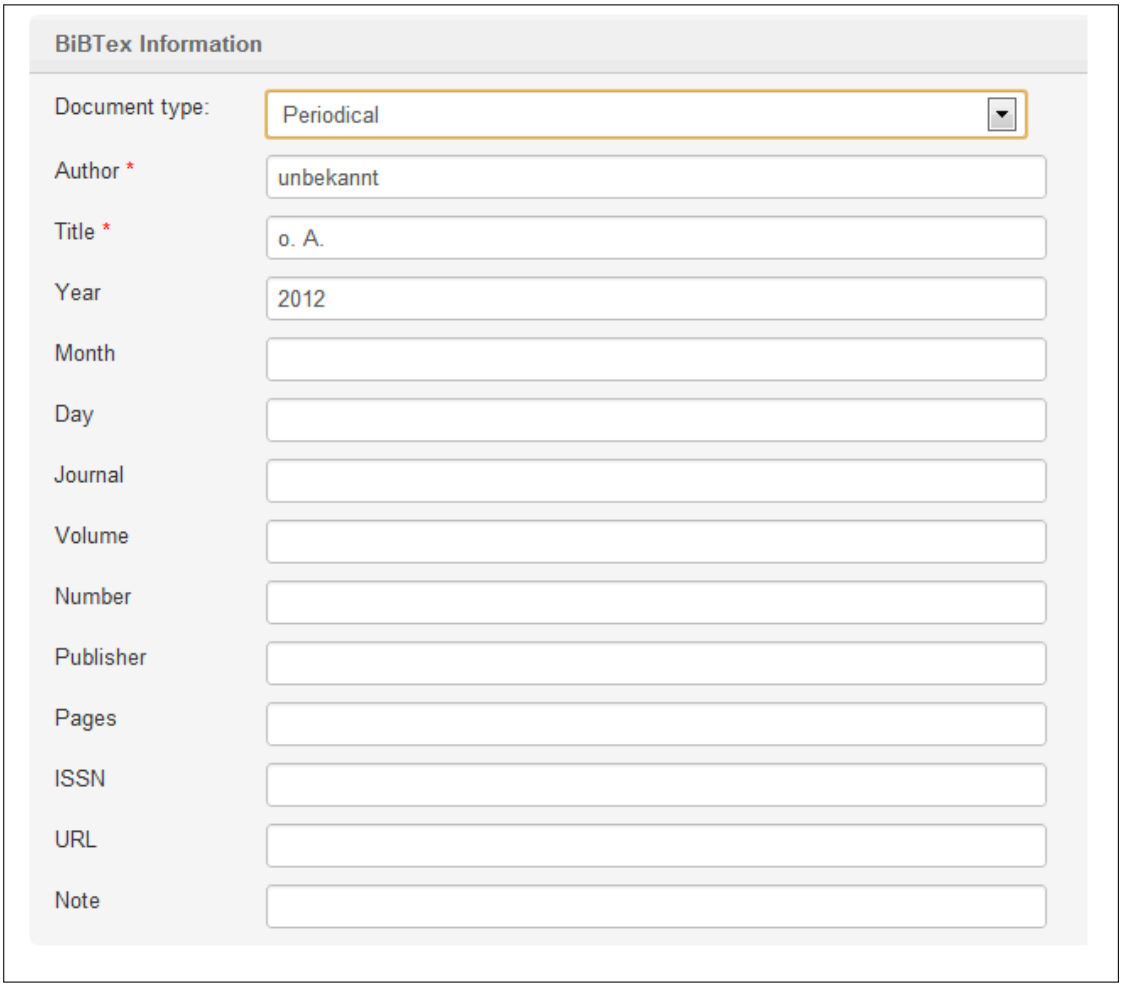


Figure 2.34.: Bibtex periodical form

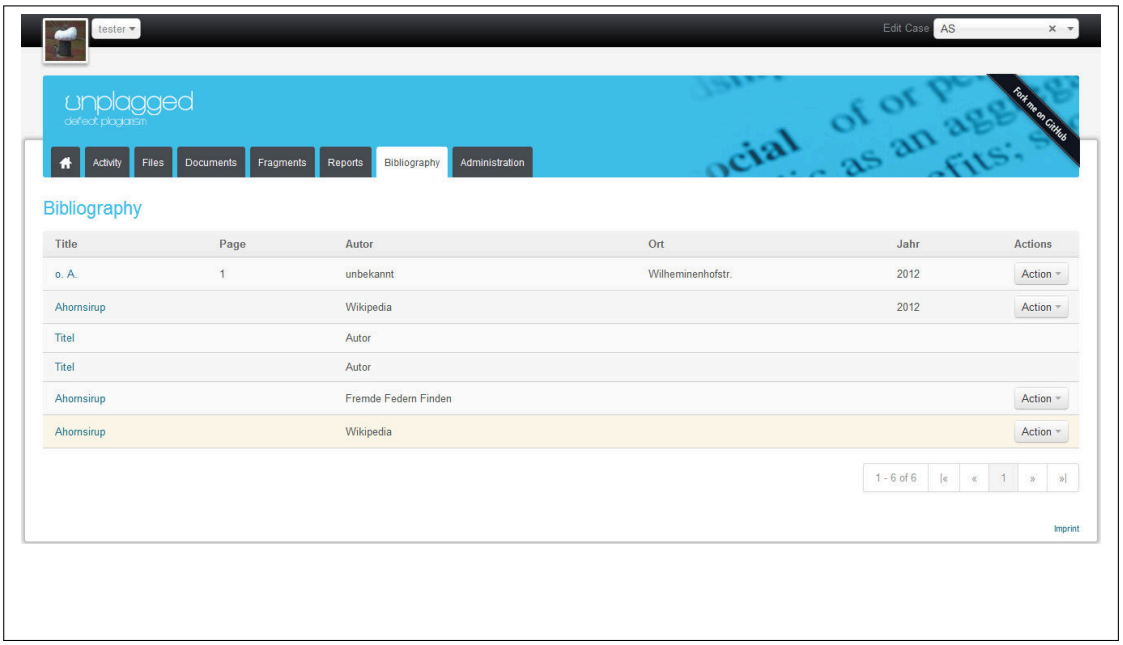


Figure 2.35.: Bibliography list

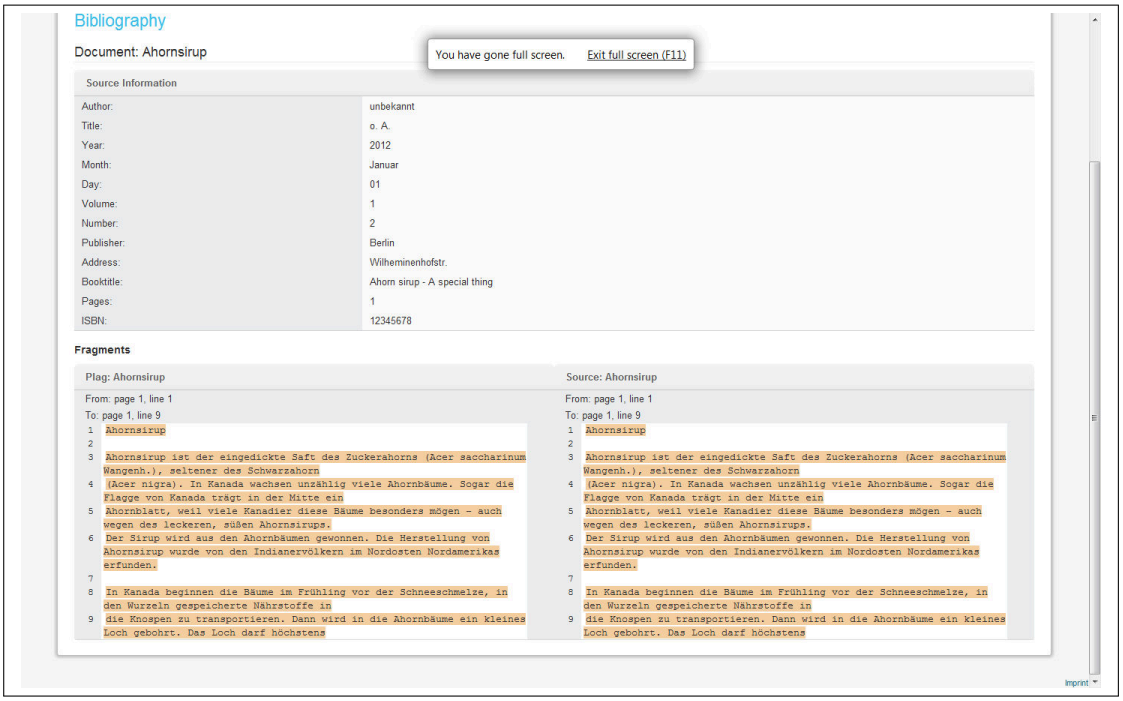


Figure 2.36.: view of bibtex information with fragments belong to it

3. Showtime

The showtime is the final part of the master project where all groups are presenting their projects and their results. Before the showtime was beginning a lot of planning was necessary. From the actual presentation and equipment to the brochures and posters. Everything needed to be planned and organized.

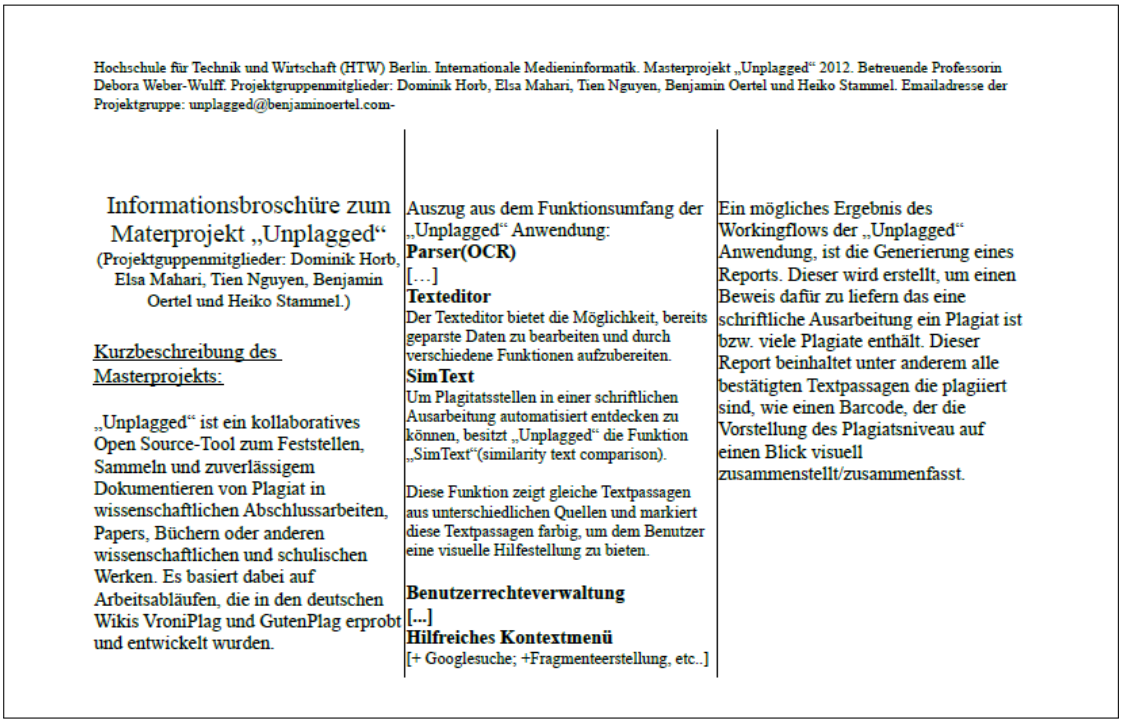
3.1. Paperwork

At the beginning it was necessary to make a first concept of the brochure and posters for the unplagged exhibition stand.

3.1.1. Brochure

The first concept of the brochure was kind of a sketch.

Below is the final outlook of the brochure.



Hochschule für Technik und Wirtschaft (HTW) Berlin. Internationale Medieninformatik. Masterprojekt „Unplagged“ 2012. Betreuende Professorin Debora Weber-Wulff. Projektgruppenmitglieder: Dominik Horb, Elsa Mahari, Tien Nguyen, Benjamin Oertel und Heiko Stammel. Emailadresse der Projektgruppe: unplagged@benjaminoertel.com-

Informationsbroschüre zum Materprojekt „Unplagged“
(Projektguppenmitglieder: Dominik Horb, Elsa Mahari, Tien Nguyen, Benjamin Oertel und Heiko Stammel.)

Kurzbeschreibung des Masterprojekts:

„Unplagged“ ist ein kollaboratives Open Source-Tool zum Feststellen, Sammeln und zuverlässigem Dokumentieren von Plagiat in wissenschaftlichen Abschlussarbeiten, Papers, Büchern oder anderen wissenschaftlichen und schulischen Werken. Es basiert dabei auf Arbeitsabläufen, die in den deutschen Wikis VroniPlag und GutenPlag erprobt und entwickelt wurden.

Auszug aus dem Funktionsumfang der „Unplagged“ Anwendung:

Parser(OCR)

[...]

Texteditor

Der Texteditor bietet die Möglichkeit, bereits geparste Daten zu bearbeiten und durch verschiedene Funktionen aufzubereiten.

SimText

Um Plagitatsstellen in einer schriftlichen Ausarbeitung automatisiert entdecken zu können, besitzt „Unplagged“ die Funktion „SimText“(similarity text comparison).

Diese Funktion zeigt gleiche Textpassagen aus unterschiedlichen Quellen und markiert diese Textpassagen farbig, um dem Benutzer eine visuelle Hilfestellung zu bieten.

Benutzerrechteverwaltung

[...]

Hilfreiches Kontextmenü

[+ Googlesuche; +Fragmenteerstellung, etc..]

Ein mögliches Ergebnis des Workingflows der „Unplagged“ Anwendung, ist die Generierung eines Reports. Dieser wird erstellt, um einen Beweis dafür zu liefern das eine schriftliche Ausarbeitung ein Plagiat ist bzw. viele Plagiate enthält. Dieser Report beinhaltet unter anderem alle bestätigten Textpassagen die plagiiert sind, wie einen Barcode, der die Vorstellung des Plagiatsniveau auf einen Blick visuell zusammenstellt/zusammenfasst.

Figure 3.1.: the first concept of the brochure

3.1.2. Posters

Title Poster

The title poster shows the name of the unplagged master project.

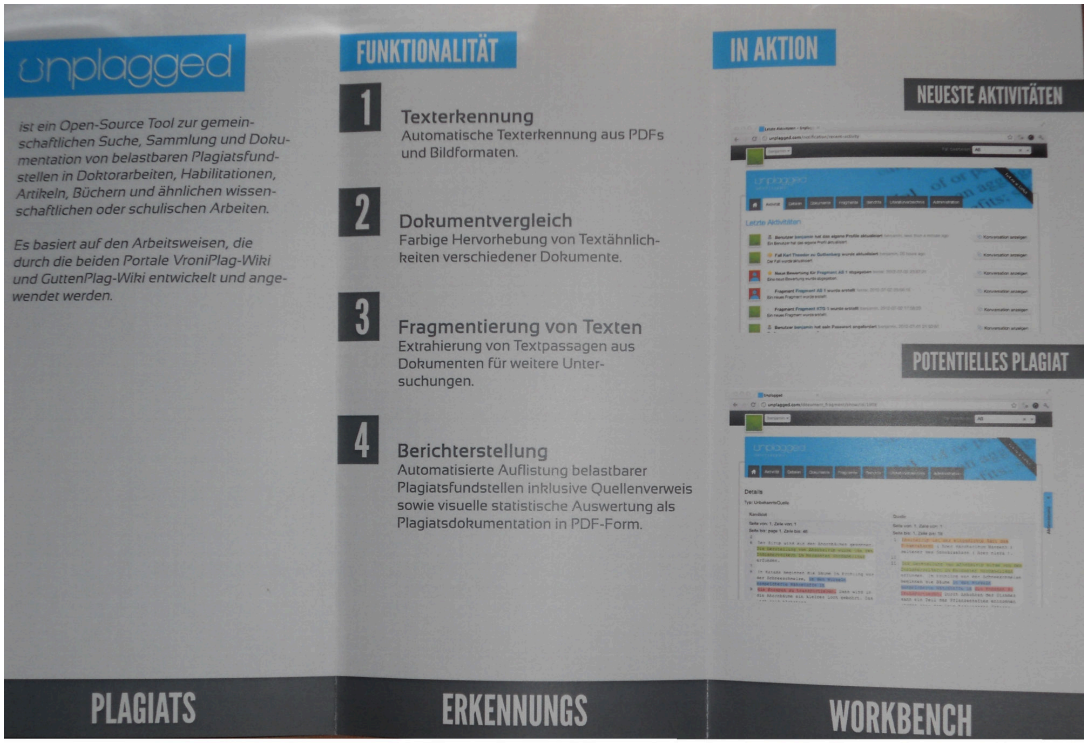


Figure 3.2.: the frontside of the final unplagged brochure

Describing Poster

Here it is the outlook of the poster which has short describings about the project.

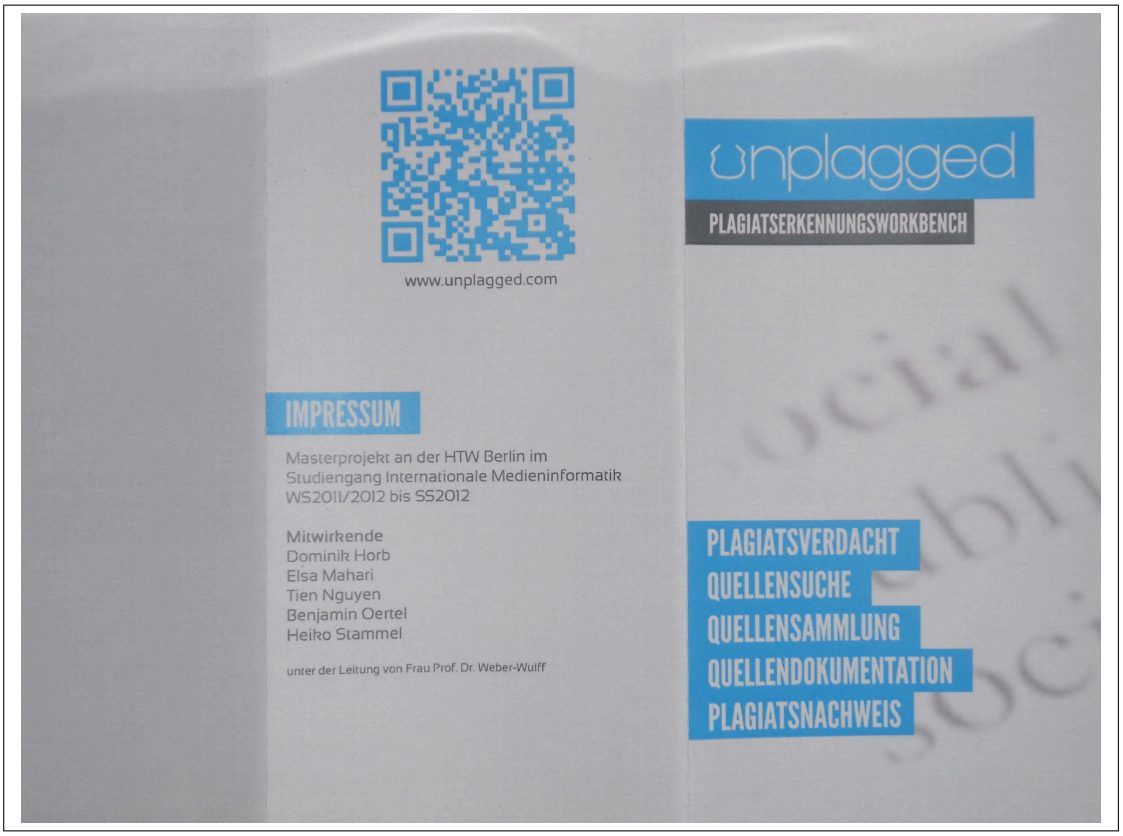


Figure 3.3.: the backside of the final unplagged brochure

Technical Poster

Below is a cutout of the technical poster.

Figure 3.4.: the title poster

3.2. Equipment

Below there is the list of the equipment which was used for the unplagged exhibition stand.

- 3 iMacs
- 3 tables
- 2 higher tables
- 3 pinboards
- 2 floodlights

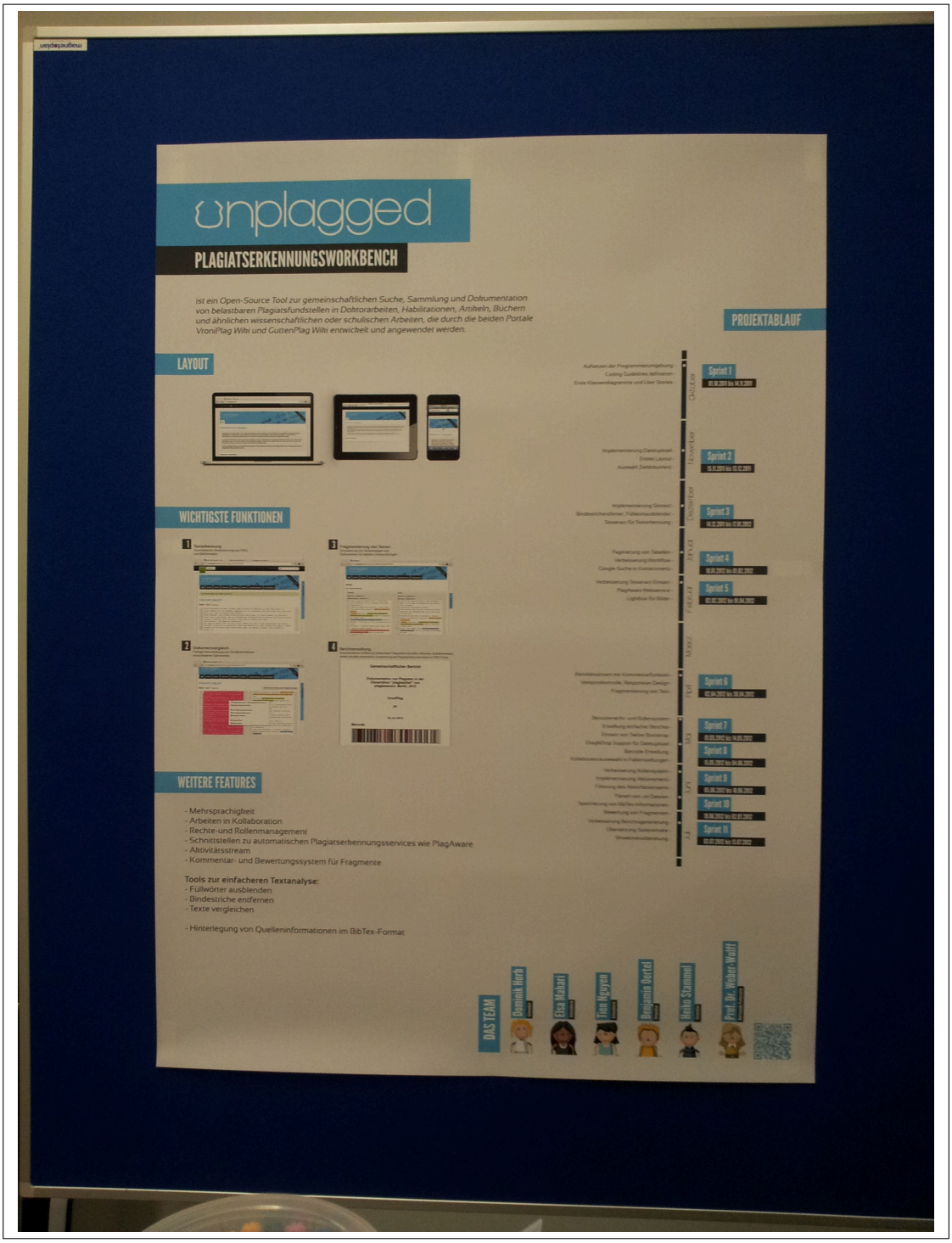


Figure 3.5.: the describing poster

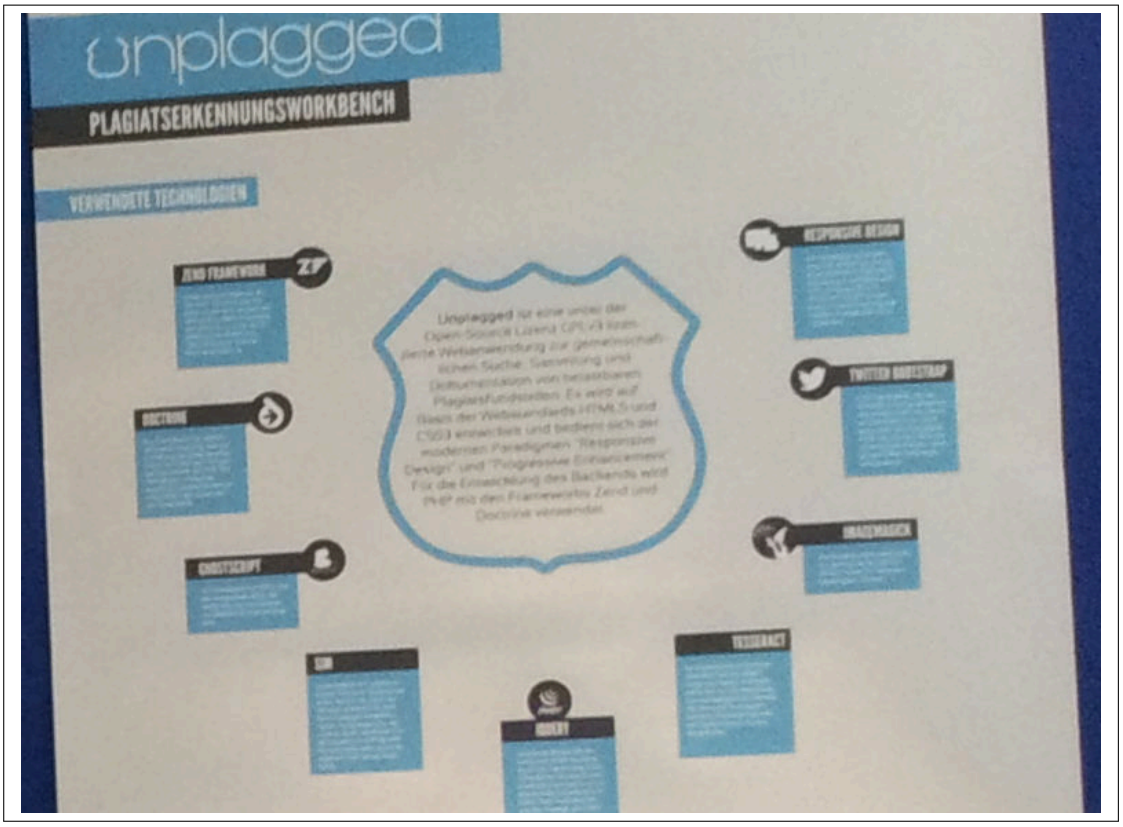


Figure 3.6.: the technical poster

3.3. Outlook of the exhibition stand

Below there are some snapshots of the unplagged exhibition stand.

Figure 3.7.: unplagged exhibition stand

Figure 3.8.: unplagged exhibition stand

Figure 3.9.: unplagged exhibition stand

Figure 3.10.: unplagged exhibition stand

4. Project Analysis and Outlook

To actually learn and take away as much as possible after nine month of development on such a big project like Unplagged we believe that thoroughly analyzing the course of the project and most importantly the mistakes is really important. The problem with this is always that there are some kind of "politics" involved, because we as a team and also every team member for themselves have an agenda when doing this. We are not graded yet, so we still want to shine with our achievements and not everybody in the team will feel the same about every topic that will or could be mentioned here. So this just as a disclaimer beforehand.

We will nonetheless try to do our best in constructively critizing ourselves and the provided environment and look at what worked well in this chapter. Additonally we will give an outlook on possible improvements and further developments of the project.

4.1. Project Analysis

What was critized after the showtime by the professors was, that the overall progress of the software that was developed, wasn't quite enough for a group of five master students working nine month on it. We sadly have to admit, that this is at least kind of true, but we still can not concur fully with this assessment.

What we feel is that – as was already said in the introduction – we laid a nice foundation for an open source project with Unplagged and additionally learned a lot, which is something that probably can get overlooked easily, when simply looking at the end result. We understand, that it is difficult to grade those "soft factors", but the goal for us as

master students is not necessarily always a product that is just looking nice on the outside, but also one that uses “best-practices” on the inside. The difference between a well salted and encrypted password to one that is stored in plain text is nothing really visible, but something that makes a difference of a few hours in development after all.

4.1.1. Using a Wiki

One of the things that we were asked to do at the beginning was to maintain a Wiki, which we created inside our project management tool Redmine. Looking at the state of it now, we have to say, that this didn't work very well.

We currently have 62 pages in there, with probably one third of them coming from meeting minutes and the rest simply being not very well written or up to date and therfore not useful, with just very few exceptions.

Starting it worked OK, but the information we put on there wasn't valid long enough in most instances, because our perception of the topics changed very fast in the beginning, which means the motivation and effort needed to keep the information relevant didn't match the usefulness. It was simply easier and even necessary after a while to ask the person that wrote the article or was currently working on that area of the system about it, because by the article you couldn't really tell if the information was still holding true. As we were just five people and mostly knew what the others worked on this was the path of least resistance and was automatically adopted pretty quickly.

It also was problematic that we had just five editors for other reasons. As we could have put an endless amout of articles on the wiki, it often wasn't the case that written articles would be corrected by someone else, but more that they stayed the way they were written, because the others were hopefully busy writing something about another topic, but maybe didn't even notice that there was something new. This lead to some kind of spiral of fewer and fewer edits, because when it didn't look like something was happening, the enticement of making edits was reduced further and further.

To make it work we believe that at least some good experiences of the eventual editors with a functioning wiki would be helpful, because with a group of just five people,

everybody has to be dedicated to keep it running. Maybe making the decision to use a wiki more optional would also help, because it's simply harder to work on something, when the usefulness wasn't discovered but the necessity got kind of "enforced" from outside of the team.

4.1.2. Agile development

All in all the agile process we figured out after a while worked pretty good and is something that everyone of us feels more comfortable using now, so doing this helped us a lot experience wise.

A problem we had with it is, that it took us really a while to pick up steam with this process that we never really used before. We are not quite sure if it also has something to do with the growing pressure to deliver something working in the end or most of us finally being comfortable with the development environment, but the sprints we did, worked much more efficiently from the start of the second project semester onwards.

What we feel is, that starting a bit earlier with coding could have helped the overall progress a lot, because the research of nearly one and a half month at the start where we didn't code, would probably have been more useful when we would have done it more along the way. Some starting points like registration and a basic setup of components could have been done right away and also would have been more in the mindset of an agile process.

4.1.3. User Interface

One of the points that was mentioned in the project description(see page 1) was to put an emphasis on the usability and to test with "*Plaggers", which is something we somehow neglected to do. It wasn't an intentional decision to not do usability testing, but rather something that we pushed farther and farther out in the schedule until we would finally feel more comfortable showing what we had achieved, but that never happened. There were still some more features to be finished and some more ideas to be integrated.

We also only had one team member that really felt comfortable with designing a user interface in Photoshop, which is really a useful skill in web development, although wireframes like we did at the start would probably have been sufficient here, to plan a bit better.

At some points we went back to review some parts of the interface, like the case selection dropdown for example, but new features were often times just added on the fly without a common process. This would be something, we would really need to enforce to be better organized for another project like this.

4.1.4. Project Managment Tools

The two main tools that we used to manage the project itself and it's code - Redmine and Git - were really a nice experience and worked pretty well for our purposes. They are both something we would recommend unconditionally and would use again without hesitation.

Simply having a mechanism to do version control locally without being depended on a server with Git is such a nice feature, that some of us even adopted it for simple tasks like writing a letter.

4.1.5. Unit Tests

In the beginning we stated the goal to write the system in a test-driven development style, but sadly that never really happened.

We believe this failed due to two key factors coming together. The first one is, that four of the five team members didn't have experience with developing test-driven and therefore never saw the nice benefits it can have after a while and the second is, that it would have been necessary to learn it in addition to the already overwhelming new development environment that we had set up.

4.1.6. Reponsibilities

The biggest problems we had often come down to unclear responsibilities. We defined some roles for the team members in the beginning, but we never emphasized them enough or gave the people that possessed them the power to enforce them. This probably happened because we didn't know each other at the start, so that we didn't know if we could trust the qualities of each others work in those areas for which the assignments were made. We also essentially had the feeling that we all should be on the same level, because we all simply were students, so no one wanted to step up and no inherent social hierarchy was clear prior to it, because we didn't knew each other for long enough time.

Overall this made us some kind of "doacracy", where things got done in the way they were started by someone, which can work very well, but it ultimately led to the problems described above with unit testing or the user interface for example.

4.2. Outlook

After a long students project like Unplagged has already been, there is always some kind of urge to continue, because so much time was already put into it and it would be sad to just abandon it afterwards. Sadly the breaking point of getting the grade can have dreadful consequences to the motivation of those thoughts. We hope that this won't be the case here and that at least some of us will continue working on it, because first of all we worked very openly at Github, so that maybe some others will step in and motivate us to keep working and secondly there are already two independent course works in the pipeline, which hopefully make up for the identified problems with unit testing and usability (and also give another two grade incentives).

We still have a whole lot of ideas that could be integrated or things that should get fixed, so to fade out this report, here is just a short sample of it, to get you an idea what is hopefully coming up:

- Create an installer script similar to Wordpress

- Switch Tesseract with it's simpler wrapper OCRopus
- Add some more view possibilites for fragments, like alternating lines from the case document and a source(zipper)
- Add arrows to the main menu indicating when a dropdown is present
- Integrate LaTeXas an alternate mechanism for report creation
- Make filtering and sorting of tables possible
- Integrate a fully automated test-suite with PHPUnit, qUnit and Selenium
- Have several rounds of usability testing

A. Logged Time

Table A.1.: Overview By Member and Month

Mitglied	**2012-4**	**2012-5**	**2012-6**	**2012-7**	**Gesamtzahl**
Dominik Horb	89.10	51.00	104.00	72.25	316.35
Benjamin Oertel	124.75	89.75	115.25	100.75	430.50
Elsa Mahari	63.25	47.00	32.50	53.00	195.75
Tien Nguyen	63.75	42.75	19.00	37.50	163.00
Heiko Stammel	8.50	4.30	13.50		26.30
Gesamtzahl	349.35	234.80	284.25	263.50	1131.90

Bibliography

[Netcraft 2012] Netcraft: *July 2012 Web Server Survey*. http://news.netcraft.com/archives/2012/07/03/july-2012-web-server-survey.html. Version: 2012

[Tiobe 2012] Tiobe: *TIOBE Programming Community Index for July 2012*. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Version: 2012

[Tuckman 1965] Tuckman, Bruce W.: Developmental sequence in small groups. In: *Psychological Bulletin* (1965)

[Weber-Wulff 2011] Weber-Wulff, Prof. D.: *Master Project: Plagiarism Detection Cockpit*. http://people.f4.htw-berlin.de/~weberwu/classes/HTW/projects/Plagiarism-Detection-Cockpit.shtml. Version: 2011, Abruf: 2012-07-21 15:14:01 +0000