4 Web Interface and Analysis Scripts

4.1 Introduction

Before the beginning of this semesterarbeit, an existing application was used to display all module qualification test results. As mentioned before, the main part of this semesterarbeit is to relaunch this application and adapt it to current needs, e.g. compatibility with new digital modules, more functionality, better user interface, etc.

4.2 Methodology

As a first step, a basic understanding of the CMS Pixel Detector (see section **??** and the module qualification tests (see section **??**) was needed in order to analyze the requirements for the web application. In the following sections, the web interface is referred to as "**MoRe-Web**" (CMS Pixel Detector Module Qualification Result Web Interface). The general procedure was then defined to be as follows:

- Analysis of existing application
- Definition of Requirements
- Establishment of a concept for the new MoRe-Web
- Implemenation of MoRe-Web
- Testing

During the entire process, a repeated quality check is needed. It analyses the following aspects:

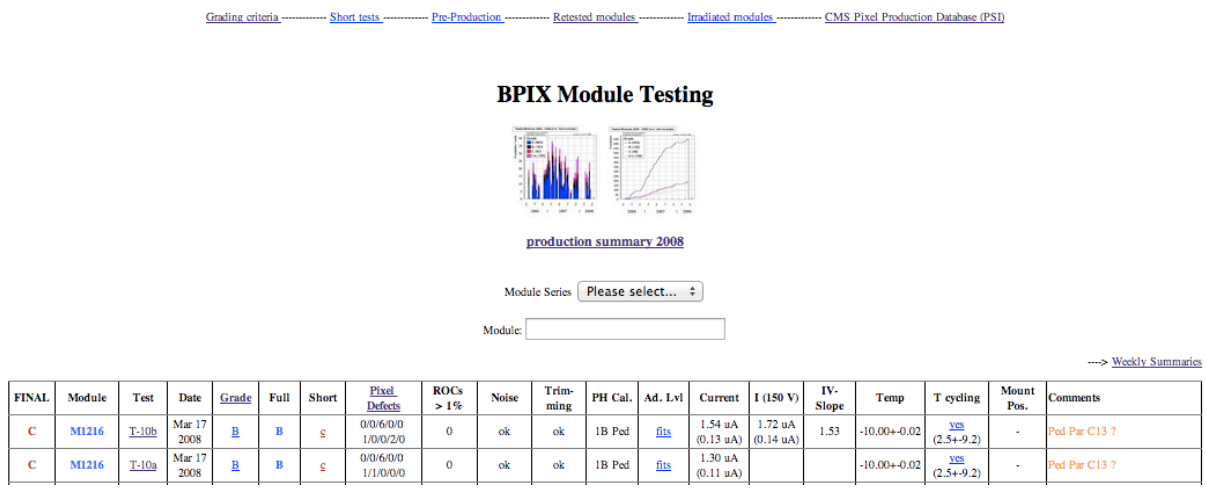
- Progress
- Meeting of requirements
- Unexpected problems
- Further improvements

4.3 Analysis of Existing Application

The starting point of the module qualification test results is `http://cmspixel.phys.ethz.ch`.

4.3.1 BPIX Module Testing

At `/moduleTests/moduleDB/prodTable.php` an application called "BPIX Module Testing" is located. The basic purpose of this script is to display an overview table for all module qualification tests and detail pages with further information for each tested module.



Grading criteria ---------- Short tests ---------- Pre-Production ---------- Retested modules ---------- Irradiated modules ---------- CMS Pixel Production Database (PSI)

BPIX Module Testing

production summary 2008

Module Series: Please select...

Module:

---> Weekly Summaries

FINAL	Module	Test	Date	Grade	Full	Short	Pixel Defects	ROCs > 1%	Noise	Trim-ming	PH Cal.	Ad. Lvl	Current	I (150 V)	IV-Slope	Temp	T cycling	Mount Pos.	Comments
C	M1216	T-10b	Mar 17 2008	B	B	c	0/0/6/0/0 1/0/0/2/0	0	ok	ok	1B Ped	fits	1.54 uA (0.13 uA)	1.72 uA (0.14 uA)	1.53	-10.00+-0.02	yes (2.5+-9.2)	-	Ped Par C13 ?
C	M1216	T-10a	Mar 17 2008	B	B	c	0/0/6/0/0 1/1/0/0/0	0	ok	ok	1B Ped	fits	1.30 uA (0.11 uA)			-10.00+-0.02	yes (2.5+-9.2)	-	Ped Par C13 ?

Figure 1: BPIX Module Testing Overview

Frontend The frontend offers an overview list of all module qualification tests as can be seen in figure 1. By clicking on a module test, an overview as seen in figure 2 is presented. A link in the upper left corner leads to an overview graph for the module test as shown in figure 3. Also, there is a link horizontally centered which leads to an address levels test graph as shown in figure 4. By clicking a specific ROC identifier, the corresponding test results show up as given in figure 5.

M1216T-10b						Address level fits						
ROC	**Total**	**Dead**	**Mask**	**Bumps**	**Trim(Bits)**	**Address**	**Noise**	**Thresh**	**Gain**	**Ped**	**Par1**	**Total**
C0	**1**	0	0	1	0 (0)	0	0	0	0	0	0	0
C1	**0**	0	0	0	0 (0)	0	0	0	0	1	0	1
C2	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C3	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C4	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C5	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C6	**2**	0	0	2	0 (0)	0	1	0	0	0	0	1
C7	**1**	0	0	1	0 (0)	0	0	0	0	0	0	0
C8	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C9	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C10	**1**	0	0	1	0 (0)	0	0	0	0	0	0	0
C11	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C12	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
C13	**0**	0	0	0	0 (0)	0	0	0	0	1	0	1
C14	**1**	0	0	1	0 (0)	0	0	0	0	0	0	0
C15	**0**	0	0	0	0 (0)	0	0	0	0	0	0	0
	6	0	0	6	0 (0)	0	1	0	0	2	0	3

Figure 2: BPIX Module Testing ROC Overview

Source The script located at `moduleTests/moduleDB/prodTable.php` generates the overview table seen in figure 3. It basically scans the `moduleTests/moduleDB/` and compares parts of the file or folder names to some parameters (e.g. series or search string). It parses the `summaryTest.txt` file of a module test and retrieves information such as test date, module number, grading, etc. from it.

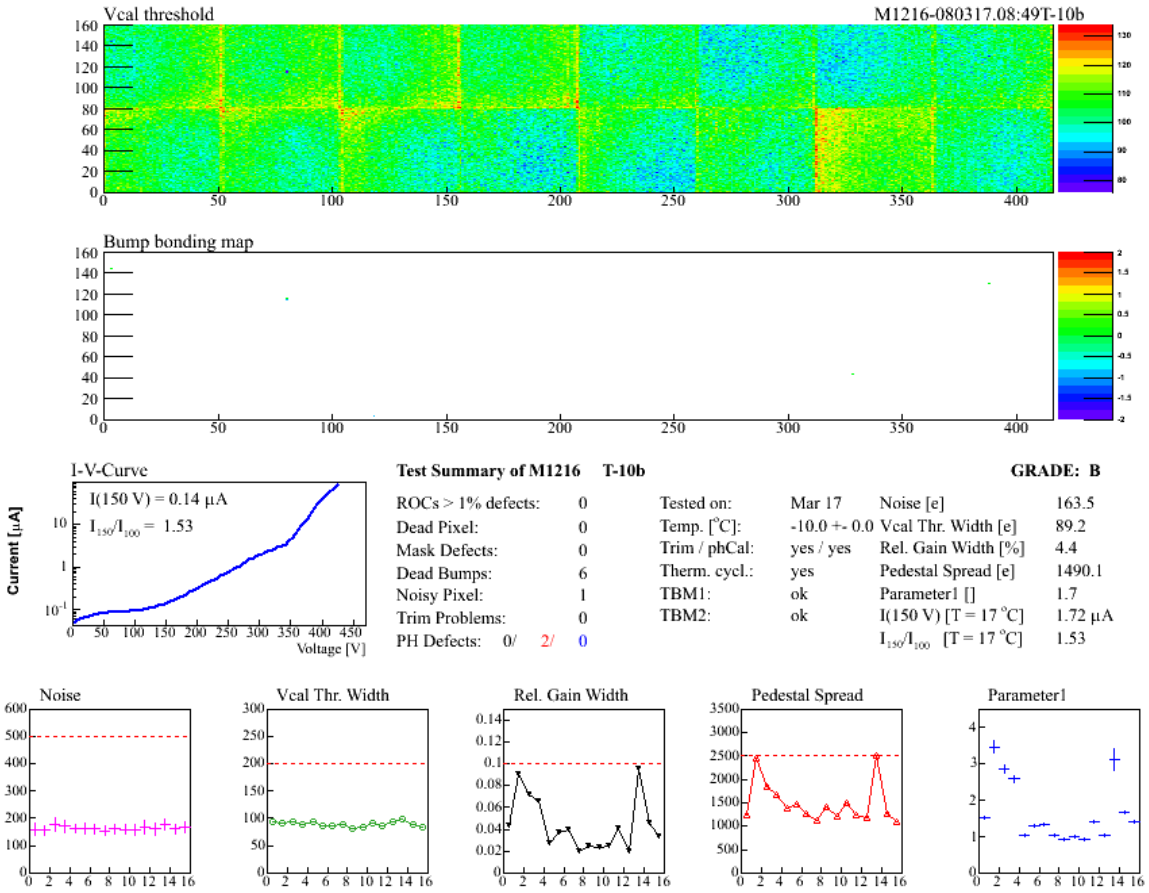


Figure 3: BPIX Module Testing Module Test Overview

4.3.2 Analysis Scripts

The analysis scripts that process the testing data an generate graphs etc. are located in `/home/peller/sdvlp/elcommandante/trunk/fitting` on the lion.ethz.ch server.

Module Summary Page The module summary page in figure 3 is generated by the script `moduleSummaryPage.C` Some important functions are listed below:

- `moduleSummary`
  Generates an image file for the module summary page. Generates the subplots using other functions in the file. Reads the data from ROOT-files. Saves the module summary as a .ps and .png file. Also generates the address level fits as .ps and .png files.

- `addVcalThreshold`
  Adds a Vcal Threshold diagram for a single chip to the Vcal Threshold overview.

- `makePlot`
  Draws the subplots "Noise","Vcal Thr. Width","Rel. Gain Width","Pedestal Spread","Parameter1".(See fitNames, nfit variables in moduleSummary())

- `grading`
  Calculates grading for a module.

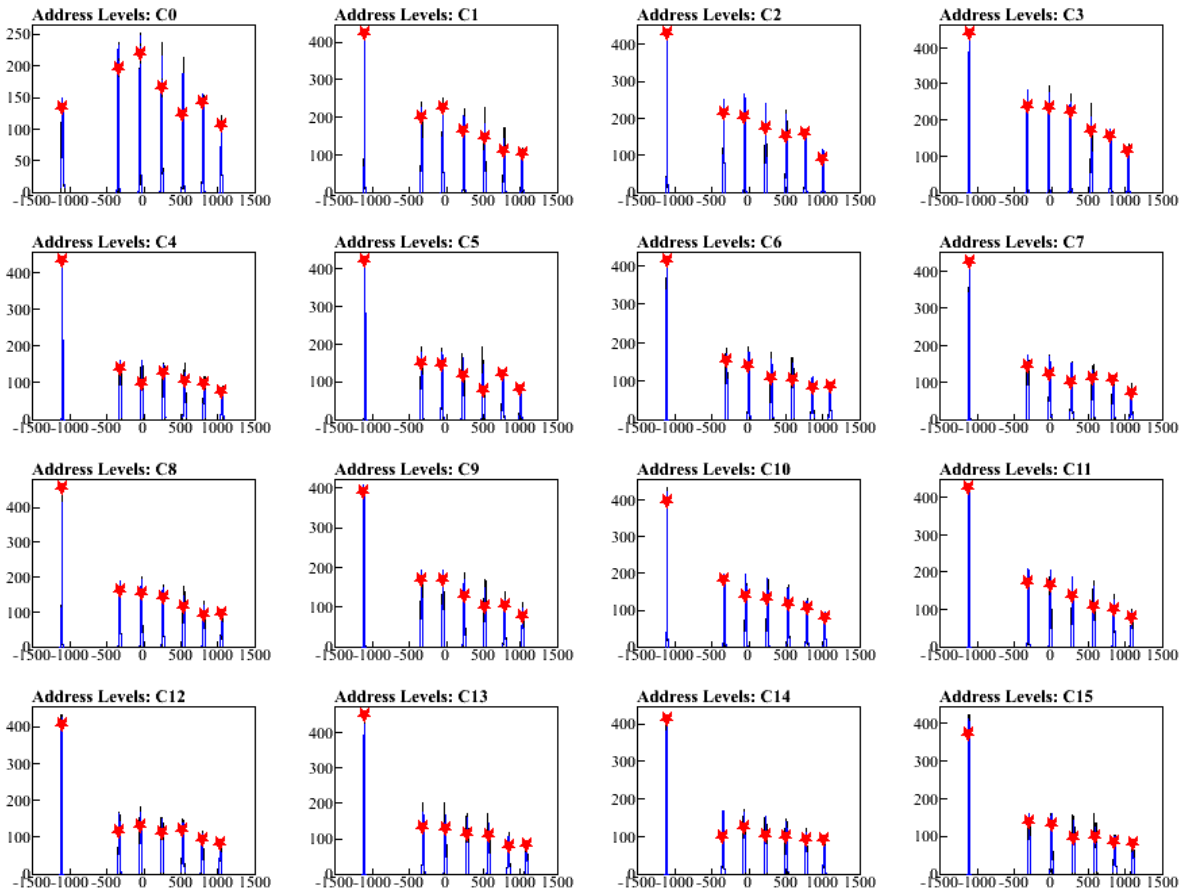


Figure 4: BPIX Module Testing Module Address Levels

- `qualification`
  Calculates the number of dead pixels, etc. needed for grading.

- `fitPeaks`
  Finds peaks for chips in subplots and calculates some statistical values.

The code is difficult to understand, since its structure is not obvious and it lacks documentation and comments.

Chip Summary Page The chip summary page in figure 5 is generated by the script `chipSummaryPage.C` Some important functions are listed below:

- `chipSummary`
  Generates an image file for the chip summary page. Generates the subplots using other functions in the file. Reads the data from ROOT-files. Saves the module summary as a .ps and .png file.

- `analyse`
  Generates plots like "ADC Measurement for ROC", "ADC Calibration for ROC", etc.

- `readCriteria`
  Reads data like "NOISE B","TRIMMING B", etc. from a file.

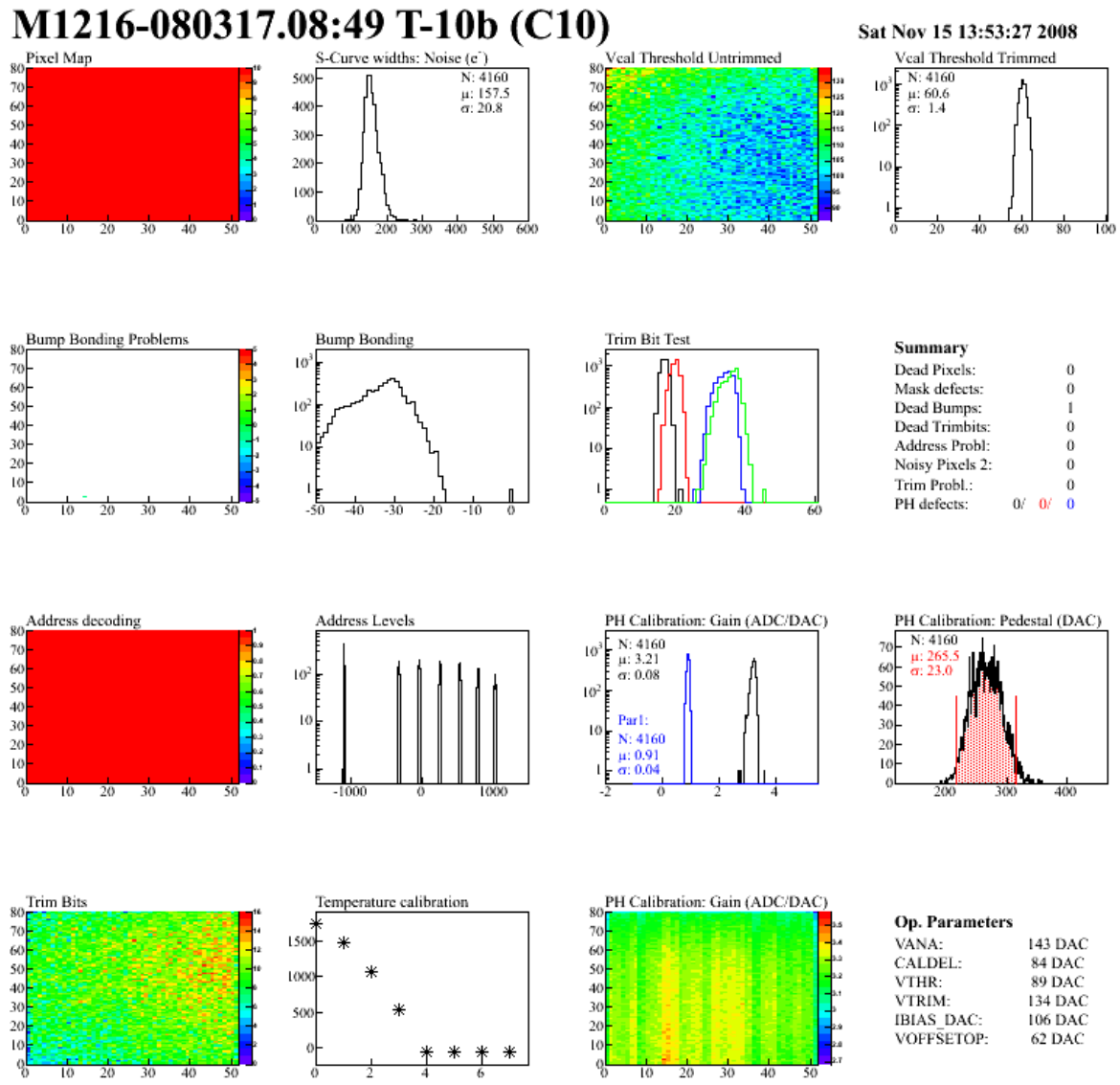


Figure 5: BPIX Module Testing ROC Test

The code is difficult to understand, since its structure is not obvious and it lacks documentation and comments.

4.4 Requirements

4.5 General

In general, MoRe-Web should be an easy-to-use interface for accessing information. It should be accessible by any client device and therefore be built on open web technologies. Furthermore, it should be optimized for display on mobile devices such as smart phones or table computers. All data should be stored in such a way that it can easily be accessed and processed by other software. It also is important that the code is easy to understand and maintain. Furthermore, the software should be extensible and flexible, in order to meet requirements that might come up in the future.

4.6 Overview

The MoRe-Web should provide an overview list of all tested modules. There might be several tests per module. This should include a sortable and searchable list with additional information available by clicking or hovering an entry, as well as filtering options. By clicking a module, the module overview together with a list of the ROCs should be displayed, similar to the existing application (see figure 2).

4.7 Module Test Overview

The overview page should offer information similar to figure 3

4.8 Module Test Details

In general, the test details should be available separately and offline, that is even without connection to MoRe-Web. The test detail page displays several plots for the test, which should be zoomable. The information available should be similar to the one in figure 5. It should also be possible to compare several tests of the same module.

4.9 Concept

The project is divided into two parts. First, the analysis scripts which are the connection point between the raw testing data and further processing. And second the web interface MoRe-Web.

4.9.1 Analysis Scripts

The analysis scripts should take data from ROOT-files and generate the following objects for both the module summary and the chip summary in the given directory structure:

- plots of test results
- HTML-file for summary pages which includes all plots and information and is accessible offline, that is it should be viewable directly from the file structure

Furthermore, the analysis scripts should insert the parsed data to a database system. For this particular case the use of sqlite is recommended, since no database server is needed and the database is directly stored in a file. Already existing entries will be replaced if the module number and the test date match. Since this database file is also available offline, it is used to generated an offline overview page.

4.9.2 MoRe-Web

The web interface is basically just a dynamic output of the overview page generated by the analysis scripts. The individual module tests consist of the offline HTML-files generated by the analysis scripts.

4.10 Implementation

4.10.1 Analysis Scripts

The analysis scripts themselves are based on the following two abstract python classes, located in `AbstractClasses/`:

- `GeneralTestResult` This class is the base for the abstracted test classes for the module summary page, the chip summary page and the address levels.
  It provides functions for populating the data from raw result files, etc. For analysing a single module, a test result object is created out of the GeneralTestResult class, which in turn contains sub test result objects that again may store sub test result objects and so on. The hierarchy is shown in figure 6.

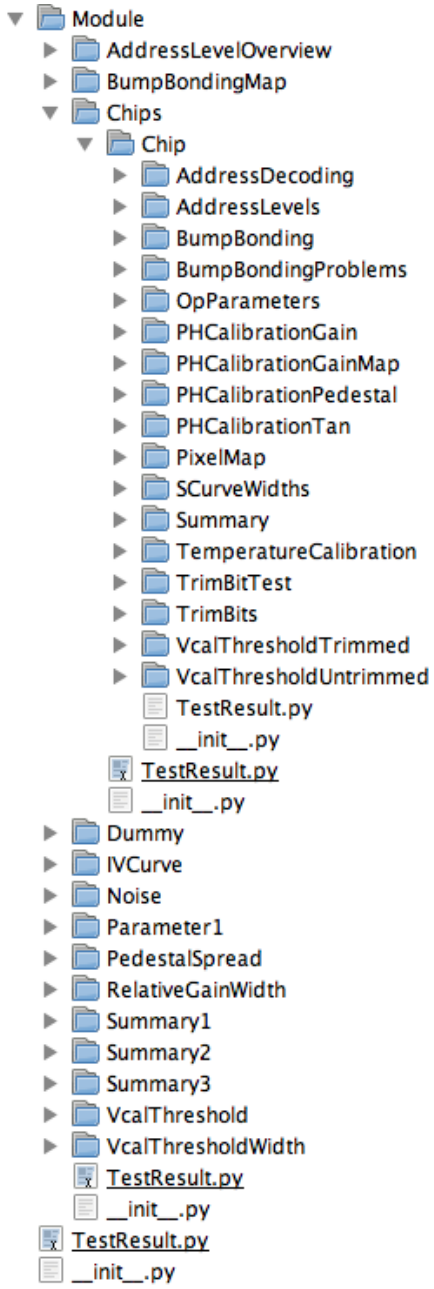


Figure 6: Class Hierarchy for the inherited GeneralTestResult classes

The actual test result data is structured in the following way:

- `Attributes`
  Hash table with attributes for the test, e.g. test date, tested object ID, module version, etc.
- `ResultData`
  - `KeyValueDictPairs`
    Hash table that stores hash tables for unique keys containing a value, a unit and a label. For example a key can be "TestTemperature", and the correspoding hash table stores -10 as the value, $\circ$C as the unit and "Test Temperature" as the label.
  - `Plot`
    Hash table that stores a ROOT-Object used for the plot, a caption, a path to the generated image file and a file format.
  - `SubTestResults`
    Hash table that stores sub test results, indexed by keys. For example, the module test result contains the IV-Curve as a sub test result.
  - `Table`
    Hash table with three lists for header, body and footer data in a table.

All the above data can be accessed by any test result object (therefore also by sub test results) since each test result object contains a reference to its parent test result object and the sub test results are indexed by a unique key.
Some important functions implemented in the class are listed below:

- `PopulateAllData`
  This function invokes the data population for the rest results and all sub test results. It needs to be called for the topmost test result instance only.
- `CustomInit`
  Can be implemented for each test result class individually to set the name, some attributes and other variables before populating data.
- `GetUniqueID`
  Returns a unique id for a ROOT-object in order to prevent memory problems.
- `GetPlotFileName`
  Returns the full path to the corresponding plot file for the current test result.
- `PopulateResultData`
  Implements how the data is populated for the current test result. That is, it can populate attributes, generate a plot file, store key value pairs and a data table.
- `WriteToDatabase`
  This function invokes the data upload to the database for the rest results and all sub test results. It needs to be called for the topmost test result instance only. Could be extended for storing all sub test results including all attributes, plots, key value pairs etc. (currently not needed, only module test result are written to database as a single row entry)

  - `CustomWriteToDatabase`
    Implements how the data is uploaded to the database for each test result.

  Moreover, it implements several functions for generating the HTML-output, explained in detail in section 4.10.2 For most test result classes, it is sufficient to implement the functions `CustomInit` and `PopulateResultData`.

- `TestResultEnvironment` This class provides general functions and variables that are needed by all test results, such as database connectivity and general parameters. The actual test result data is structured in the following way:

  - `GradingParameters`
    Hash table with grading parameters used to calculate the final grade of a module. The grading parameters are read form a configuration file.
  - `Configuration`
    Hash table containing several configuration parameters.

4.10.2 HTML-Output

Test Result HTML File As mentioned before, every test result object based on the class `GeneralTestResult` creates an output HTML-file for directly opening the test result data, such as plots, values, etc. It contains a function called `GenerateDataFileHTML` which generates the output HTML file with the result data HTML obtained by calling `GenerateResultDataHTML` for the test result data and the sub test results. The functionality in `GenerateResultDataHTML` could also be directly implemented in `GenerateDataFileHTML`, but then it would not be possible to reuse it for sub test results that are displayed on the same page. For example, the module overview page contains a data table with all ROCs, but also several subtests, which are displayed on the same page. As of functionality, there are some key features worth mentioning:

- **Navigation** Since there are references to parent and sub test results in each test result instance, a relative navigation between the test results can be implemented without connection to a database at runtime.

- **Style** The styling of the HTML document is done using CSS. The layout adapts to different display sizes by dynamically adapting the number of columns shown to the screen width. Therefore it is also possible to browse the files on a mobile device.

- **Template** The HTML-files are generated using template files by replacing markers with actual content. Hence the layout can be changed without changing the code. For this, the templating class of the TYPO3-Project (`www.typo3.org`) was ported to python and can be found at `AbstractClasses/Helper/HTMLParser.py`

Overview HTML File Using the same styling, layout and templating as in the test result HTML-files, an overview page is generated using the class `ModuleResultOverview`, which reads the data from either the local database or the global database.

4.2 Controller

A module qualification session is instantiated using the `Controller.py` program. It will instantiate the module test result object for all folders found in the `TestResultDirectory` variable defined in the configuration and all necessary sub test result objects from the classes found in `TestResultClasses/CMSPixel/ModuleTestGroup/`, provide it with a test result environment object containing data relevant for the qualification session. It also reads the configuration and sets all relevant paths.

4.2.1 System Requirements

The following requirements have to be met in order for the Controller.py to run correctly:

1. Python installed, minimum version: 2.6
2. Python packages:
   (a) ROOT (`http://root.cern.ch/drupal/content/pyroot`)

4.2.2 Step-By-Step Guide

Follow this step-by-step guide in order to perform module qualifications and generate the result output.

1. Check the minimum requirements as described in section 4.2.1
2. Open the command line on your UNIX system(Mac, Linux, etc.)
3. Change to the directory where the analysis scripts are located, i.e. where the `Controller.py` is stored, by typing `cd PathToAnalysisScripts`, where you replace `PathToAnalysisScripts` by the actual path to the analysis scripts.
4. Adjust the configuration to your current qualification session by editing the configuration files found in the `Configuration/` subfolder. The following lists describes the content of each configuration file:
   - **GradingParameters.cfg** Contains all the grading parameters. Usually needs no changes.
   - **ModuleInformation.cfg** Contains module information such as the module version, etc. for the type of modules analysed in the current qualification session.
   - **Paths.cfg** Stores the paths to the test result directory where all test results you want to qualify in this session are stored. Moreover, the path to the overview page is stored. This directory also contains the local `Overview.sqlite` sqlite database that stores all qualification results. The same database can be used for any number of qualification sessions, even when the test results in each session are stored in different folders, and therefore the resulting overview page contains all module qualification results.
   - **SystemConfiguration.cfg** Several system parameters such as the default image format, database connection parameters, etc. can be changed in this file.

5. Type the command `python Controller.py` to run the qualification session. For each module, it should display
   "Working on: 'TestDate': '1363355666', 'TestType': 'FullQualification', 'ModuleID': 'M1193' –". The current step is shown as either "Populating Data" or "Generating Final Output". If other messages occur, it is recommended to analyse their meaning, as they are most probably indicating some problems that might have occured during the qualification process.

6. If no error messages appeared, all files should have been generated and the overview page can be found in the folder specified in the configuration. Each module test result folder now contains a folder called `FinalResults`, which you can browse to access the test result files of any sub test. **Important Notice:** The HTML-files can be opened with any web browser, but for best results, make sure you use an up-to-date version of one of the applications recommended in table 6. Please note that Internet Explorer lower than version 10 is unable to display SVG-graphics. If needed, the output file format can be changed to "png", with the drawback of using the scalability of vector graphics.

Application	Minimum Version
Mozilla Firefox	4
Google Chrome	18
Safari	3
Opera	9
Internet Explorer	10

Table 1: Browser recommendations for opening the HTML-files