Automated Software Quality Monitoring in Research Collaboration Projects

Michael Sildatke
michael.sildatke@fh-aachen.de
FH Aachen - University of Applied Sciences
Jülich, Germany

Hendrik Karwanni
hendrik.karwanni@fh-aachen.de
FH Aachen - University of Applied Sciences
Jülich, Germany

Bodo Kraft
kraft@fh-aachen.de
FH Aachen - University of Applied Sciences
Jülich, Germany

Oliver Schmidts
schmidts@fh-aachen.de
FH Aachen - University of Applied Sciences
Jülich, Germany

Albert Zündorf
zuendorf@uni-kassel.de
University of Kassel
Software Engineering Research Group
Kassel, Germany

ABSTRACT

In collaborative research projects, both researchers and practitioners work together solving business-critical challenges. These projects often deal with ETL processes, in which humans extract information from non-machine-readable documents by hand. AI-based machine learning models can help to solve this problem.

Since machine learning approaches are not deterministic, their quality of output may decrease over time. This fact leads to an overall quality loss of the application which embeds machine learning models. Hence, the software qualities in development and production may differ.

Machine learning models are black boxes. That makes practitioners skeptical and increases the inhibition threshold for early productive use of research prototypes. Continuous monitoring of software quality in production offers an early response capability on quality loss and encourages the use of machine learning approaches. Furthermore, experts have to ensure that they integrate possible new inputs into the model training as quickly as possible.

In this paper, we introduce an architecture pattern with a reference implementation that extends the concept of Metrics Driven Research Collaboration with an automated software quality monitoring in productive use and a possibility to auto-generate new test data coming from processed documents in production.

Through automated monitoring of the software quality and auto-generated test data, this approach ensures that the software quality meets and keeps requested thresholds in productive use, even during further continuous deployment and changing input data.

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391478

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**; **Software verification and validation**.

KEYWORDS

Research Best Practices, Research Collaboration Management, Metrics, Lean Software Development, Software Architecture

ACM Reference Format:
Michael Sildatke, Hendrik Karwanni, Bodo Kraft, Oliver Schmidts, and Albert Zündorf. 2020. Automated Software Quality Monitoring in Research Collaboration Projects. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3391478

1 INTRODUCTION

Business-critical challenges often are predestined to be solved in collaborative research projects of practitioners and researchers. Practitioners view those projects from a business perspective and try to reach essential business goals. They often focus on time-to-market and design-to-budget strategies. On the other hand, researchers focus on theoretically rewarding challenges and try to take an innovative and creative approach. Those preconditions present several challenges, especially the different prioritization and the missing common view.

Metrics **Dri**ven Rese**a**rch Collabora**tion** (MEDIATION)[15] unifies the different views of researchers and practitioners and, thus, creates a common view on project goals. Measurable and high-level end-user metrics form the basis for an overall focus on project goals. The test-driven and incremental-iterative approach is based on the automated monitoring of all defined metrics. Acceptance tests represent those metrics programmatically. Researchers and practitioners commit to a certain quality threshold, which defines at what point of time practitioners can use an increment in production.

One of the typical underlying problems of collaborative research projects are Extract-Transform-Load (ETL) processes, in which humans extract information from non-machine-readable documents and store them in data targets by hand. Since the processed data is

informal and not standardized, researchers only can make incomplete assumptions about possible data formats and contents at the time of development.

Deterministic solution strategies fail, whenever unknown data formats appear in productive use. AI-based machine learning (ML) models can help to solve this problem. They are trained based on problem-specific known input and output data and should be able to anticipate unknown input formats.

The output quality of ML approaches may decrease over time because they are not deterministic. As a consequence, the overall quality of the application embedding ML models decreases. Hence, the software qualities at development time and in production may differ. This fact leads to the need for continuous software quality monitoring in production.

Practitioners often are skeptical about the use of research prototypes based on ML models because they are black boxes. This skepticism increases the inhibition threshold for early deployment. Continuous measurement of software quality in production offers a timely response capability on quality loss and, thus, encourages the use of ML approaches. Furthermore, experts have to integrate possible new inputs into the model training as quickly as possible.

Since MEDIATION ensures an early deployment of research prototypes, which increases business value but still have functional and technical gaps, it is essential to monitor the software quality in production continuously.

In this paper, we extend the MEDIATION approach. Based on MEDIATION, we introduce an integrated quality monitoring of the software in production. Additionally, we enable an auto-generated test data backflow from production into the development environment to minimize the manual effort for writing new tests.

The remainder of the paper is structured as follows: Section 2 describes the collaborative research project, which motivates the introduction of our approach. Section 3 describes the further development of MEDIATION and related work. Section 4 introduces **Au**toma**te**d **S**of**t**ware Qu**ali**ty Mon**i**t**o**ri**n**g In Research Collaboration Projects (ATTESTATION). Section 5 describes the concrete implementation of ATTESTATION in the collaborative research project, while Section 6 contains its evaluation. Section 7 ends the paper with a summary and an overview of future developments.

2 MOTIVATIONAL RESEARCH PROJECT

The following section motivates the introduction of the ATTESTATION pattern and describes the domain-specific use case of the research collaboration project as well as the theoretical research questions.

2.1 Domain-Specific Use Case

The underlying project which led us to introduce the ATTESTATION pattern originates from the energy industry. Our partner is a medium-sized software company that, among other things, collects tariff information from German energy suppliers. The project's focus lies on the automation of a manual ETL process in which staff members extract tariff information from product price sheets and store them into company databases.

Staff members procure required documents via the suppliers' websites, which are monitored and checked for changes by a tool. Our collaboration partner uses a company-owned data entry application to transfer manually extracted information from input documents into the company database. It includes automated plausibility checks.

Special tariffs recorded are so-called basic supply tariffs that guarantee the unconditional supply of energy to each consumer. In Germany, there are about 2,200 basic supply tariffs, all of which are published in different formats and can change daily. The range of tariffs is broad and complex. There are different tariffs for various customer groups, installed hardware, voltage levels, et cetera.

More than 90% of the documents are published in non-machine-readable PDF format and mainly contain tables listing price information as well as free text describing other contractual aspects. There is no standard tariff format, and the documents are entirely inhomogeneous.

Approximately 20 staff members manually check whether the information stored is up-to-date and adjust it as required . A second person checks the manually entered information for quality assurance.

Due to the enormous number of different document layouts, which can change with every update, a robust and flexible application is required, that can increase the degree of process automation. The development of such an application is very costly and requires the adaptation of the latest findings from research and development. Small and medium-sized enterprises (SMEs) have no opportunity to implement this alongside their daily business.

2.2 Common Research Problems

The process described has different problems of high scientific relevance. Especially the automated processing of non-machine-readable PDF documents, such as scanned ones, is of a general character and not domain-specific. How to recognize and extract semi-structured data (e.g., tables) has not yet been completely solved and represents the scientific focus of the project.

Modern approaches for object recognition are based on trained ML models and achieve a precision of 0.78 with a recall of 0.94 using image processing when recognizing tables [6]. Since these models are trained to recognize tables in scientific publications, they are not suitable for documents containing tariff information. Furthermore, the achieved precision and recall values do not meet the requirements for practical use.

Additionally, the actual processing of these documents also plays a decisive role. Due to the lack of a standardized table format, we have to develop a solution strategy that is capable of processing different and unknown input formats. In other domains, specifically trained models for automated information extraction from tables exist, but they are limited to the processing of structured data (e.g., HTML or XML) [9]. In our use case, the actual information extraction is based on data that has been extracted from unstructured data and, thus, is of inferior quality.

3 MEDIATION CONCEPT EVOLUTION & RELATED WORK

MEDIATION describes an incremental-iterative and metrics-based development approach in which each metric represents a connection to a project goal. All of the metrics are, therefore, end-user

oriented. They aim to increase the respective metrics - and thus the quality of the software - with each increment. By defining an initial threshold value for the metrics, it is determined when the software has reached a level of maturity as a Minimum Viable Product (MVP) that is suitable for productive use [15].

In our case, the partner company has to extract tariff information from PDF documents. Various attributes describe tariff information, such as the tariff name, the tariff validity, the price information, et cetera. One possible metric, for example, could be the number of correctly extracted attributes. We could define a MVP threshold in a way that the software is ready for production as soon as it, on average, extracts at least 70% of all attributes correctly.

Figure 1 shows the concept of MEDIATION and its further developments.

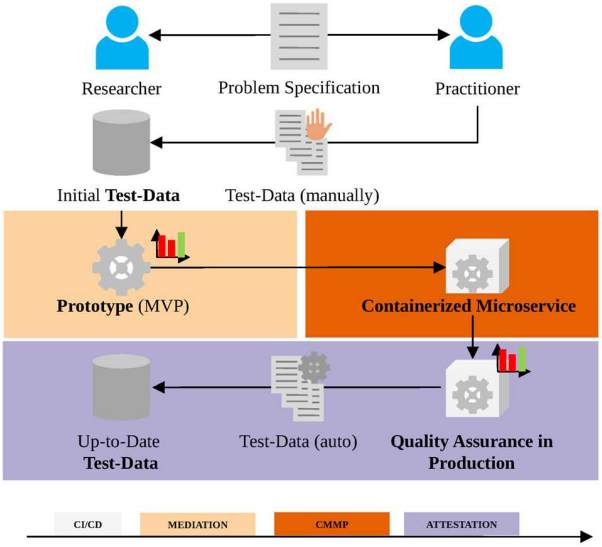

Figure 1: Evolution of MEDIATION

The original approach of MEDIATION describes the process of project initiation up to the monitoring of metrics in the development phase. Researchers and practitioners agree on project goals and define metrics. Practitioners provide realistic test cases in the form of manually defined test data on which the researchers base their work.

The software is developed iteratively until it reaches a certain threshold of metrics so that it can be defined as MVP, thus, production-ready. As part of lean product development, transparency is encouraged through this approach [11].

Collaborative Microservice MEDIATION Pattern (CMMP) as the first extension focuses on the deployment of a software increment and its actual application [13]. Among the use of containerization, this approach ensures that researchers can deploy their software in a productive environment of practitioners.

The second extension ATTESTATION described in this paper focuses on the quality monitoring of the software in productive use. It provides automated feedback through auto-generated test data originating from processed data in production.

To be profitable, practitioners focus on strategies like time-to-market or design-to-budget. Approaches like Scrum or Kanban can support them in doing this [16]. Fraser and Mancl underline the importance of benchmarks and shared information for successful research collaborations, which are supported by our approach [4]. Fowler summarizes that each metric used should be linked to the overall project-goal [3]. Classic development strategies like Continuous Integration (CI) or Continuous Delivery (CD) focus on metrics that measure code quality (like test coverage, number of lines, et cetera) [7]. Humble and Farley describe the monitoring of the software health status as an essential support mechanism on the way to production release [7].

Those strategies are useful for deterministic problems, where developers can consider all boundary conditions in the development phase. Because our underlying problems are not deterministic, the real production environment can deviate substantially. Therefore it is important to also measure the defined metrics in production to make sure that the software quality meets and keeps desired thresholds.

Weber et al. introduce a process model for the comprehensive management of ML models that considers the monitoring of trained models in an operational phase [18]. They focus on prediction problems and the detection of concept drift, for example, by measuring possible changes in the distribution of input data. We do not only focus on prediction problems where input is well-structured like statistical data tables, but also on those, where input data is only human-readable. Furthermore, we measure the actual output quality of the software.

Classic CI or CD pipelines are unidirectional [7]. That means that software increments flow from development to deployment and on to production. In our approach, we establish an auto-generated test data backflow from production into development.

Kaner et al. point out that test scripts embedding test data lead to the problem that every new test has to be programmed [8]. This method requires programming skills and is not suitable for large test data sets.

Data-Driven testing is a testing strategy in which test engineers store test data in external data tables, like Excel or CSV spreadsheets [2, 10]. In storages as such, test engineers can define test cases by hand. We extend this approach by using document-oriented data sources which do contain not only simple input and output values but also binary documents. Furthermore, the test data is generated automatically in our approach. There is no more need for time-consuming and expensive manual test data generation.

Stresnjak and Hocenski state that the overhead of Acceptance Test-Driven Development (ATDD) or Behaviour-Driven Development (BDD) can reduce costs through manually written regression tests, which are repeatable and automated [17]. Our approach enhances this effect by minimizing the manual effort for writing new tests through the automated generation of new test data.

4 ATTESTATION PATTERN

4.1 Concept

The Automated Software Quality Monitoring in Research Collaboration Projects (ATTESTATION) pattern is an extension of the

previous MEDIATION developments and delivers two important innovations:

(1) The same end-user metrics measure the quality of software during development and production.
(2) Up-to-date test data can be auto-generated by a backflow of real data into the development environment.

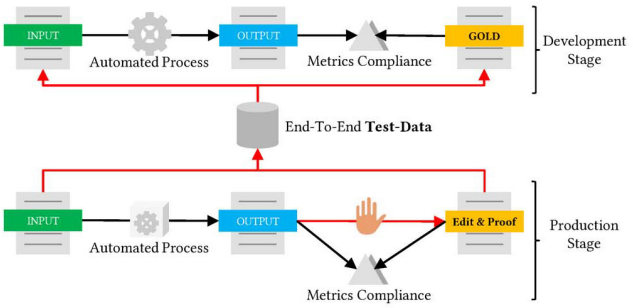

Figure 2: Conceptional View of ATTESTATION-Pattern

Figure 2 shows a conceptional view of the ATTESTATION pattern. The productive use of a research prototype enables the (partial) automation of underlying processes and minimizes the manual workload. Possible unknown data formats prevent the full automation of these processes, which leads to the requirement of manual corrections and checks.

MEDIATION describes the collection of metrics through automated tests in the development phase. GOLD-standard documents describe the expected output relating to a given input document (Section 5 gives a detailed example). In production, there are no GOLD-standard documents since unknown input data is processed. However, the consideration of manual corrections allows the subsequent measurement of the output quality. The manual corrections form the basis for measuring identical end-user metrics in production. The outputs resulting from the manual correction correspond to new GOLD-standard documents and are returned to the development stage as test data automatically.

The ATTESTATION pattern creates two main advantages:

(1) Managers can monitor software quality in the production phase, which can differ from the measured quality in development. They get an opportunity to react to quality decreases at an early stage.
(2) The auto-generated test data backflow closes gaps in the existing test data. This backflow provides fast and automated feedback. Researchers can act on this feedback early to improve the software faster (e.g., by retraining ML models).

4.2 Extended Staging

Rubin points out that in agile frameworks like Scrum, the flow of work should be organized in learning loops to acquire feedback as quickly as possible [12]. Feedback is essential to frequent and automated releases, while any change needs to trigger the feedback process. Since input data is a part of the software, developers have to test them after every structural change. End-users have to deliver feedback as soon as possible to the development team so that they act on it [7]. Different test stages should generate automated feedback [1].

In incremental-iterative approaches, end-user feedback is essential for the development team to make improvements to the software. Feedback means broadcasting information through a dashboard or other notification mechanisms. End-users or customers generate feedback on overall software quality through personal reviews [7]. We partially automate the process of feedback by an auto-generated test data backflow of processed data originating from production (cf. Figure 3). This approach ensures that changes

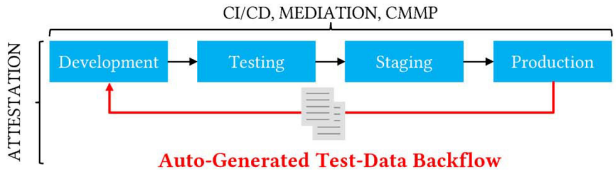

Figure 3: Extended Staging Concept of ATTESTATION

in the data structure or unknown content can be detected immediately during the development phase. There is no manual effort necessary anymore to inform the development team about real-world changes in data which can trigger the non-compliance with desired quality thresholds (cf. [15]). The automated information flow minimizes the coordination expenses between researchers and practitioners. Researchers can develop new solutions based on up-to-date data at any time.

4.3 Generic Test Management

In popular test-driven approaches like ATDD or BDD, software code embeds test data (cf. Figure 4). This handling results in the necessity to modify the software code with every update on the test data. As with the data-driven testing approach, we outsource test data into an external document-oriented database (cf. Figure 4).

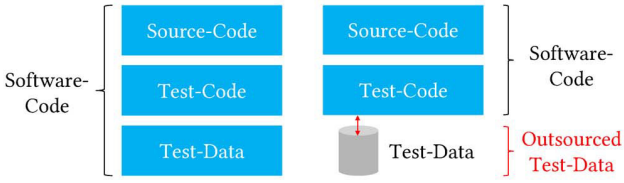

Figure 4: Generic Test Management

For ML-based applications, test and training data are of immense importance. They are responsible for the quality of the model itself and decisive for its output results. Due to the fact of possible requirement changes, they have to be retrained continuously. Our approach ensures flexible management of test and training data by separation from the actual software code.

5 APPLICATION

Within our collaborative research project, we develop an application called **P**rice**I**nformation**E**xtraction (PIE), which automatically extracts tariff relevant information from incoming PDF documents. In this section, we demonstrate a real-life application of our introduced ATTESTATION-Pattern.

5.1 Problem Scenario

The need for an optimized and highly automated process prompted our research partner to initiate the collaboration project. We focus

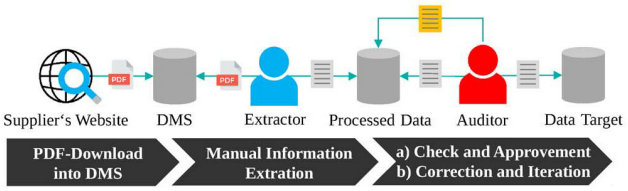

Figure 5: Problem Scenario

on a multi-stage process, shown in Figure 5. A software tool monitors the websites of all energy suppliers to check them for updates daily. Staff members download new PDF documents and store them into the central Document Management System (DMS) manually, marked as unprocessed.

Staff members act in the roles of extractors and auditors alternately. First, the extractor checks the list of all unprocessed documents and chooses one to process. He or she extracts all relevant information by hand and pushes it to a temporary database (Processed Data). An auditor checks the extracted information, and either approves or corrects it. If there are any corrections, he or she stores an update in the temporary database. Another auditor rechecks the updated information while quality assurance mechanisms enforce that one auditor never checks his or her own extractions. Staff members repeat the process manually until there are no more corrections. Finally, an auditor feeds the extracted information into the target database.

5.2 Solution Strategy & Applied Metrics

According to MEDIATION, we defined metrics and goals to measure our project progress and the quality of our application. The overall goal is to reach correct information extraction for 95% of all upcoming attributes ($Q_{gold} \geq 95\%$). We choose the initial threshold of our MVP as $Q_t \geq 70\%$. This threshold marks the limit from which a software increment gains business value. In our case, an increment that reaches $Q_t \geq 70\%$ can already minimize manual effort. But since it achieves a lower precision than required for full automation, we use it as a recommendation engine (cf. Figure 6). With the achievement of Q_{gold} we can implement full automation.

To solve the table detection task, we developed an ML approach, which originates from Gilani et al. [5]. The text extraction is OCR-based. The information extraction from tables is rule-based, while information extraction from unstructured texts works with simple dictionaries (cf. Figure 7). The solution strategy is not further detailed because it is not the focus of this paper.

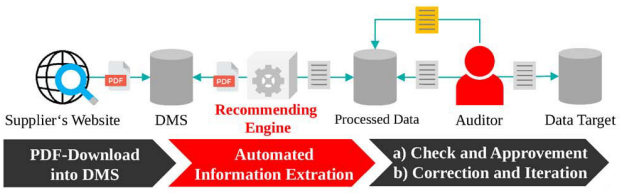

Figure 6: Recommendation Engine in ETL Process

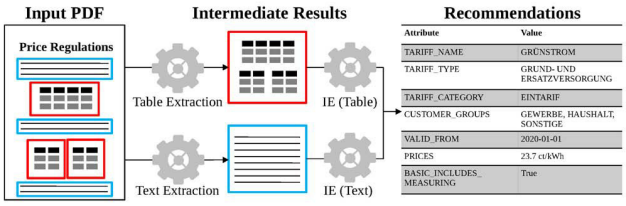

Figure 7: Attribute Recommendation Engine

Each extracted information corresponds to an attribute of the price regulation that is fed into the target database. The number of a price regulation's attributes depends on its complexity.

We define our quality metric by the correct extracted attributes of a price regulation from a PDF document provided as input

$$q_i = \frac{[Number\ of\ correct\ extracted\ Attributes]}{[Number\ of\ all\ expected\ Attributes]}$$

and the overall quality metric of the software by the arithmetic mean of all measured extraction results

$$Q = \frac{1}{n}\sum_{i=1}^{n} q_i$$

Figure 8 shows an example of extracted and expected information of a price regulation:

ATTRIBUTE	EXTRACTED	EXPECTED
TARIFF_NAME	GRÜNSTROM	GRÜNSTROM
TARIFF_TYPE	GRUND- UND ERSATZVERSORGUNG	GRUND- UND ERSATZVERSORGUNG
TARIFF_CATEGORY	EINTARIF	EINTARIF
CUSTOMER_GROUPS	GEWERBE, HAUSHALT, SONSTIGE	GEWERBE, HAUSHALT, SONSTIGE, LANDWIRTE
VALID_FROM	2020-01-01	2020-02-01
PRICES	23.7 ct/kWh	23.2 ct/kWh
BASIC_INCLUDES_MEASURING	True	True

Figure 8: Example of Extracted and Expected Information

Comparing the extracted and expected information, one can see that there are differences in *CUSTOMER_GROUPS*, *VALID_FROM*, and *PRICES*. Applying our metric would lead to the result $q_1 = \frac{4}{7} = 0.57$. Assuming that there is a second extraction result $q_2 = 1.0$, our overall quality metric would be $Q = \frac{0.57 + 1.0}{2} = 0.785$, which meets the MVP threshold.

Since there are no GOLD-standard documents in production, we compare the automatically extracted information with the corrected ones of the auditor.

5.3 Reference Architecture of PIE

The presented implementation is specialized for the described problem scenario while its basic architecture is generically reusable.

To implement our approach, we used a Python stack for PIE and a Java 11 stack with Maven and Spring Boot[1] for the manual correction utility and additional Representational State Transfer (REST) services that support our architecture. Those Spring Boot services are:

ManualCorrection This service provides a website that displays the original PDF document and the suggested extractions by PIE next to each other. With this tool, a staff member can review the suggestions and make corrections if necessary. After the user marks the extractions as correct, either by correcting any errors or by approving an already correct suggestion, the software forwards relevant data to the MongoPersist service.

MongoPersist This service acts as a layer between the ManualCorrection service and the productive MongoDB[2] and persists any document sent from the manual check into the database. The service receives the manually corrected data together with the prediction from PIE and the original PDF document. If the extracted price regulations from PIE differ from the manually corrected price regulations, MongoPersist forwards the original PDF document and its correct price regulations to the TestCaseManager service.

TestCaseManager Similar to MongoPersist, this service acts as a layer between the production environment and the End-To-End Test-Data MongoDB that stores data for the dynamically generated test cases. Additionally, this service triggers a new GitLab CI/CD pipeline via the GitLab Application Programming Interface (API) as soon as it inserted a specified number of new documents into the database. This pipeline runs the tests of PIE in the development environment.

PIE itself implements a Flask[3] web service that serves its functionality via a REST interface. PIE and the other services are all containerized with Docker[4] to ensure that those applications run the same way, regardless of the surrounding infrastructure.

To continuously store and monitor our MEDIATION metrics, we use an InfluxDB[5] and a Grafana[6] instance each in our development and production phases. To persist data in production and for test case generation, we use two different MongoDB instances. Figure 9 shows the architecture of the described implementation.

Both the productive and the End-To-End Test-Data MongoDB store their data in the structure shown in Code Listing 1. Each stored MongoDB document consists of a unique id given by MongoDB, a binary representation of the original PDF file, different ids

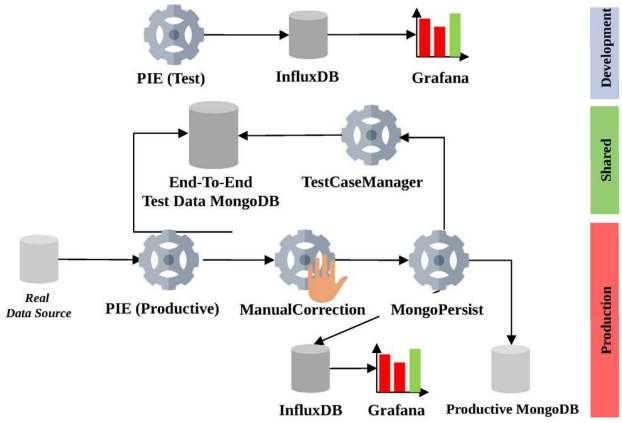

Figure 9: PIE reference architecture overview

used further in the workflow, and the price regulations that were extracted by PIE and then manually corrected.

```
{
  "_id": ObjectID("5dfb25ef1c16afd1343208b9"),
  "pdf_file": Binary(...),
  "dms_id": "P000141250",
  "market_role_id": 201,
  "price_regulations": [{
    "market_role_id": 2018,
    "tariff_name": "ORIGINALSTROM",
    "tariff_type": "GRUND- UND ERSATZVERSORGUNG",
    "tariff_category": "EINTARIF",
    "customer_groups": [
      "GEWERBE", "HAUSHALT", "LANDWIRTE", "SONSTIGE"
    ],
    "valid_from": "2017-01-01",
    "basic_includes_measuring": true,
    "prices": [{
      "type": "GRUNDPREIS",
      "category": "NETTO",
      "unit": "CENT/JAHR",
      "amount": 6555,
      "stagger_from": 0,
      "stagger_up_to": 100000,
      "hour_from": 0,
      "hour_until": 24
    }, {
      "type": "ARBEITSPREIS",
      "category": "NETTO",
      "unit": "CENT/KWH",
      "amount": 25.5,
      "stagger_from": 0,
      "stagger_up_to": 100000,
      "hour_from": 0,
      "hour_until": 24
    }]
  }]
}
```

Code Listing 1: Example document from MongoDB

We test PIE via the Python unit testing framework unittest.[7] Querying the test data sets from the End-To-End Test-Data MongoDB is done using the Python package pymongo,[8] as shown in Code Listing 2. Using the *MongoClient* provided by this package, we can query MongoDB collections very easily. In this case, the

[1] https://spring.io/
[2] https://www.mongodb.com/
[3] https://www.palletsprojects.com/p/flask/
[4] https://www.docker.com/
[5] https://www.influxdata.com/
[6] https://grafana.com/
[7] https://docs.python.org/3/library/unittest.html
[8] https://pypi.org/project/pymongo/

test runner loads all GOLD-standard documents from MongoDB without any additional query parameters.

```
GOLD_URI="mongodb://localhost:27017/gold"

@staticmethod
def load_test_cases():
    gold=MongoClient(GOLD_URI).gold
    return [(test_case,) for test_case in gold.price_sheets.find({})]
```

Code Listing 2: Load Test-Cases from MongoDB

To dynamically run a test case for each received entry from the End-To-End Test-Data MongoDB, we use the Python package parameterized.[9] Code Listing 3 shows the usage of this package. The test method *test_01_extract_prices* is called for each data entry loaded from the MongoDB previously via the decorator *parameterized.expand*. During the test run, test developers can access the current data entry via the parameter *test_case*. They can access the original document and manually extracted price regulations using the keys *'pdf_file'* and *'price_regulations'*. To determine the quality metric for each document, PIE must run its algorithm on the original PDF to extract the price regulations. After that, we compare those regulations with the correct regulations from the End-To-End Test-Data database using the approach mentioned in Subsection 5.2 and store their quality in an InfluxDB. The Python package python-influxdb[10] handles the communication with the InfluxDB.

Because we only consider the test failed when the mean quality of all tests is below a certain threshold (cf. Subsection 5.2), there is no assert in this test method. We store the quality of each extraction in the static variable *test_results*. Additionally, if the quality of a single test is below that threshold, we use the method *skipTest* to print a warning to simplify debugging. After the parameterized test method, we execute a second test method *test_02_overall* to check the overall mean quality of the extracted regulations. In there, the mean quality is calculated and asserted against the given threshold with the *assertGreaterEqual* method. This assertion ensures that the test execution only fails if the overall quality is too low. Low qualities of single extractions result in warnings.

```
test_results=[]

@parameterized.expand(load_test_cases())
def test_01_extract_prices(self, test_case):
    result=extract_prices(test_case['pdf_file'])
    matched=compare_result(result, test_case['price_regulations'])
    save_in_influx(test_case, matched)
    self.__class__.test_results.append(match)
    if match<0.7:
        raise self.skipTest("Skipped because quality threshold of "
                            "{} not matched. Quality: {}"
                            .format(0.7, match))

def test_02_overall(self):
    self.assertGreaterEqual(mean(self.__class__.test_results), 0.7,
                            "Mean quality does not match threshold."
                            "Please check skipped tests.")
```

Code Listing 3: Template for each Test-Case from MongoDB

In our setup, we manage every service in its own GitLab repository, each with its own CI/CD pipeline that runs tests, creates Docker images, and deploys the software after manual confirmation. GitLab exposes an API that, among other things, allows an external program to trigger specific CI/CD pipelines. The TestCase-Manager uses this function to automatically trigger the Python tests mentioned before after a certain number of new documents arrived from the production environment and subsequently starts the evaluation of the MEDIATION metrics in the development phase. If the overall quality of the tests is too low, the entire test execution is considered unsuccessful, and the CI/CD pipeline generates an error. The CI server informs the responsible developers via email about this unsuccessful pipeline execution.

6 EVALUATION

During the development of PIE (cf. Subsection 5.2), we developed our ATTESTATION approach to increase the acceptance for ML-based prototypes and to shorten release cycles. In addition to the project described and a concrete implementation of the ATTESTATION pattern, parts of the approach are also successfully used to measure ML model performance in production environments in a collaborative research project with a partner in the field of BIO-eCommerce [14].

As we started with the development of PIE following the MEDIATION approach, we had two main problems:

(1) The management of our research partner was skeptical about using an ML approach as the solution's basis because its concrete functionality was not transparent.
(2) The model had to be retrained frequently. This caused manual effort for the domain experts, who had to generate new test data each time.

Based on this problem, we decided to extend MEDIATION and introduced possibilities to monitor the software quality in production automatically and to auto-generate new test data. We implemented a dashboard through Grafana, shown in Figure 10.

Figure 10: PIE-Metrics dashboard based on Grafana

The first row shows the metrics that we measured during development. Point 1a shows the software quality (a) and the number of measurements (b) over time. Each measurement corresponds

[9] https://pypi.org/project/parameterized/
[10] https://github.com/influxdata/influxdb-python

to an automated test in which an input document is processed by PIE and compared with its GOLD-standard output. The horizontal line represents the threshold of $Q_t \geq 70\%$. One can see that the amount of test data increased over time while the software quality fluctuated. The widget marked with point 1b shows the current software quality that we measured during development.

The second row shows the equivalent to the software in production. Points 2a, 2b, and 2c mark the date on which we released new increments. A new release always resets the number of measurements in production.

Figure 11 shows the comparison of the measured software qualities in development (a) and production (b). The vertical dotted lines mark the dates of releases.

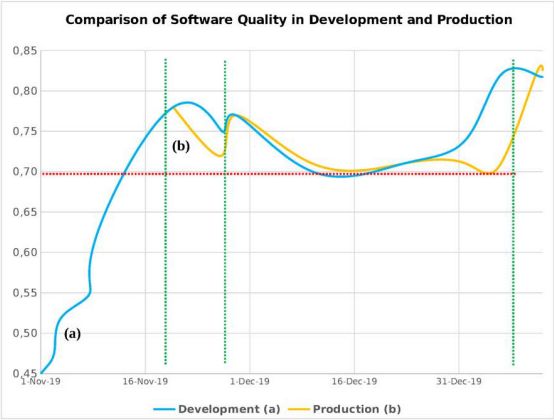

Figure 11: Comparison of software quality in development and production

Between the first and second release (first and second vertical dotted line), the measured software quality in production decreased. Due to the automated quality monitoring in production and the auto-generated test data backflow, we could recognize the quality loss and take action early to improve the application for a second release. Although the second release was more stable, we reacted to another trend of quality loss before we deployed the third release.

After applying our ATTESTATION approach, the fluctuation of quality became transparent and more stable. As a result, the development process became more effective. We could act on fluctuations of quality in time and improve the software with the help of auto-generated test data quickly. The management was convinced that the early deployment of the ML-based solution increases business value. On the other hand, researchers could continuously optimize their models with always up-to-date test data.

7 RESUME & LESSONS LEARNED

In this paper, we introduced an architecture pattern that extends the concept of MEDIATION with automated monitoring of software quality in production and a possibility to auto-generate new test data coming from real processed documents.

Our approach encourages the early deployment of ML-based applications and minimizes the risk of quality loss. It offers an opportunity to react to trends at an early stage and ensures that the software quality meets and keeps the requested thresholds in productive use, even during further continuous deployment and changing input data.

The auto-generated test data backflow provides a possibility to gain fast and automated feedback. Researchers can act according to this feedback to improve the software faster. After a while, parts of the test data may be outdated. In future work, this offers a starting point for further optimizations.

In conclusion, the presented ATTESTATION approach monitors software quality in production and generates up-to-date test data automatically. It gains business value through additional transparency and the ability to react to possible quality loss early. Since the provided approach applies in different projects, we have shown that it is not domain-specific but suitable for all projects where researchers and practitioners cannot fully determine requirements at the beginning. In particular, this applies to ML-based applications.

REFERENCES

[1] Richard Ellison. 2016. *Software Testing Environments Best Practices s.* https://www.softwaretestingmagazine.com/wp-content/cache/page_enhanced/www.softwaretestingmagazine.com/knowledge/software-testing-environments-best-practices/_index.html_gzip

[2] Mark Fewster. 1999. *Software Test Automation - Effective Use of Test Execution Tools* (01. ed.). Addison-Wesley, Amsterdam.

[3] Martin Fowler. 2013. *An Appropriate Use of Metrics.* https://martinfowler.com/articles/useOfMetrics.html#WhatsWrongWithHowWeUseMetrics

[4] Steven Fraser and Dennis Mancl. 2016. Strategies for Building Successful Company-University Research Collaborations *(SER&IP '16)*. Association for Computing Machinery, New York, NY, USA, 10–15. https://doi.org/10.1145/2897022.2897025

[5] Azka Gilani, Shah Rukh Qasim, Imran Malik, and Faisal Shafait. 2017. Table Detection Using Deep Learning. IEEE, 771–776. https://doi.org/10.1109/ICDAR.2017.131

[6] Matthias Hansen, André Pomp, Kemal Erki, and Tobias Meisen. 2019. Data-Driven Recognition and Extraction of PDF Document Elements. 7, 3 (2019), 65. https://doi.org/10.3390/technologies7030065

[7] Jez Humble and David Farley. 2010. *Continuous Delivery - Reliable Software Releases through Build, Test, and Deployment Automation* (01 ed.). Pearson Education, Amsterdam.

[8] Cem Kaner, James Bach, and Bret Pettichord. 2011. *Lessons Learned in Software Testing - A Context-Driven Approach* (01. ed.). John Wiley & Sons, New York.

[9] Nikola Milosevic, Cassie Gregson, Robert Hernandez, and Goran Nenadic. 2019. A framework for information extraction from tables in biomedical literature. 22, 1 (2019), 55–78. https://doi.org/10.1007/s10032-019-00317-0

[10] Rick Mugridge and Ward Cunningham. 2005. *Fit for Developing Software - Framework for Integrated Tests* (1. aufl. ed.). Pearson Education, Amsterdam.

[11] Donald G. Reinertsen. 2009. *The Principles of Product Development Flow - Second Generation Lean Product Development* (01. ed.). Celeritas, Redondo Beach, California.

[12] Kenneth S. Rubin. 2012. *Essential Scrum - A Practical Guide to the Most Popular Agile Process* (01. ed.). Addison-Wesley Professional, Boston.

[13] Oliver Schmidts, Bodo Kraft, Marc Schreiber, and Albert Zündorf. 2018. Continuously Evaluated Research Projects in Collaborative Decoupled Environments. In *2018 IEEE/ACM 5th International Workshop on Software Engineering Research and Industrial Practice (SER IP)* (2018-05). 2–9.

[14] Oliver Schmidts, Bodo Kraft, Ines Siebigteroth, and Albert Zündorf. 2019. Schema Matching with Frequent Changes on Semi-Structured Input Files: A Machine Learning Approach on Biological Product Data:. SCITEPRESS - Science and Technology Publications, 208–215. https://doi.org/10.5220/0007723602080215

[15] Marc Schreiber, Bodo Kraft, and Albert Zündorf. 2017. Metrics Driven Research Collaboration: Focusing on Common Project Goals Continuously. In *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*. 41–47. https://doi.org/10.1109/SER-IP.2017..6

[16] Ken Schwaber and Jeff Sutherland. 2012. *Software in 30 Days - How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust.* John Wiley & Sons, New York.

[17] Stanislav Stresnjak and Zeljko Hocenski. 2011. Usage of Robot Framework in Automation of Functional Test Regression. (10 2011).

[18] Christian Weber, Pascal Hirmer, Peter Reimann, and Holger Schwarz. 2019. A New Process Model for the Comprehensive Management of Machine Learning Models. 415–422. https://doi.org/10.5220/0007725304150422