Modernisation of a 12 Years Old Digital Archive: Experiences and Lessons Learned

Ilvio Bruder[1](✉), Martin Düffer[2], and Andreas Heuer[3]

[1] University Library Rostock, Rostock, Germany
[2] Hanse Softwarehaus, Rostock, Germany
[3] Database Research Group, University of Rostock, Rostock, Germany
{ilvio.bruder,martin.dueffer,andreas.heuer}@uni-rostock.de

Abstract. Sustainability became a very important requirement in research projects, especially in digitalisation projects and their information systems. The problem in planning sustainability of data and systems is the unknown future of technical and organisational conditions for running these systems and services. In this paper, we want to look at a digital archive project about digitised historical music sheets which was built from 2003 till 2005 and generally modernised in 2017/2018. We want to present the challenges in rebuilding a 12 years old digital archive and discuss the lessons learned.

Keywords: Software evolution · Digital archive · Long-term preservation · Software monitoring

1 Motivation

Su-Shing Chen already described the suitable "Paradox of Digital Preservation" in 2001 [3]:

> Traditionally, preserving things meant keeping them unchanged; however, our digital environment has fundamentally changed our concept of preservation requirements. If we hold on to digital information without modifications, accessing the information will become increasingly more difficult, if not impossible.

The digital preservation is also often described as the "Tamagotchi effect". This means for a digital library or a digital archive project, that we have to maintain our system regularly. This requires a relatively big amount of resources, e.g. financial and personnel costs. Early digitalisation projects did not plan for later maintenance of their systems. Therefore, many funding authorities have extended their funding criteria to include sustainable solutions. Unfortunately, these criteria do not require, how this sustainability can be achieved practically.

Hence, preserving data as well as knowledge, e.g. described in [6], and preserving technical systems are different problems. Another problem is the economic sustainability which is explicitly addressed by digital libraries [7].

A good overview of different perceptions regarding sustainability in articles of recent years is given by [5]. Many aspects of sustainability were discussed, mostly technology, management, sharing and backups as well as costs and revenue. Many articles describe concepts which are important for planning digital archives. In this paper we want to discuss the real effects of weak maintenance of a digital archive running for 12 years.

The example we want to show, is a project of a digital archive of digitised handwritten music sheets. This archive is more a tool for analysing note scribes than a platform for sharing and exhibiting digital documents. Thus, the paper focusses on the problems migrating the functionality of the analysing tool on new hard- and software. The archive was built in 2003 and 2004 and had been constantly used over the years, but barely maintained. After over 12 years of usage, the system was very slow, crashed often, and the software included critical security risks. In 2017, we had to decide to either update this archive or shut down the service. We decided to rebuild the system and challenged many expectable and non-expectable issues. In the following sections, we want to give an overview of the old and the new system, of our problems rebuilding the system and of the lessons learned. The aspect of monitoring the system in the future will be discussed in particular.

2 Starting Point: The eNoteHistory Project

In 2002, about 1000 handwritten music sheets from the 17th and 18th century were digitised at the University of Rostock. After the digitalisation, a research archive for analysing such sheets was built in the eNoteHistory project [1,2]. The project had the aim to build a digital archive, integrating knowledge components and specialised functions for writer identification in historical music scores.

Handwritten music scores were a way to record, copy and disseminate music during the late 17th century until the beginning of the 19th century. The information encoded in these music sheets, such as melody, title, time and place of origin, composer and scribe is of great interest to musicologists. Some of these data are seldom found in alphanumeric form in manuscripts, but have to be derived analysing other document features. Complex typographic and visual features such as handwriting characteristics, water marks etc. are used in practice for the analysis.

The archive consists of digitised sheets, metadata, as well as analysis tools and results. These data could be accessed in a web-based client at http://www.enotehistory.de. It was also possible to analyse new sheets with the help of feature trees or using an automatic analysis. In the manual analysis, specific features of the unknown music sheet are chosen from the feature trees, e.g. a specific handwriting of the treble clef.

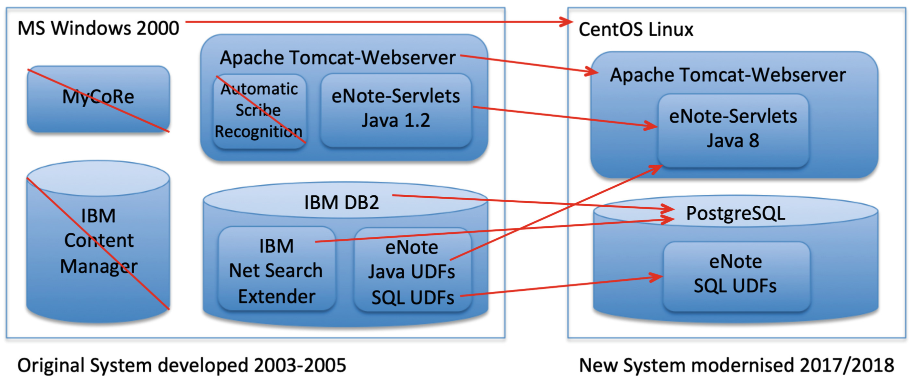


Fig. 1. Software components of the original and the modernised system.

The automatic analysis is based on the recognition of features like lines and their direction as well as the spatial arrangement of lines and points of the note heads and stems.

The system was built in a Windows 2000 Server environment on a separate server system. The programming language was mainly Java on server-side with Java Servlets for the web sites. There existed two storage solutions, first a realisation on an IBM DB2 database and second a realisation on an IBM Content Manager [4] installation with a digital library frontend based on MyCoRe[1]. The DB2-realisation included the whole analysis and access possibilities. The MyCoRe-realisation only provided simple access and browsing functionality.

3 Working Steps for Modernisation

After over 10 years of minimum maintenance, the first step was an extensive and time-consuming analysis of the system because nobody of the original development team has been working at the institution anymore. One of the developers was contacted, but could not remember much of the design. In a second step, modernisation decisions had to be made about:

- what system (hard- and software) will be used,
- which components are left as old implementation, which components have to be reimplemented (see Fig. 1),
- which functions are essential, which functions could be switched inactive to reduce the effort, and
- how to migrate the data, metadata as well as user data.

In a third step, the new system was designed and implemented. After that, the data had to be transformed and migrated into the new system. The last step was the evaluation of the system.

[1] MyCoRe is a framework for building digital libraries: http://www.mycore.de.

3.1 Analysing the Old Implementation

The analysis was the most important step, before taking any implementation and modernisation decisions. The analysis consisted of examining hard- and software, testing the functionality as well as studying the original project and reading the documentations.

As mentioned above, the eNoteHistory archive was a Java implementation on a Windows 2000 Server. The last implementation and administration steps were done over 10 years ago. There were two different implementations: based on an IBM DB2 database (version 8) and based on IBM Content Manager with a MyCoRe frontend. The DB2-based version has been running the last 10 years. The Content Manager version was available, but was obvious fragmentary and could not be reactivated without massive effort.

The executable version using IBM DB2 database was somewhat broken, too. Functionalities like search, navigation, and analysis were not accessible. Moreover, the Windows 2000 Server was a virtual server and was working very unstable.

Documentation and source code were available in different versions and it was not clear which one described the last version. Furthermore, the source code was partly incomplete. The material was overall large, but inappropriately structured for long-time maintenance. After analysing all of the eNoteHistory code variants, we decided to use the IBM DB2 version as the initial point for the restoration. Contrary to IBM DB2, the IBM Content Manager is a database and information system software, which is not further available in this form at IBM.

The Windows 2000 Server is obviously not up to date and a potential security risk. It is important to update the underlying operating system before the restoration of the eNoteHistory software itself. The plan was to use a Linux-based operating system (CentOS) in the future for a better integration into the institutes hardware environment. This means, we had to check all software components of eNoteHistory for compatibility within a Linux environment. In several cases, it was necessary to find similar software or software libraries for the Linux operating system.

Besides the database system itself, there were used extensions of the DB2 database, such as the netSearchExtender which are not available for current versions of IBM DB2. Furthermore, the eNoteHistory software used self-implemented extensions, e.g., User Defined Types and User Defined Functions. These extensions are implemented in Java with version 1.2 and corresponding Java libraries. Such extensions are not executable in modern environments with Java version 8 or 11[2] and above.

The web site as a frontend was implemented using Java Servlets. These Java Servlets were executed and content was delivered by a Tomcat web server. The version of the Tomcat-Server was 5.5 with Java 1.2. This Tomcat-Server is not supported anymore. Some of the used functions in the Servlets are not available

[2] Java version 8 and 11 are long-term support (LTS) versions.

in current versions of Tomcat anymore. Therefore, we needed new functions and a migration or transformation of the old functions.

3.2 Upgrading the Database from DB2 to PostgreSQL

One major task in modernising the eNoteHistory system was the upgrade of the database system. Sustainability, expenses and spread of current database systems are important criteria for the decision, which database will be used in the future. We decided to use the open source database system PostgreSQL which is the university-wide deployed and supported database solution.

In a first step, the database schema, the special user-defined types, and user-defined functions were analysed. The SQL standard evolved over the years and the database manufacturer have often implemented special language adaptations of SQL. The schema had to be adapted, so that some DB2-specific data types were mapped onto PostgreSQL-specific data types. For instance, the DB2 data type DATALINK is not available in PostgreSQL. These changes were made before implementing the software. Some of the problems occurred and were solved during the implementation and transfer of the data. Such problems are sizes of data types, special formats or functional differences between the database systems. Due to the significant differences between the old and new database system, a simple export from the old system and import into the new system was not possible. The original database migration tools provided by IBM for DB2 version 8 were tested, but were not applicable. Therefore, we implemented an own migration tool for the schema and data transformation, which goes through the data step by step and repeats steps on failure. For instance, if an error, that a value is out of range of a domain, is produced, the step is undone, the schema is manually adapted, and the step is repeated. A failure can occur in every row of a table. The developed tool uses an old DB2 database driver in combination with a current database driver for PostgreSQL. The transformation of schema and data with this tool was a trial and error approach.

The eNoteHistory application and clients needed access to the database. The old DB2 driver was replaced by the new PostgreSQL driver in these implementations too. Furthermore, all of the SQL-queries in the implementations had to be checked, whether they were functioning as intended or they had to be transformed and adapted into a new SQL-query for the new system.

Particularly complex were user-defined functions in the DB2 system because they use special mechanics of this old database system. No user-defined function was applicable in current database systems due to the outdated and proprietary implementation. The solution was to analyse the source code of a particular function and reimplementing the function in the new database, reimplementing the function in the application or client outside of the database, or trying to use standard queries to achieve the same functionality. A specific problem at the eNoteHistory project was, that some functions did not work correctly anymore, which increased the complexity of the analysis significantly.

ENoteHistory used two kinds of user-defined functions: user-defined functions based on SQL and user-defined functions implemented in Java. User-defined functions based on SQL had to be translated to the current SQL dialect of PostgreSQL and had to consider the made schema changes if needed. User-defined functions based on Java could not be transferred to PostgreSQL, firstly, due to the used outdated Java version and secondly, due to the lack of Java support in user-defined functions in PostgreSQL. Indeed, there is an open source add-on for Java user-defined functions in PostgreSQL, however, installation, configuration, as well as administration and updating is complicated and a significant source of failures and security issues. We decided to reimplement these Java user-defined functions within the application code of eNoteHistory outside of the database environment. This is not less complicated, but we have a better separation of database and the functionality which is more maintenance-friendly. The main advantage of user-defined functions, the performance of database-near implementation could be neglected regarding the average visitor numbers of the eNoteHistory service.

One of these Java user-defined functions was the calculation of the most similar scribes of a given music handwriting. The user-defined functions was reimplemented as a normal Java function within the eNoteHistory application (not as a user-defined function in the database). With this changes, such function was not more available within an SQL statement, but can be called within the program code. This means, that we needed new so called side tables for intermediate data, which were created automatically by the database before. Furthermore, we had to check all SQL statements, whether they used this user-defined function and possibly changed the SQL statement and the function calls.

3.3 Upgrading the Web Server and the Web Site

The original implementation of the eNoteHistory web service was based on Windows 2000, a DB2 V8 database, an Apache Tomcat Server v5.5 and Sun Java v1.2. The Windows 2000 server is replaced by a modern Linux distribution, CentOS in a current, long-term supported version. The DB2 V8 database system is replaced by PostgreSQL version 9.6 and Sun Java 1.2 by Oracle Java version 8. The Tomcat Server is used in version 8.5 with a current servlet specification.

All these updates and upgrades are significant modifications of the eNoteHistory application. Hence, many functionalities did not work anymore. Nearly every function had to be analysed and adapted, repaired or reimplemented step by step. For instance, the search functionality was implemented using the IBM DB2 Net Search Extender which is not supported anymore and was replaced by IBM DB2 Text Search. All programming code related to the Net Search Extender had to be found, analysed, and adapted to a search functionality, PostgreSQL as the new database system provides. Similar problems arose by using the current Java version due to missing some older, in eNoteHistory used code libraries. Some of these missing libraries could be replaced by current ones. Unfortunately, for few libraries, there were no current alternatives and we had to reimplement the functionality. We paid special attention to not use programs or software libraries from third party developers due to the uncertain sustainability.

Tomcat is used as the web server and contains the framework of the eNote-History application. The framework consist of Java components. The source code was available in different versions, but no version management was used. The code structure was barely described, but there were source code comments. The main problem was the first compilation of the source code after all these years. Unfortunately, it was not possible to compile the source code due to too many failures with outdated Java libraries and database interfaces. It took time and was difficult to eliminate all the failures getting a first compilation. Afterwards, reimplementation and corrections could be made.

It made sense to review every line of the source code. However, the given resources allowed only the most important and broken components to be analysed and possibly reimplemented. Further maintenance cycles may concentrate on other source code components.

3.4 Changes in Addition to the Old Application

During the rework of the eNoteHistory application, it made sense to overview other aspects of the system too. We have simply altered some textual information, corrected dead web links and provided new contact information.

Furthermore, we changed the procedure, how the files for the digitised notes are handled. In IBM DB2, the files are handled by the database using a DB2-specific functionality. In PostgreSQL, this particular functionality is not available, but a different one. Nevertheless, we decided to use a simpler way: handling the files by the operation system in the file system and using file locators as strings in PostgreSQL. This is not the recommended way, but it is less complex to maintain and is possibly better sustained. Moreover, the eNoteHistory server is dedicated only to this application and should not interfere with others.

Another aspect of the eNoteHistory application was the user management. The old user management required an authentication for some special search and analysis functionality. Nowadays, it is common to provide such functionality without any authentication. Only the altering of data should be possible for particular user. Therefore, we changed the user management significantly.

4 Lessons Learned and Measures

4.1 Lessons Learned

In the area of software evolution, there are many approaches, guidelines, information, and lessons learned about sustainability. However, most of them are from a software design (e.g. [11]) or an architectural (e.g. [8]) point of view. After finishing this work, we have learned few lessons, especially from a practical modernisation point of view:

Diversity of Failures: After running a public software over 14 years, you can find logical and content-based failures beside software errors, too.

Hard- and Software Provider: Using software, operation system, etc. you should pay attention to providers history (bigger and traditional companies are often more sustained) and to available long-term supported versions.

Standards: If possible, standards should be used, but not always the newest. For instance, SQL: no database manufacturer fulfils the newest version of SQL. Modernising eNoteHistory, we had to disable most of the newer SQL concepts like special user defined functions and the file handle concept. Sometimes, it makes sense to not extensively use available software libraries due to the dynamic and constant further development.

Maintenance Strategy: It is necessary, not only to maintain a system, but also to modernise it and maybe to reimplement some aspects from time to time.

Documentation: Not only the system needs to be documented, but also the software versions needs to be organised and described. This is also relevant for the technical organisation and technical information (which server, ports, file systems are used for what functionality).

Maintenance Plans: It is important to plan the maintenance for a minimum period of 5 to 10 years or expect severe failures after a couple of years.

Sustainability vs. Performance: From a technical point of view, the optimisation of a system regarding sustainability is more important than regarding performance.

Resources for Sustainability: Sustainability requires much resources. Projects which take into account sustainability and needed resources early, get a significant advantage. Resources like maintenance costs, hard- and software costs as well as working time and costs should be considered.

Maintenance Schedule: Hard- and software systems have to be updated regularly. A maintenance schedule depends on the planned system lifetime. Based on the experiences in eNoteHistory, the following maintenance schedule can be derived:

- system lifetime till 5 years: least effort, existing hard- and software should last the lifetime;
- lifetime 5–10 years: intermediate effort, the virtualisation of the hardware might be necessary, software and security issues might arise;
- lifetime more than 10 years: system has to be fundamentally updated (regularly, every 2–3 years on average). The longer the maintenance intervals are, the higher is the particular effort.

Monitoring the System: Moreover, the system has to be observed regularly. Failures and unavailability are bad for the user experience. It is even worse, if unavailability of services is unperceived by the operator. Nowadays, automatic tools are available for software monitoring.

4.2 Measures for Monitoring the System

Establishing a permanent observation of the eNoteHistory system was the first measure taken. Due to the software analysis of eNoteHistory, we were aware of the problems and requirements on an adequate monitoring. Besides, we had to notice that the system became more and more unstable and a permanent observation by a person was not possible. Service unavailability is caused by program failures, directory problems or missing program libraries (e.g., due to systems software updates) as well as failure of components (e.g., after a restart due to a power failure). Another problem is the reaction time until the service is available again.

Based on the experiences over the years, the following requirements emerge:

- automatic monitoring routines and reactions, as much as possible;
- periodic availability test of all web pages;
- periodic test of main functionality: login, search, and analysis steps;
- check of all web links;
- alerting per email or messenger post;
- reaction possibilities, e.g., set a maintenance status on the website or restarting the system.

The information whether the service is running or something is wrong, is very important for such a system. Therefore, we investigated in software systems for the monitoring of provided services. There are a couple of systems to be considered. Three systems are closely analysed: Selenium (https://www.seleniumhq.org), Cypress (https://www.cypress.io), and Scrapy (https://scrapy.org). Selenium and Cypress are tools explicitly for testing web sites. Scrapy is a web scraping tool for extracting data from web pages.

All of the three tools provide the functionality for checking web sites. Scrapy has rich concepts for getting data, information, and content out of a web page, but lacks features for testing and checking web sites periodically. Selenium and Cypress provide very similar, general functionality, but pursue different approaches. Selenium uses the interface of different browsers for accessing and interacting with web sites. Cypress uses a browser add-on which interacts directly with web pages. The disadvantage of Cypress is that it supports currently only the browser Chrome. The browser interface approach of Selenium is more flexible, but less powerful. Selenium is the older and technically more mature tool, which was the major reason to use it.

4.3 Website Tests Using Selenium

Selenium [9,10] provides many different test procedures. It is primarily designed as an enhancement for software tests, especially web applications. Supporting periodic tests, Selenium is also appropriate for monitoring web sites. It uses so called WebDrivers to interact with browsers. Selenium supports several browsers and is suitable for cross browser testing. The architecture is based on a Client/ Server approach. Hence, it is possible to manage and operate a couple of test browsers by one Selenium host and one test script parallelly (e.g., for stress tests). Due to the exclusive usage of browser interfaces, Selenium can not directly handle events appearing within the browser.

The following tests are provided:

– Responsiveness and adaptability test regarding screen sizes, element sizes, and element visibility;
– Tests on elements: availability, visibility of elements; interference of elements, clickability; arrangement of elements (e.g. in a grid);
– Special media tests: size of images, correct scaling;
– Tests on interacting with elements, especially form elements: buttons, radio buttons, selection elements, input elements, result checking;
– Link tests: clickability, accessibility of links (URL checks);
– Cookie tests: check of stored data.

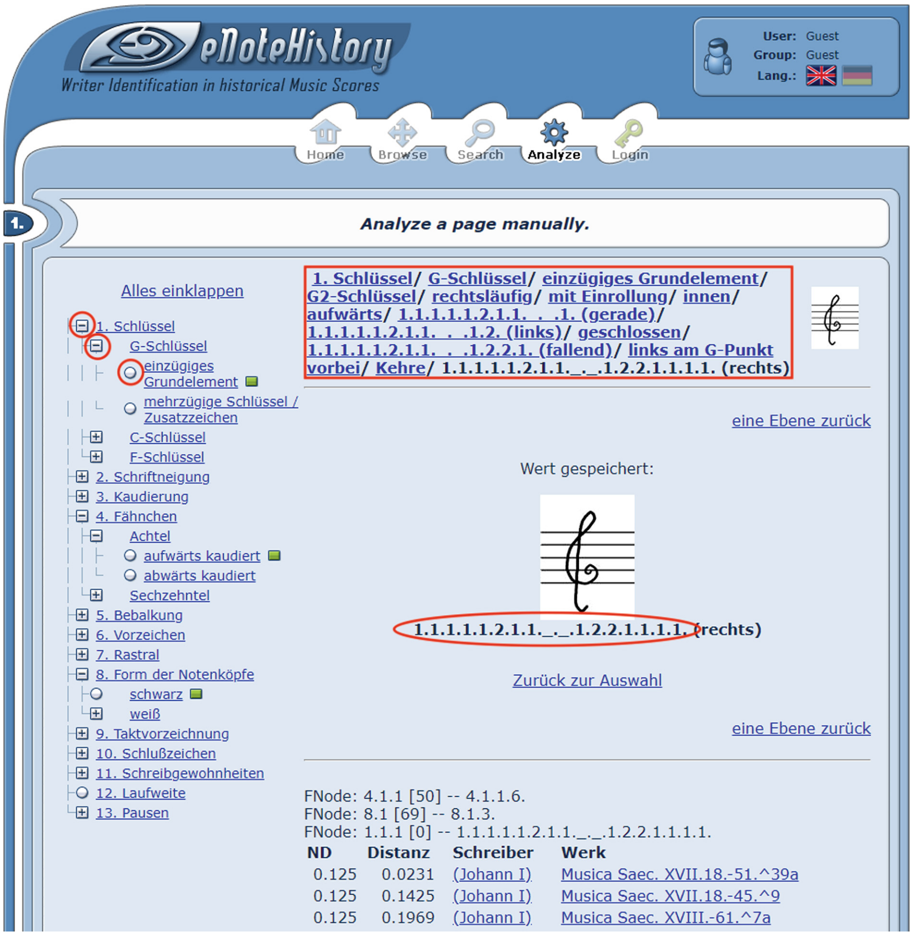
All of these tests and checks are described by test scripts, which can be implemented using a programming language. Selenium supports a couple of different programming languages. The analysis of the test results have to be explicitly implemented as well. Furthermore, Selenium provides different alerting and reaction possibilities, e.g., posting a message on twitter or altering the web server showing a maintenance site to users.

4.4 Selenium Example

Due to many different test possibilities of Selenium, we want to discuss one specific test example. Based on a series of clicks and checks, the manual analysis shall be tested. Selenium acts like a user by choosing features and checking the results. There are two reasonable test categories: First, specific features are given and the result is checked against the expected one. Second, features are randomly chosen by Selenium and the clickability as well as availability are checked.

Figure 2 shows the web page for analysing scribes of notes manually. The analysis consists of choosing properties of the notes to be analysed. On the left side, there is a tree representation of the upper three or four level of features. In the middle respective on the right side, the tree can be further explored and particular features can be chosen. After describing the available features, similar scribes can be searched. The picture shows the selection of a treble clef notation. The three upper levels of the feature tree is chosen on the left side (red circles in Fig. 2) and the other twelve levels to the final treble clef notation is chosen in the middle representation. The whole path is displayed above the choosing area (red rectangle). Besides the treble clef, the tilt of a quarter note and the note flag notation of an eighth note (visualised by a green square behind a specified feature in the tree) are specified.

In the test, a couple of features and the result is predefined. Selenium accesses the analysis web page and clicks on the topmost level of the first feature. The result is checked and should be unfolding the subtree of this element. The check also consists of the correct representation of the small feature images. After



Analyze a page manually.

Alles einklappen

- 1. Schlüssel
  - G-Schlüssel
    - einzügiges Grundelement 🟩
    - mehrzügige Schlüssel / Zusatzzeichen
  - C-Schlüssel
  - F-Schlüssel
- 2. Schriftneigung
- 3. Kaudierung
- 4. Fähnchen
  - Achtel
    - aufwärts kaudiert 🟩
    - abwärts kaudiert
  - Sechzehntel
- 5. Bebalkung
- 6. Vorzeichen
- 7. Rastral
- 8. Form der Notenköpfe
  - schwarz 🟩
  - weiß
- 9. Taktvorzeichnung
- 10. Schlußzeichen
- 11. Schreibgewohnheiten
- 12. Laufweite
- 13. Pausen

1. Schlüssel/ G-Schlüssel/ einzügiges Grundelement/ G2-Schlüssel/ rechtsläufig/ mit Einrollung/ innen/ aufwärts/ 1.1.1.1.1.2.1.1. . .1. (gerade)/ 1.1.1.1.1.2.1.1. . .1.2. (links)/ geschlossen/ 1.1.1.1.1.2.1.1. . .1.2.2.1. (fallend)/ links am G-Punkt vorbei/ Kehre/ 1.1.1.1.1.2.1.1._._.1.2.2.1.1.1.1. (rechts)

eine Ebene zurück

Wert gespeichert:

1.1.1.1.1.2.1.1._._.1.2.2.1.1.1.1. (rechts)

Zurück zur Auswahl

eine Ebene zurück

```
FNode: 4.1.1 [50] -- 4.1.1.6.
FNode: 8.1 [69] -- 8.1.3.
FNode: 1.1.1 [0] -- 1.1.1.1.1.2.1.1._._.1.2.2.1.1.1.1.
```

ND	Distanz	Schreiber	Werk
0.125	0.0231	(Johann I)	Musica Saec. XVII.18.-51.^39a
0.125	0.1425	(Johann I)	Musica Saec. XVII.18.-45.^9
0.125	0.1969	(Johann I)	Musica Saec. XVIII.-61.^7a

Fig. 2. Test the eNoteHistory manual analysis. (Color figure online)

unfolding all the levels of the regarding feature on the left side, the next feature levels are clicked in the middle area of the page until the given feature value is shown and chosen. Again, the result of every step is checked. If any given feature is looked up and chosen, the button "identify writer" is pressed by Selenium. Following, the results are compared with the expected. In case they differ, the administrator gets an e-mail with the precise error message.

5 Conclusions

The eNoteHistory application built from 2003 till 2005 was not maintained since then and has been modernised in 2017 and 2018. After 12 years, there existed many problems running such an old application. The problems were primarily instabilities of the software, data and system security issues, and the increase of service failures and malfunctions.

The eNoteHistory system was based on Windows 2000, IBM DB2 v8, Sun Java 1.2, and Tomcat v5.5. After an analysis of the components, we decided to use a current Linux CentOS, PostgreSQL, Oracle Java, and Tomcat for the new eNoteHistory server. Thus, some fundamental, complex, and extensive changes were necessary.

Overall, the effort for modernisation of such a system after 12 years is tremendous. The question whether updating and upgrading or reimplementing is highly eligible and is difficult to decide. The eNoteHistory software system is running sufficently stable and secure at the moment. An important task after the modernisation is a permanent and automatic monitoring of the services. We used the web software test suite Selenium for the monitoring and defined a couple of test procedures for checking the services, the correctness of the web pages, the web links, and the special analysis functions.

References

1. Bruder, I., Finger, A., Heuer, A., Ignatova, T.: Towards a digital document archive for historical handwritten music scores. In: Sembok, T.M.T., Zaman, H.B., Chen, H., Urs, S.R., Myaeng, S.-H. (eds.) ICADL 2003. LNCS, vol. 2911, pp. 411–414. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24594-0_41
2. Bruder, I., Ignatova, T., Milewski, L.: Knowledge-based scribe recognition in historical music archives. In: Heery, R., Lyon, L. (eds.) ECDL 2004. LNCS, vol. 3232, pp. 304–316. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30230-8_28
3. Chen, S.: The paradox of digital preservation. IEEE Comput. **34**(3), 24–28 (2001)
4. Choy, D.M.: Integration of structured and unstructured data in IBM content manager. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. ACM (2005)
5. Eschenfelder, K.R., Shankar, K., Williams, R., Lanham, A., Salo, D., Zhang, M.: What are we talking about when we talk about sustainability of digital archives, repositories and libraries? In: Proceedings of the 79th ASIS&T Annual Meeting: Creating Knowledge, Enhancing Lives Through Information & Technology. American Society for Information Science (2016)
6. Giaretta, D.: Advanced Digital Preservation. Springer, Berlin (2011)
7. Hamilton, V.: Sustainability for digital libraries. Library Rev. **53**(8), 392–395 (2004)
8. Koziolek, H.: Sustainability evaluation of software architectures: a systematic review. In: Proceedings of the Joint ACM SIGSOFT Conference on QoSA and ACM SIGSOFT Symposium ISARCS. ACM (2011)
9. Sirotkin, A.: Web application testing with selenium. Linux J. **2010**(192) (2010)
10. Vila, E., Novakova, G., Todorova, D.: Automation testing framework for web applications with selenium webdriver: opportunities and threats. In: Proceedings of the International Conference on Advances in Image Processing, ICAIP 2017, pp. 144–150. ACM (2017)
11. Zdun, U., Capilla, R., Tran, H., Zimmermann, O.: Sustainable architectural design decisions. IEEE Softw. **30**(6), 46–53 (2013)