“ediarum - from bottom-up to generic programming”

by Martin Fechner, Stefan Dumont

This lecture deals with a software project, which was developed in the context of scholarly editions. Scholarly editions make historical sources such as letters, diaries, etc. from archives accessible to researchers. For this purpose, the sources are transcribed and commented. Here at the Berlin-Brandenburg Academy of Sciences and Humanities there are editions on the philosophers Leibniz and Kant and on other famous personalities such as Karl Marx or Alexander von Humboldt.

Our software project now establishes a digital workflow for the scholarly editions. This is known as a digital scholarly edition (DSE). Digital scholarly editions follow a genuine digital paradigm.[1] This means that digital is thought not only as a tool, but as an independent method. Digital editions are usually encoded in XML format according to the guidelines of the Text Encoding Initiative (TEI). The aim is to make the edited historical sources comprehensively available for subsequent use and to be able to link them with further information and other databases. A further goal is the single-source principle, i.e. to be able to generate everything else from a data source, in this case XML documents. Both web publications and print publications are relevant presentation formats for the scholarly editions.

Our initial situation in 2011/2012 was as follows: It was clear that we needed a new workflow for many projects, as the previous one no longer met the requirements for a digital scholarly edition. The edition project *Schleiermacher in Berlin* (Friedrich Schleiermacher was a famous Berlin theologian of the 19th century), this edition project offered the opportunity to implement a new workflow. At that time, however, there was no software that would have met the requirements of the edition projects at the BBAW for the creation and publication of digital scholarly editions. Furthermore, we had only few resources that we could use to create a software solution. Consequently, the decision was made not to try a completely new development, but to use the resources carefully and to use existing software and further develop it where necessary. In our case these were the XML database *eXistdb*, the XML editing tool *Oxygen XML Author* and the typesetting engine *ConTeXt*. This approach ultimately led to success. We were able to create a software solution that we now call **ediarum**. This solution still supports the workflow of the project today, relies on a sustainable data format and makes all desired forms of publication (on the web and in print) possible.

This successful digitisation project was characterised by the following key points:

- bottom-up approach, i.e.
- concrete development for an existing project

---

[1] Patrick Sahle: Digitale Editionsformen. Teil II: Befunde, Theorie und Methodik (=Schriften des Instituts für Dokumentologie und Editorik 8). Norderstedt 2013, p. 149.

- we had to provide more resources for development than was usual for previous projects. We had about two developers with one full time equivalent (FTE) over a period of about half a year to a full year. (For comparison: previously we planned only with periods of a few weeks or months for small projects).
- close communication with the research project / edition project was necessary.
- we made use of existing, stable software with good support.
- we developed adaptations and extensions of this software by own solutions.

As a result of this pilot project, there was very positive feedback. We took this as an opportunity to immediately transfer our development and software concept to another research project: For the manuscript research project "Commentaria in Aristelem Graeca et Byzantina" we started from scratch and put together a software package similar to the one for the first pilot project. This project was also subject to the key points mentioned and was very successful. We moved one step closer to our goal - the digitization of the scientific editions at our institution. Nevertheless, with the success grew the desire to use ediarum in many other projects. But we were only able to realize one or two new projects at the same pace. At the same time, we discovered that we often reused the code already developed for one project in another. This copy-and-paste approach accelerated the implementation, but also led to existing errors being duplicated. The operation of several software solutions for individual projects, including further development and bug fixing, as well as the desire to be able to implement new projects easily, led to the consideration of restructuring our development processes. As a result, we no longer programmed for each project alone (bottom-up), but switched *ediarum* to a generic, easily adaptable program core.

The concept of *ediarum* today includes common core components which are used in all projects. There new features are implemented and bug fixes are made, from which all projects profit at the same time. Furthermore, there are project-specific extensions, i.e. for each project there is its own program code that supports the special requirements. The basis for this approach was a standardization of the data model, which is used by the projects. This point, one common data model for all projects, cannot be emphasized clearly enough. This is because all *ediarum* modules build on the data model in one way or another. A generic development of *ediarum* requires a standardized data model. (Project-specific extensions are of course possible.) 2015 the standardization of the data model took place first for a certain project type, of which we have many projects (modern German editions).

Based on this, several *ediarum* modules were developed, which take over individual tasks within a digital scholarly edition:

- **ediarum.DB** provides possibilities for project, user and data management within the XML database.
- **ediarum.BASE.edit** extends the XML editor with the necessary features to provide the researchers with a meaningful data input interface.
- **ediarum.PDF** contains the program code to generate a PDF from the XML files via the typesetting engine, which follows the layout of common print editions of scholarly editions.
- **ediarum.WEB** contains a program library with which a digital presentation for digital scholarly editions can be created without much effort.

These different modules represent the different layers of a digital edition. They are extended in the concrete implementation for a project by a project-specific component of the respective module. Our development workflow today is as follows:

- If there is a new feature request is issued on the part of a project, we check to what extent this requirement also exists in other projects.
- If it is only a project-specific need, the implementation takes place in the project-specific extension of *ediarum*.
- If there are similar requirements in other projects, the generic development process starts.
- I.e. for the development specification all similar requirements of the different projects are brought together.
- Sometimes during the implementation the possibility must be provided that the feature can be adapted project-specifically. This is usually done by integrating variables in the code that are defined in the project-specific extension.

The generic development approach has enabled us to strengthen and continuously develop the core components. Furthermore, it is relatively easy for us to set up new projects and make them ready for work. Finally, the project-specific components, as in the bottom-up approach, make it possible to implement specific requirements for individual projects.

I come to the conclusion and summarize the advantages and disadvantages, as well as the path of a change from a bottom-up approach to a generic development: Let us start with the path towards a move from bottom-up to generic programming. First, experience will be gained in individual pilot projects. More resources than usual must be made available for these pilot projects. Three conditions must be met before a changeover to generic programming can take place: 1. The pilot projects were successful. 2. Further projects are to be implemented. 3. A common core (such as a common data model and repeating program code) can be identified. The next step in the changeover is the development of a core component without project-specific requirements. The implementation of the generic component for concrete projects follows. Not to forget the migration of the pilot projects to the generic components. Once all this has been done, the further development and maintenance of the software can take place in the generic programming.

This approach requires addressing the following challenges: In science, there is only project financing. Generic development is project-independent and requires additional funds of its own. Generic development and project-specific development compete with each other. This means that the completion of important “milestones” for the projects (for evaluations, publication dates, etc.) leads to a postponement of necessary generic development. When switching from individual projects to a generic approach, similar but not identical program code must be brought together. This can be difficult and can mean additional work. Migrating the pilot projects and other existing projects to the new approach can be very time consuming. This improves the sustainability of the project, but there are no visible new features for the project.

Nevertheless, this approach has proved successful for us. Because we see the following advantages: The first prototype is ready for use more quickly in the bottom-up process. Because at first, only the requirements of one project have to be considered. The goals are

supported by the strong focus on the projects: The software is strongly oriented to the concrete needs of the users. The software is not designed too theoretically; the danger of developing without the real needs is much lower than in generically planned software. Not too little and not too much is developed. The necessary features are implemented, but no unnecessary ones. The use of generic core components and project-specific extensions will achieve a reasonable balance between desired standardization and required project-specific adaptations. The generic core components significantly simplify the maintenance of many projects.