Managing the Transition of Educational Technology from a Research Project to Productive Use

Dirk Bußler[1], Ulrike Lucke[2], Sven Strickroth[3] and Ludwig Weihmann[4]

Abstract: In research projects often new technologies are developed. A key problem, however, is the sustainability and the transition of the products to the data centers for continued operation and maintenance. Based on the experience of the development and transfer of the University Service Bus, a generic meta-infrastructure not only for e-learning, key issues and strategies (embedding into the infrastructure, documentation, competency transfer & building, rolling out, and marketing and communications) to tackle relevant transfer issues are presented and discussed. Finally, recommendations for project funders are formulated.

Keywords: Sustainability, Stabilization, Continuation

1 Introduction and Motivation

Sustainability is one of the highest demands in project funding [BO16]. Especially for application and development there is the clear expectation of funding agencies to transfer the results of research to productive use after funding. This is especially true for the digital transformation of teaching and learning. However, evaluation of funding schemes reveals that this is far from realistic; without an adaptation of local structures, processes and resource distribution research results will vanish soon [Sc18]. The reasons are well-known. Project funding is hardly limited according to available financing and timeframe, eligible costs and measures, and involved parties. So project management is forced to achieve quick results, inevitably resulting in a narrowed perspective and simplified conception. Also, the ongoing demand for innovation collides with the legitimate interest of data centers in the stable operation of its services [KLL17].

There are recommendations for continuing research prototypes [SS13], but they do not address the maintenance perspective. For IT service management, a number of procedural models such as IT infrastructure library (ITIL) [CB08] and the Capability Maturity Model Integration (CMMI) [FBS09] are established for long. They describe best practices for

[1] University of Potsdam, Center for Information Technology and Media Management, Am Neuen Palais 10, 14469 Potsdam, Germany, firstname.lastname@uni-potsdam.de, https://orcid.org/0000-0001-8714-0257

[2] University of Potsdam, Institute of Computer Science, A.-Bebel-Str. 89, 14482 Potsdam, Germany, firstname.lastname@uni-potsdam.de, https://orcid.org/0000-0003-4049-8088

[3] Ludwig-Maximilians-Universität München, Institute of Computer Science, Oettingenstraße 67, 80538 München, Germany, firstname.lastname@lmu.de; https://orcid.org/0000-0002-9647-300X

[4] University of Potsdam, Institute of Computer Science, A.-Bebel-Str. 89, 14482 Potsdam, Germany, firstname.lastname@uni-potsdam.de, https://orcid.org/0000-0002-3146-0283

Copyright © 2021 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR-WS.org/Vol-2814/paper-A6-1.pdf

design, maintenance and continuous improvement of IT systems. Though well established in commercial ITSM as well as in modern data centres even at universities, they are hardly applied in the management of research projects because of the great effort involved and the limited resources available for their implementation. This is even worse for teaching innovations where funding is often rather limited and timelines are tightly bound to academic schedules. In this article, we will analyze the success factors during the migration of educational technology from a publically funded research project to regular operation in the data center. From our set of developed tools and services [LS20] the University Service Bus (USB) is most prominent as the central switching point between single service providers and consumers across the local IT infrastructure. RESTful web services are used to provide a maximum of flexibility in composing higher value services, like learning environments or mobile apps [KZL14]. Figure 1 sketches the usage of interfaces between various services through this USB.

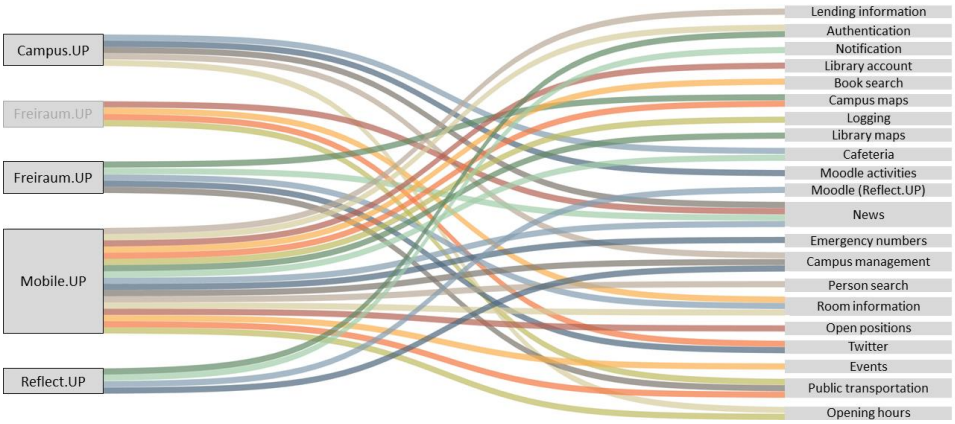

Fig. 1: The University Service Bus connects major educational platforms (left) with a variety of basic services (right) provided across the institution

Having such an architecture in place allows to quickly establish new services or features that are not limited to e-learning, as we successfully demonstrated with the deployment of online services for administration [LSS18]. However, the complexity of this approach makes perpetuation even more difficult. In this article we describe our experiences in transferring the USB approach and components to the data centre, where it is now operated. From the specific goals and context of this process we derive conditions for a sustainable design of educational technology from the very beginning of a project.

2 Issues faced when preparing the infrastructure for continuation

A researcher asking the data centre to take over an application from his research project might typically hear: “We will not be able to handle this!” Thus, designing such applications in a way that operations require minimal efforts is crucial for sustainability.

Since technical and organizational issues are closely interwoven, the focal point is to identify those issues in IT design that have an impact on permanent operation and maintenance. In the following subsections we will analyze this from five perspectives:

- Which technical aspects of the overall IT infrastructure of an institution have to be considered during first design decisions?
- Which aspects of the developed system have to be documented in which forms or using which methods to promote easy operation?
- Which competencies are required to operate the system, and how can these competencies be developed in the data center?
- Which requirements come up in the transition from restricted lab operation to roll-out in productive operation of the entire institution?
- Which communication efforts should accompany the transition of research results to broader usage?

From managing the transfer of the USB and its various APIs to the data center we have learned several lessons which will be shared in the following.

2.1 Early embedding in the overall infrastructure

In order to build or select software which can be further operated, maintained, supported and maybe also further evolved at the central data center, projects need to manage the balancing act between being highly flexible in the project for getting results quickly and building on technologies and competencies already available at the data center. This does not mean, however, that only already known technologies can be used. But, these need to be agreed on as early as possible for strategic planning, competency building and transfer (cf. section 2.3 & 2.5). This shall happen in all application-oriented EduTec projects either from the project start or as soon as usage beyond the local lab becomes feasible or desirable, so that no distinct hard switch for the transfer is necessary.

Most fundamental design questions comprise the *general architecture*, *available services* and the *technology portfolio*. It is important not to require the data center to build parallel infrastructures just for a specific project. Specifically *virtual machines* vs. *container-based setups* need to be considered. When virtual machines are chosen, the *operating system* should fit into the technology portfolio so that these can be integrated more easily into existing update and maintenance processes or that the systems are also covered by existing support contracts. – Building upon operating systems, usually other services such as *database management systems* are required. Here, data centers often already operate services which should be considered first. If this is not the case or whether there are special needs, this has to be discussed with the data center. – When software products need to be selected, the following specific (functional and non-functional) properties should be considered: *scalability* (e.g., ability to build clusters and compatibility with existing load balancers), *updatability*, *maintainability*, *configurability*, *licensing* (models, required license servers, (recurring) costs), and whether *support companies* do exist. – Avoiding

parallel structures also applies to data. A recurring point is *user management*. Here, using the central identity management solution (e.g. LDAP or Single-Signon such as Shibboleth) is strongly recommended. Experience has shown that it is better to choose a well-understood technology instead of a (potentially) more advanced one because the former one can usually be easier operated and, therefore, will likely perform better in practice.

Discussed separately are *programming languages*. Particularly, in the context of the USB with well-defined interfaces it is not necessary that every administrator and all users master all used programming languages. Users of an interface or service can transparently use it (using their preferred languages) and, therefore, do not need to fully understand its implementation. Therefore, it is sufficient that the specific language of a service is mastered by the team operating it.

Finally, processes and workflows need to be developed and agreed upon – not only for the transition to the data center (and strategic personnel planning there, cf. section 2.3) but also for the project lifetime. This way "dirty hacks" can be minimized and processes can be evaluated early. (Automated) workflows should be established for the development (such as *deployment strategies*: dev -> staging -> production environment, *testing*, keeping the *documentation* and used systems up to date; cf. DevOps and DevSecOps), the *administration* (such as maintaining data or granting permissions), *evolution of services* as well as for *providing support to users* (such as first and second level support structures). These workflows help transferring responsibilities to the data center step by step and make operation more manageable for the new "owners" (e.g., beginning with the first level and then the second level; cf. section 2.3 for competency transfer strategies). As a basis, *responsibilities* for operating, maintenance and support should be discussed as soon as the technologies are selected and being used as prototypes. Here, decentralized approaches should be considered in order to distribute specific tasks to multiple shoulders (e.g. permission management outside the data centre [LSS18]). In the specific case for the USB, workflows for *API lifecycles* and versioning are needed. Support for this is integrated in API management systems. Still, the transition from one API version to another and deprecating old APIs requires manual effort to check whether the APIs are actually in use and to actively communicate deprecations to users in order to not break their functionality.

2.2 Documentation

Documentation can make developing, using and maintaining existing software and infrastructure either an enjoyable or very unpleasant experience. So, when aiming for continuation proper documentation becomes a key issue. However, good documentation is hard to find and even harder to produce. When trying to create helpful documentation various (sets of) questions have to be considered: *1)* What does this API/component do? What is its purpose? Why was it created? *2)* How does it tie into the infrastructure? *3)* Who is responsible for this API? Who can I contact if troubles arise? *4)* How do I use this API? Which paths or functions exist? *5)* How does it actually work? (Programming language, framework, architecture, web service paradigm, runtime environment, usage of

other endpoints) *6)* How do I build and deploy this API? (build system, dependencies, runtime environment, CI/CD) *7)* Are there recurrent or urgent issues to be considered?

When trying to answer these questions from an outsider's perspective using only the existing resources flaws in the documentation (should) become obvious. Our previous approach to documenting APIs and infrastructure could certainly not answer all these questions and understanding or even finding it was not easy for an outsider. Developers familiar with the infrastructure and the "historically grown" documentation might know where to look when searching for information but anybody else surely does not. So it is important to establish well known places where a certain kind of documentation is kept and can be found for all existing APIs and infrastructure components. We decided to create a shared folder which serves as a starting point for new developers and maintainers. This folder contains text documents about each relevant component answering questions *1*, *2* and *3*. They should also point to the place where further information about each component can be found. One such place is the central API-Manager which then answers question *4* by providing an interactive list of available paths and functions. Those can be automatically generated from source code annotations based on standards such as Swagger/OpenAPI. This way consumers of an API can start at the central API-Manager and get all information needed for using an API.

The more technical questions *5*, *6* and *7* are answered in the next stage of documentation which consists of each API's GitLab repository. Here, extensive documentation should be available in the form of a README file and, in the best case, a Wiki. This documentation should explain in detail how the respective API works, which dependencies exist and how this component can be built and operated. Especially when the CI/CD (continuous deployment/ integration/testing) paradigm is used to automate as well as to document build, test and deployment the whole process can be even more complex and involve multiple systems, thus needing further explanation. This also enables others to modify a system without breaking existing functionality. Lastly, a GitLab-Repository should contain a list of tickets that give insight to the development that has already taken place and the next steps that need to be taken. Generally, documentation should be as close to the source code as possible and/or automatically generated to avoid inconsistencies caused by forgetting to update it in all places.

2.3 Competency Building & Transfer

A necessary prerequisite for competency building and transfer is a conceptual agreement on used architectures and technologies upfront and a good documentation (cf. section 2.1 and 2.2). Only then the competency building and transfer can be planned strategically, e.g. by educating staff in the data center or for designing fitting job advertisements for new employees (on vacancy or new positions). In order to maintain freedom from the project perspective and still being able to transfer knowledge to the data center, the following strategies were considered and applied: 1) early meetings, demonstrations and agreements on (potentially new) technologies and architectures, 2) workshops for teaching and exercising (new) technologies, 3) office rotation (working two days a week in the office

of a project member), 4) joint maintenance tasks, such as upgrading project services and 5) timely migration of services to the central infrastructure including a step by step transfer of responsibilities.

In our case a half-time position in the data center was included in the project proposal (with the hope to continue it after the project ended). This way the project member was able to translate between the project's and the data center's needs as well as to catch additional work caused by the project. Additionally, the data center was in the lucky position that during the last two years of the nine year lifespan of the project a vacancy needed to be staffed. The newly hired person worked two days a week in office rotation. At the beginning it was designed as an internship with demonstrations and introductory tasks, later on it moved to working together on project-related tasks. Jointly conducted maintenance tasks such as upgrading project services as well as jointly setting up new virtual machines during the transfer shows to be a good way to transfer installation, configuration and operating knowledge and also making sure policies of the data center are fulfilled. The conducted steps were documented by the data center and commented by project members in order to make sure the documentation was understood by the data center members and is correct.

2.4 Rolling Out

The transition from software development to productive use poses new requirements (including scalability, reliability and maintainability) which should also be considered during developing, cf. DevOps: How can insufficient resources or technical errors be detected automatically (on levels such as network connectivity, resource consumption, process, service, ...)? How can a safe productive operation be restored after technical errors or security incidents? A backup and restore concept is necessary. How can the lifecycle of users (especially deprovisioning of users and their data) be implemented and abandoned database entries avoided, e.g. by separating user and system data?

To ensure operation, mechanisms for tests, policies and their automation must be in place when services are taken over. This helps to avoid security risks and to guarantee system stability. In extracts penetration tests are mentioned here. Such automated mechanisms are difficult to implement and realize afterwards or in production, therefore they have to be implemented already during development. This process is called DevSecOps.

In order to meet the legal requirements for going live, legal framework conditions must be observed. When introducing IT services, these are technical and organizational measures (TOMs) and the record of processing activities to be described. The TOMs are created by the IT infrastructure operator and the record of processing activities must be created by the service operator. This serves the GDPR conformity and the documentation of the processing of personal data.

Innovation is not only service deployment and operation but further development and evolution of services. Therefore, established and qualified administrator personnel are

needed for operation and higher qualifications with the corresponding full-time remuneration are needed for further development of services.

2.5 Marketing and Communications

The technical and organizational issues described above require several ways of communication with a number of partners to be successfully established. Thus, a strategy should be developed describing which topics should be communicated to whom, wherefore, when and how. 1) Getting people to use the system, inside and outside the institution (including acquisition of development partners in other institutions) is crucial to demonstrate and increase future potential. This should start with the first gathering of requirements. 2) Negotiations with the local data centre (depending on the strategic scope of the system: with representatives from operational and/or strategic level) are necessary as soon as first technical decisions have to be made. 3) In order to justify a commitment of the institution beyond project funding, decision-makers (internal and external) need to be aware of this at a very early stage, i.e. when stable prototypes are available as well as future usage and benefits can be estimated.

We recommend considering such communication efforts in the project backlog along with technical and organizational issues. This includes a regular review of objectives, intermediate results achieved and measures implemented. If appropriate according to the scope of the intended changes, these efforts should be underpinned by participation in local strategy-building processes [LHH20].

3 Discussion, Summary & Outlook

Based on the experiences gained from developing an University Service Bus in a project within the timespan of nine years, key issues and strategies for being able to stabilize developed infrastructure and services in the local data center are presented. The lists are not exhaustive, but should give an idea on what to consider when building new services. Several mentioned points should already be best practices in software development, however, often those seem not to be considered in research projects. The key points and strategies of section 2 should be addressed and considered as early as possible during the design-cycle in order to be able to act at the end of the project. This also includes knowledge of available resources (money, time) as well as negotiation ranges (knowledge of local structures, processes and responsible persons; formal and informal).

A very specific problem in our case is that the USB is a meta-infrastructure: it is not a single service, but a complex middleware other services can build on. Therefore, there are additional barriers as it causes more communication across teams (from infrastructure to applications and support). Still an open issue is a missing holistic thinking and sense for the big picture of service providers and operators: Even if an exchange is attempted early, staff might only consider their local view and don’t feel responsible for opening their service via an API which is helpful for others. Also, diffusion of responsibility (for

technical issues, but also for security, culture etc.) is especially an issue for a distributed service-oriented architecture such as the USB. Additionally, workflows for the release of new apps must be clarified, e.g. which new apps are possible on this platform and how they can be kept up-to-date.

The authors call for more courage for participatory/cooperative development (e.g. innovation management, strategy formation) of universities in order to solve the mentioned issues. Also, project founders should take the needed resources for stabilizing developed services into account and encourage applicants to work together with central institutions already when writing a proposal in order to effectively reach sustainability. This includes having the necessary resources (staff and time) available in the project.

Bibliography

[BO16] Budde, J; Oevel, G: Innovationsmanagement an Hochschulen: Maßnahmen zur Unterstützung der Digitalisierung von Studium und Lehre. In: Proceedings Informatik 2016, LNI P-259, Bonn: Köllen, 2016, S. 947–959.

[CB08] Clifford, D; van Bon, J.: Implementing ISO/IEC 20000 Certification: The Roadmap. ITSM Library. Van Haren, 2008.

[FBS09] Forrester, EC; Buteau, BL; Shrum, S: CMMI for Services. Guidelines for Superior Service. Addison-Wesley, 2009.

[KZL14] Kiy, A; Lucke, U; Zoerner, D: An Adaptive Personal Learning Environment Architecture. In: Proc. Int. Conf. ARCS, LNCS 8350, Springer, 2014, S. 60–71.

[KLL17] Kiy, A; List, C; Lucke, U: A Virtual Environment and Infrastructure to ensure future readiness of Data Centers. European Journal of Higher Education IT (EJHEIT) 2017-1.

[LSS18] Lemcke, S; Strickroth, S; Lucke, U: Effizienz in Zeiten der Digitalisierung: Schneller, besser, kostengünstiger? In: Proc. Lernen und Arbeiten im Wandel (LAiW). CEUR-WS Vol. 2232, 2018, S. 29–43.

[LHH20] Lucke, U; Hafer, J; Hartmann, N: Strategieentwicklung in der Hochschule als partizipativer Prozess. In: Lehre und Lernen entwickeln. Potsdamer Beiträge zur Hochschulforschung 6. Universitätsverlag Potsdam, 2020, S. 99–118.

[Sc18] Schmidt, U; Heinzelmann, S; Besch, C; Andersson, M; Schulze, K; Wesolowski, A: Evaluation des Bund-Länder-Programms für bessere Studienbedingungen und mehr Qualität in der Lehre (Qualitätspakt Lehre). BMBF, 2018.

[SS13] Scanlon, E; Sharples, M et al.: Beyond Prototypes – supporting effective research on technology-enhanced learning. T.E.L. Research Programme, 2013.

[LS20] Lucke, U; Strickroth, S: Digitalisierung in Lehre und Studium. In: Lehre und Lernen entwickeln. Potsdamer Beiträge zur Hochschulforschung, 6, Universitätsverlag Potsdam, 2020, S. 235–256.