Project Description – Project Proposals in the Area of Scientific Library Services and Information Systems

LIS Call "Research Software Sustainability"

Anke Lüdeling, Berlin; Volker Gast, Jena

Project Description

A minimal infrastructure for the sustainable provision of extensible multi-layer annotation software for linguistic corpora

Introduction

The project's goal is the **design, implementation, evaluation and documentation of a minimal infrastructure** for the sustainable provision of research software. By hypothesis, an infrastructure can only be operated sustainably in an academic context if the technical and human resources that have to be provided by the respective academic institution can be minimalized for the long-term from the onset.

In a **case study providing a multi-layer annotation software for linguistic corpora**, the project exemplifies that for such an infrastructure **only four components are strictly necessary**: a *source code repository platform*; a *repository providing versions of the software* for end users; a *repository providing the software's dependencies* (e.g., software libraries); a *maintainer* who administers and publishes the infrastructure and research software, and manages the user and developer communities. Of these components, **only the maintainer arguably needs to be funded** by the respective academic institution. For all other components, potentially sustainable external infrastructure is available free of charge.

A further requirement for the sustainable operation of the infrastructure developed in the project is the **technical sustainability of the research software** it provides. In the course of the project, the **prototype *GraphAnno*** will be developed into a stable product. The product, ***Hexatomic*, has a strong use potential** across different linguistic disciplines by satisfying a verifiable high demand on the part of these scientific communities. Additionally, *Hexatomic* will satisfy the requirement for technical sustainability early on in the project, by implementing best practices of software engineering. These include, e.g., *reproducible builds* through an automated build system; *comprehensive documentation* of all aspects of the software; a permissive *open source* license; portability and *runability on different operating systems*; comprehensive test suites; public *provision of the source code*; *extensibility and adaptability* through modularization and a generic data model; extensive *compatibility with other tools and data standards*; well-structured *community processes*. The project **evaluates and documents** not only the satisfaction of the software's use potential, but also its potential for **long-term, project-independent development**. This is partly achieved through the acquisition of external contributions of functional modules to *Hexatomic*.

Moreover, the **minimal infrastructure model** is not only implemented, its **implementation is also documented and tested**. The test results are, in turn, documented and condensed into **best practices**, which represent an important project goal in and of themselves. In combining a hypothesis-driven approach with a case study, the project makes an important contribution towards the evaluation of minimal requirements for sustainable infrastructures for research software.

1 Starting point and preliminary work

Research software[1] has only recently started to play a more substantial role in the context of the larger discussion about the sustainability of research outcomes, which had hitherto mostly dealt with research data. Consequentially, the sustainability of research software has since become

[1] We follow the call for proposals in its definition of research software as "refer[ring] to the software applications and software libraries specially created for scientific knowledge gain".

the focus of a number of workshops (e.g., RE4SuSy, #hgfos16) – even a dedicated international workshop series, WSSSPE (http://wssspe.researchcomputing.org.uk/) – and conferences (e.g., RE Conferences 2013/2014). Additionally, funding agencies have started issuing calls for projects dealing with the subject, such as the one we are answering with our proposal, and have entered into an international collaboration for exchanging knowledge and experiences in the field (Hettrick 2016).

As the discourse on research software sustainability itself is relatively young, not all fundamental research has been conducted yet. The role of infrastructures for the sustainability of research software is one of the open research questions that need to be addressed. More precisely, how can infrastructures be designed and created to cater for the sustainable provision and reusability of research software?

The project we propose here aims at helping to answer this question in that it

- contributes a point of view that is based on the hypothesis that sustainability of research software can best be achieved by minimizing rather than creating infrastructure,
- defines and implements a design for a minimized infrastructure for the sustainable provision of research software that also enables re-usability,
- tests and documents this strategy and the implemented solutions, thus contributing towards best practices for sustaining research software in a financially-constrained environment with limited human resources.

As the phrase "infrastructure for the sustainable provision of research software" is ambiguous[2], we will attempt definitions of some of the key terms, namely "infrastructure", "provision", and "sustainability" in this section.

1.1 Infrastructures and provision of research software

"Infrastructures", in the context of this proposal, can be defined more precisely as technical and organisational infrastructures for the provision of research software. Infrastructures generate running operating costs. "Operating costs" include

- costs for acquisition and operation of necessary hardware, such as servers or GPU grids;
- costs for acquisition and operation of necessary software;
- costs for organisation, maintenance and extension of the above assets as well as necessary integration and development of new software, i.e., essentially personnel costs.

Furthermore, infrastructures can be classified as internal and external infrastructures. "Internal infrastructures" are infrastructures that are provided by institutions from the academic domain itself, i.e., universities, research institutes, research data centres, etc. "External infrastructures" are infrastructures that are provided by bodies outside of the academic domain, i.e., companies, foundations, extra-academic communities, etc. In order to sustainably provide research software, infrastructures, both internal and external, must be sustainable themselves. Based on the definition above, sustainability of internal infrastructures can only be achieved by minimizing their operating costs. Additionally, a single unsustainable component within an internal infrastructure leads to crucially diminished sustainability of the whole infrastructure.

We propose a model and implementation of a minimal infrastructure for the sustainable provision of research software that aims to minimize the operating costs for internal infrastructure, while at the same time employing sustainable components and enabling the re-use of the research software it provides.

"Provision of research software" by infrastructures can be read as "serving research software", i.e., providing access to the functionality of a running instance of research software to end users through web technologies such as browser-based GUIs, RESTful APIs, etc. For the context of this proposal, however, we define *provision* as providing access to source code, build

[2] The phrase may well be read and understood along the lines of "servers and supporting technology that serve research software", i.e., a single instance of a research software service which runs on a server and provides the functionality of the research software to end users. This is not how we define the phrase, and ultimately we do not believe that such infrastructures can be run in a way that merits our definition of sustainability. Monolithic remote services such as these lose, in fact, traction as a technology, cf. recent developments such as the advent of microservices, serverless architectures, etc., which are all supported by distributed rather than centralized services and technology.

artifacts, and documentation of a research software to enable long-term reproducibility, and thus re-usability and adaptability of it, independent of specific instances and personnel. The task of *serving* the software to end users (researchers) is transferred to either the end users themselves (for locally run software) or administrators, service providers, data centres, etc. (for remotely run software), which – in our definition – taken together form the *user base* of the research software.

1.2 A minimal infrastructure for the sustainable provision of research software

We argue that a minimal infrastructure model for the sustainable provision of research software consists of only four components:

1. A source code repository platform
2. A repository for build artifacts ("releases")
3. Repositories for dependency artifacts
4. A maintainer

1.2.1 Source code repository platform

We define "source code repository platform" (SCRP) as a remote platform which hosts source code and provides additional services, minimally an issue tracker and hosting of a public entry point and documentation. Examples for SCRPs include GitLab (https://about.gitlab.com/) and GitHub (https://github.com/). The SCRP represents the Single Point of Contact for users and contributors of the software. It provides means to version and access the source code, and the user and developer documentation, and links to the release repository (see below). It does not, however, provide instances of the research software itself; the operation and administration of the software is instead delegated to the user. In this infrastructure model, the term "user" includes both end users (e.g., domain researchers who use the software to process data) and administrators (i.e., providers of running instances of a software, which is accessed and used by end users). This increases the sustainability of a research software, inasmuch as Single Points of Failure are eliminated from the model: If a single instance of the software fails, it can be re-duplicated from a release, and the data – which has ideally been sustained independently of the software – can be re-imported.

The SCRP also provides a means of communication between the maintainer and the developer and user communities by way of an issue tracker. The issue tracker is used by the maintainer to track and document the implementation of new features, enhancement requests and bug fixes as well as merges of contributions into the code base, and to organize contributions and bug fixes in release roadmaps. Users use it to contribute and track bug reports and enhancement requests. Developers use it to contribute to the code base through pull requests. User and developer documentation for a research software is also hosted on the SCRP, e.g., in the form of browsable documentation (web pages, wiki, or similar), or downloadable resources (PDF, CHM, or similar). Finally, the SCRP provides a public entry point to the research software project in form of a web page or similar document.

1.2.2 Release repository

A release repository hosts different versions of build artifacts of a software. Release repository types range from a directory on a server to dedicated binary repository managers and repositories dedicated to research outcomes, including research software. Ideally, they provide a unique identifier for each version that is released.

1.2.3 Dependency artifact repository

We define “dependency artifact repositories” as build artifact repositories which enable automated collection of dependency artifacts (e.g., libraries or modules re-used in a software) during the software build process. Dependency artifact repositories also store metadata which describes an artifact. There are standard dependency artifact repositories for many of the major programming languages, such as Maven (http://maven.apache.org/) for Java, the Eclipse P2 repositories for Eclipse Plugins (https://wiki.eclipse.org/Eclipse_Project_Update_Sites), PyPI (https://pypi.python.org/pypi) for Python, Hackage (https://hackage.haskell.org/) for Haskell, CPAN (http://www.cpan.org/) for Perl, etc.

1.2.4 Maintainer

The maintainer is responsible for integration, and release strategies and roadmaps. She tests the code base and pull requests, integrates contributions, builds snapshot and stable releases and deploys them to the release repository. She also communicates with the user and developer communities, and is responsible for the maintenance of the infrastructure's repositories and projects' public entry points.

1.2.5 Internal and external components

In order to ensure minimized operating costs for the infrastructure, it is necessary to use as many external infrastructural components as possible. As both SCRPs and artifact repositories exist outside of the academic domain as indispensable all-purpose software development tools, the only component that should arguably be internal is the maintainer. As our infrastructure model is designed to be implemented in discrete configurations for each single entity of research software, depending on the requirements of the software implementation itself, the role of maintainer can usually be filled with relatively little overhead. If possible, the role could be assigned to research data officers or data centre employees, ideally with permanent positions.

1.2.6 Requirements

In order for an implementation of our infrastructure model to work reliably, it must be able to endure component changes at any given time. While it may be relatively simple to switch SCRPs and artifact repositories and move the respective objects, the infrastructure must also be made failsafe against changes in the maintainer role, which can safely be predicted to be the most common change in infrastructure components. For this reason, the *technical sustainability* of the respective research software itself is an important requirement for sustainable provision: New maintainers – but also users and developers – must be able to reconstruct the research software's architecture, its requirements, dependencies, build processes, etc., and must also be enabled to change, test, build and deploy it, independently of earlier maintainers and indeed the originators of the research software.

1.2.6.1 Technical sustainability of research software

Software sustainability in itself is still an underspecified concept. There is sparse literature on the subject itself (e.g., Tate (2005); Penzenstadler (2013); Penzenstadler and Femmer (2013); Goble (2014); Gröger and Köhn (2015); Druskat (2016a); Druskat (2016b)). Becker et al. (2015) is a helpful resource, as the authors provide a set of preliminary definitions of *software sustainability*, if not explicitly for research software, for which – as we argue further below – specific constraints exist. They define five "dimensions" of software sustainability (Becker et al. 2015, 471), of which *technical sustainability*[3] corresponds directly with the research software sustainability requirement. Druskat (2016a) refines the concept of *technical sustainability of software* via goals, which he defines as (1) ensuring the existence of the software, (2) preserving the potential for productive operation of the software, (3) creating and retaining possibilities for further development and adaptation of the software.
In comparison to non-research software, research software is ultimately in greater need of sustainability due to the high fluctuation of staff working on it, which in turn is due to the particular funding structures and funding strategies in academics; A great number of research software projects cease simply because the people who work on them move on to new projects. Additionally, the fact that research software is often funded with public money makes sustainability essentially a civic duty.

Jackson et al. (2011) cater towards Druskat's (2016a) definition by determining some of the criteria groups contributing to the technical sustainability of research software, of which some (in bold print) plausibly enable (new) maintainers in our infrastructure model to fulfill their role as outlined above: Discoverability, **Automated build system**, Understandability, **Documentation**, **Buildability**, **Installability**, Learnability, Identity, Copyright, **Licensing**, **Governance**, Community, Accessibility, **Testability**, **Portability**, Supportability, **Analysability**, **Changeability**, **Evolvability**, **Interoperability**.

[3] Becker et al. (2015, 417f.) define *technical sustainability* as "refer[ring] to longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions. It includes maintenance, innovation, obsolescence, data integrity, etc."

Druskat (2016b) further suggests that the sustainability of research software depends on a larger set of criteria than non-research software, e.g., funding status and availability of human resources for maintenance and development. This is in line with our understanding of sustainability of infrastructures as outlined above (*Internal and external components*).

If a research software fulfills the criteria listed above, it can be assumed that it will be maintainable as easily as possible, and thus any person that fills the maintainer role in our infrastructure will be enabled to fulfill the role optimally. Proving this assumption true will be one of the main goals of our project (see also section *Objectives and work programme*).

In conclusion, we define "infrastructure for the sustainable provision of research software", as a minimal technical and organizational infrastructure which enables the use, re-use, and reproduction of a research software independently of its original creators, and earlier product owners or maintainers. Such a minimal infrastructure consists of a source code repository platform, a build artifact repository, a dependency artifact repository, and a maintainer. Of these components, only the maintainer should arguably be situated within the academic domain to minimize running operating costs. In this type of infrastructure, components can be substituted relatively easily. However, in order to enable abrupt changes – even temporal gaps – in the maintainer role without risking the loss of access to the software, the research software itself must be technically sustainable.

1.3 Implementation of the infrastructure model

1.3.1 Multi-layer annotation software for linguistic corpora

In the course of the proposed project, we will implement our minimal infrastructure model in a pilot study: An infrastructure for the sustainable provision of a re-usable platform for the manual and semi-automatic processing and annotation of linguistic multi-layer corpus data. As the infrastructure model is most effectively employed for discrete applications, this kind of software presents a reasonable research object for which a case study can be completed within the fixed term of a project, including documentation and evaluation of solutions.

While there is a wide variety of corpus annotation tools available, the demand across different linguistic fields for an interdisciplinarily usable, maximally compatible platform for true multi-layer annotation of linguistic corpora is still high and unfulfilled. In the context of this proposal, providing an overview of all available annotation tools is impossible, especially since some corpus annotation projects will invariably build specialized tools without ever publishing them, thus foiling sustainability efforts by producing "hidden software"[4]. Additionally, knowledge about which tools exist is distributed over different research communities, and hence partially irrecoverable for people outside these communities.

Biemann et al. (forthcoming) provide an attempt to catalogue and categorize a larger set of available annotation software. The largest category of tools is "Specialized Single-User Tools", i.e., software that focuses on one type of annotation. We will neglect this category altogether, as these tools decidedly fall outside of the scope of multi-layer annotation software. Biemann et al. (forthcoming) also make the distinction between web-based and standalone tools, as well as between those that support workflows with multiple users and roles. The authors argue that web-based tools running on centralized infrastructure provide – among other features – shared and efficient data storage, multi-role support, support for automatic pre-annotation services (essentially NLP) and easily accessible user interfaces without the need for local installation of software. The examples for such software they give are *WebAnno* (Yimam et al. 2013) and *GATE Teamware* (Bontcheva et al. 2013). Both use a central server for data storage, while *WebAnno* also uses a server-based web application to perform the actual annotation tasks.

However, in analogy to our decision to minimize centralized (internal) infrastructure for the provision of research software, we argue that web-based annotation software served from centralized infrastructure is essentially unsustainable, as it creates running operating costs and constitutes a Single Point of Failure which implicitly also increases the risk of data loss. For some use cases, annotation software running on a web server is intrinsically impractical, e.g., for linguistic fieldwork where the annotator cannot connect to the internet, or for offline work

[4] Gil et al. (2016) implicitly define "hidden software" (called "dark software" ibid.) as unpublished software. According to Gil et al. (2016, 331), "[m]any scientists do not share their software, because they are unaware of its value, or they do not know how, or they are worried about not getting proper acknowledgment, or they do not see its value."

during travels. Additionally, the advantages of web-based annotation software named by Biemann et al. (forthcoming) can just as well be realized by locally running standalone software:

- Shared data-storage in locally running, offline software can easily be realized through the use of version control systems such as *git* (https://git-scm.com/). *git* is especially suitable for versioning and sharing text-based annotations, as it provides a system for merging changes, and works on a local repository which can be linked with one or more remote repositories. This way, the user can efficiently version data independently from available network connections.
- *git* also enables multi-role setups through its repository and branching models. The different roles defined by Biemann et al. (forthcoming) – annotators, project managers, curators, administrators – can utilize *git* for their specific needs: annotators work on specific branches reserved for on-going annotation work; project managers can employ different repositories for different annotation projects while monitoring progress and annotator performance, and discovering issues via *git*'s logging and blaming functions; curators can pull changes from annotation branches into dedicated curation branches at any time; administrators – relieved of the duty to run a whole infrastructure – can set up repositories, configure access, etc.
- There is no reason why standalone tools should not be able to interface with pre-annotation services, either by embedding the respective libraries, or by calling remote APIs which would need to be available for utilization in both application types, local and remote, anyway.
- Easily accessible user interfaces, that is, ones that are easy to learn and use, depend more on accessible user interface design metaphors than on the implementation details of a software. In fact, it seems more sustainable to make the software in question extensible with a variety of different interfaces. This can be achieved, for example, by utilizing a module-based application framework, as outlined below.
- Portable desktop software can simply avoid complicated installation procedures by providing a single archive, which – after extraction – provides a ready-to-use application.

All in all, web-based annotation tools, in contrast to extensible standalone tools, can be said to implement different concepts, and fulfill a different set of targets: While remote tools are concerned with standardized and accessible workflows, extensible standalone platforms offer a greater potential for sustainability and application in diverse use cases. In consideration of the aspects discussed so far, we will implement an extensible multi-layer annotation platform as a standalone application rather than a remote infrastructure (cf. *section Objectives: Research software*).

To the best of our knowledge, none of the existing tools – whether web-based or standalone – can handle *true* multi-layer annotation. By "capabilities for true multi-layer annotation" we mean features that provide for, e.g., unrestricted multiple segmentation[5], enriching existing corpora with custom annotation types, and merging different annotation layers into one and the same corpus. The implementation of such capabilities requires a powerful data model which is able to represent potentially unlimited types of annotations, as well as merge different kinds of annotation structures into the same corpus model to facilitate combined analysis. While true multi-layer annotation search tools like *ANNIS* (preliminary work by Lüdeling, cf. Zeldes et al. (2009)) or *KorAP* (Bański et al. 2014) and data formats (e.g. PAULA XML (Chiarcos et al. 2008) or TCF (Heid et al. 2010)) exist, we see a rising demand in annotation tools and workflows that support such true multi-layer annotation. Some of the existing annotation tools like *GraphAnno* (preliminary work by Gast, cf. Gast et al. (2015a, 2015b)) or *WebAnno* already use a flexible annotation data model which would at last theoretically be able to handle this kind of multi-layer data. However, in order to facilitate true multi-layer annotation as defined above, annotation software must also provide features that allow the user to harness the full potential of the data model. These features must arguably be implemented in user interface components that permit, e.g., multiple segmentation, the addition of arbitrary annotation layers, and annotation layer merges. In turn, this requirement implies that the annotation software in question must be extensible, as it is impossible to predict all use cases in a multi-layer annotation workflow. In order to sustainably provide extension capabilities in the sense that technical barriers for

[5] Multiple segmentation, such as multiple tokenizations of multiple data sources within one corpus, are needed for the analysis of historical corpora, e.g., RIDGES Herbology (Odebrecht et al. 2016), as they often contain data sources for original script, transcription, normalization, etc., all of which must be tokenized for particular annotations.

potential contributors are kept as low as possible, the software in question should ideally use a modularization framework that allows to add and remove modules to the product without changing the core architecture and functionality at all. Neither of the above-mentioned potential software candidates for true multi-layer annotation are modularized in that sense.

Considering this, it seems that in order to create a technically sustainable true multi-layer annotation software, it must be built on top of a modularized application framework. Additionally, as annotations can be persisted in a wide variety of data formats, multi-layer annotation software must be able to merge annotations from these formats in order to facilitate the enrichment of a given corpus with additional annotations. Therefore, it should be compatible with as many formats as possible, or at least provide a way to extend it so it can be made compatible with a given format. Furthermore, as many annotations now come from automatic, or semi-automatic, annotation processes, the software should provide an interface to existing natural language processing (NLP) tools or infrastructures.

In conclusion, it is necessary for the implementation of our infrastructure model to employ an annotation tool with at least the following requirements.

- Generic data model, capable of representing *true* multi-layer data
- Extensible architecture, ideally on top of an existing modularized application framework
- Interfaces to existing NLP tools
- Compatibility with a large number of linguistic formats

As none of the existing tools fulfill all these requirements and we do not intend to implement such an annotation tool from scratch in the course of the project we propose, we will re-use existing software and combine them into the envisaged technically sustainable platform. As, in contrast to other tools mentioned above, *GraphAnno*'s data model and feature set (cf. Gast et al. 2015a) is the most generic and universally applicable in the context of multi-layer annotation, we will use it as a starting point. Additionally, an architectural prototype for an extensible annotation platform already exists in the form of *Atomic* (preliminary work by Gast, cf. Druskat et al. 2014). In fact, the former has originally been designed to serve as a functional prototype for the latter, so that we anticipate a relatively straightforward transfer of *GraphAnno*'s functionality to the architectural setup of *Atomic*, merging both into the target software we call *Hexatomic* (**H**ighly **ex**tensible **a**nnotation **to**ol for **m**ult**i**-layer **c**orpora).

1.3.1.1 Research software for a case study: *GraphAnno*

GraphAnno (https://github.com/LBierkandt/graph-anno) is a productivity prototype for a browser-based annotation software for multi-layer corpora, written in Ruby, whose graph-based model is suitable for the representation of annotations from diverse research approaches. It has originally been developed in the context of a typological project, but has since grown into a user-friendly, lightweight software for universal corpus annotations, and is actively used by a growing community of linguists. In combination with *GraphPynt* (https://github.com/VolkerGast/GraphPynt), a Python interface for *GraphAnno*, the latter can interface with the *Natural Language Toolkit* (*NLTK*) as well as a number of existing import modules for different linguistic formats. *GraphAnno* can also export to CSV.

In *GraphAnno*, the user works with an embedded command line interface, backed by a native command language, a small number of control widgets, and a direct visualization of the graph model. Additionally, *GraphAnno* includes a number of assistance views and supports embedding of multimedia files. For the exploration of corpus data and direct annotation of query results, a native query language is provided.

GraphAnno can potentially be applied to a large number of research questions from a variety of linguistic disciplines, especially due to its generic data model and interface to other (Python-based) tools. In contrast to this potential, *GraphAnno* has not been developed towards sustainability, which is reflected in its monolithic architecture and consequentially its lack of modularity. On the one hand, this impedes distributed future development independently of the original developers. On the other hand it will be difficult to extend *GraphAnno* with features for re-use outside of the original use case, for example with features for new annotation types and corpus structures; with optimizations and extensions of the user interface (by, e.g., less abstract editors for specific annotation tasks); with annotation task control independent of the command line such as mouse-based graphical annotation. Extensibility, however, is not only a key

requirement for potential re-use of research software in general, it also serves as a fall-back mechanism for our minimal infrastructure: While the latter is explicitly designed for fail-safety (see above), extensibility and modularization enable the research software in question to sustain even without a maintainer. New features and other changes do not have to be integrated into a monolithic product, but can instead be added "offline" (e.g., in a local copy of the source code pulled from the SCRP) as modules. As soon as a new maintainer for a project is found, these changes can again be integrated into releases of the software.

Both *GraphAnno* and *GraphPynt* have been developed by single developers. In the case of *GraphAnno*, the developer has already left academia. Additionally, no dedicated permanent positions are available for the development of either *GraphAnno* or *GraphPynt*.

However, as the demand for an annotation software with the comprehensive feature set and capabilities of *GraphAnno* in combination with *GraphPynt* is unfulfilled as of yet, and our infrastructure model is based on the assumption that the research software to be provided is technically sustainable, we will make *GraphAnno* sustainable in the course of our project.

We will achieve this through the modularization of *GraphAnno*'s architecture, and the inclusion of *GraphAnno* and *GraphPynt* in an existing software, developer, user and maintainer infrastructure: *corpus-tools.org* (Druskat et al. 2016). Specifically, we will follow Artaza et al.'s (2016) good practice of "not re-inventing the wheel" (as well as others of course), by using a well-established plugin-based application framework for the modularization of *GraphAnno*, the Eclipse Platform (https://wiki.eclipse.org/Rich_Client_Platform, see also section *Objectives: Research software*). In accordance with our infrastructure model, the sustained *GraphAnno* will serve as a pilot object to be provided to the linguistic community by an exemplary infrastructure, to be implemented in the course of our project.

1.3.1.1.1 *GraphAnno* - status, usage, suitability, potential

GraphAnno, despite being a prototype, has already been, and is being used for conducting linguistic research, e.g., Gast et al. (2015a), Gast et al. (2015b), Gast (forthcoming a) and Gast (forthcoming b). As already detailed above, *GraphAnno* already has almost all of the features that a technically sustainable software for multi-layer corpus annotation should have: It has a generic graph-based data model which allows the representation of diverse annotations and thus re-use in use cases beyond the original one. It interfaces with the NLTK and other Python-based NLP tools via *GraphPynt*. It is compatible with common corpus annotation formats such as *Computational Natural Language Learning* (*CoNNL*), and more specific schemes such as *TimeML*. In order to leverage *GraphAnno*'s potential as a universally applicable multi-layer annotation platform, it simply needs to be modularized so that additional user interfaces can be added to work on its data model.

Hexatomic, the new, modularized version of *GraphAnno* which also integrates with *corpus-tools.org*, has great potential to be used far beyond the proposed project. As it interfaces directly with *ANNIS* – cf. section *Objectives: Research software* –, it will fill the "annotation gap" for the large group of projects which already use *ANNIS* for search in and analysis of their corpora. It also has the potential to unlock completely new use cases for whole linguistic communities and their data, e.g., in language documentation or historical linguistics. Some of the potential use cases that are not covered by the attached letters of intent include

- The initial creation and further annotation of multi-layer corpora: Through compatibility with different linguistic formats, *Hexatomic* can be used to merge different annotation types from different sources into one corpus and work directly on that corpus, enabled by the generic graph-based data model.
- Collaborative annotation: By using a decentralized version control system such as *git* for locally persisted files, *Hexatomic* can be used for collaborative annotation. Additionally, through the use of *corpus-tools.org*'s conversion framework (which will be embedded in *Hexatomic*, cf. below section *Objectives: Research software*), corpus data can be converted into the format that is most suitable for the respective collaboration use case.
- Re-use of existing corpus data: Through the use of *corpus-tools.org*'s conversion framework, existing corpus data can easily be enriched with new annotations, which is, for example, a common use case in final theses.

1.3.2 Infrastructure components

In consideration of industry standards for the development of open source and research software, the following components are obvious candidates for an implementation of our infrastructure model.

- Source code repository platform: *GitHub*
- Repository for build artifacts: *Zenodo*
- Repositories for dependency artifacts: *Maven Central* and *Eclipse P2 repositories*
- Maintainer: Changing (see section *Objectives and work programme*)

GitHub (https://github.com) is a source code repository and platform. It works with different version control systems (*git*, *SVN* (https://subversion.apache.org/)), and provides a wide range of services which make it suitable for use as the SCRP component in our infrastructure. An SCRP must provide at least the following features: an issue tracker; a means to host a public entry point; a means to host documentation. GitHub offers an on-site issue tracker with pull request functionality. Public entry points can be implemented on GitHub as enriched repository landing pages. This way, the page can be used as an entry point, displaying all relevant information on the research software. GitHub also offers different options for hosting documentation, such as a wiki for each repository, and GitHub pages, where websites can be hosted directly from the source code repository and can thus be used to display software documentation content and links. We will use GitHub as our SCRP, thereby providing source code, documentation and a single entry point.

Zenodo (https://zenodo.org/) is a general-purpose open access research data repository which hosts user files up to a size of 50GB. It provides a dedicated data type "software" and a unique identifier (DOI) for each hosted artifact. Zenodo also integrates closely with GitHub: Release-ready code can be tagged as a "release" on GitHub, and if the GitHub repository is linked with Zenodo, the release is automatically archived there. This not only secures long-term availability of releases, it also makes releases citable, contributing to the sustainability of the research software in its own right through documenting and increasing its impact and thus enlarging the community of potential stakeholders (cf. Smith et al. 2016). We will use Zenodo as our release repository, thereby providing citable build artifacts.

The Maven Central Repository (http://repo1.maven.org/maven2/), is a repository hosting software artifacts (e.g., libraries), which are accompanied by metadata. Apache Maven (https://maven.apache.org/) is a build automation tool primarily used for projects implemented in the Java programming language. During a Maven build, dependencies of the software being built (e.g., libraries used by the software) can automatically be pulled from repositories and included in the build artifact. The Maven Central Repository is the default source for automatically collected dependencies. It also provides a minimal quality insurance mechanism in that only artifacts with quality Maven metadata can be deployed to the repository.

The Eclipse P2 repositories are the default provider for plugins and plugin bundles (called "features") used in applications based on the Eclipse Platform. The Eclipse P2 ("Provisioning Platform") project focuses on provisioning technology for OSGi-based applications, and while it has specific support for installing Eclipse-based technology, it includes a general-purpose provisioning infrastructure that can be used as the basis for provisioning solutions for a wide variety of software applications.

Eclipse Tycho (https://eclipse.org/tycho/) provides a plugin for Maven build automation which pulls dependency artifacts from P2 repositories and includes them in the build artifact. We will use both Maven alone and Maven with Tycho as our build automation tools, and Maven Central and the Eclipse P2 repositories as our dependency artifact repositories, thereby enabling reproducible and automatable builds.

The maintainer is arguably the central component in our infrastructure as she commands not only the integration, release and testing workflows, but is also responsible for communication with users and contributors. Additionally, she manages and maintains the other infrastructure components, and substitutes them if necessary. Within the proposed project, the maintainer is also responsible for documenting infrastructural decisions, builds and releases, maintainer changes, etc. We will start the project assigning the maintainer role to one of the project team's researchers. In the course of the project, however, we will test the infrastructure's capability of

enduring changes by re-assigning the role, first to the other researcher in the team in a simulation phase, then to a student assistant within the team as a proper test, subsequently to a student assistant outside of the core team as a stress test of the infrastructure, and finally to the person filling either the existing permanently funded Research Software Engineer position at the Corpus Linguistics and Morphology research group, or a permanently funded Research Data Management position currently in planning at the Philological Faculty II (both at Humboldt-Universität zu Berlin). The last step will ensure the autonomy of the infrastructure from external funding after the end of the project (cf. section *Work packages*, WP3).

1.3.2.1 Sustainability of infrastructure components

While to achieve sustainability for our infrastructure, single components must be replaceable at any time, the risk of potential complete failure of single components should be minimized from the onset. At the same time, in order to make the infrastructure as a whole sustainable, it is necessary to reduce running operating costs for academic stakeholders to a minimum. This makes it necessary to use external infrastructural components, as defined above. While it is obviously impossible to control these external components and prevent them from failing completely, it is possible and necessary to choose those components that have the greatest potential to sustain in the long-term. The following paragraphs describe the potential for sustainability of the components we have chosen for our pilot implementation.

GitHub is not only one of the most prominent source code repository providers, it is also the largest host of source code in the world (> 85.5 mio. repositories[6]). However, size alone does not make a component sustainable.[7] The Software Heritage archive (Cosmo and Zacchiroli 2017) has made it its "long term goal [...] to collect all publicly available software in source code form together with its development history, replicate it massively to ensure its preservation, and share it with everyone who needs it" (ibid.). The project is thus building a distribute infrastructure (in this case, a large set of peer server nodes for massive re-duplication of content) and are in the process of growing a multi-stakeholder network of peers (“The Software Heritage Archive” 2017). One of the main features of the project is that the archive includes all public repositories from GitHub, making GitHub a sustainable component for our own infrastructure, as our source code will be recoverable even if its host disappears.

Zenodo's technical infrastructure is funded by the European Organization for Nuclear Research (CERN), while its staff is funded both by CERN and through OpenAIRE grants. While the OpenAIRE project runs out in 2017, CERN has a five year rolling plan (Grootveld and Nowak 2017). Additionally, the Zenodo team is collaborating with funders to secure cloud-credit schemes. Zenodo provides long-term bit-level preservation in the CERN Data Center. It is in the process of applying for the Data Seal of Approval, and CERN is additionally working towards ISO certification of its technical and organisational infrastructure. Zenodo has grown into a well-regarded repository in its own right and is recommended as a target for research output in the European Commission's data management guidelines for Horizon 2020 projects (European Commission 2017).

Maven Central is a central infrastructure for the automated build of applications with different build tools, and Apache Maven uses it as its default repository (“‘About Maven Central’” 2017). It is free, highly available through geographically distributed dedicated servers in North America and Europe, and a number of major open source organizations such as the Apache Software Foundation, the Eclipse Foundation, JBoss as well as a multitude of individual open source projects publish their artifacts there. As Maven Central is business-critical infrastructure for an unfathomable number of software companies throughout the world – it hosts well over 1,100,000 software components, a lot of which are required dependencies of even more software projects – it can arguably be expected to exist for as long as software is developed in Java and built with Apache Maven. The Maven build tool itself is provided by the Apache Software Foundation, a US 501(c)(3) charitable open source software foundation, which is dedicated to sustaining the open source solutions it provides. As of the annual report for the

[6] As queried via the GitHub API on 20 March 2017. Query: https://api.github.com/repositories?since=85000000.

[7] Although, arguably, the sheer numbers of stakeholders for GitHub would seem to make a difference should it be necessary, e.g., in case GitHub, Inc. went bankrupt. At present, GitHub must be regarded as the de facto industry standard for open source code hosting and communities, and its disappearance would have a major impact on the software industry as a whole.

fiscal year 2015-2016, the Apache Software Foundation has a cash reserve of "1.755 million [USD], or 24.1 months at the FY 15 monthly spend of $72.9K" (Apache Software Foundation 2016). It is financed in part through sponsorship by large software companies such as Google, Microsoft, Yahoo!, Facebook, IBM, and many others, as well as infrastructure sponsors partly from the academic domain (e.g., Freie Universität zu Berlin). The Eclipse P2 repositories providing Eclipse Platform components are "retained indefinitely" (https://wiki.eclipse.org/Eclipse_Project_Update_Sites). They are funded by the Eclipse Foundation, a US 501(c)6 not-for-profit foundation, whose more than 250 members include large industry corporations such as IBM, Oracle, Bosch and SAP.

The maintainer is the one component in our infrastructure model which is kept replaceable by design. The academic workforce is, on average, highly mobile due to a general prevalence of fixed-term contracts. While the person filling the role will probably have to be substituted on a fairly regular basis – usually whenever a funding period comes to its end – the role itself must be supported by measures securing the sustainability of the infrastructure in general. These measures mainly consist of comprehensive documentation of the different aspects of the role: technical details as well as design and architectural decisions regarding the research software to be provided and the infrastructure supporting the provision; community rules including codes of conduct, etc.; workflows (build procedure, release procedure, contribution procedure, maintainer change procedure, infrastructural substitution procedure, documentation procedure, report procedure, etc.). This documentation must be provided via the infrastructure itself, and must be kept up-to-date by the maintainer.

In conclusion, the infrastructure model we envisage enables the provision of research software, and does so sustainably, as it maximally reduces the cost for research institutions by harnessing external infrastructure wherever possible. These external infrastructure components are sustainable in themselves. In order for our model to be successfully implemented and employed, the research software it is meant to provide must also have a high degree of technical sustainability, as defined above. The infrastructure model as a whole is independent of our prototypical implementation, as the components can be picked with respect to the use case: For a web application built on the MEAN stack (MongoDB, Express.js, Angular, Node.js), for example, the SCRP could be GitHub, Gitlab, Bitbucket or any other alternative and the dependency artifact repository would probably be NPM (https://www.npmjs.com/); The build artifact repository could be Zenodo or any other suitable repository, and the maintainer component must be implemented depending on the project setup anyway. Of course, our infrastructure model is not suitable for any and all projects. Research software providing crowdsourcing functionality, for example, cannot be provided in this way, as arbitrary users cannot be expected to coordinate via a SCRP. Instead, a server-based solution is more suitable in this case. Similarly, research software that processes Big Data will not be useful as a download on a single computer. Instead, such software will most likely be provided backed by a computational grid. Nevertheless, for cases such as ours where manual or semi-automatic processing of relatively small batches of data is the core function of the software, we believe our infrastructure model to be a perfect fit, and highly sustainable.

1.4 Project-related publications

1.4.1 Articles published by outlets with scientific quality assurance, book publications, and works accepted for publication but not yet published.

Volker Gast

Gast, V. forthcoming a. „The scalar operator even and its German equivalents: Pragmatic and syntactic factors determining the use of auch, selbst and sogar in the Europarl corpus“. In De Cesare, A. & C. Adorno (eds.), *Focus on Additivity. Multifaceted Views on Focusing Modifiers*. Amsterdam: Benjamins.

Gast V. forthcoming b. „*Even* and *so much as* in downward entailing contexts. A quantitative study based on data from the British National Corpus“. In: Emonds, J. & J. Markéta (eds.): Language Use and Linguistic Structure. Proceedings of the Olomouc Linguistics Colloquium 2016. Olomouc: Palacký University.

Gast, V., L. Bierkandt, S. Druskat & C. Rzymski (2016).“Enriching TimeBank: Towards a more precise annotation of temporal relations in a text“. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Druskat, S., V. Gast, T. Krause & F. Zipser. 2016. „corpus-tools.org: An Interoperable Generic Software Tool Set for Multi-layer Linguistic Corpora“. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Gast, V., L. Bierkandt and C. Rzymski. 2015. „Annotating modals with GraphAnno, a configurable lightweight tool for multi-level annotation“. In M. Nissim & P. Pietrandrea (eds.): *Proceedings of the Workshop on Models for Modality Annotation*, 19-28. Stroudsburg, PA : Association for Computational Linguistics (ACL).

Anke Lüdeling

Lüdeling, A. & M. Kytö, eds. 2009. Corpus Linguistics. An International Handbook. Handbücher zur Sprach- und Kommunikationswissenschaft. Vol. 1+2. Berlin: Mouton de Gruyter.

Odebrecht, C., M. Belz, A. Zeldes, A. Lüdeling & T. Krause. 2016. “RIDGES Herbology: Designing a Diachronic Multi-Layer Corpus.” *Language Resources and Evaluation*, 1–31. doi:10.1007/s10579-016-9374-3.

Krause, T., U. Leser & A. Lüdeling. forthcoming. “graphANNIS: A Fast Query Engine for Deeply Annotated Linguistic Corpora.” *Journal for Language Technology and Computational Linguistics. Special Issue on Korpuslinguistische Softwarewerkzeuge*.

Lüdeling, A., H. Hirschmann & A. Shadrova. forthcoming. “Linguistic Models, Acquisition Theories, and Learner Corpora: Morphological Productivity in SLA Research Exemplified by Complex Verbs in German.” *Language Learning Special Issue on Language Learning Research at the Intersection of Experimental, Corpus-Based and Computational Methods: Evidence and Interpretation*.

Sauer, Simon, and Anke Lüdeling. 2016. “Flexible Multi-Layer Spoken Dialogue Corpora.” *International Journal of Corpus Linguistics* 21 (3): 419–38.

1.4.2 and 1.4.3 do not apply

2 Objectives and work programme

2.1 Anticipated total duration of the project

We anticipate a total project duration of 3 years. This duration is necessary mainly in order to investigate, implement and thoroughly test our minimal infrastructure model, and to make the research software it provides technically sustainable, re-usable, and ready for further development beyond the scope of the project. We do not anticipate that the project duration needs to be extended beyond the first 3 years, as we expect both the infrastructure and the research software to be sustainable and self-financing at the end of the project.

2.2 Objectives

The main objective of our project is to model, implement, test and document an infrastructure for the sustainable provision of research software that enables a potential for further development of the research software it provides in the long term. In accordance with the notion that a sustainable infrastructure must minimize running operating costs for the infrastructure provider, we will implement a minimal infrastructure as detailed above. We will furthermore document the implementation details, and test the implementation setup in different ways. We will also document the test results and formulate best practices based on them.

As such a minimal infrastructure assumes technical sustainability of the research software it provides, we will make the prototype of our research software technically sustainable, and provide the product of this process via the implemented infrastructure. We will furthermore test whether the potential for further development independently of the original developers can be realized, and document the results. Furthermore, we will document uptake of the software by the research community, and document the impact the infrastructure setup will have had on community uptake and development contributions.

Table 1: *Project objectives*

Infrastructure	Provided research software
1a: Implement the infrastructure model	2a: Make the prototype technically sustainable
1b: Document the implementation	2b: Provide the product via the infrastructure
1c: Test the implementation	2c: Test long-term development potential
1d: Document the test results	2d: Document community uptake
1e: Extract best practices from test results	2e: Document infrastructure impact on uptake and development

We will meet these objectives and measure our success as detailed in the following sections.

2.2.1 Objectives: Infrastructure

We will **implement our infrastructure model (*1a*)** with the following components: SCRP: *GitHub*; Dependency artifact repositories: *Maven Central, Eclipse P2 repositories*; Release repository: *Zenodo*; Maintainer: *changing* (cf. section *Work packages*).

The implementation of the infrastructure model is successful if (a) *Hexatomic* is provided via this infrastructure, i.e., is being made available both as source code and as runnable release; (b) comprehensive documentation is being made available through the infrastructure (user and developer documentation); (c) the infrastructure provides a Single Entry Point in form of a landing page; (d) it provides a Single Point of Contact, i.e., an issue tracker. We will **document the implementation of the infrastructure (*1b*)** including design decisions in, e.g., a technical report, conference or journal publication ("publication").

We will heavily **test the implementation (*1c*)** by (a) exchanging the maintainer component regularly (cf. below *WP3*); (b) testing other build artifact repositories than Zenodo; (c) testing other SCRPs other then GitHub. As both the Eclipse P2 repositories and Maven Central are the only feasible options for a dependency artifact repository in our case, we will not be able to exchange them. The tests will be successful if they provide information about the stability of the infrastructure. That means, they will (a) prove whether single components in the model can be exchanged without compromising the stability of the infrastructure as a whole, and (b) disclose any potential break points and other weaknesses in the implementation. We will **document these tests and their results (*1d*)** in a publication.

And finally, we will **publish best practices for implementing our minimal infrastructure model for provisioning research software (*1e*)**, based on the tests and our experiences in a publication.

2.2.2 Objectives: Research software

We will **make the research software prototype *GraphAnno* technically sustainable (*2a*)**. In accordance with the requirements for sustainable multi-layer annotation software for linguistic corpora (*generic data model*, *extensible architecture*, *interfaces to NLP tools*, *compatibility with linguistic formats*), we will modularize it, and embed it into the *corpus-tools.org* software set. More precisely, we will re-use all parts of the current set of *corpus-tools.org* in the implementation: The modular architectural prototype *Atomic* (*extensible architecture*), the Java API of the generic graph-based meta model *Salt* (preliminary work by Lüdeling, cf. Zipser et al. 2015, Zipser & Romary 2010) (generic data model), the conversion framework *Pepper* (preliminary work by Lüdeling, cf. Zipser et al. 2015) (compatibility with linguistic formats), and the search engine of *ANNIS* (Krause et al. forthcoming).

Atomic has been created as an architectural prototype for a multi-layer annotation tool. It is based on the Eclipse Platform, a plugin-based modular application platform written in Java, which makes it highly extensible through plugins that can be added to the platform at runtime. Using *Atomic* as *Hexatomic*'s application platform allows for the modularization of *GraphAnno*'s feature set, and for its extension with the user interfaces required for multi-layer annotation, e.g., graphical editors for specific annotation types. Additionally, *Atomic* already uses *Salt* as its data model. *Salt* – just like *GraphAnno*'s current graph model – allows for the representation of potentially unlimited types of annotation which can be modeled as any combination of nodes and vertices. This will contribute greatly to *Hexatomic*'s technical sustainability as it allows the re-use of the software for the most diverse annotation tasks. Instead of implementing whole new tools from scratch for each new use case, users will simply have to implement the interface they need in a plugin, to be able to add their specific annotations to a given corpus, represented in a generic model. *Pepper*, which is also already embedded in the *Atomic* prototype, is a modular conversion framework for linguistic data. Data from format A is mapped to a *Salt* main memory model and can then be mapped to target format B, reducing the number of needed conversion routes for n formats from n^2-n to $2n$ in comparison to direct conversion tools. There already is a large, and ever-growing, number of import and export modules for different formats available (http://corpus-tools.org/pepper/knownModules.html). The use of *Pepper* again contributes to *Hexatomic*'s technical sustainability, as it will make it compatible with a large number of linguistic formats. Additionally, *Pepper* makes *Hexatomic* compatible with *ANNIS* via the *Pepper* exporter for the *ANNIS* format. *ANNIS* is used in a wide variety of projects internationally. Its search engine is also embedded directly in *Atomic*, which will provide *Hexatomic* with a suitable

replacement for *GraphAnno*'s own search functionality from the onset. *ANNIS*' search engine employs a native query language to conduct powerful searches in linguistic corpora, far beyond simple key-value searches. It can find complex linguistic structures, metadata, etc.

We will transfer the functionalities that *GraphAnno* provides into single plugins for *Hexatomic*. We will also transfer the format compatibilities that *GraphPynt* provides by implementing new *Pepper* import modules where necessary, and test available modules, e.g., for the CoNLL formats. Additionally, we will investigate the implementation of a stable interface between the Natural Language Toolkit (NLTK, written in Python) and Java, via the Jython implementation of Python for the Java Virtual Machine. As this is a non-trivial task due to partial incompatibilities between Jython and the reference implementation of Python, CPython, we will fall back to an alternative solution should the interface turn out not to be implementable within the scope of the project. The alternative solution has two tiers, the first being the inclusion of another NLP toolkit – Apache OpenNLP – which is written in Java and also provides maximum entropy and perceptron-based machine learning. The second tier includes the implementation of *Pepper* import modules for output formats of the NLTK to establish compatibility, if not seamless integration. We will also take more general measures (Table 2) to make *Hexatomic* technically sustainable, according to the criteria groups in section *Technical sustainability of research software*.

Table 2: *Measures taken for each criteria group towards technical sustainability of Hexatomic.*

Criteria group	Measure
Licensing	License *Hexatomic* under the permissive Apache License, Version 2.0 (allows different copyright holders for different parts, and commercial re-use)
Automated builds	Use the automated Apache Maven build system with the Eclipse Tycho plugin
Documentation	Create comprehensive user and developer documentation, incl. in-code API documentation
Buildability	Document build process and dependencies, use Maven and Tycho to automatically pull the right dependencies into the build
Installability	Releases come in the form of an archive file (ZIP or similar), no installation beyond extracting the archive is necessary
Testability	Create comprehensive automatable tests (unit tests, GUI tests, integration tests), and document test process and coverage
Portability	Provide *Hexatomic* for the main operating systems Windows, Mac OS, and Linux
Analysability	Provide source releases, publish code on GitHub, provide source code comments, adhere to best practices (e.g., Boswell and Foucher (2011)), document architectural decisions
Changeability	Allow external contributions, create contribution policy, contributor retains copyright
Evolvability	Publicize development roadmap, create public API and documentation for extensions
Interoperability	Provide interoperability with other tools via data im-/export modules for embedded *Pepper*

The technically sustainable implementation of *GraphAnno* in *Hexatomic* is successful if: *Hexatomic* includes the **complete feature set of *GraphAnno*** and can be used productively; It **can be extended** with new plugins; It can **work directly on *Salt*** main memory models; It can **import from and export to all formats** for which *Pepper* modules exist, and from and to all formats which *GraphPynt* supports; It provides an **interface to one or more NLP** tools, in the best case to the NLTK directly, alternatively to Apache OpenNLP directly and to the NLTK indirectly via *Pepper* support for its output formats; It is **licensed with the Apache License, Version 2.0**, which is compatible with the Eclipse Public License 1.0 used for the Eclipse Platform; It can be **built automatically** with Maven and Tycho; It **runs on Windows, Linux and Mac OS** machines; It is **comprehensively documented** for users and developers, including documentation of the build and test processes, test coverage, architectural design, contributor policy, community policies (including code of conduct), development roadmap, and source code comments; It can be **installed by simply downloading and extracting** an archive file; Comprehensive automatable **tests exist** for it in the form of unit tests, GUI tests, and integration tests; Its sources are **published on GitHub** and in source releases, and its code adheres to best practices of clean coding; It has a **public API** supporting the extension via new plugins; It can be **extended by third parties**.

We will **provide *Hexatomic* via the defined implementation of our minimal infrastructure model (*2b*)**. The provision is successful if users and potential contributors can find and learn about *Hexatomic* through GitHub and access all documentation there, can download releases of it from Zenodo, can communicate with the maintainer and the community via GitHub, can report on and contribute to *Hexatomic* via GitHub, can download and build the

source code into a runnable version of the software including all dependencies, can cite specific release versions of *Hexatomic* through the DOI provided by Zenodo.

We will **test *Hexatomic*'s long-term development potential (*2c*)** by acquiring external contributors from the linguistic community and ask them to provide new functionality to the software for their own research. The test will be successful if (a) the external contributor is able to extend *Hexatomic* with new functionality; (b) the external contributor does not have to resort to direct support from the original developers, i.e., the project team, apart from the maintainer.

We will **document community uptake of *Hexatomic* (*2d*)**. Ideally, we will be able to successfully report attestable uptake in the form of publications citing the use of the software. Alternatively, we will provide testimonials of productive use of the software in other projects. If no uptake occurs, we will proactively investigate the reasons by approaching stakeholders and documenting their specific reasons.

We will **document infrastructure impact on the uptake and further development of *Hexatomic* (*2e*)** by conducting a qualitative study with users and contributors, and publish the results.

2.3 Work programme and proposed research methods

2.3.1 Work packages

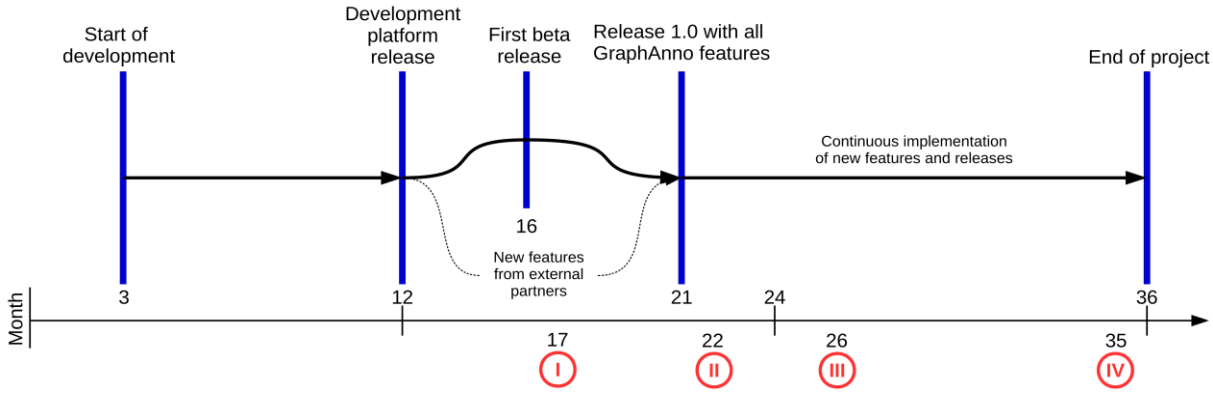

Figure 1: *Roadmap with milestones for* Hexatomic *(blue lines) and maintainership transfers (red circles).*

We distribute the project work over four work packages. In *WP1* (Infrastructure evaluation) and *WP4* (End-of-project planning) we will plan the start and the end of the project. These work packages have a shorter runtime of 3 months each. *WP2* (Implementation, test, documentation of the multi-layer annotation tool) and *WP3* (Implementation, test, documentation of the minimal infrastructure) have a longer runtime and will use most of the staff resources. The latter two work packages are aligned with both the development process of *Hexatomic* and the implementation, test and documentation of the proposed minimal infrastructure.

An important part of the project are the maintainership changes, which will test our infrastructure implementation and produce concrete research results regarding our infrastructure model. Figure 1 contains a graphical overview of maintainership changes as well as development milestones for *Hexatomic*. As for the latter, we will first implement *Hexatomic*'s modular core architecture which will be used as a development platform for new plugins. This first implementation will consist of the plugin-based framework itself, the *Salt* data model, the embedded *Pepper* conversion framework, and the embedded *ANNIS* search engine. It will not yet provide any user interfaces beyond those necessary for the management of corpus projects. Additionally, we will develop procedures for builds, releases and documentation, and document the procedures themselves. After the development platform is available, we will start to transfer the feature set of *GraphAnno* onto it, modularizing the features in the process. In parallel, acquired project partners (cf. *WP2*) can start to contribute features not originally included in *GraphAnno*. At all stages over the course of the project, the currently responsible maintainer will supervise the testing and inclusion of internal and external contributions, and will perform new releases as well as oversee, produce and adapt the software documentation. All source code and all releases will be available for the public from the release of the development platform

onwards, thus external partners can build on the development platform as early as possible and can also give feedback on design decisions. Between the following first beta release, which will already include several features including at least the generic graph-based editor from *GraphAnno*, in month 16 and the first stable release in month 21, there will be more public beta releases if needed. After that release of *Hexatomic* version 1.0, which includes the complete *GraphAnno* feature set, new plugins with new features will be developed and released frequently until the end of the project.

Maintainership changes are planned to happen in month 17, 22, 26 and 35. This schedule ensures that there will be at least one release to manage for each maintainer. All maintainership changes will be carried out under realistic premises and challenges. They will be documented by the respective new maintainer, as she will be able to identify any shortcomings in documentation as well as obstacles in the process. This way we can identify issues early, and fix them during the project before the final maintainership change to an institutional maintainer is performed. Figure 2 presents an overview of work package distribution over project time, and assignment to team member.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
RSE1 (FSU)	WP1			WP2									WP2 + WP3				WP2																	WP4		
RSE2 (HU)	WP1			WP2													WP2 + WP3					WP2											WP4		
SHK1 (FSU)	WP1			WP2																																
SHK2 (HU)	WP1			WP2																		WP3				WP2										
SHK3 (FSU)																									WP3											

Figure 2: *Distribution of work packages over person months for 2 researchers (with job description of Research Software Engineer, RSE) and 3 student assistants (SHK). Columns = months, months in red: maintainership change.*

We assume that maintainers will have limited time for their maintainership task. Therefore, neither researcher will ever work full-time as a maintainer. When a researcher is assigned the maintainer role, their time will be split equally between maintainership and development tasks, i.e., 25% of a full-time position per task set during maintainership assignments, as both researchers are part-time positions with 50% of a full-time position. Student assistants have more limited working hours and will therefore work on maintainership tasks exclusively when assigned the role. This divided workload leads to uneven numbers in the total person months used per work package.

Since the team is located at both Jena and Berlin, we will use the proposed sustainable infrastructure to plan, design and document the features we will implement and to coordinate releases. In addition to these infrastructure components, which are part of the sustainable infrastructure, we will use temporary tools and platforms for internal communication within the project team, like video conference software, chat systems, telephone and e-mail. These tools and platforms don't need to be sustainable since they can be easily exchanged depending on the preferences of the collaborators.

WP1: Infrastructure evaluation

Before we start the implementation of our infrastructure model it is important to evaluate the proposed infrastructure and software components as well as possible alternatives, especially for their sustainability features. In the process, we will create and publish guidelines on how we identify sustainable infrastructure in the context of our project. All proposed infrastructure components will be re-evaluated using these guidelines. If a proposed infrastructure no longer fulfills the requirements we will search for new alternatives.

Deliverables:

- Guidelines on how we determine the sustainability of an infrastructure/software component
- Documentation about used external infrastructure and its sustainability

Sum: 3 person months researcher, 4.5 person months student assistant

WP2: Implementation, test, documentation of the multi-layer annotation tool

This work package bundles the work that is needed to transform the *GraphAnno* prototype into the technically sustainable *Hexatomic* product. *Hexatomic* will include at least all *GraphAnno* features needed to further annotate existing corpora that have been created in projects that have previously used *GraphAnno*, so that with the release of *Hexatomic* 1.0, these projects

could switch to *Hexatomic* without losing functionality. Additionally, we will add further annotation editors for specific annotation types which will have to be determined in the course of the project in collaboration with stakeholders and interested parties.

This work package also includes the creation of comprehensive documentation of all aspects of *Hexatomic*, i.e., user and developer documentation including API documentation, documentation of the build and test processes, test coverage, architectural design, and source code comments. We will constantly document our findings and plan to present them at conferences for linguistic resources, digital humanities and computer science, in order to get feedback from different communities, and to acquire stakeholders and project partners with the aim to generate further contributions to *Hexatomic*. For further details, cf. section *Objectives: Research software*.

Deliverables:

- Stable, technically sustainable, extensible multi-layer annotation software (*Hexatomic*) which can be productively used by linguistic researchers from different communities
- Comprehensive documentation of all aspects concerning the software (cf. section *Objectives: Research software*)
- Conference publications on *Hexatomic*, specifically its sustainable design and extension capabilities for annotation tasks from different linguistic disciplines

Sum: 27.75 person months researcher, 45.5 person months student assistant

WP3: Implementation, test, documentation of the minimal infrastructure

This work package includes the implementation, test and documentation of the infrastructure model with the target components as specified in WP1. The work itself includes the initial set up of the components, i.e.: creation of a GitHub project with users and roles as needed; creation of a dedicated Zenodo account; establishment of the link between GitHub and Zenodo for releases; creation of a fork of the *GraphAnno* and *Atomic* GitHub repositories to secure access for our project; transfer and test of re-usable parts of *Atomic* in preparation for use as the development platform for *Hexatomic*; definition and documentation of a modularization strategy for *GraphAnno*; setup and test of the build, test, and release processes with Maven and Tycho; creation of initial documentation for all maintenance steps as well as software-related documentation as specified above; etc. The work also includes all potential changes in infrastructure components, including their research, implementation, tests, and documentation.

For an active community-driven software project, which we envisage *Hexatomic* to be, it is important to have a dedicated maintainer. We assume that the maintainership is the one component in our infrastructure model which will change relatively regularly. Therefore, changes in maintainership must be well-prepared, which is also part of this work package. A maintainership change plan includes (a) giving the maintainer administration access to all infrastructure components; (b) update documentation of the build, test and release processes before the change; (c) conduct a release where the new maintainer is supervised by the old one; (d) document issues encountered during the change, adapt respective documentation accordingly, and re-iterate the change process if necessary.

In order to subject the infrastructure implementation to stress tests, we will simulate the maintainership transfer several times during the project, under increasingly realistic circumstances. The first transfer (red circled Roman I, researcher 1 to researcher 2) will be of a simulatory nature – both researchers are intimately familiar with the project setup – and aims at testing the basic processes and identifying obvious shortcomings in the documentation. The second transfer (red circled Roman II, researcher 2 to student assistant 2) is more realistic, as the student assistant will be somewhat familiar with the project setup, but the time she can dedicate to maintaining the software is limited and she will arguably have less experience with software maintenance than researchers with a job description of Research Software Engineer. The third transfer (red circled Roman III, student assistant 2 to student assistant 3) will be more realistic yet, as student assistant 3 will only join the project for the last year and will thus be unfamiliar with the project setup, lacking the experience the rest of the team will have from the first two years of runtime. Additionally, she will be associated with the Research Data Management Helpdesk at the Friedrich Schiller University Jena, and hence in terms of expertise (and association) somewhat removed from the rest of the team. Additionally, integrating student

assistant 3 in a research data management institution provides the opportunity to test a realistic setting for a research software maintainer, as in our opinion such institutions represent an obvious location for this kind of role. Furthermore, if planning permits, the target maintainer of the final transfer (cf. *WP4* below) will fulfill a similar role in a permanent position at Humboldt-Universität zu Berlin (cf. section *Infrastructure components*).

Deliverables:

- Technical report on the infrastructure implementation, including design decisions
- Documentation of infrastructure tests and results in a publication
- Comprehensive documentation of maintenance processes (build, test, release processes) as maintainer guidelines
- Technical reports about each maintainership transfer

Sum: 2.25 person-months researcher, 10 person-months student assistant

WP4: End-of-project planning

At the end of the project a last transfer (red circled Roman IV, student assistant 3 to permanent maintainer) is made to this "final" maintainer, who fills a permanent position at Humboldt-Universität zu Berlin.

In addition to *WP3* which plans and simulates maintainership transfers, we will also plan for the sustained existence of the minimal infrastructure, and further development of *Hexatomic* beyond the end of the project. This includes consolidating the large body of documentation which has been iteratively updated throughout the project, and completing it where necessary. It also includes the evaluation and implementation of rules for maintainers for organizing the user and developer communities for *Hexatomic*, including, for example, contributor policies and agreements detailing copyright and licensing policies and a code of conduct.

If problematic components have been identified in the course of the project, we will evaluate alternative infrastructure components and perform any necessary transfers before the project ends. Additionally, all potential legal issues for the maintainership transfer to the institution that hosts *Hexatomic* after the project ends need to be identified and solved.

Deliverables:

- Community policies, including code of conduct
- Maintainership transferred to a permanent position at Humboldt-Universität zu Berlin
- Complete consolidated documentation published under a CC-BY 4.0 license
- Publication on best practices for implementing our infrastructure model

Sum: 3 person-months researcher

2.4 Measures to meet funding requirements and handle project results

By including the research community in the development process and explicitly test how a maintainer can coordinate and incorporate external contributions, we also get feedback from the community on what to improve in the contribution process and in our annotation tool. Additionally, we are in contact with corpus developers in our own working groups and with external groups to make sure the software is actually usable for the annotation tasks we want to support. We also use conversion tools like *Pepper* to assure we are compatible with as many corpus annotation file formats used by the community. More details about how we plan to meet the funding requirements are described in the Objectives section.

The source code of software developed in the project will be made available as open source and will be published under the Apache License, Version 2.0. Using the Apache License allows use by commercial and non-commercial entities with as few restrictions as possible. The code will be published on an external source code repository platform like GitHub, and releases will be additionally published on the Zenodo repository to ensure long-term access. This ensures that the software developed in this project is accessible for academic and non-academic users long after the project is finished. Along with the source code, we will publish the corresponding documentation, guidelines and reports about the maintainership transfers under a CC-BY license. The documentation will also be added to GitHub and Zenodo. When publishing scientific research papers we will prefer open access journals and conferences. With these

measurements for handling the project results, we follow the official and binding research data policy of the Humboldt-Universität zu Berlin (https://www.cms.hu-berlin.de/de/dl/dataman/hu-fdt-policy/at_download/file).

Additionally, to further ensure that the source code and documentation are available, we also guarantee a continuity in active maintainership of the software, by transferring it to a permanent position at Humboldt-Universität zu Berlin (see description of *WP4* for details).

2.5 Information on scientific and financial involvement of international cooperation partners

n/a

3 Bibliography

- "About Maven Central". 2017. http://central.sonatype.org/pages/about.html. Last accessed 22 Feb 2017.
- Apache Software Foundation. 2016. "Annual Report. Fiscal Year 2015–2016." Tech. rep. Apache Software Foundation. https://s3.amazonaws.com/files-dist/AnnualReports/ASFAnnualReport-FY2015-2016FINAL.pdf.
- Artaza, H., N. Chue Hong, M. Corpas et al. 2016. „Top 10 metrics for life science software good practices [version 1; referees: 2 approved]". *F1000Research* 2016, 5(ELIXIR):2000. doi: 10.12688/f1000research.9206.1
- Bański, P., J. Bingel, N. Diewald, E. Frick, M. Hanl, M. Kupietz, P. Pęzik, C. Schnober & A. Witt. 2014. "KorAP: The New Corpus Analysis Platform at IDS Mannheim." In: Vetulani, Z. & H. Uszkoreit (eds.): *Human Language Technology Challenges for Computer Science and Linguistics : 6th Language & Technology Conference*. December 7-9, 2013, Poznań, Poland. 586–87. http://ids-pub.bsz-bw.de/frontdoor/index/index/docId/3261.
- Becker, C., R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff & C. C. Venters. 2015. "Sustainability Design and Software: The Karlskrona Manifesto." In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2:467–76. doi:10.1109/ICSE.2015.179.
- Biemann, C., K. Bontcheva, R. Eckart de Castilho, I. Gurevych & S. M. Yimam. forthcoming. "Collaborative Web-Based Tools for Multi-Layer Text Annotation." In: N. Ide & J. Pustejovsky (eds.): *The Handbook of Linguistic Annotation.* Springer Netherlands. doi:10.1007/978-94-024-0881-2.
- Bontcheva, K., H. Cunningham, I. Roberts, A. Roberts, V. Tablan, N. Aswani & G. Gorrell. 2013. "GATE Teamware: A Web-Based, Collaborative Text Annotation Framework." *Language Resources and Evaluation* 47 (4): 1007–29. doi:10.1007/s10579-013-9215-6.
- Boswell, D. & T. Foucher. 2011. *The Art of Readable Code*. O'Reilly Media, Inc.
- Chiarcos, C., S. Dipper, M. Götze, U. Leser, A. Lüdeling, J. Ritz & M. Stede. 2008. "A Flexible Framework for Integrating Annotations from Different Tools and Tag Sets." *Traitment Automatique Des Langues* 49 (2): 271–93.
- Cosmo, R. Di & S. Zacchiroli. 2017. "Software Heritage. Preserving the Free Software Commons." Talk given at Free and Open Source Software Developers' European Meeting (FOSDEM'17), Brussels.
- Druskat, S. 2016a. "A Proposal for the Measurement and Documentation of Research Software Sustainability in Interactive Metadata Repositories." In: Allen et al. (eds.): *Proceedings of the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4)*. University of Manchester, Manchester, UK. http://ceur-ws.org/Vol-1686/.
- Druskat, S. 2016b. "Entwurf Eines Metadatenrepositoriums Zur Erfassung Technischer Nachhaltigkeit von Forschungssoftware." Talk given at Helmholtz Open Science Workshop „Zugang zu und Nachnutzung von wissenschaftlicher Software", Helmholtz-Zentrum Dresden-Rossendorf; Zenodo. doi:10.5281/zenodo.168383.
- Druskat, S., V. Gast, T. Krause & F. Zipser. 2016. "corpus-tools.org: An Interoperable Generic Software Tool Set for Multi-Layer Linguistic Corpora." In: Calzolari et al. (eds.): *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. 23–28. Paris, France: European Language Resources Association (ELRA).
- Druskat, S., L. Bierkandt, V. Gast, C. Rzymski, F. Zipser. 2014. „Atomic: an open-source software platform for multi-level corpus annotation". In J. Ruppert & G. Faaß (eds.): *Proceedings of the 12th Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2014)*, October 2014, 228-234.
- European Commission. 2017. "Guidelines on FAIR Data Management in Horizon 2020. Version 3.0." Tech. rep. http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf.
- Gast, V., L. Bierkandt & C. Rzymski. 2015a. "Annotating Modals with GraphAnno, a Configurable Lightweight Tool for Multi-Level Annotation." In: Nissim, M. & P. Pietandrea (eds.): *Proceedings of the Workshop on Models for Modality Annotation.*, 19–28. Stroudsburg, PA: Association for Computational Linguistics (ACL).
- Gast, V., L. Bierkandt & C. Rzymski. 2015b. "Creating and Retrieving Tense and Aspect Annotations with GraphAnno, a Lightweight Tool for Multi-Level Annotation." In: Bunt, H. (ed.): *Proceedings of the 11th Joint ACL-ISO Workshop on Interoperable Annotation*. 23–28. Tilburg: Tilburg Center for Cognition; Communication.
- Gil, Y., D. Garijo, S. Mishra & Varun Ratnakar. 2016. "OntoSoft: A Distributed Semantic Registry for Scientific Software." In *Proceedings of the Twelfth IEEE Conference on EScience*. Baltimore, MD. doi:10.1109/eScience.2016.7870916.
- Goble, C. 2014. "Better Software, Better Research." *IEEE Internet Computing* 18 (5): 4–8.
- Gröger, J. & M. Köhn. 2015. "Nachhaltige Software. Dokumentation Des Fachgesprächs 'Nachhaltige Software' Am 28.11.2014." Dokumentationen 07/2015. Umweltbundesamt. http://www.umweltbundesamt.de/en/publikationen/nachhaltige-software.
- Grootveld, M. & K. Nowak. 2017. "Q&A Session 'Open Research Data in H2020 and Zenodo Repository'." Technischer Bericht. OpenAIRE. https://www.openaire.eu/public-documents?id=843&task=document.viewdoc.

- Heid, U., H. Schmid, K. Eckart & E. Hinrichs. 2010. “A Corpus Representation Format for Linguistic Web Services: The D-SPIN Text Corpus Format and Its Relationship with ISO Standards.” In: Calzolari, N. et al. (eds.): *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10)*. Valletta, Malta: European Language Resources Association (ELRA).
- Hettrick, S. 2016. “Research Software Sustainability: Report on Knowledge Exchange Workshop.” http://repository.jisc.ac.uk/6332/.
- Jackson, M., S. Crouch & R. Baxter. 2011. “Software Evaluation: Criteria-Based Assessment.” *Software Sustainability Institute*.
- Krause, T., U. Leser & A. Lüdeling. forthcoming. “graphANNIS: A Fast Query Engine for Deeply Annotated Linguistic Corpora.” *Journal for Language Technology and Computational Linguistics. Special Issue on Korpuslinguistische Softwarewerkzeuge*.
- Odebrecht, C., M. Belz, A. Zeldes, A. Lüdeling & T. Krause. 2016. “RIDGES Herbology: Designing a Diachronic Multi-Layer Corpus.” *Language Resources and Evaluation*, 1–31. doi:10.1007/s10579-016-9374-3.
- Penzenstadler, B. 2013. “Towards a Definition of Sustainability in and for Software Engineering.” In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 1183–5. SAC ’13. New York, NY, USA: ACM. doi:10.1145/2480362.2480585.
- Penzenstadler, B. & H. Femmer. 2013. “A Generic Model for Sustainability with Process- and Product-Specific Instances.” In *Proceedings of the 2013 Workshop on Green in/by Software Engineering*, 3–8. GIBSE ’13. New York, NY, USA: ACM. doi:10.1145/2451605.2451609.
- Smith, A. M., D. S. Katz, K. E. Niemeyer & FORCE11 Software Citation Working Group. 2016. “Software Citation Principles.” *PeerJ Computer Science* 2 (e86). PeerJ. doi:10.7717/peerj-cs.86.
- Tate, K. 2005. *Sustainable Software Development: An Agile Perspective*. Boston, Mass.: Addison-Wesley.
- “The Software Heritage Archive.” 2017. https://www.softwareheritage.org/. Last accessed 20 Mar 2017.
- Yimam, S. M., I. Gurevych, R. Eckart de Castilho & C. Biemann. 2013. “WebAnno: A Flexible, Web-Based and Visually Supported System for Distributed Annotations.” In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 1–6. Sofia, Bulgaria: Association for Computational Linguistics. http://www.aclweb.org/anthology/P13-4001.
- Zeldes, A., J. Ritz, A. Lüdeling & C. Chiarcos. 2009. “ANNIS: A Search Tool for Multi-Layer Annotated Corpora.” In *Proceedings of Corpus Linguistics 2009*. Liverpool, UK.
- Zipser, F., T. Krause, A. Lüdeling, A. Neumann, M. Stede, A. Zeldes. 2015. “ANNIS, SaltNPepper & PAULA: A multilayer corpus infrastructure”. *Final Conference of the SFB 632 Information Structure: Advances in Information Structure Research 2003 - 2015*. Berlin, 08-09 May 2015.
- Zipser, F. & L. Romary .2010. “A model oriented approach to the mapping of annotation formats using standards”. In: *Proceedings of the Workshop on Language Resource and Language Technology Standards, LREC 2010*. Malta. URL: http://hal.archives-ouvertes.fr/inria-00527799/en/