DHQ: Digital Humanities Quarterly
2017
Volume 11 Number 4

Continuous Integration and Unit Testing of Digital Editions

Bridget Almas <balmas_at_gmail_dot_com>, The Alpheios Project, Ltd.
Thibault Clérice <tthibault_dot_clerice_at_enc-sorbonne_dot_fr>, Centre Jean-Mabillon (École des chartes) - PSL

Abstract

Over the last few years, the Perseus Digital Library (PDL) and the Open Philology Project (OPP) have been moving towards enabling better interoperability and citability of their texts by implementing the Canonical Text Services URN standard and the Epidoc subset of the TEI P5 guidelines. This is a resource-intensive effort necessitating a scalable workflow centered on continuous curation of these texts, from both within and outside the PDL/OPP ecosystem. Key requirements for such a workflow are ease of maintenance and speed of deployment of texts for use by a wide variety of analytical services and user interfaces. Drawing on software engineering best practices, we have designed an architecture meant for continuous integration with customizable services that test individual files upon each contribution made to our public git repositories. The services can be configured to test and report status on a variety of checkpoints from schema compliance to CTS-ready markup designed for flexibility and interoperability.

Introduction

In 2012, the *Perseus Digital Library (PDL)* [Almas 2013] decided to apply a nascent norm in the digital classics world, the *Canonical Text Services* (CTS) protocol [Smith and Blackwell 2012], to its corpus of primary source Greek and Latin texts (see Figure 1). This effort coincided with a rather aggressive *Optical Character Recognition* (OCR) campaign by its sister project, the *Open Philology Project* (OPP) in Leipzig, aimed “at providing at least one version for all Greek and Latin sources produced during antiquity”. Through this effort OPP is adding thousands of new Greek and Latin texts to open access repositories, with a focus on post-classical corpora available online [Crane et al 2013]. With hundreds of pre-existing *PDL* texts needing to be made *CTS* compliant as well as upgraded from the *Text Encoding Initiative (TEI) P4* Guidelines [TEI-Consortium 2002] to the *Epidoc* [Elliott, Bodard, Cayless et al. 2006] subset of *TEI P5* [TEI-Consortium 2007], together with the incoming hundreds or thousands of texts coming out of the *OPP* pipeline, the work of a curator would require much tedious checking of technical details. 1

In addition, management of resources needs to be scaled within the context of a non-uniform corpus. Both the conversion process of pre-existing *TEI XML* files and the integration of new files needs to be validated against the agreed upon norms. While the *TEI* norm and any of its subsets are a good first step towards unification of resources, norms like *CTS* and digital libraries like *Perseus* require some specific technical solutions that can be both scalable and cost-efficient. 2

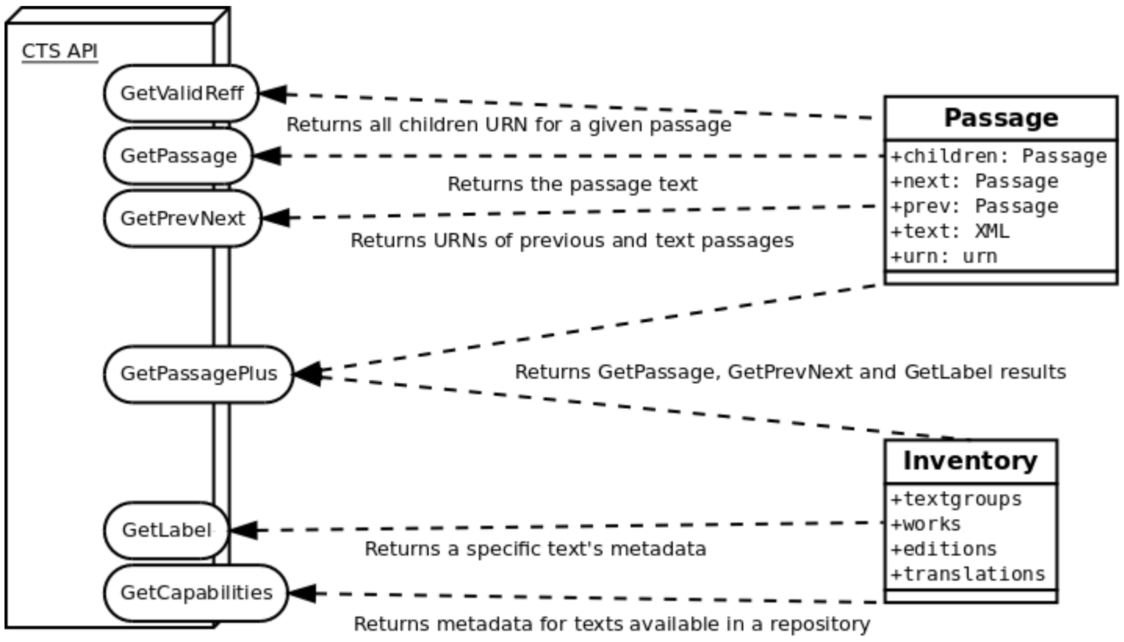

CTS API
GetValidReff
GetPassage
GetPrevNext
GetPassagePlus
GetLabel
GetCapabilities
Returns all children URN for a given passage
Returns the passage text
Returns URNs of previous and text passages
Returns GetPassage, GetPrevNext and GetLabel results
Returns a specific text's metadata
Returns metadata for texts available in a repository
Passage
+children: Passage
+next: Passage
+prev: Passage
+text: XML
+urn: urn
Inventory
+textgroups
+works
+editions
+translations

Figure 1. CTS API Requests Explanation Diagram

Motivating Factors, Decisions and History

Lessons learned from the long history of managing the Perseus corpus and its supporting applications drove some of the technical decisions of this project. Ingesting new and updated texts in the legacy *Perseus 4.0* application is a tedious process. *Perseus 4.0* is a traditional 3-tier Java web application which is deployed under *Tomcat*. The views it presents the users combine the results of relational database queries of a *MySQL* database [MySQL 2004] with static data served directly from the filesystem. Much of the runtime analytical functionality (frequency calculations, word lookups, entity identification) relies on textual data being parsed and pre-loaded into tables in the supporting *MySQL* database. The binary offset location of text within the *XML* source files is used to synchronize the relational data with the *XML* source. This tight coupling between application code, database and raw data means that any time a text is touched, the entire database needs to be reloaded. A more scalable solution was needed that would enable Perseus to serve new and updated texts in real time as they became available, with the confidence that they would work correctly and not break other parts of the application. This requires a distributed architecture. Implementation of *CTS* is one step in this direction, as it allows us to identify and serve text passages by their canonical identifiers, using persistent stable identifiers and a technology-independent *API*. 3

Another primary objective for the *Perseus* and *OPP* projects is to provide a fully open-access and self-describing corpus of texts which can stand on its own and support a wide variety of scholarly needs. Any solution which embeds knowledge of text content or structure in software application or database code is antithetical to this goal. 4

Structural Markup Guidelines

As previously noted, the *CTS* service protocol allows us to identify and serve text passages by their canonical identifiers, using persistent stable identifiers and a technology-independent API. The *CTS URN* notation is based on a strict hierarchical concept of the text, where its passages are sub-ordered down to the word level with no limitations applied to the depth of the passages tree. In this context, *XML* fits the technical requirements. But to implement *CTS* we must decide upon a single “canonical” hierarchical *XML* markup structure for each text. External indices and transformations can be used to present alternative schemes or visualizations, in addition to or instead of relying upon embedded milestones to deal with issues of overlapping citation hierarchies. 5

From scholarly tradition to *XML* encoding

Most scholarly tradition is easily transferred from text to tree [Renear, Mylonas and Durang 1993]: hierarchical models of lines, verses, books or chapters are easily expressed using traditional *TEI*[1]. Verses (in the context of Antiquity, poetry and theater) and paragraph-based citation schemes translate perfectly to a tree system. Use of the `tei:n` attribute to denote the identifier of a passage allows for a fast, real-time traversing of the tree, with technologies such as *XPath* and *XQuery*, to reconstitute passages such as Homer's *Iliad* 1.1. Identification of passages becomes scalable and encoder-friendly and respects both *TEI* guidelines and the scholarly tradition. 6

However, a complex situation emerges from another tradition: page-based citation schemes. Most of Perseus' prose resources, whose citation schemes are inherited from scholarly traditions, are quoted by semantical unit (book, chapter, section, etc.) whereas some systems have preferred topological ones (mostly pages). Cicero's and Plato's work, two of the most studied authors in Greek and Latin, follow a page based scheme [Franzini and Foradi 2014]. In this context, we find ourselves with two concurrent trees: one that reflects paragraphs and divisions through markup; a second one that embodies the topographical citation scheme. This leads to the use of the fairly common `<tei:pb>` or `<tei:milestone>`, identifying the name and the identifier of the canonical citation scheme if required. With the constraint of an *XML* based delivery of passages, however, this structure fails and collides with the tree oriented query system of XML, namely *XPath*. 7

The *Perseus Digital Library* needs not only to be scalable in terms of speed but also in terms of code efficiency. Ideally a single technical implementation of the *CTS protocol* should be able to support the entire corpus. And to deliver a rather fast response to the *GetValidReff* request for passages in the *Iliad* - which, without refinement, can necessitate the transfer and the identification of the 15,693 *URNs* corresponding to the complete set of line identifiers available in the text [2]- the XPATH for passage retrieval needs to be cost-efficient. The first solution to this problem is a shift from the traditional citation scheme to a more logical one, with the publication of an equivalences registry between one scheme and another. A second one is the manipulation of the markup rules, with attributes which would indicate that one paragraph and its sibling actually belong to a common unit. 8

Self-containing text vs. outer metadata: *CapiTainS Guidelines*

CTS is built around three major sets of information which are covered by its guidelines and which come from three different sources: metadata from the library, with authorship and edition information, metadata from the data repository, including the object identifier, and metadata from scholarship, as embodied by the citation scheme. The *CapiTainS Guidelines* [Almas, Clérice and Munson 2017] are designed specifically for a *XML* based implementation of the *CTS protocol*. They supplement the core *CTS specification* and provide a solution to the challenges of enabling reuse and scalability. The *CapiTainS Guidelines* include : 9

- a directory and file naming convention (see Figure 2),
- expression of the *CTS citation scheme* and edition specific metadata inside the edition *XML* file,
- shared metadata files at the textgroup and notional work level

URN Structure			CTS Namespace	Textgroup	Work	Edition or Translation
	urn:	*cts:*	*latinLit:*	*phi0472.*	*phi001.*	*perseus-lat1*

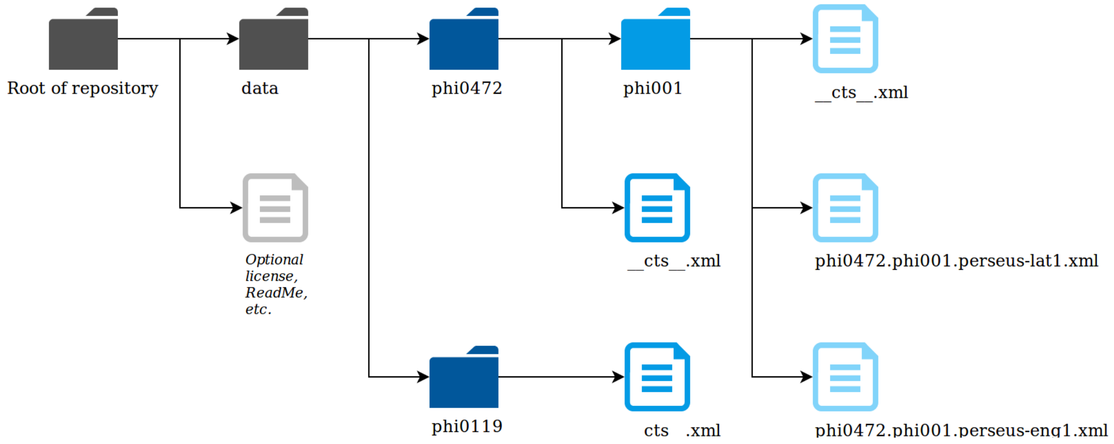

Root of repository
data
phi0472
phi001
__cts__.xml
Optional license, ReadMe, etc.
__cts__.xml
phi0472.phi001.perseus-lat1.xml
phi0119
__cts__.xml
phi0472.phi001.perseus-eng1.xml

Figure 2. Directory structure

10 The rationale behind this approach is to avoid unnecessary duplication of information while still allowing for a completely self-describing corpus structure. Texts adhering to the guidelines can then be integrated into the corpus with a much lighter dependency on the current implementation of the services and tools built to support it, while shared metadata can be of use separately from the text itself.

11 To facilitate text identification, the identifier of the text should be accessible from both inside and outside the markup. While the naming convention of files does cover external identification using the work identifier, a simple query on the text should also be able to return its full *URN*. In *TEI P5*, the deepest required common node is the `<tei:body>`. In the subset commonly used in *Epidoc*, the deepest required node is one level deeper, the first `<tei:div>` inside one text, which identifies the text as being of `@type` translation or edition. The *CapiTainS Guidelines* add to this a required `@n` attribute containing the *CTS URN* of the text. This is enough for the *CTS API* to identify the author, the work and the edition or translation specific metadata from internal markup or external databases. We use the `@n` attribute on the div enclosing the text or translation, rather than metadata in the *TEI* `<header>`, because the *TEI P5* (and the *Epidoc* subset) allow for multiple editions or translations to be included in one file, and we want the *URN* to be unambiguously associated with the text it identifies.

12 In addition to the individual file naming convention, applying a similar approach to the hierarchical directory structure allows us to easily support human browsing of the resources in the source repository [Crane et al. 2015]. Our guidelines call for the first level of the directory structure to be named for the *CTS textgroup* and to include a file containing the *CTS* metadata for the textgroup, named as “__cts__.xml”. The second level of the directory structure is named after the identifier of the notional work and itself contains a metadata file which contains the *CTS* metadata for the work, edition and translation. These metadata files can be used by the service application to dynamically construct a complete *CTS TextInventory*, a required output of the applications implementing the *CTS API*.

13 As for the citation scheme, the *TEI P5* specifically already defines a set of nodes, the `<tei:cRefPattern>`, as children of `<tei:refsDecl>`, that are built for this specific task: identifying references through the traversal of the tree using *regular expressions* and *XPath*. The *CapiTainS Guidelines* call for implementation of this `<tei:refsDecl>` structure, using the `@n` attribute to identify it as the CTS reference declaration and the definition of `<cRefPattern>` for each level of citation to allow for the internal description to perform information retrieval (see Code Sample 1). Applications which serve the corpus and which want to implement the *CTS API* can aggregate this information with that provided by the external *CTS metadata* files to dynamically report the citation scheme as part of the *TextInventory*.

```
<refsDecl n="CTS">
 <cRefPattern n="line" matchPattern="(.+).(.+)"

replacementPattern="#xpath(/tei:TEI/tei:text/tei:body/tei:div[<att>@n</att>='$1']//tei:l
[<att>@n</att>='$2'])">
  <p>This pointer pattern extracts book and line</p>
 </cRefPattern>
 <cRefPattern n="book" matchPattern="(.+)"
   replacementPattern="#xpath(/tei:TEI/tei:text/tei:body/tei:div[<att>@n</att>='$1'])">
  <p>This pointer pattern extracts book.</p>
 </cRefPattern>
</refsDecl>
```

Example 1. Implementation of *CTS* `<refsDecl>` for an edition of the Iliad

Unit Tests

From text to software : defining properties and functions

Unit testing is a software engineering practice which focuses on ensuring the functional capacity of software following changes to it by running tests on the smallest unit in a non-deployment environment to prevent propagation of errors in the software base [Huizinga and Kolawa 2007, 75]. Test results can be expressed in many different ways : through percentage relative to the last test, or absolutely, or in a simple binary fashion with a passed/not passed information. Tests can generally be developed automatically but might be expanded once specific bugs needing testing surface. Unit tests are intended to check the valid output and/or the consistency of resources, whether they are compute-free or not. (Constants and properties are examples of compute-free resources, whereas functions and objects are examples of the opposite, because a specific input should give a specific output.) Unit tests on *XML* documents are focused on testing properties of the document against a schema such as *TEI* using *RelaxNG* [Clark 2001]. *RelaxNG* is a description language for *XML* that specifies how an *XML* document should be structured, such as what values are acceptable for attributes and what nodes allow or require as their descendants. The scope of what we can test with a *RelaxNG* schema is limited to these tests and the content and structure of a given document. It has no external data access and is not designed for computing variable document structures. 14

The first step to properly apply *unit testing* in this context is to define, for an encoded text, the parts which are “properties” and the parts which are “functions”. Identifying these parts helps design the general test scenario by grouping resources which are less compute intensive. In a CTS corpus, we can think of metadata such as the *CTS URN* identifier and the text markup as properties, i.e. they should be present and respected but they are not to be computed upon. Additional testable properties, given the *CapiTainS Guidelines*, include information from the outer metadata files about the work and author, along with their translations. 15

Adherence to and application of a specific text encoding scheme falls in between function and property. In the context of *Object Oriented Programming (OOP)* [Pierce 2002, 225], the *TEI* Encoding, and its subsets, represents the architecture of the *proto-object* or the *parent class*. Objects derived from this class should respect the parent structure. In this context, *XML* compliance, and moreover, schema and *DTD* compliance, can be thought of as required properties of those objects. 16

Passage retrieval is the only specific function that one encounters in *CTS*. The presence of the `<refsDecl>` in the *XML* file of a text is a property, but the accuracy of the `<refsDecl>` and the presence within the text of at least one element for each level of citation is a requirement for the text to be functional. In addition, for any text, the `@replacementPattern` given for any level of citation should not, when completed, resolve to more than one passage for any given identifier at any level of the hierarchy 17

These then are the base cases for our tests (see Figure 3), but experience tells us that additional properties and tests will likely be discovered to be necessary, and need to be added to the existing texts. For example, with the expansion to semitic languages, the existence of right-to-left markers should be checked against language rules. 18

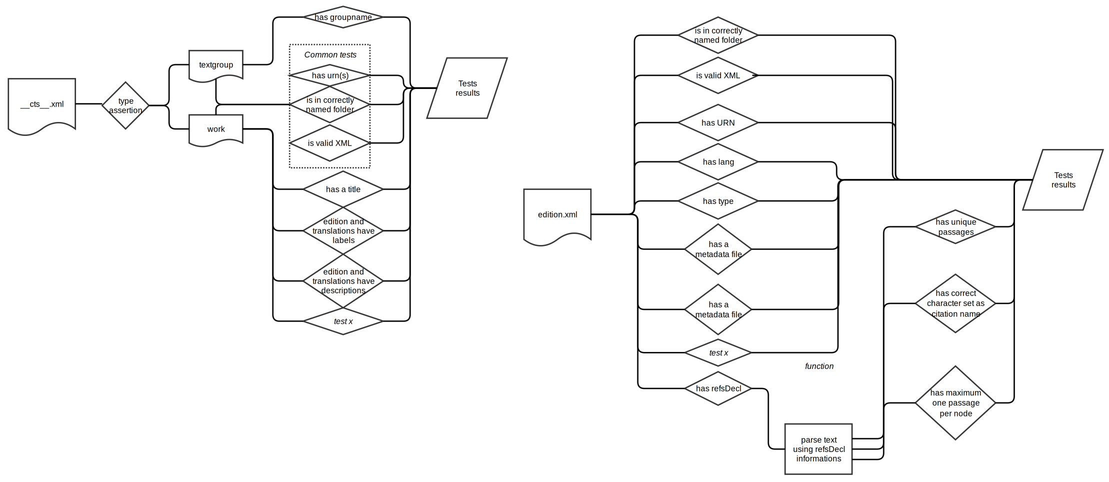


Figure 3. Base Test Diagram

Reuse, present and future development

Taking the software engineering paradigm further, we can treat the corpora as a whole as a set of software packages, where each text is a unit representing an individual code base. The test should happen in three different steps: *object discovery*, *test attribution*, and *unit tests*. Within this context, test discovery means detection of *XML* files. Then in the test attribution step, objects are dispatched by a type detector: here, metadata files adhering to the __cts__.xml name are automatically sent to a specific metadata test class while others are sent to a text test class. Finally, objects are dealt with in a test object whose output is sent back to the main test process. In case the results are needed for further tests, such as the presence of metadata about author and *notional work*, those are made available in this process. 19

Tests rely on different technical resources, and some do not require custom coding: for example, schemes are tested against *TEI* or *Epidoc* using jingtrang [Clark 2001 (2)] and the respective *RelaxNG* resources. Other tests, such as those which check the naming conventions, are implemented simply as regular expressions. And finally, the *CapiTainS Guidelines* for the definition and resolution of *CTS* passages are exercised through tests written in *Python*. 20

The open source software for this test framework is designed to enable extensibility and reuse. An entirely different type of document, for example, a repository of Treebank data (see Code Sample 2), could be tested through reuse of the archetypal test class objects and coding of new rules for the the file resolver. The *archetypal unit test class* takes a path, a “parsable” method for testing ingestion, a “logs” property and a “test” method for starting the tests. This class also has two constants which need to be supplied: “test”, which contains the list of method names to be used for tests, and “readable”, which should provide human readable explanation of the tests. 21

```
class TreebankUnit(HookTest.units.TESTUnit):
    tests = ["parsable", "has_root"]
    readable = {
        "parsable": "File parsing",
        "has_root": "Root declared"
    }

    def __init__(self, path):
        super(HookTest.units.TESTUnit, self).__init__(path)

    def has_root(self):
        # Process
        self.log("If something needs to be verbose")
        has_root = True  # Assign result as a boolean
        yield has_root

    def test(self, scheme):
        tests = [] + CTSUnit.tests
        tests.append(scheme)

        for testname in tests:
            # Show the logs and return the status
            for status in getattr(self, testname)():
                yield (
                   TreebankUnit.readable[testname],
                   status,
                   self.logs
                )
        self.flush()
```

Example 2. Code sample, Pseudo-python sample integration of Treebank Unit Test class

Continuous Integration

Context and architecture

Continuous Integration is a software development practice in which programmers sharing the same project commit different changes to a code base. These commits lead to the running of a series of tests to check on compatibility of the new code and finally to the delivery of the community accepted changes to a production or a stage environment [Fowler 2006]. 22

Perseus data has been hosted on GitHub since July, 26th 2013. Before this, Perseus resources were hosted internally and distributed at release points only on SourceForge. This made incorporating contributions of corrections from external sources difficult. Opening the data of Perseus had two goals. The first one is simply openness. Hosting resources and giving access to them in a raw fashion not dependent on any application or *API* has been a best practice espoused by numerous projects in the Humanities, such as the Pleiades project [Ragnall, Talbert, Horne and Elliott 2008]. The second point of giving access to the data on these collaboration platforms is to allow for citizen scientists, fellow researchers and classical studies enthusiasts, to participate in the correction of Perseus resources the same way. 23

In this context, the library curator finds themselves in a situation where they should ensure that changes proposed, made in the form of *pull requests*, are correct from both the technical and the philological perspective (see Figure 4). Developing a *webhook* to check on technical validity, built on the capacity of GitHub to ping services when changes are proposed, has allowed us to significantly lighten the work required of the curator. It also allows us to measure and report on progress, from the highest level (the percentage of the entire repository which is fully CTS CapiTainS Compliant) to the individual object test result (percentage of tests passed). Results of these tests can then also be checked automatically by deployment scripts for the CTS-enabled applications serving the texts. 24

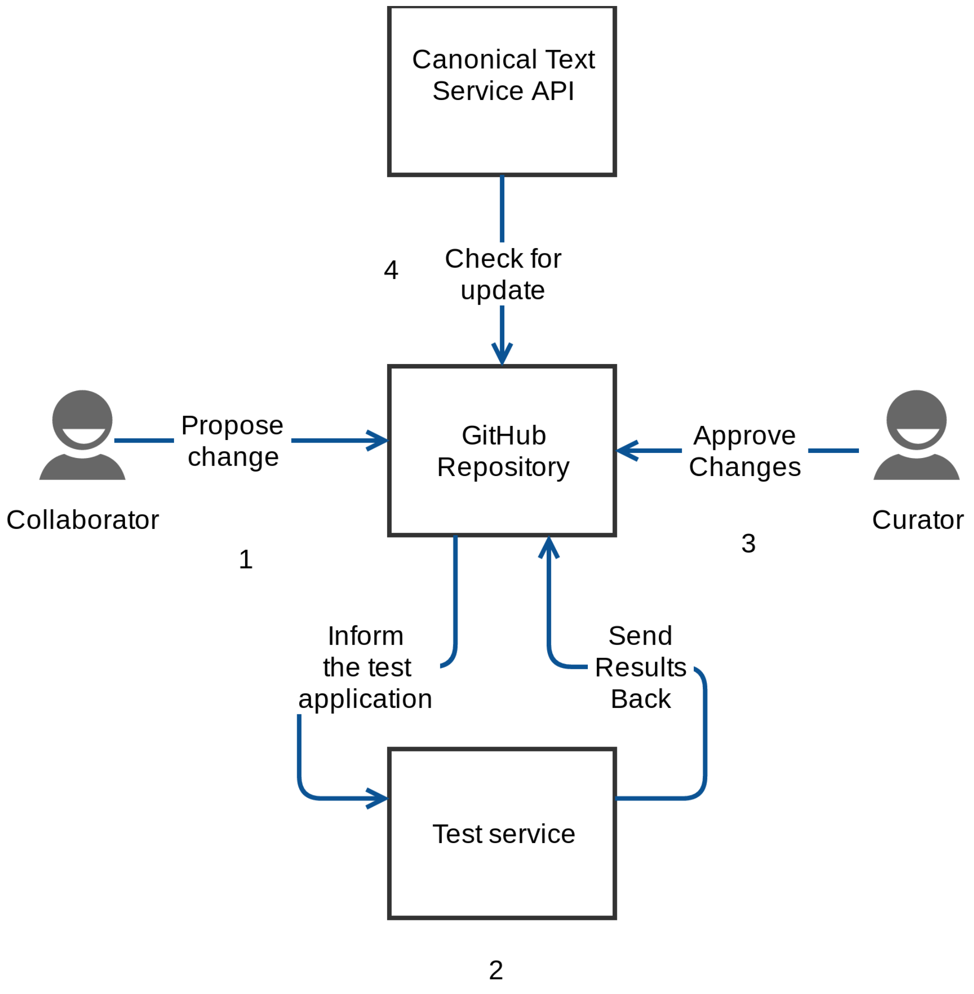


Figure 4. Continuous integration workflow

Scalability and deployment

The tool suite used for this continuous integration environment makes use of free online services and is divided into two separate code bases, each presenting its own set of challenges. The user interface, *Hook* ([Almas and Clérice 2017]), needs to offer an *API* endpoint for the test results and user management for registering *API* access to the GitHub repositories. *Hook* acts as the archival service, listening for test results and annotating pull requests or commits on the source repositories with a summary. On each transaction between Github and *Hook*, identification tokens are exchanged along the required data via the *oAuth protocol* [Hardt 2012]. The user interface is itself a lightweight Python *Flask* web application [Ronachter 2010].

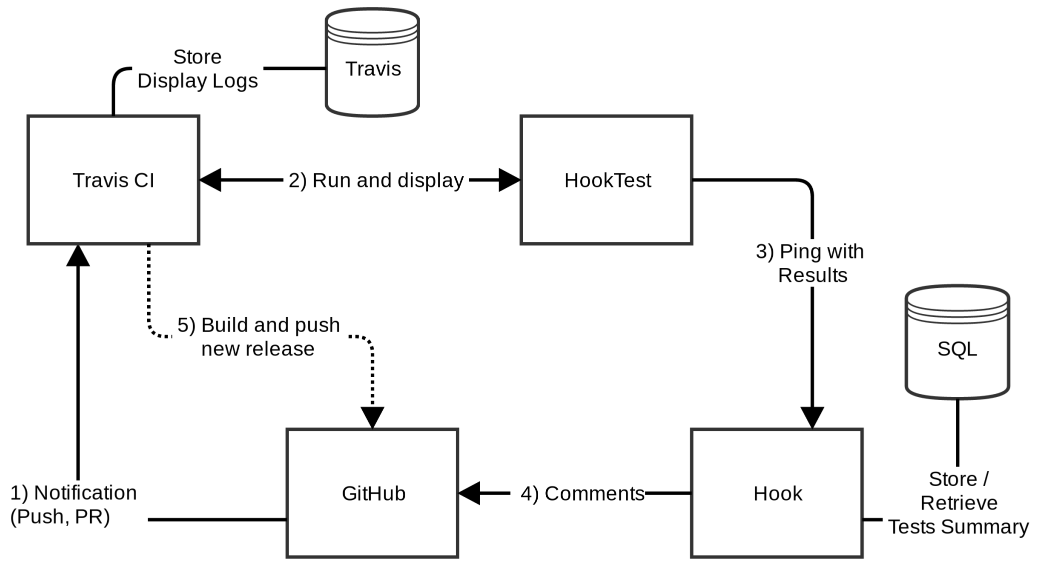


Figure 5. Hook Testing Architecture

The second application, *HookTest* [Almas, Clérice and Munson 2017 b], is the testing software that actually runs the tests. *HookTest* has been designed for its stable release 1.0.0 as a tool that can be both run on local machines or on free services for *Continuous Integration* such as Travis-CI. Depending on the size of the corpus, different types of verbosity of the results are made available so text status messages are manageable even on really large corpora. *HookTest* also provides a second set of optional services to package the corpus into a set of only valid files (i.e. files passing tests) and push this package back as a release to Github. 26

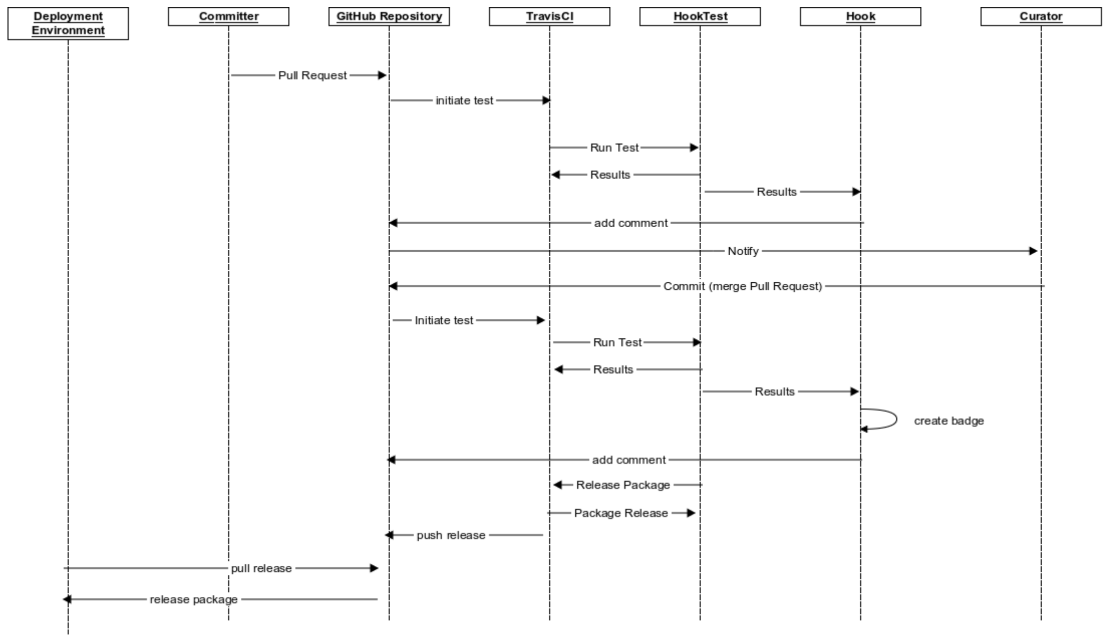
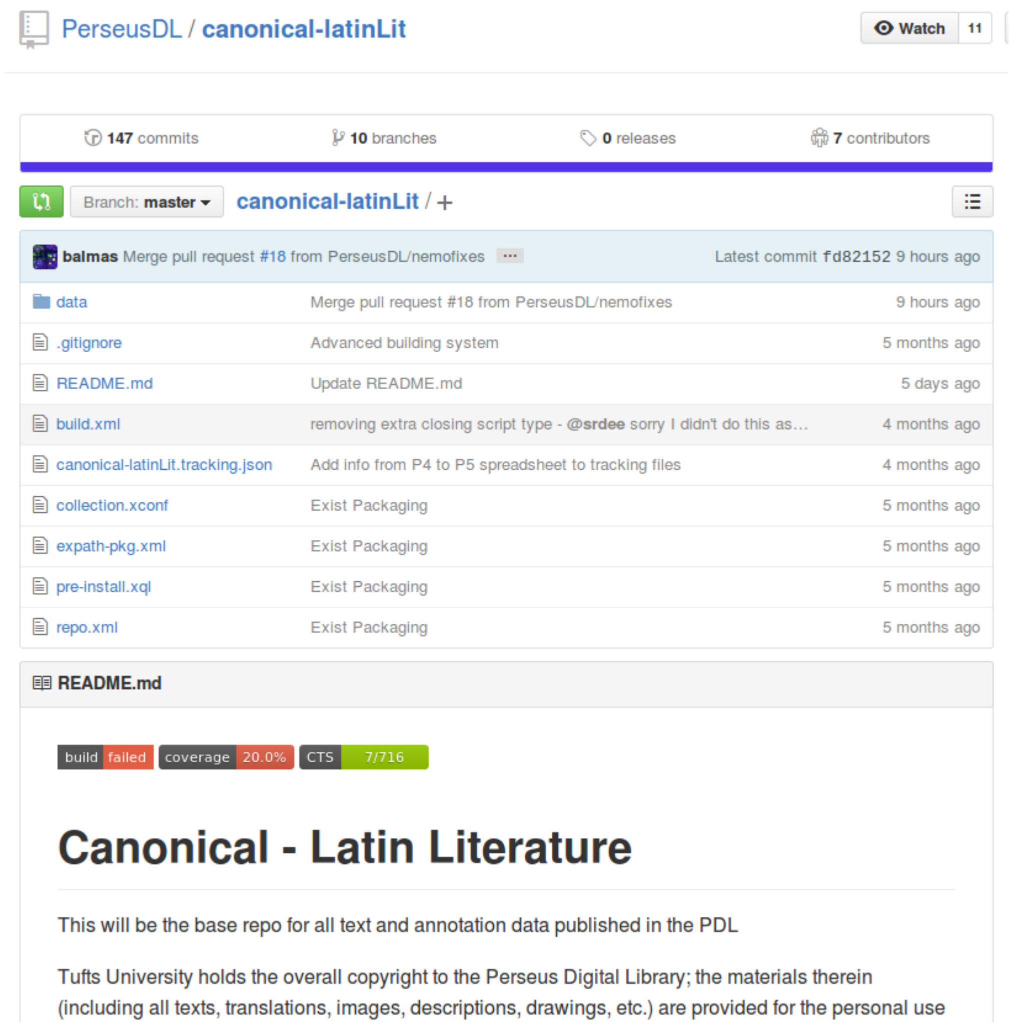


Figure 6. Continuous Integration Workflow Sequence

In a configuration which leverages both *Hook* and *HookTest* together with the Travis-CI service there are two steps to the feedback process. At the end of the test, *HookTest* displays on Travis the results of the tests in a table (see Figure 6) and dispatches the results to *Hook*[3]. The *Hook* application adds a comment to the resource on Github (i.e. the *Pull Request* or the *Commit* which triggered the test) with a score, a binary result (passed/failed) and a difference status 27

(New text passings, number of new nodes, etc.). In addition to the code comments, *Hook* creates and serves icons, in the form of badges which can be referenced from the *README* of the repository, for the users of the repository and the application to be able to quickly access information and status from the GitHub repository home page (see Figure 7)



balmas Merge pull request #18 from PerseusDL/nemofixes … Latest commit fd82152 9 hours ago

data	Merge pull request #18 from PerseusDL/nemofixes	9 hours ago
.gitignore	Advanced building system	5 months ago
README.md	Update README.md	5 days ago
build.xml	removing extra closing script type - @srdee sorry I didn't do this as…	4 months ago
canonical-latinLit.tracking.json	Add info from P4 to P5 spreadsheet to tracking files	4 months ago
collection.xconf	Exist Packaging	5 months ago
expath-pkg.xml	Exist Packaging	5 months ago
pre-install.xql	Exist Packaging	5 months ago
repo.xml	Exist Packaging	5 months ago

README.md

Canonical - Latin Literature

This will be the base repo for all text and annotation data published in the PDL

Tufts University holds the overall copyright to the Perseus Digital Library; the materials therein (including all texts, translations, images, descriptions, drawings, etc.) are provided for the personal use

Figure 7. PerseusDL/canonical-latinLit GitHub homepage

If the release packaging service is enabled, each new version of the corpus that has been released can then automatically be deployed into production and test environments, in the same manner as a software update (see Figure 8).

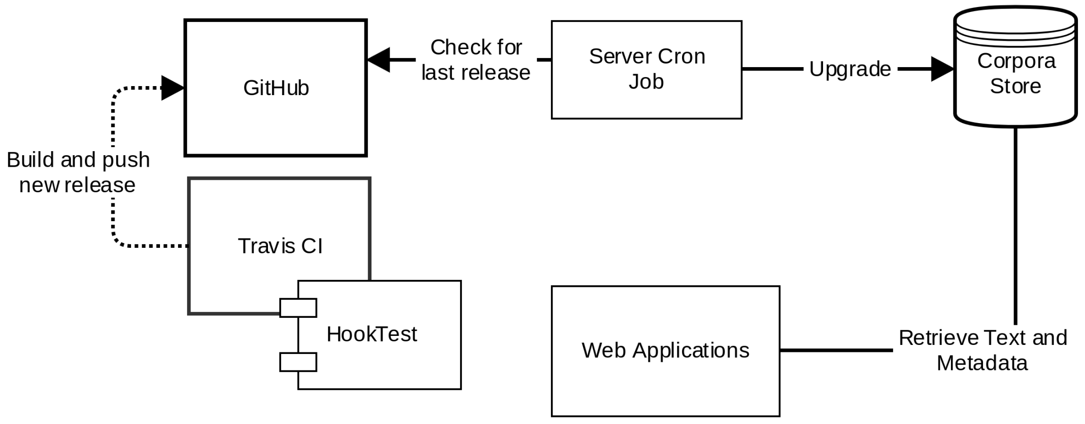
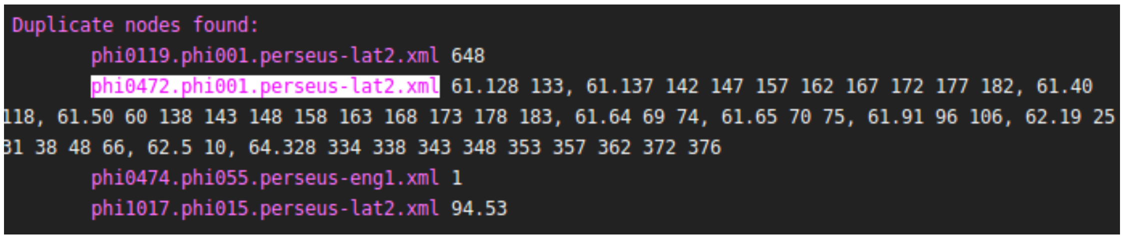


Figure 8. Hook Update/Integration Architecture

The comments added by *Hook* to *Pull Requests* and *Commits* on the GitHub repository enables the curator to easily assess changes made by other contributors. The test results can be found on the GitHub resources, and also activate GitHub-managed notifications (mail or web) that states the summary results of the tests with links back to the detailed results in Travis. These notices are sent to the curator owners of the GitHub repository and issuers of the Pull Request or Commit, and also can be subscribed to by other interested parties. 29

Whether working on a new corpus or converting an existing corpus to comply with the *CapiTainS* specification, the tests allow for detection of errors that could not be easily caught by schema validation with RelaxNG or Schematron. One common example of an error of this sort is the duplication of passage identifiers. Because passage identifiers are built by combining identifiers of elements at different levels of the hierarchy, this cannot be done without a programmatic test. These errors are identified in the *HookTest* results. When a text conversion is done and the push request made, *Hook* provides a list of duplicate passages and writes in the summary on the *Pull Request*. If there is no new text passing, the curator and the contributor can check the output and could find the report written by *HookTest* on Travis (see Figure 9). 30



```
Duplicate nodes found:
        phi0119.phi001.perseus-lat2.xml 648
        phi0472.phi001.perseus-lat2.xml 61.128 133, 61.137 142 147 157 162 167 172 177 182, 61.40
118, 61.50 60 138 143 148 158 163 168 173 178 183, 61.64 69 74, 61.65 70 75, 61.91 96 106, 62.19 25
31 38 48 66, 62.5 10, 64.328 334 338 343 348 353 357 362 372 376
        phi0474.phi055.perseus-eng1.xml 1
        phi1017.phi015.perseus-lat2.xml 94.53
```

Figure 9. Logs example for PerseusDL/canonical-latinLit

Conclusion

With around 100 million words available on PDL, and millions more words still to come through OPP, in a context of opening contributions up to wide ranging communities of users, dealing with ingestion of new texts scalably is a matter of security, flexibility and efficiency. Developing stronger and more flexible guidelines has helped the project move towards generalization of its norms and reduced the cost to encode, develop and curate. 31

With a strong continuous integration service in place, we can now support not only a wider range of genres and languages, but also a wider diversity of contributors. We can delegate the tedious tasks of checking markup to the machine, leaving curators free to focus on the scholarship. We also expect that automating checks on the integrity and the adaptability of textual objects for specific frameworks can reduce the error rate and allow for shorter feedback loops to contributors and users of our corpora. 32

Notes

[1] Tree, or as put by [Renear, Mylonas and Durang 1993] “Ordered hierarchy of content objects (OHCO)”, is a model that many texts of western classical literature can fit. This modelization is the same that supports the real bases of *TEI*. See “Complicating the Issue” in the *TEI Guidelines* [TEI-C 2007]

[2] *GetValidReff* for Homer's *Iliad*, with a level parameter set to 2, should return identifiers for all 15,693 lines to the *API* client. See http://www.perseus.tufts.edu/hopper/CTS?request=GetValidReff&urn=urn:cts:greekLit:tlg0012.tlg001.perseus-grc1

[3] In a local-only configuration, *HookTest* displays results of tests on the console or a local log file.

Works Cited

Almas 2013 2013-05-01 *Perseus CTS API* Bridget Almas

Almas and Clérice 2017 Bridget Almas Thibault Clérice *Hook Hook* 2017-06-19 Zenodo

Almas, Clérice and Munson 2017 Bridget Almas Thibault Clérice Matthew Munson *CapiTainS Guidelines 2.0.0* 2017-05-02

Almas, Clérice and Munson 2017 b Bridget Almas Thibault Clérice Matthew Munson *HookTest: 1.1.2* 2017-06-23 Zenodo

Clark 2001 Clark James *The Design of RelaxNG* 2001

Clark 2001 (2) Clark James *JingTrang* 2001

Crane et al 2013 Gregory R. Crane *Open Greek and Latin Project* Humboldt Chair of Digital Humanities 2013-12-13

Crane et al. 2015 Gregory R. Crane *Perseus Digital Library Canonical Latin Literature Repository* 2015

Elliott, Bodard, Cayless et al. 2006 2006-2016 Tom Elliott Gabriel Bodard Hugh Cayless *EpiDoc: Epigraphic Documents in TEI XML*

Fowler 2006 Martin Fowler *Continuous Integration* 2006-05-01

Franzini and Foradi 2014 Elena Franzini Maryam Foradi *Latin and Greek Texts: What Are We Reading in Schools and Universities?* Humboldt Chair of Digital Humanities 2014-09-10

Hardt 2012 2012 Dick Hardt *The OAuth 2.0 authorization framework*

Huizinga and Kolawa 2007 *Automated defect prevention: best practices in software management* Dorota Huizinga Adam Kolawa 2007-01-22 Wiley-IEEE Computer Society Pr

MySQL 2004 2004 MySQL AB *MySQL database server*

Pierce 2002 Benjamin C. Pierce *Types and Programming Languages* 2002 MIT Press

Ragnall, Talbert, Horne and Elliott 2008 Roger Bagnall 2008 Richard Talbert Ryan Horne Tom Elliott *PLEIADES, A community-built gazetteer and graph of ancient places*

Renear, Mylonas and Durang 1993 *Refining our notion of what text really is: The problem of overlapping hierarchies.* 1993-01-06 Allen Renear Elli Mylonas David Durand

Ronachter 2010 *Flask (A Python Microframework)* Armin Ronachter

Smith and Blackwell 2012 Neel Smith Christopher Blackwell 2012 *An overview of the CTS URN notation* Homer Multitext project

TEI-C 2007 TEI Consortium *Complicating the Issue* 2007

TEI-Consortium 2002 TEI Consortium *TEI: P5 Guidelines* 2002

TEI-Consortium 2007 TEI Consortium *TEI: P4 Guidelines* University of Virginia Press 2002