eCOMMONS

Loyola University Chicago
Loyola eCommons

Computer Science: Faculty Publications and Other Works

Faculty Publications

11-2015

Towards Sustainable Digital Humanities Software

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Shilpika Shilpika
sshilpika@luc.edu

Nicholas J. Hayward
Loyola University Chicago, nhayward@luc.edu

Saulo Aguiar
Loyola University Chicago, saguiar@luc.edu

Konstantin Läufer
Loyola University Chicago, klaeufer@gmail.com

Recommended Citation

George K. Thiruvathukal, Shilpika, Nicholas Hayward, Saulo Aguiar, and Konstantin Läufer, Towards Sustainable Digital Humanities Software, Chicago Colloquium on Digital Humanities and Computer Science, 2015.

This Presentation is brought to you for free and open access by the Faculty Publications at Loyola eCommons. It has been accepted for inclusion in Computer Science: Faculty Publications and Other Works by an authorized administrator of Loyola eCommons. For more information, please contact ecommons@luc.edu.

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License.

Towards Sustainable Digital Humanities Software: Initial Efforts to Assess Quality in DH Projects

George K. Thiruvathukal, Shilpika, Nicholas Hayward,
Saulo Aguiar, and Konstantin Läufer
Loyola University Chicago - CS Dept, CTSDH
Contact: gkt@cs.luc.edu

http://wssspe.researchcomputing.org.uk/

"**Progress in scientific research is dependent on the quality and accessibility of software** at all levels and it is now **critical to address many new challenges related to the development, deployment, and maintenance of reusable software**. In addition, it is essential that scientists, researchers, and students are able to **learn and adopt a new set of software-related skills and methodologies**. Established **researchers are already acquiring some of these skills**, and in particular a specialized class of software developers is emerging in academic environments who are an integral and embedded part of successful research teams. WSSSPE provides a forum for discussion of the challenges, including both positions and experiences, and a forum for the community to assemble and act."

Doesn't this sound like something the DH community needs to do as well? We thought so, so we decided to take our existing NSF-funded project, focused on software engineering for scientific software development and exploring its potential for DH projects.

Position Paper at WSSSPE

Software Engineering Need not be Difficult by Carver and Thiruvathukal

We argued for two general sets of practices for agile projects using free/open source hosting (which most scientific software projects use):

- make use of tools/techniques available by default
- add tools/techniques possible with not much additional effort (possibly developed by the community) to the mix

Our work focuses on the development of one of these tools, metrics, as a dashboard/service being made available to the community

Tools/Techniques Everyone Can Use

- Source code management (a.k.a. version control)
- Issue Tracking
- Wikis (i.e. documentation matters!)
- Project Management (we often use Trello)
- Continuous Integration (Jenkins, Travis, etc.)

Tools/Techniques Everyone Could (Should) Use

- Unit Testing
- Test-Driven Development
- High-Level Requirements (Use Cases, User Stories, UML)
- Metrics
- Documentation (a user manual is easy to write with tools like Sphinx)
- Continuous Integration
- Code Review
- Abstraction - modern, high-level languages often lead to more maintainable code (luckily, they’re often used in DH programming)

Selected DH-related Projects on GitHub

We are *only* looking at software (not content) projects with active commits and open issues.

Project	Commits	Open Issues	Defect Density	Spoilage
Abjad - Musicology	19,271	134	4.125	0.151
Zotero - Bibliographies	6,769	321	2.181	0.259
Omeka - Publishing	5,267	52	0.162	0.04
TEIC Stylesheets - Editing	5,501	22	0.055	0.018
CollateX - Editing	2,791	11	0.109	0.022
NLTK - Natural Language	11,629	241	0.113	0.295

Measures in software engineering

process related

- development time (attribute), time (measure)
- issue opened (attribute), date (measure)
- tests passed (attribute), number of tests (measure)

code related

- code size (attribute); kloc, ncloc (measure)
- function complexity (attribute), cyclomatic number (measure)

All of these are direct (base) metrics. They can be measured directly without performing a calculation.

Derived metrics

Issue/Defect Density - calculated from reported issues/defects and the size of a product (i.e. code size)

Issue Spoilage - calculated from reported issues/defects by looking at how long it takes to address the issue

Test Effectiveness - calculated from number of tests passed and number of issue

Uses of Metrics

To understand the quality of the software.

To improve the software development process.

While some practices such as version control, issue tracking, and testing (via continuous integration) are used in open source projects, the use of metrics is not prevalent.

Misuses of Metrics

To evaluate individual performance.

Example:

- Programmer A writes 10 lines of code and introduces 10 bugs
- Programmer B writes 100,000 lines of code and introduces 100 bugs
- Programmer C writes 1 line of code and fixes 10 bugs

Who is the best programmer?

While interesting to managers, this is of little use when it comes to software engineering. If the software is of high quality, it is a team success.

Our current work: In-Process Metrics

Three works in progress:

- Defect (Issues by state) Density
- Issue Spoilage
- Test Effectiveness

We'll cover each of these.

Why Digital Humanities and Computational Science projects?

- We noticed that there are many *interdisciplinary* projects that don't often have formally trained computer scientists and/or software engineers on the team.
- We're interested to know whether software metrics can help to find evidence of quality in existing projects.
- We're also interested in providing tools that any project can start using to understand opportunities for process improvement.
- We know, anecdotally, that software engineering is eschewed in many circles, even among professional programmers. What if we could make methods available to any project without requiring any complex software to be installed: Software Engineering as a Service (SEAS).

Quality - Deming Style

The overarching goal of quality is to provide total customer satisfaction:

- Who are the customers?
- How do the customers provide feedback?
- When do we acquire and lose customers?

Software quality concerns itself with:

- functional quality - conformance to a design, often driven by formal requirements
- structural quality - non-functional req’s, e.g. maintainability, reliability, correctness

Defect Density

- Defect Density is the number of *confirmed defects* detected in software/component during a defined period of development/operation divided by the size of the software/component.
- Defect Density = Number of Defects / Module Size
- Defect density for a given project in a Github repository is calculated as
  ∑ Issues / ∑ KLOC
- $\sum KLOC = \sum_{i \ \forall \ i=(1 \text{ to } n)} \sum KLOC_{(i,j),\ j=(1 \text{ to } m)} =$
  $\sum_{i \ \forall \ i=(1 \text{ to } n)} (KLOC_{(i,1)} + KLOC_{(i,2)} + KLOC_{(i,3)} + \ldots + KLOC_{(i,m)})$
  where n = number of files in a repository and m = granularity requested by the client, e.g. week, month.

Defect Density - Code Snippet

- written in Scala
- https://github.com/sshilpika/metrics-dashboard-storage-service/blob/master/src/main/scala/service/commitIssueCollection.scala#L14-L63

Issue Spoilage

- Spoilage is a measure of how much effort is spent fixing faults rather than building new code.
- Issue Spoilage = Efforts spent fixing issues/ total project effort.

  $= \sum_{i \ \forall \ i=(1 \text{ to } n)} T_{Issue(i)} / (T_{LastCommit} - T_{FirstCommit})$ where n = number of issues

- A well maintained project should have a spoilage < 1. However, if the project development/maintenance has stopped with large number of open issues, this could cause spoilage to cross 1 which is a red flag and an indication for immediate measures for fixing unresolved issues.

Issue Spoilage Code Snippet

- written in Scala
- https://github.com/sshilpika/metrics-dashboard-storage-service/blob/master/src/main/scala/service/CommitDensityService.scala#L274-L330

Visualizations - Tests

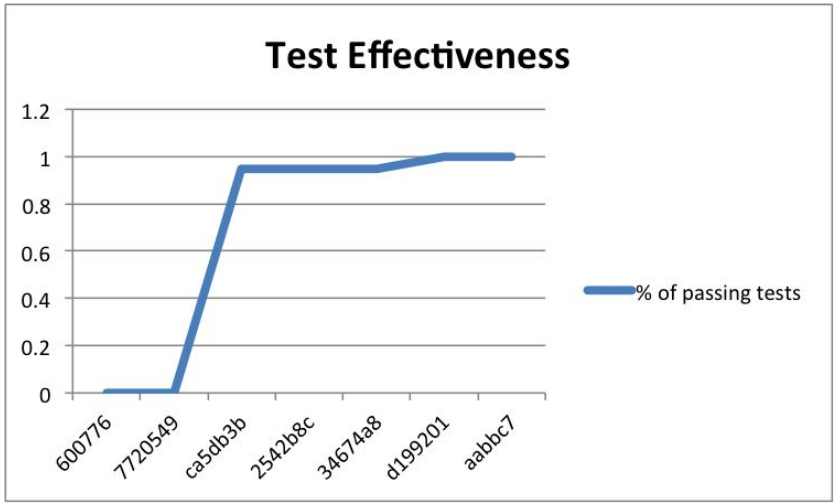
- Software Metrics' Tests

Test Effectiveness (work in progress)

- Test Effectiveness is the ratio between the number of *tests executed* in a software/component and the *number of passing test cases* in a given snapshot of the project.
- This feature is under development
  - For every snapshot of the project
    - run and filter test data output
      - how many tests were executed?
      - how many of them were passing?
- Challenges
  - Requires specific implementation for each language
    - Knowledge of test suits for every language
  - Management of process execution ... what if a bad written test doesn't stop executing?

Test Effectiveness

- Example
  - Process Tree Scala (one of our operating systems teaching projects)
  - https://github.com/LoyolaChicagoCode/processtree-scala



Conclusions and Future Work

- Everything here to be live at metricsdashboard.cs.luc.edu
- We’re in the middle of a transition to AWS for performance and availability
- Working to replace precomputing with a (near) real-time analytical system based on Apache Spark (running on supercomputers at ANL)
- etc…