EvaP

Jennifer Stamm, Stefan Neubert

Winter Term 2015/16

Inhaltsverzeichnis

1 Introduction

1.1 EvaP – An Evaluation System

The online platform EvaP is used for evaluation of courses at the Hasso Plattner Institue (HPI). Its development started in 2011 when the student representatives decided to redevelop the former system EvaJ. Now, it is an Open Source project hosted on GitHub, even though the main developers are still part of the student representative team.

EvaP's time to shine is at the end of each semester. Each university course is evaluated with specific questionaires chosen by the lecturer. Students are allowed to give anonymous feedback to different aspects of the lecture, the lecturer itself and additional tutors through a grading system and comments. EvaP encourages evaluation by rewarding participation with points. These reward points can be redeemed for currency to use at HPI events. EvaP's latest feature is the distribution of course grades through the platform.

1.2 Motivation – Why Should EvaP Be Analyzed, Verified and Tested?

EvaP is an important source for feedback at HPI. The platform offers students a way to express their critique anonymously. It documents the feedback for lecturers. Thus, they do not need to collect and save the feedback themselves. Additionally, they can take their time to evaluate the student's feedback. Furthermore, the evalutation of all courses is saved centrally. This allows to gain an overview over the quality of HPI courses and compare feedback over time as well as with other courses. Therefore, EvaP is an important tool at the HPI.

Since EvaP is developed by students, the responsible persons and main developers change regularly as older students graduate and new ones enroll. The change of responsible persons shifts the view of which features, programming paradigms and quality assurance are most important. Consequently, the requirements change. The change of main developers means a loss of knowledge about the existing code. Besides, the Open Source aspect allows several developers without inside knowledge to add code. Even though all code is reviewed and checked by the main developers, they may not grasp it as well as self-written code. Events like Hackdays or Hacking Hours are used to promote EvaP's development. While these practices ensure the advancement of EvaP, they may endanger its quality.

Because of EvaP's importance at the HPI, its quality should be ensured. Therefore, we will analyze, verify and test the software within the scope of this lecture.

2 Milestone 1

As suggested the duration of milestone 1 is from the beginning of the project until the end of december. Milestone 1 includes the set up of the software project which led to

the discovery of the first bug. It includes the first steps taken to gather information, get comfortable with the project and planning of the project. Lastly, it includes the phase of testing the project regarding graph coverage.

2.1 Set Up

- set up of vagrant on windows does not work for submodules - unnoticed by developers because of different os

2.2 First Steps

Incidentally, this is the first semester for the student representatives to host *EvaP Hacking Hours* biweekly. This is an event to give students a space to develop and work on EvaP. We will use it as a possibility to stay in contact with the main developers. As testers of EvaP it is a valuable resource to be able to talk directly with developers. This way we can gather current information easily. We are able to verify the content of old artifacts with them as well as our future findings.

2.2.1 Application Survey

The current main developer of new features is Johannes Wolf. He is supported by Johannes Linke who is mainly responsible for a good coding style, including code review and refactoring. We interview them about information regarding the first steps of our project.

Development Paradigm and Development Languages There are no development paradigms explicitly determined. But as stated there is a main developer responsible for code review. As it is all newly developed code is reviewed before it is accepted. Everyone is able to review code and discuss it with the coder and other reviewers. This practice shall ensure readable code and distribute knowledge about code changes. Additionally, it is implicitely assumed that paradigms of the used development languages are followed. The development languages are:

- Python 3 through the Django framework
- HTML
- Javascript
- CSS

As an example for paradigms given by the languages we checked the document known as *PEP 0008*. This is the style guide for Python code written by Guido van Rossum, the author of Python, and followed by most open source projects in Python. Link: `https://www.python.org/dev/peps/pep-0008/`

Requirements / Specification / Documentation / Artefacts Requirements were elaborated at the start of EvaP's predecessor EvaJ several years ago. No original artifacts were stored even though most requirements still hold. Examples are:

- Evaluation should be anonymously
- Written feedback is only readable to a small, strongly specified selection of people
- Written feedback is reviewed before it is available to the lecturer

As expected of an open source project managed by students there is no formal specification of EvaP. Though there are different information gathered in a wiki hosted on Github. (Link: `https://github.com/fsr-itse/EvaP/wiki`) The responsibility to specify new features is on the main developer, Johannes Wolf. He collaborates directly with the users of the feature.

Current testing status / Bug repositories The project is hosted open source on GitHub. Different tools allow to include badges on the overview page to display the status of the latest build (Graphic 1). The following tools are already used:

- *Travis CI*: A continuous integration service to build and test projects hosted on GitHub
- *Gemnasium*: An automated service for monitoring project dependencies for possible updates
- *Landscape*: A service checking the code for errors, code smells and deviations from stylistic conventions
- *COVERALLS*: A service relying on Travis CI that tracks the code coverage
- Another feature of Github is used to track bugs: the label *[T] Bug* for *issues*.

Since this is already an extensive list, we will not to spend much time on researching further tools.

Abbildung 1: Badges of testing tools on the GitHub overview page (01.12.2015)

Personal involvement Our first contact with EvaP was as users. As HPI bachelor students we evaluated courses of the first semester. As a tutor for a course Stefan used EvaP to read feedback. As a member of the student representatives Jennifer reviewed comments before publishing the evaluation. During the *Evap Hackday 2014* and *Evap Hackday 2015* Jennifer joined the team developers. She familiarized herself with the development practice to fork, code and create pull requests for review and solved several small issues.

2.2.2 Initial Test Plan

We developed an initial plan based on our findings in the application survey. As suggested we followed the questions discussed during the lecture. The test plan includes evaluating and enhancing the existing artifacts, cross-checking the results found by the tools used and eventually improving the testing status.

Five V&V Questions We started with evaluating EvaP regarding the five basic verification and validation questions:

1. When do verification and validation start? When are they complete?

   Verification and validation started along the start of the project and will be an important part during the duration of the project.

2. What particular techniques should be applied during development?

   As EvaP is a software too big to be wholly tested by us within the scope of the seminar project, we will try to apply as many techniques as possible on a small subpart of the project. We will evaluate the techniques and make recommendations for further testing to the main developers.

3. How can we assess the readiness of a product?

   Since EvaP is already in use it is certainly ready. Our testing will not influence the readiness.

4. How can we control the quality of successive releases?

   Additionally to continous integrationa and automated tests, a code review before merge is already implemented in the development process to control the quality of successive releases. We hope to enable developers to gain more knowledge about the system by enhancing the available artifacts and documentation.

5. How can the development process itself be improved?

   The main weakness of the development process is the sparse distribution of knowledgeand the change of developers over time. Since only students are interested in developing EvaP we do not see a possibility to improve this part of the process.

Test Classifications and Approaches Considering the following test approaches we came to the conlusion described below:

1. Validation vs. Defect Testing

   Since there is no requirements document or even a list of features, it is hard to implement validation testing. As we will not draw up a corresponding document, we will not apply validation testing. However, if possible, we will try to detect defects in the software by finding inputs leading to incorrect behaviour. Thus, we will apply defect testing.

2. Development, Release, User Testing

   Because of the applied continuous integration process: Development testing is already in use. Release testing and acceptance is not possible; there are no specified releases. User testing is applied in a way as most users familiar with the developer report encountered bugs directly.

3. Unit/Component, Integration, System Testing

   EvaP is a web application, so there are no hardware components directly involved. While there are software dependencies, this aspect is already covered by the tool Gemnasium. For a web application it would be important to work in the most popular web browsers (desktop and mobile). Since there are no complaints known about EvaP not working in a web browser and no plans exist about developments that would use different web browser features, we will not focus on integration testing. Similar to validation testing, without a specified requirements we can not execute system testing. Instead we will focus on unit and component testing.

In conclusion, we will use tests to find unnoticed defects and we will apply different testing techniques on units or components of EvaP. These testing approaches align with our goals to help EvaP's developers and to learn about different testing techniques.

Available Artifacts The most thorough artifact, the GitHub wiki, offers two artifacts with potential regarding coverage-based testing:

- A finite state machine describing the states of courses in the evaluation process (`https://github.com/fsr-itse/EvaP/wiki/Evaluation-States`)
- Description of a few use cases with UML use case diagrams (`https://github.com/fsr-itse/EvaP/wiki/Use-Cases`)

Initial Test Plan Based on our findings, the initial test plan is as follows:

- Create a control flow graph of a function, apply coverage criteria to define test sets and implement tests
- Check and if necessary update the FMS of evaluation states

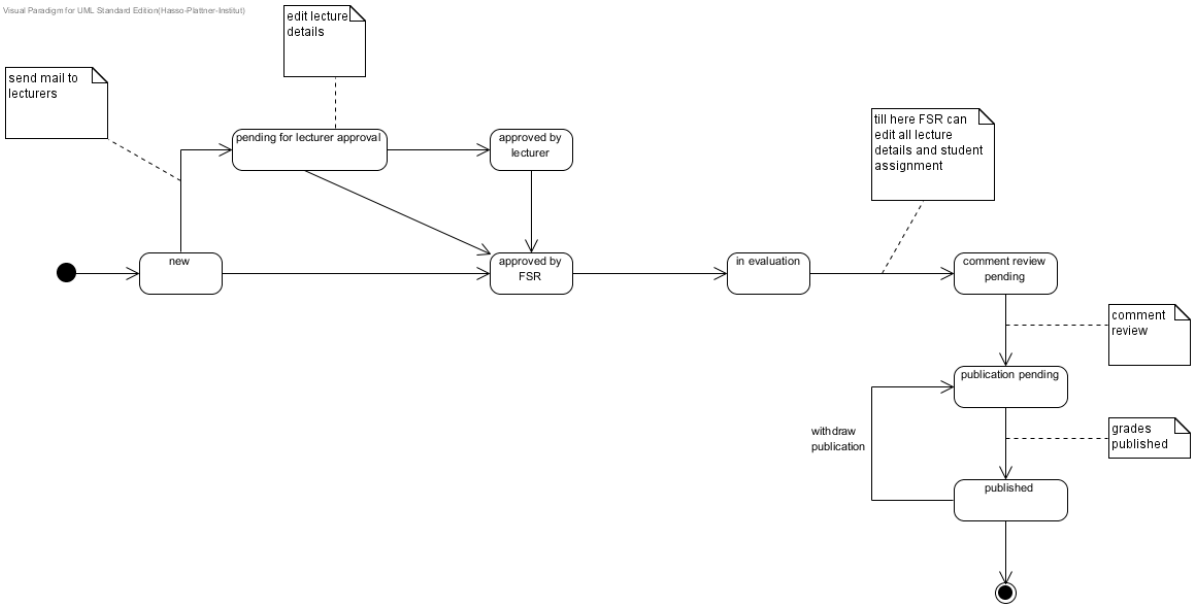

Abbildung 2: Original FSM: Possible states of a course

- Use or create a UML use case diagram including its elaboration to develop an activity diagram
- Investigate coverage found by the tool COVERALLS

2.2.3 Test Automation

The project is already covered by several tools. Our research showed the used tools are popular in the Python community if the project is hosted on GitHub. Because of this and since the developers are comfortable with their choices, we will observe the functionality of the tools during the project. We will focus this section on describing our experience with the set up of running tests locally.

Content of this part: Stefan has fun with PyCharm

2.3 Graph Coverage

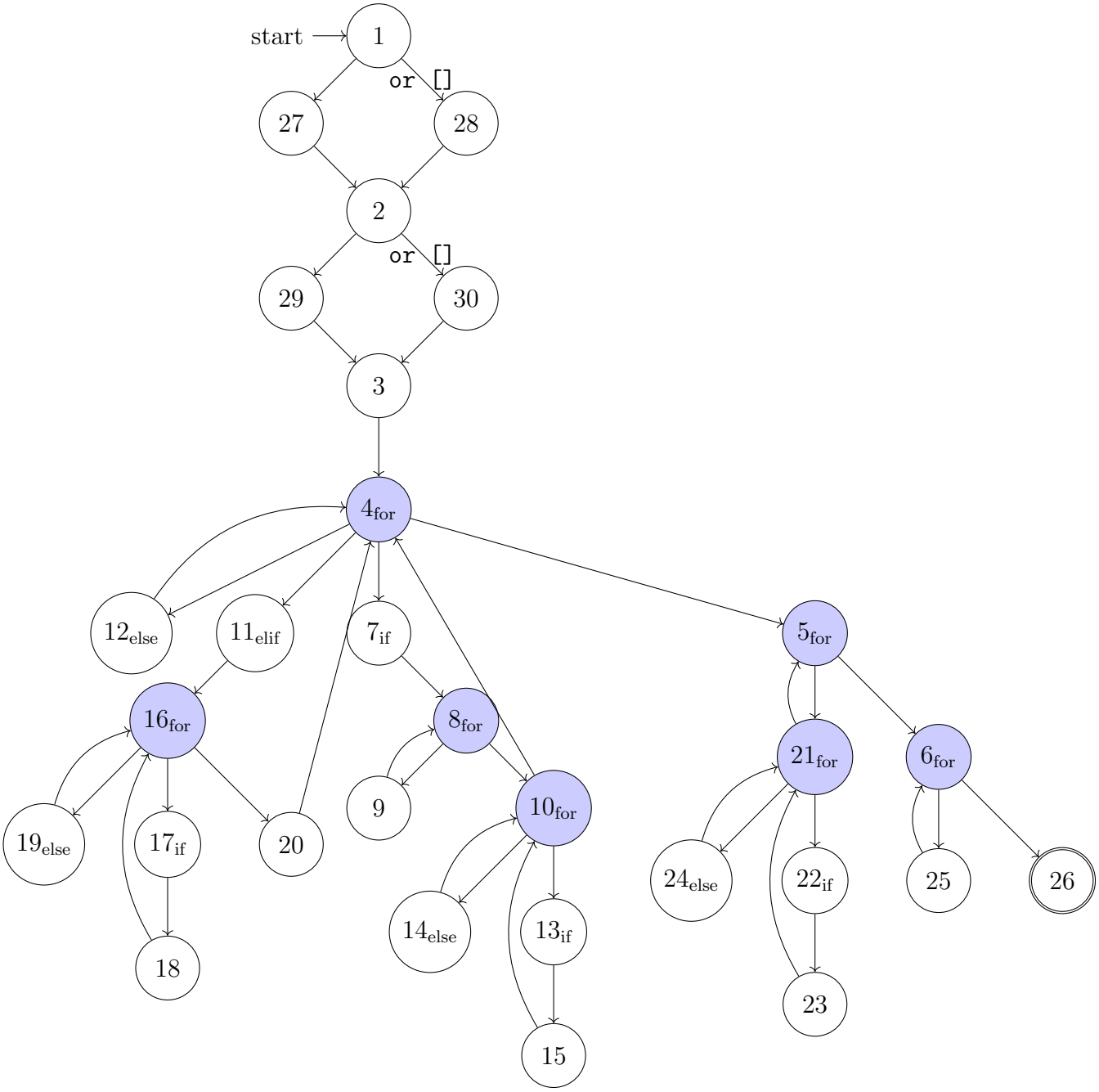

Abbildung 3: Control Flow Graph of the Function `send_publish_notifications` in `evap/evaluation/tools.py`