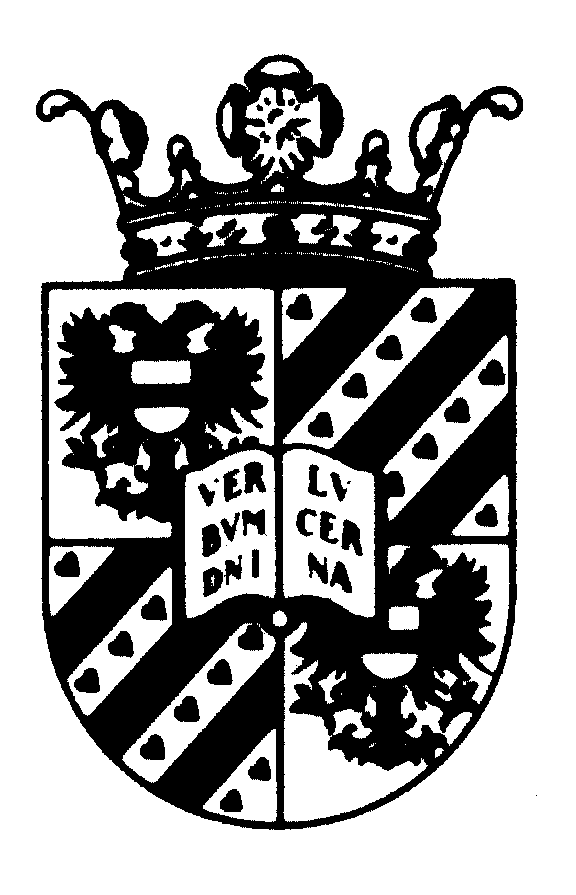
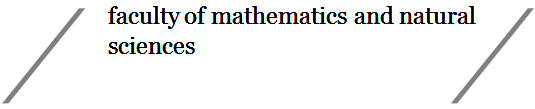
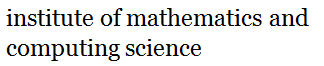
**Final Project Bachelor Computing Science**

**Course code: WBCS13000**

**Credits (EC): 15 points (420 hours)**

**1. General information**

**Project title: “A Dashboard for the Automatic Monitoring Python Web Services”**

**Date:   
  
First supervisor: Mircea Lungu  
  
Second supervisor: Vasilios Andrikopoulos**

**External supervisor (if applicable): TBD**

**2. Project description:**

“The web is the only true object oriented system”, says Alan Kay, the inventor of object oriented programming. This because the service oriented architecture of the web embodies the early ideas of the OO paradigm much better than the way it has been implemented in the mainstream programming languages. Indeed, the web becomes increasingly **a system of interconnected services**.

One of the most popular programming languages of the recent years is Python. Combining a simple structure with a large offer of libraries Python is also one of the most popular languages for implementing web applications and web services. Within the Python ecosystem, one of the **most popular solutions for implementing web services in Python is Flask** and the Flask-REST package.

One of the downsides of services is the fact that the developers of the web service do not have information about which parts of their web services are used the most and which are used the least or even not at all. In a Java system, for example, one can perform static analysis to discover such dependencies, but on the web, everything is dynamic. However, there is no equivalent way of analyzing the dependencies to a given web service. The only way to do it is to monitor the actual service being deployed and collect statistical information about it.

Another challenge is monitoring the impact of the system evolution on the quality of the service, and in particular, on the performance of the service. To do this, one must keep track of the performance of the web services over the evolution of the system. Two solutions are evident in this context:

1. Observing the evolution of the performance of unit tests as the system evolves over time.
2. Observing the evolution of the live service performance for the deployed services over the evolution of the system.

Finally, since these problems are problems that every service developer will encounter and since there is no existing off-the-shelf solution for Python, the goal of this thesis, is to implement such a service. Moreover, to make it really easy to use, the students will research a way of automatically generating a dashboard for an existing service that involves the minimal effort from the part of the service developers.

The project will generally follow the following steps:

- The students get acquainted with the technology of Flask and Python by implementing a very simple web-service

- The students implement a very simple Python module that allows for simple annotations to be added to the endpoints of interest in order to allow the instrumentation of the given services

- The students implement a basic dashboard that presents the information about: endpoint usage, and evolving endpoint performance

- As a validation, the endpoints deploy the infrastructure on Zeeguu, a research project developed within RUG which provides **a series of web services that support the learners of a foreign language** to accelerate the speed of their acquisition of the vocabulary of a second language. The web service is implemented in Python with Flask, so it will be the perfect test case for the technology.

In parallel with the practical steps outlined above, the students are going to conduct a survey of the state of the art in automatic monitoring of service performance by searching for relevant articles in the relevant journals.

**3. Methodology and timeline:**

The deadlines for this project are the following:

1. Week 1-2: Getting acquainted with Flask + Python. Possibly they look into the usage of Flask-REST too.
2. Week 3-4: Investigating function wrappers and implementing a basic monitoring module. Ensuring that the monitoring can be done evolutionary with test cases, but also *in vivo* on the deployed system. Finally, the statistics about the client usage for the different endpoints should also be collected.
3. Week 5-6: Investigating the alternatives for dashboard design and implementing a first version of the dashboard.
4. Week 7 - 8: Validating the system by deploying it on the Zeeguu services.
5. Week 9-10: Thesis writing + Presentation Preparation
6. Week 11: Final presentation
7. Week 12: Final touches on the thesis

**4. Division of tasks**

The project is targeted at two students working on complementary aspects: e.g. monitoring backend & dashboard. Other divisions of labor could also be envisioned. Two independent theses will be submitted

**5. Deliverables:**

* Software source code
* Validation as deployed system
* Source code documentation + high-level design documentation
* BSc thesis

**6. Grading**

* **Process:** 25%
* **Thesis:** 25%
* **Technical contributions:** 15%
* **Scientific contribution:** 20%
* **Final presentation:** 15%