GEOSOFTWARE II, WiSe 2020/21
Open Earth Observation Data Processing

Edzer Pebesma, Daniel Nüst, Christian Knoth

October 2020

Introduction

Important links

LSF/QISPOS: 142952

LearnWeb: Geosoftware II WS 2020/21

Mattermost: https://zivmattermost.uni-muenster.de/geosoft2/

Course goals and schedule

Geosoftware II is set up to be **the most challenging course of your studies**. Students work as a group on a project to solve a complex geoinformatics challenge, which no one has worked on before and where the right path to an acceptable solution is not known, not even by the teachers. The solution will involve choosing the best tool independent of previous programming language experiences.

The students conduct joint group work, forming a company/team that is the *contractor*, who bids for a project tender published by the *customer*, the teachers. The customer publishes a problem statement and an invitation to bid[1] for solving a relevant problem or business case. The *contractor* provides the managers, architects, designers, and developers of a software system to advance the state-of-the-art in geoinformatics, in this year in the context of big Earth data processing in cloud infrastructures. In this project, students get to know the latest geospatial data processing concepts and libraries, especially for big data, and gain valuable experience for their future jobs as software developers, researchers, project managers, etc.

The group conducts the project using AGILE methodologies, i.e., Scrum project organisation. Due to the complexity of the task and the shortened semester due to the COVID-19 pandemic, the teachers are especially supporting the task planning and scheduling in the first few weeks. The main lessons learned from Geosoftware II should be practical experiences in understanding someone else's problem, designing and implementing a software solution to it, and staying within the project deadlines. The core skills built evolve around the code and programming, especially in *collaborative software development*. Therefore, during the project work, the responsibilities of non-programming tasks rotate, so that every student spends the majority of their time as a developer while gaining experiences in other roles. Each student serves for three weeks as either project lead (incl. public relations and customer contacts), tester/technical writer (documentation), or scrum master.

[1] https://en.wikipedia.org/wiki/Invitation_for_bid

Topics for Introductory Presentations

In the first two course sessions, the following topics will be presented by individual students or groups of two. These topics provide background on the project setting and goals. The presenting students are then go-to experts on the respective topic if further questions arise during the course of the semester. The students must provide a handout written in R Markdown in form of a pull request to the course's GitHub repository Geosoft2/geosoft2-2020 and may use any suitable means to present their findings.
The topics and allocations are managed in GitHub issues with label 'topic'.

Student roles and project management

The students work together as one project team at a fictitious company. This company has made a commitment to follow the Scrum methodology of AGILE software development. The majority of the people and person hours are naturally dedicated to software development, which is in line with the course's main goals. However, for 15 people to effectively work together, there is a need for coordination and management. Therefore, every student will take on a non-developer role for a specific period of time during the semester. There will always be exactly one student per role, who hands over the current lists of tasks and lessons learned to the next person. To reduce friction and increase learning experience, important experiences and lessons learned are noted in one document per role, which is regularly updated by the respective role holders. These documents are shared across the whole project team to allow students to learn from experiences beyond the role realised by themselves. Students are encouraged to take on challenging roles beyond their current areas of comfort.

The roles and responsibilities are as follows. Details may be adjusted during the project duration by the whole project team.

- **Project lead** ensures project success, e.g., organise required resources, manage schedule and budget, document progress, is mindful of project scope and overall goals, manages risk; together with Scrum master: plan next sprints; together with product owner: decide on scope + features and set priorities
- **Account manager / product owner** ensures the client's needs are met, and that he client is always up to date on the project, e.g., preparing of presentations, reports, communication about problems/questions, eliciting feedback on prototypes; together with lead decide on scope + features and set priorities; together with testers: prepare regular demonstrations;
- **Scrum master** ensures Scrum methodology is applied effectively, e.g., oversee and facilitate team communication, manage backlog together with project lead, plan + moderate backlog grooming and sprint planning meetings, document any lessons learned from the development process, manage backlogs, be aware of friction and problems, arbitrate technical and personal conflicts (the latter in coordination with teachers!); together with project lead: plan next sprints
- **Software testers (2)** ensure quality of the product, e.g., they create tests plans (identify what to test), gather/manage test data, manage and monitor test environments, execute tests, fix bugs, document project progress together with project lead, gather (performance) metrics, review code; together with account manager: prepare regular demonstrations;

Table 1: Possible assignment of special roles if not all participants (here: 9) take on a special role. Overlaps ensure smooth transitions, final week is "all hands on deck".

Week	Project lead	Product owner	Scrum master	Software tester
1	A	C	E	G & H
2	A	C	E	G & H
3	A	C	E	G & H
4	A	C	E	G & H
5	A/B	C/D	E/F	G&H/I&J
6	B	D	F	I & J
7	B	D	F	I & J
8	B	D	F	I & J
9	B	D	F	I & J
10	A/B	C/D	E/F	G/H/I/J

For the sake of completeness, the role of **project team member/developer** is responsible for implementing the understood and agreed tasks, including deciding what tasks are in the next sprint, writing software + tests, and estimating the efforts to complete tasks.

The course duration is 10 weeks. This gives 50 weeks of special roles to fill for 8 students, and therefore about 3 weeks per assignment. The final week before the submission deadline will bring together all role holders to ensure the project success. The role teams prepare the final versions of the experience documentation. Detailed task assignments during the final week are left to the project team.

Efforts and grading

9 ECTS correspond to 225 – 270 working hours (following calculations assume the lower boundary for WiSe 2020/21). We allot 23 hours for preparing introductory presentations, attending introductory presentations, and attending the final presentation, and 2 hours for writing the individual report. That leaves 190 hours over 10 weeks (not counting Christmas holidays) for the implementation. This gives an average workload of a little bit over **two full working days per week** during the semester for each student!

What	Work load (hrs)	Grade contribution
presentations & report (topic preparation, attendance, final presentation, personal report)	25	15%
planning (bid, schedule)	16	15%
implementation (incl. technical documentation, demonstration, regular meetings, project management tasks)	190	70%

Grading of the implementation is based (a) on an overall grade for the result, the project progression, especially adjusting and reacting to challenges and internal communication and cooperation between colleagues, and (b) individual code contributions as shown in the software development platform. Because of the latter, we require each student to contribute under their **own account** and co-author joint commits[2]. A fork & pull development model is highly encouraged because it allows to list the relevant pull requests in a final **individual report**. This report must present personal *learning achievements*, *role(s) in the team*, and individual *contributions* in a single PDF document (max. 3 pages) - see "Individual reports" section in the GitHub repository.

[2] https://github.blog/2018-01-29-commit-together-with-co-authors/

1 Invitation to bid (Lastenheft)

1.1 Project title

An openEO back-end driver for geospatial data science using the Pangeo software stack

1.2 Problem statement

The amount of remote sensing data is constantly increasing. Processing such datasets in an effective and transparent way is challenging for data scientists. To process big geospatial data effectively, users turn to cloud-based scalable infrastructures and parallel computing. Here it can be hard for non-experts to leverage the latest powerful software libraries around. To process big geospatial data transparently, researchers require open infrastructure that they can fully inspect, extend, and, if need be, deploy themselves.

The openEO project developed an API for exposing the power of big Earth observation cloud infrastructures in a standardised way based on the data cube paradigm[3]. The *openEO API* allows users to easily switch between service and infrastructure providers and use the most suitable or cost-effective solution. openEO's success is in part due to the availability of different implementations in multiple programming languages and using different processing stacks in the background.

Therefore, a further openEO back-end driver should be implemented using latest Python-based numerical processing libraries.

1.3 About the customer

The customer is a secretive SME (small and medium-sized enterprise) that plans to sell services that create value-added products based on satellite imagery (Copernicus, Landsat, Planet) and model predictions (e.g., weather, climate) to downstream users, where downstream users include

- weather services, e.g., involved in predicting risks of extreme weather conditions
- agricultural service providers, e.g., involved in predictive harvesting
- insurance companies, interested in location-specific adjustment of insurance rates, e.g., for storm damage, floods, droughts
- forestry services, e.g., advising forestry companies to adjust forest management due to climate change (drought risk, heat waves)

The customer is interested in using this product both to create the processes for value-added products itself, as well as allowing customers to modify these processes or develop their own processes. The revenue comes from providing cloud services that provide these products with up-to-date (near real-time) observation (satellite) or predicted (modelling) data at a lower cost due to synergy effects and specialisation.

[3] https://doi.org/10.1080/20964471.2017.1398903

1.4 State-of-the-art

The European Commission, after establishing that the data from the Copernicus Programmd[4] had to be freely accessible, soon discovered that for many (if not most) users the data would be too large to download. It then funded five DIAS (Data and Information Access Services) projects all meant to make cloud processing of Copernicus data easier (but not free). Each of these projects are meant to be commercially viable after project funding (5M € per project over 3 years) ends. We can see, e.g., from the WeKeo DIAS' price list that a user can get access to the data on the cloud platform by renting one or more virtual machines, and set up these machines according to own wishes. That is great, but it

- costs money,
- only removes the download step, while it still forces the user to install all software, and proceed on the VM as if the user was working "locally",
- does not abstract the collection of images (tiles, files) to image collections and/or data cubes requiring the user to preprocess/transform data before analysis, and
- hence, makes it very hard to reproduce a sequence of steps to share which processes lead to some scientific result.

The openEO project/initiative was first launched in a r-spatial blog post in 2016, arguing that in a heterogeneous cloud processing ecosystem, an open API could fill the role that GDAL[5] has as a software layer abstracting file formats behind a common data structure. This API was missing, and openEO successfully proposed to fill this role. A 2M € H2020 project was funded by the EU (2017-2020) and a follow-up project ("openEO Platform") was funded by ESA, aiming at providing services that are production-ready/commercializable. The full openEO H2020 project proposal is found on https://zenodo.org/record/1065474.

A parallel development that took place was the STAC (spatio-temporal asset catalogs); the openEO project, in person of Matthias Mohr, contributed significantly to this specification. Initially designed as a modern and light-weight service to describe assets (such as a satellite image) in terms of its sensor, location and time properties, openEO helped develop its ability to describe *collections*: sets of assets coming from the same sensor (or source), describing the same phenomenon. This development was done in collaboration with Google Earth Engine engineers, in such a way that Google could describe its image collections accordingly.

1.5 Project goals

The openEO API is nothing more than a communication layer between clients (e.g., a Jupyter notebook, RStudio session, or browser session) and back-ends (compute engines that have direct access to the data, running in some cloud, running some processing software). An openEO back-end driver is the software for an openEO back-end. An openEO back-end provides the access to the data (in terms of describing which data it has access to (`/collections`), provides access to processes that a user can run on a back-end (`/processes`), and can manage running jobs (`/jobs`) and existing process graphs. Currently existing back-ends can be browsed on the openEO Hub at `https://hub.openeo.org/`.

[4]`https://en.wikipedia.org/wiki/Copernicus_Programme`
[5]`https://en.wikipedia.org/wiki/GDAL`

We want to develop and deploy **a novel openEO API back-end driver** using **the Pangeo software collection**[6] and publish the same functionality, as exposed in the Pangeo Gallery via notebooks, using an openEO client and the openEO Hub. The new driver should be easy to install on any cloud provider based on containerisation. This way, we hope to provide a competitive offer and win many researchers as users in the area of geospatial data processing in science, because they can freely choose the client library and service provider for their workflows.

Concrete requirements and the process to fulfil this goal now follow.

1.6 Requirements

The requirements of this project are to create an instance, and the software to run an instance, that

- serves three years of Sentinel-2 data of an area around Münster (DE) consisting of more than one Sentinel-2 tiles, and that can compute monthly averaged NDVI values on a 100 m x 100 m cell resolution
- serves daily SST data on a 0.25-degree global grid for 1981-2020, and allows the computation of monthly or yearly mean sea surface temperatures (data can be downloaded by anonymous ftp from ftp.cdc.noaa.gov, in directory `/Projects/Datasets/noaa.oisst.v2.highres`, as yearly netCDF files)
- is compliant to the openEO API
- can be validated and examined using the openEO Hub software

The instance with the full data should be running at time of project delivery and two weeks afterwards; one or multiple Dockerfile(s) that, when built and started, runs an instance of the service shall be made available using trimmed-down data sets.

1.6.1 Functional requirements

The following functional requirements extend and clarify the above overall requirements, but they may not cover all features need to realise the project goals.

Data ingest

- load data via direct file upload to the server
- load data via an online reference (i.e., a download URL) passed through a bespoke API
- support at least NetCDF and Sentinel's SAFE file formats

Processes

- implement at least two processes (NDVI for Sentinel-2, mean sea surface temperatures) supporting two different resolutions
- each process has at least two configuration parameters

[6] `https://pangeo.io/`

Hub integration, API, and deployment

- all collections and processes are listed when an instance is connected to the openEO Hub
- the instance must support the capabilities endpoint `/`
- the instance must support CORS
- Docker/docker-compose configurations are used for packaging and deployment
- the developed software can be deployed to at least one cloud provider, chosen in coordination with the customer

1.6.2 Non-functional requirements

Training and Demonstration For training and demonstration purposes, a one-click set-up to deploy a test scenario must be provided. The scenario must include a data set and process that allows a quick evaluation, i.e., processing times less than 20 seconds. These small datasets may be included within the software containers. The contractor provides a test script or test suite to execute demo process and validate its response with a single command or click. In addition, the openEO back-end validator must be integrated in the test suite.

An independent instance of the openEU Hub is provided to demonstrate the functional requirements.

Maintainability The developed software must be published as open source software under an OSI-approved license according to the license requirements. The contractor must ensure license compatibility of other used or extended software. The contractor is free to use and extend existing open source software projects, e.g., those part of the openEO organisation on GitHub, or alternative Python implementations, e.g., SARScripts/openeo_odc_driver.

An established build and packaging workflow as well as dependency management must be applied, e.g., Gradle for Java, npm for JavaScript, R packages for R, or PyPI packages for Python. If more than one programming language is used in a component then different build and packaging steps must be scripted with a Makefile.

Documentation on building, installing, testing, and configuring the system must be provided in Markdown-formatted documentation files using the appropriate markup for lists, links, etc. The documentation must be rendered to an online website.

A common test software, e.g., pytest for Python or testthat for R, must be used to realise unit and integration tests for all requirements/user stories.

User friendliness The system must support intuitive use to the extend that targeted group can use its features without further documentation. Common practices for web-based user interfaces and interaction paradigms should be applied. The user interface and visualisation must be suitable for color-blind users. The system must be tested and fully functional with browsers representing at least 80% of current users.

Performance In general, a user interaction must result in a display change within 1 second to allow users to stay focused on their current train of thought; complex page contents may be loaded asynchronously; interactive visualisations must react within 0.1 seconds to give a user the impression of direct manipulation.[7] The full workflow of

[7]Source: `http://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/`

area selection, data retrieval, and visualisation must not take longer than 20 seconds for a demonstration case.

The contractor provides a test script to evaluate the performance using at least 5 different HTTP operations or views with an appropriate number of repetitions.

Deployment Docker must be used to ensure easy deployment. Used images must be based on Dockerfiles published as part of the source code, and all base images should be hosted on Docker Hub (or comparable).

Project management The contractor shall apply an agile development process oriented at Scrum. The customer must be given access to the contractor's online task management to observe the progress. The contractor may include the customer in regular meetings for updates and feedback on the progress.

1.7 Deliverables and deadlines

Bid The bid (Pflichtenheft[8]) must be submitted to the customer via WWU's Learnweb no later than **November 25, 2020, 12:00 CET**. It must comprise an implementation plan for all requirements of the invitation to bid (Lastenheft) as well as a schedule for the implementation as a single PDF document. The bid must include *user stories* realising all requirements. The bid document may be English or German. Its content should follow common standards for content and structure of bids.

Changes The bid is a binding agreement between customer and contractor. All changes to the bid after first acceptance require a written confirmation (email) by the other party. The contractor may direct *change requests* to the customer via email to both `daniel.nuest@uni-muenster.de` and `christianknoth@uni-muenster.de`. A change request includes a short summary of the changes in the email body and an updated version of the bid as an attachment.

Final delivery The final delivery must be submitted on the day before the final presentation on **February 10, 2020**, and contain the following items:

1. project report
   (a) a single PDF document submitted via Learnweb
   (b) cover page contents: project title, team name (optional), references to all parts of the delivery (code, documentation, etc.)
   (c) screen-shots of all relevant UI components and workflow steps
   (d) description how all requirements/user stories from the bid were fulfilled
2. commented source code in a single code repository on ZIVGitLab, GitLab.com, or GitHub.com; delivery in more than one repository must be formally accepted by the customer
3. ready-to-use Docker images in a public registry (if multiple images with docker-compose configuration) and Dockerfiles
4. installation and maintenance documentation (online website, source may be included in repository a separate repository)
5. API test suite (e.g., as documented collection of requests in a Jupyter/R Markdown notebook, or a Postman/SoapUI Open Source project)

Acceptance criteria The acceptance criteria encompass the fulfilment of all functional and non-functional requirements as described in the accepted bid, and the complete and on-time submission of all deliverables.

[8]https://de.wikipedia.org/wiki/Pflichtenheft, https://wiki.induux.de/Pflichtenheft, and http://www.infrasoft.at/downloads/Anleitung_zum_Pflichtenheft.pdf