PA1489 Assignment Descriptions

Mikael Svahnberg*

2025-06-23

1 Introduction

This document describes the graded assignments in the course PA1489 Basic Software Engineering

The course consist of the following assignments:

Assignment	Size	Description
Written Assignment 1	2.5 hp	Tools: Git, Docker, Testing and Debugging
Written Assignment 2	2.5 hp	Implementation and Documentation
Written Assignment 3	2.5 hp	Engineering Diary

All assignments are submitted and graded individually, but it is ok to collaborate in their creation.

2 Assignment 1: Tools

This assignment covers basic software engineering tools. Do note that in order to work with these tools, you also need a basic understanding of how your computer works, your development environment, what a file is and how to find and use it, and (to some extent) working with a command line interface.

This assignment consists of three parts (described further in subsequent sections):

1. Git (Configuration Mangement)
2. Docker and Docker Compose (Develop using Containers)
3. Testing and Debugging

Each of these parts are examined separately and individually. They are examined by showing a teaching assistant that you know how to work with and use the concepts. The TA uses a checklist to mark the parts that you have shown, and this is used to calculate a score for the entirety of the tools assignment. This score is then translated to a grade as follows:

*Mikael.Svahnberg@bth.se

Score	Percent	Grade
20	100%	
18	>90%	A
16	>80%	B
14	>70%	C
13	>65%	D
12	>60%	E
>12	<60%	F

3 Assignment 1 Tools: Git

3.1 TASKS Get Started with Git

3.1.1 Register an account

- Register an account at some git server:
  - `https://github.com/signup` or `https://education.github.com/pack`
  - `https://gitlab.com/users/sign_up`
  - `https://www.atlassian.com/software/bitbucket/bundle`
  - `https://codeberg.org/`
- Github is still very popular for open source projects
  - In some trouble for how they may use the code you upload ther
- Many migrated over to Gitlab when Microsoft bought Github
- Atlassian and BitBucket are very well integrated with the rest of their produxcts
  - Used to have very generous offers for students and universities (unclear status these days)
- Codeberg.org is specifically focussed on open source projects.

3.1.2 Create and Clone a Repository

- Easiest to start in the web interface
- Name the project to something creative, e.g. `gitExample`
- When you are done, there should be a link, e.g. under `<> Code` that can be used to clone the project.
  - Example: `git clone https://codeberg.org/mickesv/gitex.git`
  - This will set up `remote/origin` for you.

3.1.3 Create some git history

1. Create some files
2. Add them to the staging area and commit
3. Change one of the files; add and commit again
4. Repeat a couple of times.
5. Create a branch
6. Create some files, add and commit.
7. Edit some of your first files and commit.
8. Check the log.
9. Check status.
10. Push to the server
11. Check the status.

3.1.4 (Optional) Fork a colleagues repository

1. Find the account of a colleague (on the same server)
2. Pick a repository and fork it (for example the example account that you just created)
3. Clone it to your computer and create some more git history
4. When you have pushed everything to your fork, create a `pull request` in your colleagues repository (via the web interface)

3.1.5 (Optional) Handle a Pull Request

When your colleague have created a pull request to your repo, handle it.

- Inspect every commit to see what has been changed
- Can it be merged automatically? This should be indicated somewhere.
- Create a merge commit.

Create some more commits in your respective forks.

- Create a new pull request.
- This time, *deny* the pull request.

3.1.6 (Optional) More participants in the same project

- Divide into groups of around 5 people
- Pick a colleagues repository
- Enter `Settings/Collaborators` and add all of you to the same project
- Clone the repo

Now you are only allowed to work in a specific file `charlie-foxtrot.txt`

- You may add new text
- You may edit the existing text
- You may insert text; between two lines, and in the middle of a line
- You may remove text

Commit regularly (max 2-3 changes per commit) Push after every commit.

- You may need to do a `fetch/merge` in order to be allowed to do a `push`

Handle the merge conflicts
Discuss in small groups: What can you do to get fewer conflicts?

3.2 Show the TA ASSIGNMENT

Show the TA the following:
File management:

1. ☐ Create a directory
2. ☐ Create some files in the directory
3. ☐ Open files in your development enviromment; edit; save.
4. ☐ Close your development environment and find the files via a file explorer (not your development environment).

Working with git:

5. ☐ Git clone from an online repository to a new directory.
6. ☐ Edit some files in the git repo using your development environment.
7. ☐ Find the files in a file explorer (not your development environment).
8. ☐ Git add and git commit of your edited files.
9. ☐ Git push to your online repository.
10. ☐ Show the files in your online repo (using web browser).

Each completed item gives 1 point for a total of 10 points.

4 Assignment 1 Tools: Docker and Docker Compose

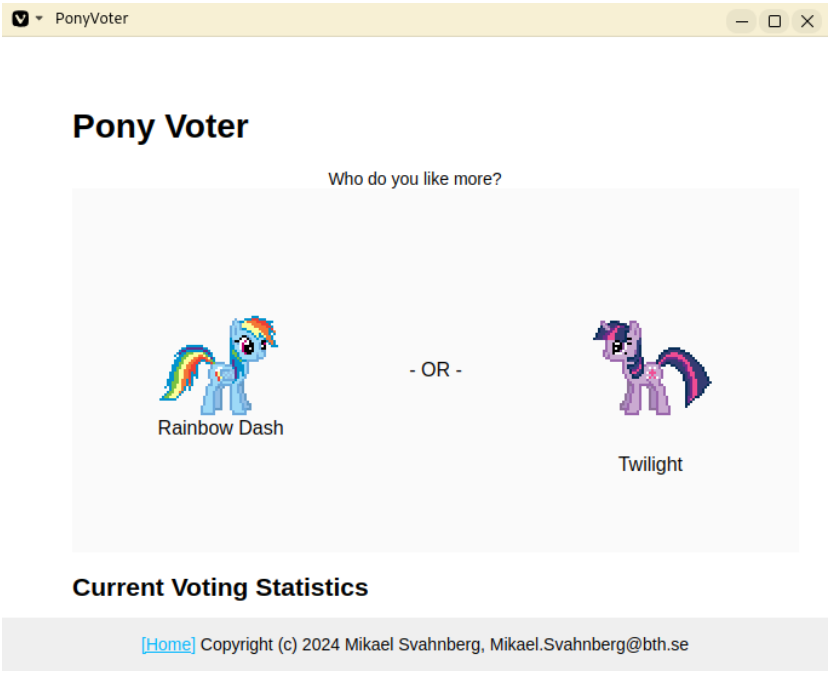
4.1 Tasks: The PonyVoter Microservice app

4.1.1 Get Started

1. Make sure Docker is installed
2. (Optional) The official tutorials are quite good:
   - Docker `https://docs.docker.com/get-started/`
   - Docker with node.js `https://docs.docker.com/language/nodejs/`
   - Docker with python `https://docs.docker.com/guides/python/`
3. Clone the *PonyVoter* project `https://codeberg.org/mickesv/PonyVoter.git`

4.1.2 Info PonyVoter INFO

- PonyVoter presents two options and you vote by clicking on one of them
- The votes are registered in a database so that you can keep track of which pony is the most popular.



4.1.3 Technical Overview

- PonyVoter consists of three containers and a database

PonyVoter the "front page" of the application that serves web pages to the users

VoteCounter registers votes and stores them to the database

StatsPresenter calculates how many votes each pony has, and summarises this

MongoDB the database where the votes are stored.

- PonyVoter is hopelessly over-designed and at the same time under-implemented:
  - `VoteCounter` and `StatsPresenter` are extremely simple and probably did not need to be separate containers.
  - Many things are hard coded.
  - To keep things simple, no rendering engine (e.g. `Pug`) is used to generate the HTML code.
  - To keep the project small, there are only six ponies to choose between.
  - Completeness? Only the bare necessities are implemented.
  - Scalability? What happens when the total number of votes increases?
  - Bugs! Of course there are bugs.
  - Security?

Think about:

1. What containers should be made available to the user?
2. How do you ensure that these become available?
3. How can you start all the containers with a single command?

4.1.4 Inspect and Test

- Study the file `ponyvoter.yaml`
  - Which *Services* are launched?
  - Where can you find the source code for each of these?
  - What is specified for each container?
  - Can you see how to access each container (e.g. which network ports to use)?
- Study the file `makefile`
  - How do you build the project?
  - How do you run the project?

Run the project:

1. Start the application (using `make` or `docker compose -f ponyvoter.yaml up`)
2. Visit `http://localhost:8080` and test the application

- Keep an eye on the terminal while running. What is printed?

3. Abort by pressing `Ctrl-C` in the terminal.
   - What happens?
   - Check with `docker images` what images you have
   - Check with `docker ps -a` what container are running or no longer running
4. Start again (same command)
   - What happens?
   - Note that the statistics are not reset despite all containers being restarted.
     - Why not?
     - How can you find out more information about this?

4.1.5 Modify

Your task is to build a new StatsPresenter, `newStats`

- `votecounter` has an API endpoint `/listVotes` that returns a json object with all votes currently cast.
  - Call this in order to get an updated list of all votes (instead of querying the database)
- Your new API will consist of:
  - `/listponies` returns an array with the names of all ponies (see example below)
  - `/stats/<ponyname>` returns a json object (see example below)
  - `/` is kept for historical reasons and will always return an empty array.
- Make sure that there is console output in each of these functions so that you can see what they do in the log output.

```
/listpoinies    returns  ["applejack", "pinkiepie", "rainbow"]
/stats/rainbow  returns  {_id: "rainbow", count: 2, name: "rainbow"}
/               returns  []
```

Tasks

1. Modify `ponyvoter.yaml`
   - add the service `newStats` with appropriate configuration
   - modify `services/ponyvoter/environment/STATSPRESENTER_HOST` to point towards `newStats` (If applicable, change the port number too)
2. Implement `newStats` (Run and Test as needed)
3. You will have to make one change in the `PonyVoter` container: replace calls to `createStatistics()` with `newStatistics()` .

4.2 Show the TA ASSIGNMENT

Show the TA the following:

1. ☐ A local clone of the PonyVoter repository
2. ☐ Start the system using `docker compose`
3. ☐ Test the system via a web browser.
4. ☐ Describe each Container
   - Which containers are there
   - When is each container "invoked" in the log output?
   - What does each container do?
   - What is the difference between image and container?
5. ☐ Show and explain your edited `ponyvoter.yaml`
6. ☐ Show and explain the code for your `newStats` container
7. ☐ Show that you are indeed using the `newStats` container (i.e. that `PonyVoter` is calling `newStatistics()`)

Each completed item gives 1 point for a total of 7 points.

5 Assignment 1 Tools: Testing and Debugging

5.1 Tasks: Testing and Debugging SorterTool

5.1.1 Get Started

1. Clone the project `https://codeberg.org/mickesv/SorterTool`
2. Read the source code and make sure you understand it.
3. Make sure that `pytest` is installed: `pip install pytest`
   - (You may need to set up a python venv before this)
4. Run the existing tests: `pytest -v`
   - Read the output and make sure you understand it.
   - One test fails. Why? How can you find out more information?

```
git clone https://codeberg.org/mickesv/SorterTool
cd SorterTool/Python

python -m venv .venv
source .venv/bin/activate

pip install pytest
```

5.1.2 Write More Tests

Only one algorithm is currently tested, i.e. QuickSort, and only with a hard-coded array of numbers.

1. **TODO:** Write tests for BubbleSort and InsertionSort
2. **TODO:** Generate a new array of random length (between 5–20) containing random numbers (between 1–1000) for each test.

Tip:

- You can write separate tests for each algorithm, or you can use the same tests and use `@pytest.mark.parametrize()`

5.1.3 Use the Debugger

1. Add a breakpoint on the first line in the `bubbleSort` method
2. Debug the program and try to understand how the bubblesort algorithm works (Tip: *Sorting out Sorting* : `https://youtu.be/HnQMDkUFzh4?si=xMEF8GgU_4fXxxB6`)
3. **Practice** using *breakpoints*, *step over*, *step in*, and *step out*
4. Add a `watch` expression and continue debugging until you see it change value

Fix the error:

- Can you now figure out why one of the initial tests failed?
- Can you fix the code so that it passes?
  - If so, please fix the error and re-run the tests.

5.2 Show the TA ASSIGNMENT

Show and explain to the TA the following:

1. ☐ Which tests have you added? Show them and explain what they do.
2. ☐ Run `pytest -v .` Explain the output. If any tests still fail, explain why.
3. ☐ Show while you run the debugger on the `quickSort` method
   - Show while you use *breakpoints*, *step over*, *step in*, and *step out*
   - Show the current values of local variables

Each completed item gives 1 point for a total of 3 points.

6 Assignment 2: Implementation and Documentation

This assignment focus more on good implementation practices such as well-structured source code, and documented source code. The task is to implement a small web app, i.e. **BackupOrganiser** (further described below). This assignment spans several weeks, and is examined by showing a teaching assistant your running application and your source code.

The TA uses a checklist to mark the parts that you have shown, and this is used to calculate a score for the entirety of the assignment. This score is then translated to a grade as follows:

Score	Percent	Grade
33	100%	
30	>90%	A
26	>80%	B
23	>70%	C
21	>65%	D
19	>60%	E
>19	<60%	F

You may collaborate with other student colleagues, but you present your final work and is examined individually.

6.1 Introducing the Backup Organiser App INFO

- **Backup Organiser** is an app to keep track of your data and when you last did a backup.
- For now, it is self-reported by manually updating the information for a data collection when a new backup is created.

A *data collection* consists of:

- name (text)
- description (text)
- creation date (text; managing actual dates require more logic!)
- last modified date (text)
- still-updated (boolean flag)
- a list of *backup entries*

A *backup entry* consists of:

- name (text)
- date (text)
- location (text)

Functionality

- Create:

  Collection Add a data collection

- Read:

  Overview Overview of all data collections – displaying name and the date of the latest backup entry

  List Detailed list of all data collections – displaying all information about all data collections

  Info Detailed information about a single data collection

  Search Find all data collections where the name contains the search term (display name and the date of the latest backup entry)

- Update:

  Backup Add a backup to a data collection

  Edit Change the last modified date or the still-updated flag of a data collection

- Delete:

  Delete Delete a specific data collection

  Unbackup remove a specific backup from a specific data collection

Notes:

- This is a typical *CRUD* (create-read-update-delete) - pattern
- The names of the functionalities are chosen to be as unique as possible, as early as possible. This will make it easier to create a command-line UI.
- You do not need to save any data between sessions, but it will help you during testing (and actually make the application usable).

6.2 Tasks Week 1

0. Create the project *BackupOrganiser* so that you are ready to start coding.
1. Create a class `DataCollection` that contains attributes as listed earlier
2. Create a class `CollectionManager`:
   - This class contains a list of `DataCollection` objects.
   - It contains *at least* the following methods:
     - `add_collection(name, description, creation_date, modification_date, updated)` creates a `DataCollection` object and adds it to its list.
     - `overview()` returns a lists of strings, one string per `DataColection` object
     - `detailed_overview()` returns an array containing lists of strings, one list per `DataColection` object

        * Tip: create a method `DataCollection.brief_str()` to generate the list of strings.
    - `info(collection_name)` returns a list of strings that represent a single `DataCollection` object
        * Tip: create a method `DataCollection.full_str()` to generate the list of strings.
    - `get(collection_name)` returns a `DataCollection` object
    - You may wait with: `search`, `edit`, and `delete`

3. Create a class `BackupManager` that *at least* contains:

    - `add_backup(collection_object, backup_name, backup_date, backup_location)` modifies the `DataCollection` object it was given and appends a new Backup entry
        - Tip: Call `CollectionManager.get()` to find the `collection_object`
    - You may wait with `unbackup`

4. Create a `main()` function that

    - Creates one `CollectionManager` and one `BackupManager` object.
    - Tests each of the methods above on those two objects.

5. You *may* create a class `ConsoleInterface` to help you with this.

Remember

- One class per file
- Document your methods. Use the docstring.
- Create new methods if/when you need to.
- Take a step back and look at the code you have written – can you improve it?
- Keep all the `print()` functionality in one place and *not* in the classes `DataCollection`, `CollectionManager`, or `BackupManager`.

6.3 Tasks Week 2

1. Create a file `restinterface.py` where you implement a http interface to your app.

    - `POST /api/Collection` add a new collection
    - `GET /api/Overview` get a JSON object with an overview of all data collections
    - `GET /api/List` get a JSON object with a detailed list of all data collections
    - `GET /api/Info?name=data-collection-name` get a JSON object with details for one specific data collection
    - `POST /api/Backup` add a backup

- You may wait with: `GET /api/Search?name=text`
- You may wait with: `POST /api/Edit`
- You may wait with: `POST /api/Unbackup`
- You may wait with: `DELETE /api/Delete?name=data-collection-name`

Note

- This is why we separated all `print()` statements into a single place; it is now "easy" to create a new frontend.
- You will need some http interface (e.g. `flask` in Python) for this – see lab from previous weeks.
- You will need to modify your `main()` function to start the `RestInterface` for you.

Tips:

- create a method `DataCollection.full_json()` to generate a JSON object for a single DataCollection object.
- similarly, you want to add methods `overview_json()`, `detailed_overview_json()`, and `info_json()` in your `CollectionManager` class.
- You may wish to run your app inside Docker already now.
- The `POST` endpoints should accept a JSON object; see next slide for examples.

6.3.1 Example JSON objects

```
# Add a Collection
POST http://localhost:5000/api/Collection
content-type: application/json

{
  "name": "web collection",
  "description": "created over the interwebz",
  "creation_date": "just now",
  "modification_date": "only recently",
  "still_updated": true
}

# Add a Backup
POST http://localhost:5000/api/Backup
content-type: application/json

{
  "name": "collection 1",
  "backupname": "DVD Burn",
  "date": "May 2025",
  "location": "Top drawer"
```

```
}

# Unbackup
POST http://localhost:5000/api/Unbackup
content-type: application/json

{
  "name": "collection 1",
  "backupname": "DVD Burn",
  "date": "May 2025"
}

# Edit
POST http://localhost:5000/api/Edit
content-type: application/json

{
  "name": "collection 1",
  "modification_date": "yesterday",
  "still_updated": true
}
```

6.4 Tasks Week 3

1. Complete the remaining functionalities
2. Make a web interface to use the `RESTInterface` entrypoints you made.
3. Add a `Dockerfile` and a `compose.yaml` so you can start your application using Docker.

Note

- You can get away with a single html page that converts `<form>` data into JSON requests to your already implemented `restinterface`
- You may instead create separate pages for each of the functionalities and add new routes to manage these.
  - That's why we included `/api/` in each URL: to separate them from "normal" web endpoints.

6.5 Show the TA ASSIGNMENT

When the full app is done, you are expected to show a TA when you:

1. Start the application as a docker container `docker compose up`
2. Demo the application in a web browser
   - show each of the functionalities listed earlier

3. Present your implementation and be ready to answer questions from the TA.

The TA will look at:

- The exent to which your app is working and supports all the listed functionalities
- The structure of your source code
- How well documented your source code is
- Your understanding of the source code

They will *not* look at

- How pretty your web page is.

The assessment is done as follows:

1. Functionality, 1 point each for:
   (a) ☐ Add Collection
   (b) ☐ Overview of Collections
   (c) ☐ Detailed list of Collections
   (d) ☐ Detailed Info for one Collection
   (e) ☐ Search Collections by Name
   (f) ☐ Add Backup to Collection
   (g) ☐ Edit Collection
   (h) ☐ Delete Collection
   (i) ☐ Remove a Backup from a Collection
2. Execution, 1 point each for:
   (a) ☐ The application runs inside Docker containers
   (b) ☐ The application is accessible from a web browser
3. Source Code, 1 point each for:
   (a) ☐ Methods have names that accurately describe what they do
   (b) ☐ Methods are short and to the point
   (c) ☐ Methods have docstrings
   (d) ☐ Methods are implemented in the right file/class (where they logically belong)
   (e) ☐ Comments are used to explain
   (f) ☐ The application is built using classes and objects
   (g) ☐ Each class is implemented in a separate file
4. Understanding, a maximum of **5 points** each for:
   (a) ☐ At a high level explaining what each file/class does

(b) ☐ Explaining how some functionality (TA selects) is implemented and showing the relevant source code.

(c) ☐ Explaining how some other functionality (TA selects) is implemented and showing the relevant source code.

This gives a total of 33 points.

7 Assignment 3: Engineering Diary

In this assignment, the focus is on your learning and reflections. As you work with new tools, or implement new code, an engineering diary is a vital help to be able to re-trace previous decisions, to note down quick ideas, or items that require further reading. In real life this engineering diary is your personal tool and life-saver, but in this course we also use it as part of the examination in order to bootstrap your use of a diary as a vital engineering tool.

Your engineering diary is individual, and is **submitted to Canvas** once you are done with assignments 1 and 2.

A checklist is used to assess the contents of your diary, and this is used to calculate a score for the entirety of the assignment. This score is then translated to a grade as follows:

Score	Percent	Grade
24	100%	
22	>90%	A
19	>80%	B
17	>70%	C
16	>65%	D
14	>60%	E
>14	<60%	F

7.1 Generic Suggestions

- Try to get into the habit of often reflecting upon what you have done, what you have learnt, and what you must find out.
- It is better to write a little every day rather than a lot once a week.
- Even an entry such as "2025-06-22: Finished implementing the Frobnicator class; all went well and now I want to go out in the sun and play!" is valuable.

7.2 Required Sections

Your engineering diary *must* contain reflections on:

1. Configuration Management
   - Why should we use configuration management?
   - Which commands are you using?
   - What works well?

- What is difficult?
- How did you solve your challenges? What could you have done differently?
- What did you not manage to solve? Why not?

2. Container Development
   - Which commands are you using?
   - What works well?
   - What is difficult?
   - How did you solve your challenges? What could you have done differently?
   - What did you not manage to solve? Why not?
   - How is a Dockerfile structured?
   - How is a docker compose file structured?
3. Testing and Debugging
   - Which commands are you using?
   - What works well?
   - What is difficult?
   - How did you solve your challenges? What could you have done differently?
   - What did you not manage to solve? Why not?
4. Implementation and Documentation
   - What are you working on?
   - In what order are you implementing things?
   - How do you keep your code "clean" and well structured?
   - What works well?
   - What is difficult?
   - How did you solve your challenges? What could you have done differently?
   - What did you not manage to solve? Why not?

Please **timestamp** each entry in your diary.

7.3 Assessment

The following criteria will be used to assess your engineering diary:

1. ☐ There is a discussion about configuration management
2. ☐ There is a discussion about container development
3. ☐ There is a discussion about testing and debugging

4. ☐ There is a discussion about the Implementation and documentation assignment
5. ☐ It is clear when each entry was made.
6. ☐ There are regular updates to the engineering diary (more than five, spanning the entire course)
7. ☐ The engineering diary contain summaries of the work that has been done since the last update.
8. ☐ The engineering diary contain insightful reflections and identifies opportunities for improvements.

Each completed item gives a maximum of 3 points for a total of 24 points.