

# Roadmap: TruCol

A decentralised collaboration protocol for test-driven development

September 8, 2022

## 1 Introduction

Welcome, this document presents our market analysis for the TruCol company. The objective of this document is to provide some basic insight into the order of magnitude of the potential of the TruCol company to generate returns for its potential investors. Based on various pitch templates, [7], and private communications, we intend to convey this information through sharing our model and estimate of the following market parameters for the TruCol company:

- **Total addressable market (TAM)**, or total available market, is the total market demand for a product or service, calculated in annual revenue or unit sales if 100% of the available market is achieved[3].
- **Serviceable available market (SAM)** is the portion of TAM targeted and served by a company's products or services[3].
- **Serviceable obtainable market (SOM)**, or share of market, is the percentage of SAM which is realistically reached[3].

Since we currently have little experience on this topic within our team, we are making our data and assumptions as transparent as possible, both in this document as in our code. This way, we hope to improve our model based on your feedback by enabling you to experiment with it yourself. Additionally, because the market analysis consists of a rough estimate, three different estimation methods are considered for generating the TAM, SAM and SOM estimates. The redundancy is introduced in an attempt to establish some frame of reference within the results. The assumptions and data points for the respective models are specified explicitly in this document with an identifier, and where relevant, this identifier is used in the comment of the code to link the document and code.

The mathematical models are described in section 3. (the Python models themselves are included as appendices.

The results of these models are presented in section 6. To shed some light on how sensitive the model is to for example changes in assumptions, a sensitivity analysis is presented for each model in section 7. Next, the results and sensitivity of the models are discussed in section 8 and a conclusion is provided in section 9.

We invite you to tinker with the assumptions and models yourself! The data and plots in this report are automatically updated if you run `python -m code.project1.src`. If you experience any difficulties in running the code, simply reach out to us, (click on issues on the GitHub page) and we are happy to get you running the code.

## 2 TruCol Company Business Model

Since the market size estimation modelling is somewhat of an abstract/subjective task, a Monte-Carlo simulation is created of a Top-Down model.

### 2.1 Revenue Format

Before the model is presented, it is important to realise that we propose an optimisation service. This means that if a certain activity, e.g. a logistics company has operational cost of 5 \$million/day, our service is only able to earn at most the margin of improvement we are able to bring our customer. Suppose the independent usage of the TruCol provides the customer with a 2% optimisation in their operational costs, yielding them  $5.000.000 \cdot 0.02 = \$100.000/day$ . Suppose our expertise is able to enable them to yield a 3% optimisation by identifying the relevant development/system processes and supporting them in improved test specification. In that assumption our company would bring them an additional  $3-2=1\%$  which would translate roughly to \$50.000. That would be the value we bring to the logistics company in this hypothetical scenario.

In reality this example is oversimplified, the 2% the company could get by themselves would involve some risk pertaining to inaccurate test specification which could lead to loss of the bounty. Our company reduces this risk by providing test-specification security expertise. Furthermore, our interaction with the client may bring the client

experience that can be applied in future applications of the TruCol protocol, hence the value to we bring to the client is larger than the amount they gain in terms of optimisation w.r.t. the case where they use the protocol themselves.

### 3 Market Analysis Models

This section describes the model that is used to perform the market analysis for the TruCol company service. The model will be used to estimate the yearly revenue that is projected for this company. Typical estimation models to do this for startups are:

- *Top Down Model* - Starts with a large population with known size that make up the target market, and then narrows the market size down to the specific market segment.
- *Bottom Up Model* - Takes current pricing and/or usage of product as a starting point and extrapolates up/outwards to compute the potential market size.
- *Value Theory Model* - estimates the value provided to customers and estimates how much of that value can be reflected in the product pricing.

Since the TruCol company does not yet have a large body of current pricing and product usage, the Bottom Up Model is not used in this market analysis document. Similarly, the Value Theory Model is omitted in this market analysis as it is most powerful on historical data which is not yet available for the TruCol protocol. Since the market sizes of most sectors in which the TruCol company intends to operate, the Top Down Model is used to derive a rough estimate of the projected yearly revenue for the TruCol company.

### 4 Market Sizing

The market sizing parameters are listed in table 1. The description is given in section 4.1. Ps. **bold** numbers are model parameters (and possibly assumed), blue numbers follow from the model parameters.

Table 1: Cost Model Parameters in €(/hr or absolute, unless specified otherwise)

Parameter	Value
nr simulations	300.0
profit dhl billion	4.099999
profit dhl	4.100000e+09
profit fedex billion	1.29
profit fedex	1.290000e+09
profit ups billion	7.7
profit ups	7.700000e+09
profit dhl fedex ups	1.309000e+10
sam factor	0.003
tam factor	0.004
logistics market share dhl fedex ups	0.15
logistics market share dhl fedex ups percentage	15.0
avg algo optimisation profit percentage	0.1
avg algo optimisation profit	0.001
TruCol cut on profit percentage	1.0
TruCol cut on profit	0.01

#### 4.1 Introduction

To use the Top Down Model, section 4.2 describes the TAM in which the TruCol company will operate. To this end, section 4.2.1 discusses the market size of the logistics market, and the profit within that market. Due to time-constraints and lack of data, some datapoints and assumptions of the logistics market are applied to other sectors such as the automated trading market and pharmaceuticals market in section 6.3 to get some insight in their respective market sizes. Furthermore, section 6.3.1 provides some qualitative insight in the potential future markets that are highly suited for the TruCol protocol. These emerging markets are accordingly expected to be relevant markets to address in the near future.

## 4.2 Total Addressable Market

To compute the TAM, SAM and SOM, some form of market definition can be used. To this end, it is considered valuable to specify what the TruCol company does, where it adds value and how it does that. Furthermore, since these three aforementioned estimates pertain to a potential future, the potential, yet deemed feasible, activities of the TruCol company are included.

The TruCol company provides advice and support to companies on how they can get the most out of the TruCol protocol. To understand this, the following assumptions are shared. Under these assumptions, one can conclude that an economically rational company would try to off-load as much of their required tasks into the TruCol protocol. This is because it would minimise their operational costs and/or improve algorithmic efficiency of their solutions.

- **asu-0:** Solutions to tasks that are completed using the TruCol protocol are deterministically verifiable.
- **asu-1:** Solutions to tasks that are completed using the TruCol protocol are of sufficient quality.
- **asu-2:** Tasks that are completed using the TruCol protocol can be solved for the lowest cost price that is currently available in this world.
- **asu-3:** No personnel needs to be attracted, screened, hired nor fired for tasks that are completed using the TruCol protocol.
- **asu-4:** Companies can benefit from public particular solutions to their task specifications.
- **asu-5:** By sampling from a bigger talent pool (this world), the average performance of the solutions will be better than what is produced by the in-house talent pool, or, for equal solution performance, a faster rate of development can be obtained on average for an equal or lower price.

We help companies identify the tasks for which they can use the TruCol protocol, and we assist them in writing safe test specifications that are not easily hackable. This implies that under the given set of assumptions, the TAM for the TruCol protocol can be defined as the total costs that the companies (and consumers) in this world are willing to pay for assistance on using the TruCol protocol.

### 4.2.1 TruCol Total Addressable Logistics Market

This sub-sub section illustrates a rough method of estimating the logistics sub-segment of the TAM for the TruCol protocol. To do this, an example of algorithmic optimisation within the logistics market as presented by McKinsey & Company is generalised conservatively to a rough estimate of the total logistics market size.

A clear example of a logistics company successfully hiring a support for algorithmic optimisation is documented by McKinsey & Company in the "how they help their clients" segment of their website[4]. The study how reports McKinsey's team, among which McKinsey's Strategic Network Analytic Center, helped an Asian logistics company. With McKinsey's team, the logistics company realised an *in line haul network cost* reduction of 3.6% while reducing their *transit time* with 0.8%, yielding an overall 16% increase in profit for the logistics company, without compromising the quality. To use this report as a valuable resource to generate some rough estimates on market size, the following assumptions are made:

- **asu-6:** The logistics company made a net profit by hiring McKinsey & Company in this particular ordeal.
- **asu-7:** The example of a 16% increase in profit is generalizable to a conservative potential **0.1%** of profit increases through algorithmic optimisation across the entire logistics industry.
- **asu-8:** Companies are willing to pay at least **1.0 %** of their potential profit increases for the assistance the TruCol company provides in identifying opportunities for optimisation and for improving test-specification security.

Based on those assumptions, one could find a potential yearly profit increase across the entire logistics sector by summing the net profit of the logistics sector. [2] claims that this company [8] valued the logistics market at 8.1 trillion in 2016. Additionally, [2] claims [8] estimates the logistics market value will grow to 15.5 trillion in 2023. However, no figures on profit are found. Therefore, individual companies are explored.

For DHL one can find on pdf page 37/170 in [5] that the annual profit for DHL in 2019 was **4.099999** billion.

For UPS one can find on pdf page 4/257 in [9] that the annual unadjusted operating profit for UPS in 2020 was **7.7** billion. Note, [10] says UPS had a net operating profit of 1.1 billion in Q1 of 2020, implying they had to almost double their average profit in the remaining three quarters of 2020 to be consistent with an annual **7.7** billion.

For FedEx the net income as reported for 2020 has been **\$1.29** billion in pdf page 2/17 [6].

- **Asu-9:** The net income as reported (GAAP) by FedEx can be interpreted as the profit by FedEx.

Next, the claim that fragmentation of the global market implied in 2016 that Deutsche Post DHL, Ceva Logistics, UPS, and FedEx, control less than **15.0%** of that global market, allows estimating a limit on the net global profit made in the logistics market based on the following assumptions:

- **Asu-10:** The market segment in the global logistics market maintained by the combination of DHL, UPS and FedEx is at most **15.0%** in 2020.
- **Asu-11:** The profit in the remaining **85%** of the global logistics market has the same average yearly profitability per percent market share as the combination of DHL, UPS and FedEx.

## 5 Results

### 5.1 TAM

Based on assumptions 1-11 one could estimate an upperbound of

$$\begin{aligned}
 net - profit_{DHL+UPS+FedEx} &= 4.099999 + 7.7 + 1.29 = 13.09 \text{billion} \\
 \frac{net - profit_{global\_logistics}}{net - profit_{DHL+UPS+FedEx}} &= \frac{0.85}{0.15} \\
 net - profit_{global\_logistics} &= net - profit_{DHL+UPS+FedEx} \cdot \frac{0.85}{0.15} \\
 net - profit_{global\_logistics} &= \frac{13.09 \cdot 0.85}{0.15} \\
 net - profit_{global\_logistics} &= 74.2 \text{billion}
 \end{aligned} \tag{1}$$

**TODO:** verify in model and text this is TAM or TSAM. TSAM if factor of what is practically addressable is taken into account, e.g. 0.0001% of market in first year, that is 1 in every 10000 logistics companies. There are **X** logistics companies, so that requires us to reach out to **Y** companies and work with **Z** companies. A follow rate of **R** is required on a reachout of **Y**.

Note, if TSAM is too small, realise the factor 160 might be too small (16% demonstratable profit gain through algorithmic optimisation, 0.1% assumed). See Monte-Carlo simulation for a larger spread on assumptions.

## 6 Revenue: Monte-Carlo Simulation

### 6.1 Top Down Model

The code listed in the appendices generated the following estimates for the total addressable market sizes for the TruCol company.

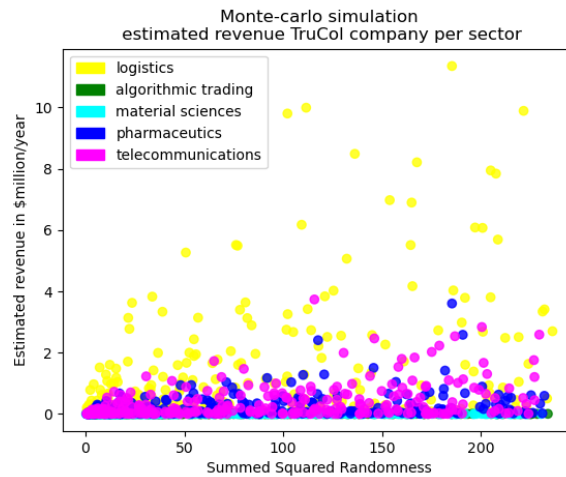


Figure 1: A scatterplot generated by a Monte-Carlo simulation to provide an impression on the estimated projected total addressable market per sector for the TruCol company.

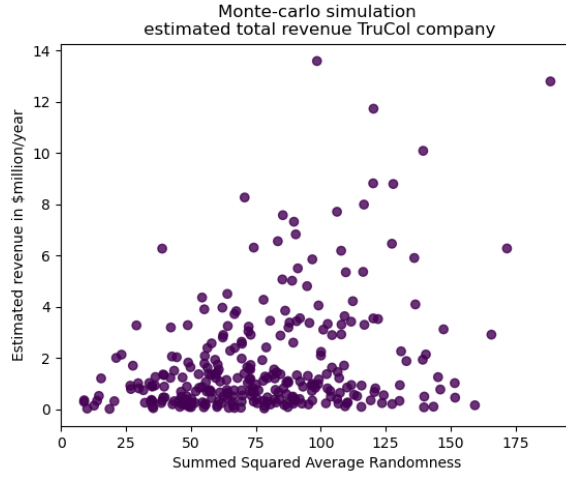


Figure 2: A scatterplot generated by a Monte-Carlo simulation to provide an estimate on the projected total addressable market for the TruCol company.

## 6.2 TruCol Revenue

Hence, if each of those companies in the logistics sector could increase their profits on average annually by **0.1%** using algorithmic optimisation, and if they would use the TruCol protocol to do that, and if they would be willing to invest **1.0%** of that profit in our support and assistance in getting the most out of the TruCol protocol, we would currently estimate that this would yield roughly an income of  $74.2 \cdot 0.001 \cdot 0.01 = \$0.74\text{million}$

## 6.3 Additional addressable markets

Since the TruCol company is market agnostic, we also seek to assist in algorithmic optimisation outside the logistics market. Several markets are worth mentioning in particular as we expect them to either heavily rely on algorithmic optimisations, or because they are particularly suited for the TruCol protocol.

- **(Automated) trading** In the highly competitive market of (automated) trading, algorithmic optimisations are key to making successful trades.
- **Space Sector** The space engineering sector already has a relatively high test driven development[?], this lowers the adoption costs of the TruCol protocol relative to most industries. Furthermore, space applications are heavily mass constrained, which generally makes them highly energy constrained as well. These energy constraints emphasise the importance of algorithmic optimisations, for example in telecommunications satellites and swarm robots.
- **Innovative Materials Research** The domain of material science has been adopting algorithmic search strategies to find new materials [1].
- **Pharmaceutical Industry** Another example of a large market that has been shifting to adopt algorithmic search strategies to find new medicines.

Each of these are multi-billion dollar markets which can contribute to the TAM of the TruCol company.

### 6.3.1 Emerging markets

Beyond those listed markets, the following emerging markets could be great opportunities for the TruCol company to latch in and grow along in.

- **Neuromorphic Computing** This field is developing new complexity theory to adapt to the unconventional computation methods. This is an interesting opportunity to explore the versatility of the TruCol protocol.
- **Quantum Computing** This is another upcoming field with many new algorithmic implementations. The newness of the field may suggest that the amount of optimisation and exploration to be done is relatively high, possibly indicating a relatively large potential for the TruCol protocol. However, currently our team does not yet contain experience in this type of algorithmic developments.
- **Artificial Intelligence** With the introduction of GPT-3 and GitHub Copilot, the world has seen examples of AI engines that are able to generate code for some basic tasks. The TruCol protocol could catalyse the usage of such AI engines that are able to write code based on requirement specification. We expect that users of the TruCol protocol will develop a tactical advantage on requirement specifications for AI engines.

## 7 Sensitivity Analysis

### 7.1 Top Down

Omitted due to time constraints. Can be generated upon request.

## 8 Discussion

We expect that the main points for improvement in this analysis are the datapoints that are used. In particular, the generalisation of the profit margin of the logistics sector to other sectors could be replaced by the actual data of the other sectors. Furthermore, the profit margin of the logistics sector could be searched directly instead of deriving it based on the of its larger companies.

## 9 Conclusion

A rough estimate based on various datasources has been generated to estimate the yearly revenue of the TruCol company. Additional iterations with improved datapoints is recommended to obtain a more accurate estimate. The market analysis does not yet include the growth that may be captured in diversification, emerging markets such as neuromorphic computing and in-house automation/AI-engines. Before taking these potentials into account, however, a more accurate assessment of the starting market is recommended.

## References

- [1] Zahed Allahyari and Artem R Oganov. Coevolutionary search for optimal materials in the space of all possible compounds. *npj Computational Materials*, 6(1):1–10, 2020.
- [2] Andrew Allen. Logistics industry to be worth \$15.5tn by 2023. <https://www.cips.org/supply-management/news/2016/november/logistics-industry-forecast-to-be-worth-155tn-by-2023/>. Accessed: 2021-07-11.
- [3] The businessplan shop. Tam sam som - what it means and why it matters. [https://www.thebusinessplanshop.com/blog/en/entry/tam\\_sam\\_som](https://www.thebusinessplanshop.com/blog/en/entry/tam_sam_som). Accessed: 2021-07-11.
- [4] McKinsey & Company. Algorithmic route optimization improves revenue for a logistics company. <https://www.mckinsey.com/business-functions/mckinsey-analytics/how-we-help-clients/algorithmic-route-optimization-improves-revenue-for-a-logistics-company#>. Accessed: 2021-07-11.
- [5] DHL. 2019 financial year - focussing on our profitable core. <https://www.dpdhl.com/content/dam/dpdhl/en/media-center/investors/documents/annual-reports/DPDHL-2019-Annual-Report.pdf>. Accessed: 2021-07-11.
- [6] FedEx. Fedex corp. reports fourth quarter and full-year earnings. [https://s21.q4cdn.com/665674268/files/doc\\_news/investor/2020/FedEx-Q4-FY20-Earnings-Release.pdf](https://s21.q4cdn.com/665674268/files/doc_news/investor/2020/FedEx-Q4-FY20-Earnings-Release.pdf). Accessed: 2021-07-11.
- [7] Haje Jan Kamps. *Pitch Deck Design*. Apress, Berkeley, CA, 2020.
- [8] Transparency Market Research. Logistics market - snapshot. <https://www.transparencymarketresearch.com/logistics-market.html>. Accessed: 2021-07-11.
- [9] UPS. Notice of 2021 annual meeting of shareowners and proxy statement & 2020 annual report on form 10-k. [https://investors.ups.com/\\_assets/\\_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1110/9465/annual\\_report/UPS\\_2021\\_Proxy\\_Statement\\_and\\_2020\\_Annual\\_Report%3B\\_Form\\_10-K.pdf](https://investors.ups.com/_assets/_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1110/9465/annual_report/UPS_2021_Proxy_Statement_and_2020_Annual_Report%3B_Form_10-K.pdf). Accessed: 2021-07-11.
- [10] UPS. Ups q21 earnings call. [https://investors.ups.com/\\_assets/\\_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1111/9835/file/UPS\\_1Q21\\_Earnings\\_Webcast\\_Deck\\_Final.pdf](https://investors.ups.com/_assets/_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1111/9835/file/UPS_1Q21_Earnings_Webcast_Deck_Final.pdf). Accessed: 2021-07-11.

## A Appendix src/Activity.py

```
1  """Object that is used to store the activities that compose a Gantt
   ↪ chart."""
2  import copy
3
4
5  # pylint: disable=R0902
6  class Activity:
7      """Used to create an activity in a Gantt chart."""
8
9      # pylint: disable=R0913
10     def __init__(
11         self,
12         description,
13         duration,
14         new_tag,
15         colour=None,
16         parent=None,
17         starts_at_child_nr_start=None,
18         starts_at_child_nr_end=None,
19         font_size=None,
20         hourly_wage=None,
21         hours_per_day=None,
22         min_parallel_workers=1,
23         max_parallel_workers=1,
24         milestone=None,
25     ):
26         self.parent = parent
27         self.description = description
28         self.duration = duration
29         self.children = []
30         self.font_size = font_size
31         self.starts_at_child_nr_start = starts_at_child_nr_start
32         self.starts_at_child_nr_end = starts_at_child_nr_end
33         self.hourly_wage = hourly_wage
34         self.min_parallel_workers = min_parallel_workers
35         self.max_parallel_workers = max_parallel_workers
36         self.milestone = milestone
37
38         if hours_per_day is None:
39             self.hours_per_day = 8
40         else:
41             self.hours_per_day = hours_per_day
42         if self.parent is None:
43             if colour is None:
44                 raise Exception("Parent activity needs a colour")
45             self.colour = colour
46             self.tag = []
47         else:
48             if (self.parent.hourly_wage is not None) and (hourly_wage
49                 ↪ is None):
50                 self.hourly_wage = self.parent.hourly_wage
51             self.colour = parent.colour
52             self.tag = copy.deepcopy(self.parent.tag)
53
54         # create tag
55         self.tag.append(new_tag)
56
57     def add_children(self, children):
58         """Stores the child activities in an activity.
```

```
59         :param children:
60         """
61         self.children = children
62
63     def get_tag(self):
64         """Converts the tags of an activity into a string, with the
65             ↪ activity
66             indexes separated by underscores."""
67         return "_".join([str(tag) for tag in self.tag])
68
69     def addTwo(self, x):
70         """adds two to the incoming integer and returns the result of
71             ↪ the
72             computation.
73
74         :param x:
75         """
76         return x + 2
```

---



## B Appendix src/Cost\_model.py

```
1  """List of cost model parameters."""
2
3  from src.export_data.export_params_to_latex import
4      ↪ dict_to_latex_table
5
6  params = {
7      "wages": {
8          "blockchain_dev": 75 + 1,
9          "front_end_dev": 40 + 1,
10         "human_resources": 35 + 1,
11     },
12     # Bounties used to attract initial protocol users.
13     "bounty_subsidising": 100000,
14     "buffer": 100000, # Buffer to take unknown costs into account.
15     "daily_operational_costs": 200,
16     "days_to_operational_break-even": 270,
17 }
18
19 # pylint: disable=R0902
20 class Cost_model:
21     """Computes the total costs."""
22
23     # pylint: disable=R0913
24     def __init__(
25         self,
26         parent_costs: dict,
27         the_params: dict,
28     ):
29
30         self.parent_costs = parent_costs
31         self.operational_costs: int = (
32             the_params["daily_operational_costs"]
33             * the_params["days_to_operational_break-even"]
34         )
35         (
36             self.total_costs,
37             the_params["labour_costs"],
38         ) = self.total_costs_to_dict(the_params)
39
40         the_params["total_cost"] = self.compute_total_cost()
41         the_params["non_labour_costs"] = (
42             the_params["total_cost"] - the_params["labour_costs"]
43         )
44
45         # Writes the costs model parameters to latex table.
46         dict_to_latex_table(
47             "latex/Tables/total_costs_table.tex",
48             self.total_costs,
49             "Description",
50             r"Cost [\euro]",
51             "Total Expected Investment Costs",
52             "total_costs_params",
53         )
54
55     def total_costs_to_dict(self, the_params):
56         """Generates a dictionary including all expected project
57             ↪ costs."""
58         total_costs = {}
59         labour_costs = 0
```

```

59 # Get the order of the parent activities.
60 order = []
61 for key in self.parent_costs.keys():
62     order.append(key)
63 # The stored total costs are cumulative costs, so subtract
64     ↳ the costs of
65 # the previous parent to get the actual costs of the parent.
66 for i, key in enumerate(order):
67     if i == 0:
68         total_costs[key.description] = self.parent_costs[key]
69     else:
70         total_costs[key.description] = (
71             self.parent_costs[key] - self.parent_costs[order[
72                 ↳ i - 1]]
73         )
74     labour_costs = labour_costs + total_costs[key.description]
75     ↳ ]
76
77 total_costs["Bounty Subsidising"] = the_params["
78     ↳ bounty_subsidising"]
79 total_costs["Buffer"] = the_params["buffer"]
80 total_costs["Operational Costs"] = self.operational_costs
81 return total_costs, labour_costs
82
83 def compute_total_cost(self):
84     """Generates the expected summed total costs."""
85     total_cost = 0
86     for value in self.total_costs.values():
87         if isinstance(value, tuple):
88             total_cost = total_cost + int(value[0])
89         else:
90             total_cost = total_cost + int(value)
91     return total_cost

```

---

## C Appendix src/Create\_python\_gantt.py

```
1  """This file is used to create a Gantt chart using Python code. It is
   ↳ like a
2  PlantUML API. The reason to use Python instead of directly specifying
   ↳ a Gantt
3  in a .uml file, is because it is easier to update/modify the Gantt
   ↳ chart.
4
5  For example, if you want to add an activity somewhere, in the .uml
   ↳ you
6  have to manually update all the tag numbers to preserve the order. (
   ↳ And
7  do that again for the colour specifications etc.) That is tedious
   ↳ work.
8  In this Python API, you can simply say: start the activity new
   ↳ activity
9  after activity X, and make activity Y start at the end of the new
10 activity. Then it automatically updates all the tags, for all
11 properties, e.g. activity descriptions, colours, order etc.
12 """
13 from src.Activity import Activity
14 from src.export_data.Milestone import Milestone
15
16
17 # pylint: disable=R0914
18 def create_python_gantt(
19     wages: dict, milestone_font_size=None, start_date=None
20 ):
21     """Specifies the data for a Gantt chart."""
22     # Create a list to store the parent activities
23     parents = []
24     date_milestones = [
25         Milestone(
26             description="Complete CI deployment",
27             font_size=milestone_font_size,
28             date=start_date,
29         )
30     ]
31     # parent one
32     protocol = Activity(
33         description="Develop protocol",
34         duration=120,
35         new_tag=0,
36         colour="Green",
37         hourly_wage=0,
38     )
39     # children
40     onchain = Activity(
41         description="On-chain: Solidty+VRF",
42         duration=60,
43         new_tag=0,
44         parent=protocol,
45         hourly_wage=wages["blockchain_dev"],
46     )
47     git_tellor = Activity(
48         description="Git integration: Tellor",
49         duration=90,
50         new_tag=1,
51         parent=protocol,
52         starts_at_child_nr_start=0,
53         hourly_wage=wages["blockchain_dev"],
```

```

54 )
55 git_chainlink = Activity(
56     description="Git integration: Chainlink",
57     duration=90,
58     new_tag=2,
59     parent=protocol,
60     starts_at_child_nr_start=0,
61     hourly_wage=wages["blockchain_dev"],
62     milestone=Milestone(
63         description="Support all languages",
64         font_size=milestone_font_size,
65     ),
66 )
67 alt_chains = Activity(
68     description="Alternative Chains",
69     duration=90,
70     new_tag=3,
71     parent=protocol,
72     starts_at_child_nr_start=0,
73     hourly_wage=wages["blockchain_dev"],
74 )
75
76 # grandchildren
77 ci = Activity(
78     description="(Decentralised) Continuous integration",
79     new_tag=0,
80     duration=30,
81     parent=git_chainlink,
82 )
83 # git_chainlink.add_children([ci])
84 security = Activity(
85     description="Security & Robustness",
86     duration=60,
87     new_tag=1,
88     parent=git_chainlink,
89 )
90 # ci.add_children([security])
91
92 # merge
93 protocol.add_children(
94     [onchain, git_tellor, git_chainlink, alt_chains, ci, security
95     ↪ ]
96 )
97 parents.append(protocol)
98
99 # parent_two
100 platform_eco = Activity(
101     description="Platform & ecosystem",
102     duration=120,
103     new_tag=1,
104     colour="DarkOrchid",
105     starts_at_child_nr_start=0,
106     hourly_wage=wages["front_end_dev"],
107 )
108 # children
109 website = Activity(
110     description="Website", duration=50, new_tag=0, parent=
111     ↪ platform_eco
112 )
113 marketing_platf = Activity(
114     description="Marketing platform",
115     duration=30,

```

```

114         new_tag=1,
115         parent=platform_eco,
116     )
117     bounties = Activity(
118         description="Subsidize bounties",
119         duration=10,
120         new_tag=2,
121         parent=platform_eco,
122     )
123     platform_buffer = Activity(
124         description="Platform Planning Buffer",
125         duration=30,
126         new_tag=3,
127         parent=platform_eco,
128         milestone=Milestone(
129             description="First Customer Usage",
130             font_size=milestone_font_size,
131             # TODO: remove date, make it place on the right position.
132             date="2023-03-17",
133         ),
134     )
135
136     # Grandchildren
137     api = Activity(description="API", duration=50, new_tag=0, parent=
138         ↪ website)
139     gui = Activity(
140         description="GUI",
141         duration=50,
142         new_tag=1,
143         parent=website,
144         starts_at_child_nr_start=0,
145     )
146     forum = Activity(
147         description="Forum",
148         duration=10,
149         new_tag=2,
150         parent=website,
151         starts_at_child_nr_start=0,
152     )
153     website.add_children([api, gui, forum])
154
155     platform_eco.add_children(
156         [website, marketing_platf, bounties, platform_buffer]
157     )
158     parents.append(platform_eco)
159
160     # parent_three
161     company = Activity(
162         description="Launch company",
163         duration=150,
164         new_tag=2,
165         colour="Yellow",
166         starts_at_child_nr_start=0,
167         hourly_wage=wages["human_resources"],
168     )
169     # children
170     partners = Activity(
171         description="Qualitative partner research",
172         duration=20,
173         new_tag=0,
174         parent=company,
175     )

```

```

175 organisation = Activity(
176     description="Establish organisation",
177     duration=80,
178     new_tag=1,
179     parent=company,
180 )
181 marketing_company = Activity(
182     description="Marketing", duration=30, new_tag=2, parent=
        ↳ company
183 )
184 company_buffer = Activity(
185     description="Organisation Planning Buffer ",
186     duration=20,
187     new_tag=3,
188     parent=company,
189     milestone=Milestone(
190         description="Operational Break Even",
191         font_size=milestone_font_size,
192         # TODO: remove date, make it place on the right position.
193         date="2023-06-09",
194     ),
195 )
196
197 # Grandchildren
198 auditing = Activity(
199     description="Auditing", duration=10, new_tag=0, parent=
        ↳ organisation
200 )
201 hiring = Activity(
202     description="Hiring", duration=20, new_tag=1, parent=
        ↳ organisation
203 )
204 administration = Activity(
205     description="Administration",
206     duration=10,
207     new_tag=2,
208     parent=organisation,
209 )
210 legal = Activity(
211     description="Legal", duration=20, new_tag=3, parent=
        ↳ organisation
212 )
213 financial = Activity(
214     description="Financial", duration=20, new_tag=4, parent=
        ↳ organisation
215 )
216 organisation.add_children(
217     [auditing, hiring, administration, legal, financial]
218 )
219
220 company.add_children(
221     [partners, organisation, marketing_company, company_buffer]
222 )
223 parents.append(company)
224
225 # Create Milestones
226
227 return parents, date_milestones
228
229
230 def addTwo(x):
231     """adds two to the incoming integer and returns the result of the

```

```
232     computation.""
233     return x + 2
```

---

## D Appendix src/Gantt.py

```
1  """Creates the Gantt chart specified in create_python_gantt."""
2
3  from src.Create_python_gantt import create_python_gantt
4  from src.export_data.Milestone import (
5      get_milestone_for_date,
6      get_milestone_style,
7      get_milestone_uml_line,
8  )
9
10
11 # pylint: disable=R0902
12 class Gantt:
13     """Creates the Gantt chart specified in create_python_gantt."""
14
15     def __init__(self, filepath: str, params: dict):
16         self.start_line = "@startgantt"
17         self.project_start_date = "2022/10-01"
18         self.closed_days = ["saturday", "sunday"]
19         self.gantt_font_size = 100
20         self.gantt_font_size_line = (
21             f"skinparam classFontSize {self.gantt_font_size}"
22         )
23         self.box_font_size = "30"
24
25         # self.font_size="skinparam defaultFontSize 100"
26         self.parents, self.date_milestones = create_python_gantt(
27             params["wages"], self.box_font_size, self.
28             ↪ project_start_date
29         )
30         self.end_line = "@endgantt"
31         self.lines, self.parent_costs = self.get_plantuml_gantt_lines
32             ↪ ()
33         self.write_gantt(filepath, self.lines)
34
35         self.costs = None
36
37     def get_plantuml_gantt_lines(self):
38         """Gets the list of lines of the UML file to create a Gantt
39             ↪ chart."""
40         lines = []
41         lines.append(self.start_line)
42         lines.append(f"project starts the {self.project_start_date}")
43         lines = self.add_closed_dates(lines)
44         lines.append(self.gantt_font_size_line)
45         # Add milestone style here
46         lines.append(get_milestone_style("blue", 100, "red", "yellow"
47             ↪ ))
48
49         # Include date milestones
50         for date_milestone in self.date_milestones:
51             lines.append(get_milestone_for_date(date_milestone))
52
53         # Writes acitivity line, appends milestone line below
54             ↪ activity.
55         lines, parent_costs = self.loop_through_parents_printing(
56             ↪ lines)
57         lines.append(self.end_line)
58         return lines, parent_costs
59
60     def loop_through_parents_printing(self, lines):
```



```

55     """Prints all relevant data of the parent activities.
56
57     :param lines:
58         """
59     # print descriptions
60     for i, _ in enumerate(self.parents):
61         lines = self.print_parent_descriptions(lines, self.
            ↪ parents[i])
62
63     # print order
64     for i, _ in enumerate(self.parents):
65         if i > 0:
66             if not self.parents[i].starts_at_child_nr_start is
            ↪ None:
67                 lines.append(
68                     f"[{self.parents[i].get_tag()}] starts at ["
69                     + f"{self.parents[i].starts_at_child_nr_start
            ↪ }]"
70                     + "'s start"
71                 )
72             else:
73                 print(
74                     f"parent_descr={self.parents[i].description},
            ↪ and "
75                     + "starts at: "
76                     + f"{self.parents[i].starts_at_child_nr_start}"
77                     + " and tag="
78                     + f"{self.parents[i].get_tag()}, writing end"
79                 )
80                 lines.append(
81                     f"[{self.parents[i].get_tag()}] starts at ["
82                     + f"{self.parents[i-1].get_tag()}]'s end"
83                 )
84             lines = self.print_parent_order(lines, self.parents[i])
85
86     # print colour
87     for i, _ in enumerate(self.parents):
88         lines = self.print_parent_colour(lines, self.parents[i])
89
90     # compute costs
91     total_costs = 0
92     parent_costs = {}
93     for i, _ in enumerate(self.parents):
94         parent, total_costs, lines = self.print_parent_costs(
95             total_costs, lines, self.parents[i]
96         )
97         parent_costs[parent] = total_costs
98     return lines, parent_costs
99
100 def print_parent_descriptions(self, lines, parent):
101     """Creates the lines with the descriptions of the parents.
102
103     :param lines:
104     :param parent:
105         """
106     lines.append("")
107     lines = self.print_descriptions(parent, lines)
108     return lines
109
110 def print_parent_order(self, lines, parent):
111     """Creates the lines that specify the order of the parent
        ↪ activities.

```

```

112         :param lines:
113         :param parent:
114         """
115         lines.append("")
116         lines = self.print_order(parent, lines)
117         return lines
118
119     def print_parent_colour(self, lines, parent):
120         """Creates the line that specifies the colour of the parent.
121
122         :param lines:
123         :param parent:
124         """
125         lines.append("")
126         lines = self.print_colour(parent, lines)
127         lines.append("")
128         return lines
129
130     def print_parent_costs(self, total_costs, lines, parent):
131         """Computes the costs of a parent activity based on the costs
132         ↪ of its
133         children.
134
135         :param total_costs:
136         :param lines:
137         :param parent:
138         """
139         lines.append("")
140         total_costs, lines = self.print_costs(total_costs, parent,
141         ↪ lines)
142         if isinstance(total_costs, tuple):
143             total_costs = total_costs[0]
144         print(f"parent={parent.description}, total_costs={total_costs}
145         ↪ ")
146
147         lines.append("")
148         return parent, total_costs, lines
149
150     def print_descriptions(self, activity, lines):
151         """Creates the UML lines that specify the activity
152         ↪ description.
153
154         :param activity:
155         :param lines:
156         """
157
158         if activity.font_size is not None:
159             font_size = activity.font_size
160         else:
161             font_size = self.box_font_size
162         lines.append(
163             f"<size:{font_size}>{activity.description}] as ["
164             + f"{activity.get_tag()}] lasts {activity.duration} days"
165         )
166
167         # If activity has milestone, include it here.
168         if activity.milestone is not None:
169             lines.append(
170                 get_milestone_uml_line(activity.milestone, activity.
171                 ↪ get_tag())
172             )

```

```

169     for child in activity.children:
170         lines = self.print_descriptions(child, lines)
171     return lines
172
173 def print_order(self, activity, lines):
174     """Creates the order in which the activities are performed.
175         ↳ The orders
176         are stored relatively to eachother. E.g. activity y starts at
177         ↳ the end
178         of activity x.
179
180     :param activity:
181     :param lines:
182     """
183     # Write order for all children in an activity
184     for i, _ in enumerate(activity.children):
185         if i == 0:
186             lines.append(
187                 f"[{activity.children[i].get_tag()}] starts at ["
188                 + f"{activity.get_tag()}]'s start"
189             )
190         else:
191             # check if it starts at the start or end of some
192             ↳ activity
193             if not activity.children[i].starts_at_child_nr_start
194             ↳ is None:
195                 lines.append(
196                     f"[{activity.children[i].get_tag()}] starts
197                     ↳ at ["
198                     + f"{activity.children[i].
199                     ↳ starts_at_child_nr_start}"
200                     + "]"'s start"
201                 )
202             else:
203                 lines.append(
204                     f"[{activity.children[i].get_tag()}] starts
205                     ↳ at ["
206                     + f"{activity.children[i-1].get_tag()}]'s end
207                     ↳ "
208                 )
209
210     # start recursive loop to write order of each of the
211     ↳ children
212     for child in activity.children:
213         lines = self.print_order(child, lines)
214     return lines
215
216 def print_colour(self, activity, lines):
217     """Creates the colours of the activities.
218
219     :param activity:
220     :param lines:
221     """
222     lines.append(
223         f"[{activity.get_tag()}] is colored in {activity.colour}
224         ↳ "
225     )
226     for child in activity.children:
227         lines = self.print_colour(child, lines)
228     return lines
229
230 def print_costs(self, total_costs, activity, lines):

```

```

221     """Creates the lines that specify the costs of the activities
222         ↪ .
223
224     # TODO: output to LaTeX variables.
225
226     :param total_costs:
227     :param activity:
228     :param lines:
229     """
230     activity_costs = (
231         activity.duration * activity.hours_per_day * activity.
232         ↪ hourly_wage
233     )
234     lines.append(
235         f"[{activity.description}] takes: {activity.duration}[
236         ↪ days] "
237         + f"equating to:{activity.duration*activity.hours_per_day
238         ↪ }[hours]"
239         + f" and costs:{activity.hourly_wage} per hour, yielding
240         ↪ activity"
241         + " costs:"
242         + f"{activity_costs}"
243         + " Euros."
244     )
245     total_costs = (
246         total_costs
247         + activity.duration * activity.hours_per_day * activity.
248         ↪ hourly_wage
249     )
250     for child in activity.children:
251         total_costs, lines = self.print_costs(total_costs, child,
252         ↪ lines)
253     return total_costs, lines
254
255 def add_closed_dates(self, lines):
256     """Adds the recurring days on which no work is performed in
257         ↪ the Gantt.
258
259     :param lines:
260     """
261     for closed_day in self.closed_days:
262         lines.append(f"{closed_day} are closed")
263     return lines
264
265 def write_gantt(self, filepath, some_list):
266     """Writes the lines of a gant to an output file.
267
268     :param filepath:
269     :param some_list:
270     """
271     with open(filepath, "w", encoding="utf-8") as f:
272         for item in some_list:
273             # pylint: disable=C0209
274             f.write("%s\n" % item)
275     f.close()
276
277 def addTwo(self, x):
278     """adds two to the incoming integer and returns the result of
279         ↪ the
280         computation.
281
282     :param x:

```

274

"""

275

**return** x + 2

---

## E Appendix src/\_main\_.py

---

```
1  """Entry point for this project, runs the project code and exports
   ↪ data if
2  export commands are given to the cli command that invokes this script
   ↪ ."""
3
4
5  from src.arg_parser import parse_cli_args
6  from src.export_data.export_data import export_data
7
8  # Project code imports.
9  from src.revenue_model.Model_top_down import Model_top_down
10
11 # Export data import.
12
13
14 # Parse command line interface arguments to determine what this
   ↪ script does.
15 args = parse_cli_args()
16
17 # run monte-carlo for revenue estimation
18 model = Model_top_down()
19
20
21 # Run data export code if any argument is given.
22 if not all(
23     arg is None for arg in [args.l, args.dd, args.sd, args.c2l, args.
   ↪ ec2l]
24 ):
25     export_data(args)
```

---

## F Appendix src/arg\_parser.py

```
1  """This is the main code of this project nr, and it manages running
   ↪ the code
2  and outputting the results to LaTeX."""
3  import argparse
4
5
6  def parse_cli_args():
7      """Parses the command line arguments to determine what this code
   ↪ should
8      do."""
9      # Instantiate the parser
10     parser = argparse.ArgumentParser(description="Optional app
   ↪ description")
11
12     # Include argument parsing for data exporting code.
13     # Compile LaTeX
14     parser.add_argument(
15         "--l",
16         action="store_true",
17         help="Boolean indicating if code compiles LaTeX",
18     )
19
20     # Generate, compile and export Dynamic PlantUML diagrams to LaTeX
   ↪ .
21     parser.add_argument(
22         "--dd",
23         action="store_true",
24         help=(
25             "A boolean indicating if code generated diagrams are
   ↪ compiled"
26             + " and exported."
27         ),
28     )
29     # Generate, compile and export Static PlantUML diagrams to LaTeX.
30     parser.add_argument(
31         "--sd",
32         action="store_true",
33         help=(
34             "A boolean indicating if static diagrams are compiled and
   ↪ "
35             + " exported."
36         ),
37     )
38
39     # Export the project code to LaTeX.
40     parser.add_argument(
41         "--c2l",
42         action="store_true",
43         help="A boolean indicating if project code is exported to
   ↪ LaTeX.",
44     )
45
46     # Export the exporting code, and the project code to LaTeX.
47     parser.add_argument(
48         "--ec2l",
49         action="store_true",
50         help=(
51             "A boolean indicating if code that exports code is
   ↪ exported"
52             + " to LaTeX."
```

```
53         ),
54     )
55     # Load the arguments that are given.
56     args = parser.parse_args()
57     return args
```

---



## G Appendix src/export\_data/Hardcoded\_data.py

```
1  """Specify hardcoded output data."""
2
3
4  # pylint: disable=R0902
5  # pylint: disable=R0903
6  class Hardcoded_data:
7      """Contains a list of parameters that are used in this project
8          ↪ that
9          combines Python code with a LaTeX document."""
10
11     def __init__(self):
12
13         # Specify code configuration details
14         # TODO: include as optional arguments.
15         self.await_compilation = True
16         self.verbose = True
17         self.gantt_extension = ".uml"
18         self.diagram_extension = ".png"
19
20         # Filenames.
21         self.main_latex_filename = "main.tex"
22         self.export_data_dirname = "export_data"
23         self.diagram_dir = "Diagrams"
24         self.plantuml_java_filename = "plantuml.jar"
25
26         # Appendix manager filenames
27         self.export_code_appendices_filename = "
28             ↪ export_code_appendices.tex"
29         self.export_code_appendices_filename_from_root = (
30             "export_code_appendices_from_root.tex"
31         )
32         self.project_code_appendices_filename = "
33             ↪ project_code_appendices.tex"
34         self.project_code_appendices_filename_from_root = (
35             "project_code_appendices_from_root.tex"
36         )
37         self.automatic_appendices_manager_filenames = [
38             self.export_code_appendices_filename,
39             self.export_code_appendices_filename_from_root,
40             self.project_code_appendices_filename,
41             self.project_code_appendices_filename_from_root,
42         ]
43
44         self.manual_appendices_filename = "manual_appendices.tex"
45         self.manual_appendices_filename_from_root = (
46             "manual_appendices_from_root.tex"
47         )
48         self.manual_appendices_manager_filenames = [
49             self.manual_appendices_filename,
50             self.manual_appendices_filename_from_root,
51         ]
52         self.appendix_dir_from_root = "latex/Appendices/"
53
54         # Folder names.
55         self.dynamic_diagram_dir = "Dynamic_diagrams"
56         self.static_diagram_dir = "Static_diagrams"
57
58         # Specify paths relative to root.
59         self.path_to_export_data_from_root = f"src/{self.
60             ↪ export_data_dirname}"
```

```

57     self.jar_path_relative_from_root = (
58         f"{self.path_to_export_data_from_root}"
59         + f"/{self.plant_uml_java_filename}"
60     )
61     self.diagram_output_dir_relative_to_root = (
62         f"latex/Images/{self.diagram_dir}"
63     )
64
65     # Path related variables
66     self.append_export_code_to_latex = True
67     self.path_to_dynamic_ganttts = (
68         f"{self.path_to_export_data_from_root}/"
69         + f"{self.diagram_dir}/{self.dynamic_diagram_dir}"
70     )
71     self.path_to_static_ganttts = (
72         f"{self.path_to_export_data_from_root}/"
73         + f"{self.diagram_dir}/{self.static_diagram_dir}"
74     )

```

---

## H Appendix src/export\_data/Milestone.py

```
1  """Represents a milestone object that can be added to the end of an
   ↪ activity."""
2
3
4  import re
5
6
7  # pylint: disable=R0903
8  class Milestone:
9      """Used to create a milestone at the end or start of an activity
   ↪ in a Gantt
10     chart."""
11
12     # pylint: disable=R0913
13     def __init__(
14         self,
15         description,
16         font_size,
17         at_end=True,
18         date=None,
19     ):
20         self.font_size = font_size
21         if font_size is None:
22             raise Exception("Error, milestone fontsize not set.")
23         self.description = description
24         self.at_end = at_end
25         self.date = date
26
27         # Create an alphanumeric tag with underscores.
28         self.tag = re.sub(r"^[^A-Za-z0-9_]+", "_", self.description).
   ↪ lower()
29         # Specify milestone stile.
30
31
32     def get_milestone_style(colour, fontsize, backgroundcolour,
   ↪ linecolour):
33         """Returns the UML style specification of a milestone.
34
35         :param colour:
36         :param fontsize:
37         :param backgroundcolour:
38         :param linecolour:
39         """
40         left = """
41         <style>
42         ganttDiagram {
43         milestone {
44         """
45         middle = f"""
46             FontColor {colour}
47             FontSize {fontsize}
48             FontStyle italic
49             BackGroundColor {backgroundcolour}
50             LineColor {linecolour}
51         """
52         end = """
53         }
54     }
55 </style>
   """
56
```

```

57     return left + middle + end
58
59
60 def get_milestone_uml_line(milestone, activity_tag):
61     """Returns the uml line for a milestone.
62
63     :param milestone: param activity_tag:
64     :param activity_tag:
65     """
66     if milestone.date is not None:
67         milestone = get_milestone_for_date(milestone)
68     else:
69         milestone = get_milestone_for_activity(milestone,
70             ↪ activity_tag)
71     print(f"milestone={milestone}")
72     return milestone
73
74 def get_milestone_for_activity(milestone, activity_tag):
75     """returns a milestone for a certain activity.
76
77     :param milestone: param activity_tag:
78     :param activity_tag:
79     """
80     left = get_milestone_left(milestone)
81     right = get_milestone_right(milestone, activity_tag)
82     return left + right
83
84
85 def get_milestone_for_date(milestone):
86     """Return milestone for a certain date.
87
88     :param milestone:
89     """
90     left = get_milestone_left(milestone)
91     if milestone.date is not None:
92         # TODO: verify date format is valid yyyy-mm-dd
93         right = f" {milestone.date}"
94     milestone = left + right
95     print(f"milestone={milestone}")
96     return milestone
97
98
99 def get_milestone_left(milestone):
100     """Returns the left part of a milestone description.
101
102     :param milestone:
103     """
104     left = (
105         f"<size:{milestone.font_size}>{milestone.description} - "
106         + f"{milestone.tag}] as [{milestone.tag}] happens "
107     )
108     return left
109
110
111 def get_milestone_right(milestone, activity_tag):
112     """Returns the right part of a milestone description.
113
114     :param milestone: param activity_tag:
115     :param activity_tag:
116     """
117     if milestone.at_end:

```

```
118         right = "at [" + activity_tag + "]"'s end"
119     elif not milestone.at_end:
120         right = "at [" + activity_tag + "]"'s start"
121     return right
```

---

## I Appendix src/export\_data/Plot\_to\_tex.py

```
1  """File used to export plots to a latex directory.
2  Call this from another file, for project 11, question 3b:
3  from Plot_to_tex import Plot_to_tex as plt_tex
4  multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints), dtype=
    ↪ int);
5  # actually fill with data
6  lineLabels = [] # add a label for each dataseries
7  plt_tex.plotMultipleLines(plt_tex,single_x_series,multiple_y_series,"
    ↪ x-axis
8  label [units]","y-axis label [units]",lineLabels,"3b",4,11)
9  4b=filename
10 4 = position of legend, e.g. top right.
11 For a single line, use:
12 plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),dataseries,"
    ↪ x-axis
13 label [units]","y-axis label [units]",lineLabel,"3b",4,11)"""
14
15 import os
16
17 import matplotlib.pyplot as plt
18 import numpy as np
19 from matplotlib import lines
20
21 # You can also plot a table directly into latex, see
    ↪ example_create_a_table(..)
22 # Then put it in latex with for example:
23 # \begin{table}[H]
24 #     \centering
25 #     \caption{Results some computation.}\label{tab:some_computation}
26 #     \begin{tabular}{|c|c|} % remember to update this
27 # % to show all columns of table
28 #         \hline
29 #         \input{latex/project3/tables/q2.txt}
30 #     \end{tabular}
31 # \end{table}
32
33
34 class Plot_to_tex:
35     """Object used to output plots to latex directory of project."""
36
37     def __init__(self):
38         self.script_dir = self.get_script_dir()
39
40     # plot graph (legendPosition = integer 1 to 4)
41     def plotSingleLine(
42         self,
43         x_path,
44         y_series,
45         x_axis_label,
46         y_axis_label,
47         label,
48         filename,
49         legendPosition,
50         project_name,
51     ):
52         """
53
54         :param x_path: param y_series:
55         :param x_axis_label: param y_axis_label:
56         :param label: param filename:
```

```

57 :param legendPosition: param project_name:
58 :param y_series: param y_axis_label:
59 :param filename: param project_name:
60 :param y_axis_label: param project_name:
61 :param project_name:
62
63 """
64 # pylint: disable=R0913
65 # TODO: reduce 9/5 arguments to at most 5/5 arguments.
66 fig = plt.figure()
67 ax = fig.add_subplot(111)
68 ax.plot(x_path, y_series, c="b", ls="--", label=label,
69         ↪ fillstyle="none")
70 plt.legend(loc=legendPosition)
71 plt.xlabel(x_axis_label)
72 plt.ylabel(y_axis_label)
73 plt.savefig(
74     os.path.dirname(__file__)
75     + f"/../../../latex/{project_name}"
76     + "/Images/"
77     + filename
78     + ".png"
79 )
80
81 # plt.show();
82
83 # plot graphs
84 def plotMultipleLines(
85     self,
86     x,
87     y_series,
88     x_label,
89     y_label,
90     label,
91     filename,
92     legendPosition,
93     project_name,
94 ):
95     """
96
97     :param x: param y_series:
98     :param x_label: param y_label:
99     :param label: param filename:
100     :param legendPosition: param project_name:
101     :param y_series: param y_label:
102     :param filename: param project_name:
103     :param y_label: param project_name:
104     :param project_name:
105
106     """
107     # pylint: disable=R0913
108     # TODO: reduce 9/5 arguments to at most 5/5 arguments.
109     fig = plt.figure()
110     ax = fig.add_subplot(111)
111
112     # generate colours
113     cmap = self.get_cmap(len(y_series[:, 0]))
114
115     # generate line types
116     lineTypes = self.generateLineTypes(y_series)
117
118     for i in range(0, len(y_series)):

```

```

118         # overwrite linetypes to single type
119         lineTypes[i] = "_"
120         ax.plot(
121             x,
122             y_series[i, :],
123             ls=lineTypes[i],
124             label=label[i],
125             fillstyle="none",
126             c=cmap(i),
127         )
128         # color
129
130     # configure plot layout
131     plt.legend(loc=legendPosition)
132     plt.xlabel(x_label)
133     plt.ylabel(y_label)
134     plt.savefig(
135         os.path.dirname(__file__)
136         + f"/../../../latex/{project_name}"
137         + "/Images/"
138         + filename
139         + ".png"
140     )
141
142     # Generate random line colours
143     # Source: https://stackoverflow.com/questions/14720331/how-to-generate-random-colors-in-matplotlib
144     # how-to-generate-random-colors-in-matplotlib
145     def get_cmap(self, n, name="hsv"):
146         """Returns a function that maps each index in 0, 1, ..., n-1
147             ↪ to a
148             distinct RGB color; the keyword argument name must be a
149             ↪ standard mpl
150             colormap name.
151
152             :param n: param name: (Default value = "hsv")
153             :param name: Default value = "hsv"
154             """
155         return plt.cm.get_cmap(name, n)
156
157     def generateLineTypes(self, y_series):
158         """
159
160         :param y_series:
161
162         """
163         # generate varying linetypes
164         typeOfLines = list(lines.lineStyles.keys())
165
166         while len(y_series) > len(typeOfLines):
167             typeOfLines.append("-.")
168
169         # remove void lines
170         for i in range(0, len(y_series)):
171             if typeOfLines[i] == "None":
172                 typeOfLines[i] = "_"
173             if typeOfLines[i] == ":":
174                 typeOfLines[i] = ":"
175             if typeOfLines[i] == " ":
176                 typeOfLines[i] = " "
177         return typeOfLines

```



```

177 # Create a table with: table_matrix = np.zeros((4,4),dtype=object
178     ↪ ) and pass
179 # it to this object
180 def put_table_in_tex(self, table_matrix, filename, project_name):
181     """
182     :param table_matrix: param filename:
183     :param project_name: param filename:
184     :param filename:
185     """
186     cols = np.shape(table_matrix)[1]
187     some_format = "%s"
188     for _ in range(1, cols):
189         some_format = some_format + " & %s"
190     some_format = some_format + ""
191     # TODO: Change to something else to save as txt.
192     np.savetxt(
193         os.path.dirname(__file__)
194         + f"/../../../latex/{project_name}"
195         + "/tables/"
196         + filename
197         + ".txt",
198         table_matrix,
199         delimiter=" & ",
200         fmt=format,
201         newline=" \\\\ \\hline \n",
202     )
203
204
205 # replace this with your own table creation and then pass it to
206 # put_table_in_tex(..)
207 def example_create_a_table(self):
208     """Example on how to create a latex table from Python."""
209     project_name = "1"
210     table_name = "example_table_name"
211     rows = 2
212     columns = 4
213     table_matrix = np.zeros((rows, columns), dtype=object)
214     table_matrix[:, :] = "" # replace the standard zeros with
215         ↪ empty cell
216     print(table_matrix)
217     for column in range(0, columns):
218         for row in range(0, rows):
219             table_matrix[row, column] = row + column
220     table_matrix[1, 0] = "example"
221     table_matrix[0, 1] = "grid sizes"
222
223     self.put_table_in_tex(table_matrix, table_name, project_name)
224
225 def get_script_dir(self):
226     """returns the directory of this script regardless of from
227         ↪ which level
228     the code is executed."""
229     return os.path.dirname(__file__)
230
231
232 def export_plot(self, some_plt, filename):
233     """
234     :param plt:
235     :param filename:
236     """

```

```

236         self.create_target_dir_if_not_exists("latex/Images/", "graphs
237         ↪ ")
238         some_plt.savefig(
239             "latex/Images/" + "graphs/" + filename + ".png", dpi=200
240         )
241     def create_target_dir_if_not_exists(self, path, new_dir_name):
242         """
243
244         :param path:
245         :param new_dir_name:
246
247         """
248         if os.path.exists(path):
249             if not os.path.exists(f"{path}/{new_dir_name}"):
250                 os.makedirs(f"{path}/{new_dir_name}")
251         else:
252             raise Exception(f"Error, path={path} did not exist.")
253
254 if __name__ == "__main__":
255     main = Plot_to_tex()
256     main.example_create_a_table()
257

```

---

## J Appendix src/export\_data/create\_dynamic\_diagrams.py

---

```
1  """Generates .uml PlantUML files based on Python code.
2
3  Typically used for Gantt charts.
4  """
5  from src.export_data.plantuml_compile import (
6      compile_diagrams_in_dir_relative_to_root,
7  )
8  from src.export_data.plantuml_generate import
9      ↪ generate_all_dynamic_diagrams
10 from src.export_data.plantuml_to_tex import export_diagrams_to_latex
11
12 def create_dynamic_diagrams(args, hd):
13     """Generates the dynamic diagrams."""
14     # Generate PlantUML diagrams dynamically (using code).
15     if args.dd:
16         generate_all_dynamic_diagrams(
17             f"{hd.path_to_export_data_from_root}/Diagrams/
18             ↪ Dynamic diagrams"
19         )
20
21     # Compile dynamically generated PlantUML diagrams to images.
22     compile_diagrams_in_dir_relative_to_root(
23         hd.await_compilation,
24         hd.gantt_extension,
25         hd.jar_path_relative_from_root,
26         hd.path_to_dynamic_gantts,
27         hd.verbose,
28     )
29
30     # Export dynamic PlantUML text files to LaTeX.
31     export_diagrams_to_latex(
32         hd.path_to_dynamic_gantts,
33         hd.gantt_extension,
34         hd.diagram_output_dir_relative_to_root,
35     )
36
37     # Export dynamic PlantUML diagram images to LaTeX.
38     export_diagrams_to_latex(
39         hd.path_to_dynamic_gantts,
40         hd.diagram_extension,
41         hd.diagram_output_dir_relative_to_root,
```

---

## K Appendix src/export\_data/create\_static\_diagrams.py

---

```
1  """This file creates the diagrams that are written directly in plain
2      ↪ .uml
3      files."""
4  from src.export_data.plantuml_compile import (
5      compile_diagrams_in_dir_relative_to_root,
6  )
7  from src.export_data.plantuml_to_tex import export_diagrams_to_latex
8
9  def create_static_diagrams(args, hd):
10     """Generates .png images of the static diagram .uml files.
11
12     :param args:
13     :param hd:
14     """
15     # PlantUML
16     if args.sd:
17         # Compile statically generated PlantUML diagrams to images.
18         compile_diagrams_in_dir_relative_to_root(
19             hd.await_compilation,
20             hd.gantt_extension,
21             hd.jar_path_relative_from_root,
22             hd.path_to_static_gantts,
23             hd.verbose,
24         )
25
26         # Export static PlantUML text files to LaTeX.
27         export_diagrams_to_latex(
28             hd.path_to_static_gantts,
29             hd.gantt_extension,
30             hd.diagram_output_dir_relative_to_root,
31         )
32
33         # Export static PlantUML diagram images to LaTeX.
34         export_diagrams_to_latex(
35             hd.path_to_static_gantts,
36             hd.diagram_extension,
37             hd.diagram_output_dir_relative_to_root,
38         )
```

---

## L Appendix src/export\_data/export\_data.py

```
1  """File used to export data to latex."""
2  from pprint import pprint
3
4  from src.export_data.create_dynamic_diagrams import
5      ↪ create_dynamic_diagrams
6  from src.export_data.create_static_diagrams import
7      ↪ create_static_diagrams
8  from src.export_data.Hardcoded_data import Hardcoded_data
9  from src.export_data.latex_compile import compile_latex
10 from src.export_data.latex_export_code import export_code_to_latex
11
12 def export_data(args):
13     """Parses the Python arguments that specify what should be
14     ↪ compiled.
15
16     :param args:
17     """
18
19     hd = Hardcoded_data()
20
21     # Parameter exporting is done in the models themselves.
22
23     # Generating PlantUML diagrams
24     create_dynamic_diagrams(args, hd)
25     create_static_diagrams(args, hd)
26
27     # Plotting graphs using Python code, and export them to latex.
28     # Generate plots.
29     # Export plots to LaTeX.
30
31     # Export code to LaTeX.
32     if args.c2l:
33         # TODO: verify whether the latex/{project_name}/Appendices
34         ↪ folder
35         # exists before exporting.
36         # TODO: verify whether the latex/{project_name}/Images folder
37         ↪ exists
38         # before exporting.
39         export_code_to_latex(hd, False)
40     elif args.ec2l:
41         # TODO: verify whether the latex/{project_name}/Appendices
42         ↪ folder
43         # exists before exporting.
44         # TODO: verify whether the latex/{project_name}/Images folder
45         ↪ exists
46         # before exporting.
47         export_code_to_latex(hd, True)
48
49     # Compile the accompanying LaTeX report.
50     if args.l:
51         compile_latex(True, True)
52         print("")
53     print("\n\nDone exporting data.")
54
55 def get_latex_param_lines(params):
56     """Exports the model parameters and computed values to LaTeX
57     ↪ variables."""
58     # Export model parameters to .tex file with LaTeX variables.
```

```

53 param_lines = []
54 # TODO: flatten dict
55 # print(f'params={params}')
56 pprint(params)
57
58 for key, value in params.items():
59     if key == "wages":
60         for wages_key, wages_value in params[key].items():
61             param_lines.append(
62                 "\\newcommand"
63                 + chr(92)
64                 + str(wages_key.replace("_", ""))
65                 + "{"
66                 + str(wages_value)
67                 + "}"
68             )
69     else:
70         param_lines.append(
71             "\\newcommand"
72             + chr(92)
73             + str(key.replace("_", "").replace("-", ""))
74             + "{"
75             + str(value)
76             + "}"
77         )
78 return param_lines

```

---

## M Appendix src/export\_data/helper\_bash\_commands.py

---

```
1  """Code used to execute Bash commands from Python."""
2  import subprocess # nosec
3
4  from src.export_data.plantuml_compile import
   ↪ get_output_of_bash_command
5
6
7  def run_a_bash_command(await_compilation, bash_command, verbose):
8      """Runs a bash commands from Python.
9
10     :param await_compilation:
11     :param bash_command:
12     :param verbose:
13     """
14     if await_compilation:
15         if verbose:
16             subprocess.call(
17                 bash_command,
18                 shell=True, # nosec
19                 stdout=subprocess.PIPE,
20                 stderr=subprocess.STDOUT,
21             )
22         else:
23             # pylint: disable=E1129
24             # pylint: disable=R1732
25             subprocess.call(
26                 bash_command,
27                 shell=True, # nosec
28                 stderr=subprocess.DEVNULL,
29                 stdout=subprocess.DEVNULL,
30             )
31     else:
32         if verbose:
33             # pylint: disable=R1732
34             subprocess.Popen(
35                 bash_command,
36                 shell=True, # nosec
37                 stdout=subprocess.PIPE,
38                 stderr=subprocess.STDOUT,
39             )
40         else:
41             with subprocess.Popen(
42                 bash_command,
43                 shell=True, # nosec
44                 stderr=subprocess.DEVNULL,
45                 stdout=subprocess.DEVNULL,
46             ) as process:
47                 get_output_of_bash_command(process)
```

---

## N Appendix src/export\_data/helper\_dir\_file\_edit.py

```
1  """A helper file that is used for directory and file editing."""
2  import glob
3  import os
4  import shutil
5
6
7  def file_contains(filepath, substring):
8      """Returns True if a file exists. None otherwise.
9
10     :param filepath:
11     :param substring:
12     """
13     with open(filepath, encoding="utf-8") as f:
14         return substring in f.read()
15
16
17  def get_dir_filelist_based_on_extension(dir_relative_to_root,
18     ↪ extension):
19     """
20     :param dir_relative_to_root: A relative directory as
21     seen from the root dir of this project.
22     :param extension: File extension that is used/searched in this
23     ↪ function.
24
25     """
26     selected_filenames = []
27     # TODO: assert directory exists
28     for filename in os.listdir(dir_relative_to_root):
29         if filename.endswith(extension):
30             selected_filenames.append(filename)
31     return selected_filenames
32
33  def create_dir_relative_to_root_if_not_exists(dir_relative_to_root):
34     """
35
36     :param dir_relative_to_root: A relative directory as
37     seen from the root dir of this project.
38
39     """
40     if not os.path.exists(dir_relative_to_root):
41         os.makedirs(dir_relative_to_root)
42
43
44  def dir_relative_to_root_exists(dir_relative_to_root):
45     """
46
47     :param dir_relative_to_root: A relative directory as
48     seen from the root dir of this project.
49
50     """
51     if not os.path.exists(dir_relative_to_root):
52         return False
53     if os.path.exists(dir_relative_to_root):
54         return True
55     raise Exception(
56         f"Directory relative to root: {dir_relative_to_root}"
57         + " did not exist, nor did it exist."
58     )
```



```

59
60
61 def get_all_files_in_dir_and_child_dirs(extension, path,
    ↳ excluded_files=None):
62     """Returns a list of the relative paths to all files within the
        ↳ some path
63     that match the given file extension. Also includes files in child
64     directories.
65
66     :param extension: File extension that is used/searched in this
        ↳ function.
67     The file extension of the file that is sought in the appendix
        ↳ line. Either
68     ".py" or ".pdf".
69     :param path: Absolute filepath in which files are being sought.
70     :param excluded_files: Default value = None) Files that will not
        ↳ be
71     included even if they are found.
72     """
73     filepaths = []
74     for r, _, f in os.walk(path):
75         for file in f:
76             if file.endswith(extension):
77                 if (excluded_files is None) or (
78                     excluded_files is not None)
79                     and (file not in excluded_files)
80                 ):
81                     filepaths.append(r + "/" + file)
82     return filepaths
83
84
85 def get_filepaths_in_dir(extension, path, excluded_files=None):
86     """Returns a list of the relative paths to all files within the
        ↳ some path
87     that match the given file extension. Does not include files in
88     child_directories.
89
90     :param extension: File extension that is used/searched in this
        ↳ function.
91     The file extension of the file that is sought in the appendix
        ↳ line.
92     Either ".py" or ".pdf".
93     :param path: Absolute filepath in which files are being sought.
94     :param excluded_files: Default value = None) Files that will not
        ↳ be
95     included even if they are found.
96     """
97     filepaths = []
98     current_path = os.getcwd()
99     os.chdir(path)
100    for file in glob.glob(f"*.{extension}"):
101        print(file)
102        if (excluded_files is None) or (
103            excluded_files is not None) and (file not in
            ↳ excluded_files)
104        ):
105            # Append normalised filepath e.g. collapses b/src/../d to
            ↳ b/d.
106            filepaths.append(os.path.normpath(f"{path}/{file}"))
107    os.chdir(current_path)
108    return filepaths
109

```

```

110 def sort_filepaths_by_filename(filepaths):
111     """
112
113     :param filepaths:
114
115     """
116     # filepaths.sort(key = lambda x: x.split()[1])
117     filepaths.sort(key=lambda x: x[x.rfind("/") + 1 :]) # noqa: E203
118     for filepath in filepaths:
119         print(f"{filepath}")
120     return filepaths
121
122
123 def get_filename_from_dir(path):
124     """Returns a filename from an absolute path to a file.
125
126     :param path: path to a file of which the name is queried.
127     """
128     return path[path.rfind("/") + 1 :] # noqa: E203
129
130
131 def delete_file_if_exists(filepath):
132     """
133
134     :param filepath:
135
136     """
137     try:
138         os.remove(filepath)
139     except OSError:
140         pass
141
142
143 def convert_filepath_to_filepath_from_root(filepath,
144     ↪ normalised_root_path):
145     """
146
147     :param filepath:
148     :param normalised_root_path:
149
150     """
151     normalised_filepath = os.path.normpath(filepath)
152     filepath_relative_from_root = normalised_filepath[
153         len(normalised_root_path) : # noqa: E203
154     ]
155     return filepath_relative_from_root
156
157 def overwrite_file(filepath, lines):
158     """
159
160     :param filepath:
161     :param lines:
162
163     """
164     with open(filepath, "w", encoding="utf-8") as the_file:
165         for line in lines:
166             the_file.write(f"{line}\n")
167
168
169 def append_lines_to_file(filepath, lines):
170

```

```

171     """
172
173     :param filepath:
174     :param lines:
175
176     """
177     with open(filepath, "a", encoding="utf-8") as the_file:
178         for line in lines:
179             the_file.write(f"{line}\n")
180
181
182 def append_line_to_file(filepath, line):
183     """
184
185     :param filepath:
186     :param line:
187
188     """
189     with open(filepath, "a", encoding="utf-8") as the_file:
190         the_file.write(f"{line}\n")
191         the_file.close()
192
193
194 def remove_all_auto_generated_appendices(hd):
195     """
196
197     :param hd:
198
199     """
200
201     # TODO: move identifier into hardcoded.
202     all_appendix_files = get_all_files_in_dir_and_child_dirs(
203         ".tex", hd.appendix_dir_from_root, excluded_files=None
204     )
205     for file in all_appendix_files:
206         if "Auto_generated" in file:
207             delete_file_if_exists(file)
208
209
210 def delete_dir_relative_to_root_if_not_exists(dir_relative_to_root):
211     """
212
213     :param dir_relative_to_root: A relative directory as
214     seen from the root dir of this project.
215
216     """
217     if os.path.exists(dir_relative_to_root):
218         # Remove directory and its content.
219         shutil.rmtree(dir_relative_to_root)

```

---

## O Appendix src/export\_data/helper\_tex\_editing.py

```
1  """Helper that modifies LaTeX file to allow automatically including
   ↪ Pyathon
2  code as appendices."""
3  import os
4
5  from src.export_data.helper_dir_file_edit import (
6      append_line_to_file,
7      append_lines_to_file,
8      convert_filepath_to_filepath_from_root,
9      delete_file_if_exists,
10     get_filename_from_dir,
11 )
12
13
14 def code_filepath_to_tex_appendix_filename(
15     filename, _, is_project_code, is_export_code
16 ):
17     """
18
19     :param filename:
20     :param from_root:
21     :param is_project_code:
22     :param is_export_code:
23
24     """
25
26     # TODO: Include assert to verify filename ends at .py.
27     # TODO: Include assert to verify filename doesn't end at .py
   ↪ anymore.
28     filename_without_extension = os.path.splitext(filename)[0]
29
30     # Create appendix filename identifier segment
31     verify_input_code_type(is_export_code, is_project_code)
32     if is_project_code:
33         identifier = "Auto-generated-project-code-appendix-"
34     elif is_export_code:
35         identifier = "Auto-generated-export-code-appendix-"
36
37     appendix_filename = f"{identifier}{filename_without_extension}"
38     return appendix_filename
39
40
41 def verify_input_code_type(is_export_code, is_project_code):
42     """
43
44     :param is_export_code:
45     :param is_project_code:
46
47     """
48     # Create appendix filename identifier segment
49     if is_project_code and is_export_code:
50         raise Exception(
51             "Error, a file can't be both project code, and export
   ↪ code at"
52             + " same time."
53         )
54     if not is_project_code and not is_export_code:
55         raise Exception(
56             "Error, don't know what to do with files that are neither
   ↪ project"
```

```

57         + " code, nor export code."
58     )
59
60
61 def tex_appendix_filename_to_inclusion_command(appendix_filename,
62     ↪ from_root):
63     """
64     :param appendix_filename:
65     :param from_root:
66
67     """
68     # Create full appendix filename.
69     if from_root:
70         # Generate latex inclusion command for latex compilation from
71         ↪ root dir.
72         appendix_inclusion_command = (
73             f"\\input{{latex/Appendices/{appendix_filename}.tex}} \\
74             ↪ newpage"
75         )
76         # \input{latex/Appendices/Auto_generated_py_App8.tex} \
77         ↪ newpage
78     else:
79         # \input{Appendices/Auto_generated_py_App8.tex} \newpage
80         appendix_inclusion_command = (
81             f"\\input{{Appendices/{appendix_filename}.tex}} \\newpage
82             ↪ "
83         )
84     return appendix_inclusion_command
85
86
87 def create_appendix_filecontent(
88     latex_object_name, filename, filepath_from_root, from_root
89 ):
90     """
91     :param latex_object_name:
92     :param filename:
93     :param filepath_from_root:
94     :param from_root:
95
96     """
97     # Latex titles should escape underscores.
98     filepath_from_root_without_underscores = filepath_from_root.
99     ↪ replace(
100         "-", r"\-"
101     )
102     lines = []
103     lines.append(
104         rf"\{latex_object_name}{{Appendix "
105         + rf"\{filepath_from_root_without_underscores}}}"
106         + rf"\label{{app:{filename}}}"
107     )
108     if from_root:
109         lines.append(rf"\pythonexternal{{latex/..{filepath_from_root
110         ↪ }}}}")
111     else:
112         lines.append(rf"\pythonexternal{{latex/..{filepath_from_root
113         ↪ }}}}")
114     return lines

```

```

111 def create_appendix_manager_files(hd):
112     """
113
114     :param hd:
115
116     """
117     # Verify target directory exists.
118     if not os.path.exists(hd.appendix_dir_from_root):
119         raise Exception(
120             "Error, the Appendices directory was not found at:"
121             + f"{hd.appendix_dir_from_root}"
122         )
123
124     # Delete appendix manager files.
125     list(
126         map(
127             lambda x: delete_file_if_exists(f"{hd.
128                 ↪ appendix_dir_from_root}{x}"),
129             hd.automatic_appendices_manager_filenames,
130         )
131     )
132
133     # Create new appendix_manager_files
134     list(
135         map(
136             # pylint: disable=R1732
137             lambda x: open(
138                 f"{hd.appendix_dir_from_root}{x}", "a", encoding="utf
139                 ↪ _8"
140             ),
141             hd.automatic_appendices_manager_filenames,
142         )
143     )
144
145     # Ensure manual appendix_manager_files are created.
146     list(
147         map(
148             # pylint: disable=R1732
149             lambda x: open(
150                 f"{hd.appendix_dir_from_root}{x}", "a", encoding="utf
151                 ↪ _8"
152             ),
153             hd.manual_appendices_manager_filenames,
154         )
155     )
156
157     # pylint: disable=R0913
158     def create_appendix_file(
159         hd,
160         filename,
161         filepath_from_root,
162         latex_object_name,
163         is_export_code,
164         is_project_code,
165     ):
166         """
167
168         :param hd:
169         :param filename:
170         :param filepath_from_root:
171         :param latex_object_name:

```

```

170 :param is_export_code:
171 :param is_project_code:
172
173 """
174 verify_input_code_type(is_export_code, is_project_code)
175 filename_without_extension = os.path.splitext(filename)[0]
176 if is_project_code:
177     # Create the appendix for the case the latex is compiled from
178     ↪ root.
179     appendix_filepath = (
180         f"{hd.appendix_dir_from_root}/Auto_generated_proj"
181         + f"ect_code_appendix_{filename_without_extension}.tex"
182     )
183
184     # Append latex_filepath to appendix manager.
185     # append_lines_to_file(
186     #     f"{hd.appendix_dir_from_root}{hd.
187     ↪ project_code_appendices_filename}",
188     #     [tex_appendix_filepath_to_inclusion_command(
189     ↪ appendix_filepath)],
190     # )
191
192     # Get Appendix .tex content.
193     appendix_lines_from_root = create_appendix_filecontent(
194         latex_object_name, filename, filepath_from_root, True
195     )
196
197     # Write appendix to .tex file.
198     append_lines_to_file(appendix_filepath,
199     ↪ appendix_lines_from_root)
200 elif is_export_code:
201     # Create the appendix for the case the latex is compiled from
202     ↪ root.
203     appendix_filepath = (
204         f"{hd.appendix_dir_from_root}/Auto_generated_export"
205         + f"_code_appendix_{filename_without_extension}.tex"
206     )
207
208     # Append latex_filepath to appendix manager.
209     # append_lines_to_file(
210     #     f"{hd.appendix_dir_from_root}{hd.
211     ↪ export_code_appendices_filename}",
212     #     [tex_appendix_filepath_to_inclusion_command(
213     ↪ appendix_filepath)],
214     # )
215
216     # Get Appendix .tex content.
217     appendix_lines_from_root = create_appendix_filecontent(
218         latex_object_name, filename, filepath_from_root, True
219     )
220
221     # Write appendix to .tex file.
222     append_lines_to_file(appendix_filepath,
223     ↪ appendix_lines_from_root)
224 # TODO: verify files exist
225
226 def export_python_project_code(
227     hd, normalised_root_dir, python_project_code_filepaths
228 ):
229     """

```

```

224 :param hd:
225 :param normalised_root_dir:
226 :param python_project_code_filepaths:
227
228 """
229 is_project_code = True
230 is_export_code = False
231 from_root = False
232 for filepath in python_project_code_filepaths:
233     create_appendices(
234         hd,
235         filepath,
236         normalised_root_dir,
237         from_root,
238         is_export_code,
239         is_project_code,
240     )
241     create_appendices(
242         hd,
243         filepath,
244         normalised_root_dir,
245         True,
246         is_export_code,
247         is_project_code,
248     )
249
250
251 def export_python_export_code(
252     hd, normalised_root_dir, python_export_code_filepaths
253 ):
254     """
255
256     :param hd:
257     :param normalised_root_dir:
258     :param python_export_code_filepaths:
259
260     """
261     is_project_code = False
262     is_export_code = True
263     from_root = False
264     for filepath in python_export_code_filepaths:
265         create_appendices(
266             hd,
267             filepath,
268             normalised_root_dir,
269             from_root,
270             is_export_code,
271             is_project_code,
272         )
273         create_appendices(
274             hd,
275             filepath,
276             normalised_root_dir,
277             True,
278             is_export_code,
279             is_project_code,
280         )
281
282
283 # pylint: disable=R0913
284 def create_appendices(
285     hd,

```



```

286     filepath,
287     normalised_root_dir,
288     from_root,
289     is_export_code,
290     is_project_code,
291 ):
292     """
293
294     :param hd:
295     :param filepath:
296     :param normalised_root_dir:
297     :param from_root:
298     :param is_export_code:
299     :param is_project_code:
300
301     """
302     # Get the filepath of a python file from the root dir of this
303     ↪ project.
304     filepath_from_root = convert_filepath_to_filepath_from_root(
305         filepath, normalised_root_dir
306     )
307     print(f"from_root={from_root}, filepath_from_root={
308         ↪ filepath_from_root}")
309
310     # Get the filename of a python filepath
311     filename = get_filename_from_dir(filepath)
312
313     # Get the filename for a latex appendix from a python filename.
314     appendix_filename = code_filepath_to_tex_appendix_filename(
315         filename, from_root, is_project_code, is_export_code
316     )
317
318     # Command to include the appendix in the appendices manager.
319     appendix_inclusion_command =
320     ↪ tex_appendix_filename_to_inclusion_command(
321         appendix_filename, from_root
322     )
323
324     append_appendix_to_appendix_managers(
325         appendix_inclusion_command,
326         from_root,
327         hd,
328         is_export_code,
329         is_project_code,
330     )
331
332     # Create the appendix .tex file.
333     # TODO: move "section" to hardcoded.
334     if from_root: # Appendix only contains files readable from root.
335         create_appendix_file(
336             hd,
337             filename,
338             filepath_from_root,
339             "section",
340             is_export_code,
341             is_project_code,
342         )
343
344     def append_appendix_to_appendix_managers(
345         appendix_inclusion_command, from_root, hd, is_export_code,
346         ↪ is_project_code

```

```

344 ):
345     """
346
347     :param appendix_inclusion_command:
348     :param from_root:
349     :param hd:
350     :param is_export_code:
351     :param is_project_code:
352
353     """
354     # Append the appendix .tex file to the appendix manager.
355     if is_project_code:
356         if from_root:
357             # print('from_root={from_root}Append to:
358             # +f"{hd.project_code_appendices_filename_from_root}')
359             append_line_to_file(
360                 f"{hd.appendix_dir_from_root}"
361                 + f"{hd.project_code_appendices_filename_from_root}",
362                 appendix_inclusion_command,
363             )
364         else:
365             # print(f'from_root={from_root}Append to:"
366             # +f"{hd.project_code_appendices_filename}')
367             append_line_to_file(
368                 f"{hd.appendix_dir_from_root}"
369                 + f"{hd.project_code_appendices_filename}",
370                 appendix_inclusion_command,
371             )
372
373     if is_export_code:
374         if from_root:
375             append_line_to_file(
376                 f"{hd.appendix_dir_from_root}"
377                 + f"{hd.export_code_appendices_filename_from_root}",
378                 appendix_inclusion_command,
379             )
380         else:
381             append_line_to_file(
382                 f"{hd.appendix_dir_from_root}"
383                 + f"{hd.export_code_appendices_filename}",
384                 appendix_inclusion_command,
385             )

```

---

## P Appendix src/export\_data/helper\_tex\_reading.py

---

```
1  """Helper script to parse Latex files, to support including LaTeX
2  appendices."""
3  from src.export_data.helper_dir_file_edit import file_contains
4
5
6  def verify_latex_supports_auto_generated_appendices(
7      ↪ path_to_main_latex_file):
8      """Ensures the Latex file supports including automatically
9      ↪ appending code
10     as appendices.
11
12     :param path_to_main_latex_file:
13     """
14     # TODO: change verification to complete tex block(s) for
15     ↪ appendices.
16     # TODO: Also verify related boolean and if statement creations.
17     determining_overleaf_home_line = (
18         r"\def\overleafhome{/tmp}% change as appropriate"
19     )
20     begin_apendices_line = "\\begin{appendices}"
21     print(f"determining_overleaf_home_line={
22         ↪ determining_overleaf_home_line}")
23     print(f"begin_apendices_line={begin_apendices_line}")
24
25     if not file_contains(
26         path_to_main_latex_file, determining_overleaf_home_line
27     ):
28         raise Exception(
29             f"Error, {path_to_main_latex_file} does not contain:\n\n"
30             + f"{determining_overleaf_home_line}\n\n so this Python
31             ↪ code "
32             + "cannot export the code as latex appendices."
33         )
34
35     if not file_contains(
36         path_to_main_latex_file, determining_overleaf_home_line
37     ):
38         raise Exception(
39             f"Error, {path_to_main_latex_file} does not contain:\n\n"
40             + f"{begin_apendices_line}\n\n so this Python code cannot
41             ↪ export"
42             + "the code as latex appendices."
43         )
44
45     )
```

---

## Q Appendix src/export\_data/latex\_compile.py

---

```
1  """Compiles the latex report using the compile script."""
2
3
4  from src.export_data.helper_bash_commands import run_a_bash_command
5
6
7  def compile_latex(await_compilation, verbose):
8      """Compiles the LaTeX report of this project using its compile
9          ↪ script.
10
11      :param await_compilation: Make python wait until the PlantUML
12          ↪ compilation
13      is completed.
14      :param project_name: The name of the project that is being
15          ↪ executed/ran.
16      :param verbose: True, ensures compilation output is printed to
17          ↪ terminal,
18      False means compilation is silent.
19
20      Returns:
21          Nothing.
22
23      Raises:
24          Nothing.
25      """
26
27      # Ensure compile script is runnable.
28      bash_make_compile_script_runnable_command = (
29          "chmod +x latex/compile_script.sh"
30      )
31      run_a_bash_command(
32          await_compilation, bash_make_compile_script_runnable_command,
33          ↪ verbose
34      )
35
36      # Run latex compilation script to compile latex project.
37      bash_compilation_command = "latex/compile_script.sh"
38      run_a_bash_command(await_compilation, bash_compilation_command,
39          ↪ verbose)
40      print(f"ran:{bash_compilation_command}")
```

---

## R Appendix src/export\_data/latex\_export\_code.py

```
1  """Exports code to latex appendices."""
2  import os
3
4  from src.export_data.helper_dir_file_edit import (
5      get_all_files_in_dir_and_child_dirs,
6      get_filepaths_in_dir,
7      remove_all_auto_generated_appendices,
8      sort_filepaths_by_filename,
9  )
10 from src.export_data.helper_tex_editing import (
11     create_appendix_manager_files,
12     export_python_export_code,
13     export_python_project_code,
14 )
15 from src.export_data.helper_tex_reading import (
16     verify_latex_supports_auto_generated_appendices,
17 )
18
19
20 def export_code_to_latex(hd, include_export_code):
21     """This function exports the python files and compiled pdfs of
22         ↪ jupyter
23     notebooks into the latex of the same project number. First it
24         ↪ scans which
25     appendices (without code, without notebooks) are already manually
26         ↪ included
27     in the main latex code. Next, all appendices that contain the
28         ↪ python code
29     are either found or created in the following order: First, the
30         ↪ __main__.py
31     file is included, followed by the main.py file, followed by all
32         ↪ python code
33     files in alphabetic order. After this, all the pdfs of the
34         ↪ compiled
35     notebooks are added in alphabetic order of filename. This order
36         ↪ of
37     appendices is overwritten in the main tex file.
38
39     :param main_latex_filename: Name of the main latex document of
40         ↪ this project
41     number.
42     :param project_name: The name of the project that is being
43         ↪ executed/ran.
44     The number indicating which project this code pertains to.
45     """
46     script_dir = get_script_dir()
47     latex_dir = script_dir + "../..../latex/"
48     path_to_main_latex_file = f"{latex_dir}{hd.main_latex_filename}"
49     root_dir = script_dir + "../..../"
50     normalised_root_dir = os.path.normpath(root_dir)
51     src_dir = script_dir + "../.."
52
53     # Verify the latex file supports auto-generated python appendices
54     ↪ .
55     verify_latex_supports_auto_generated_appendices(
56         ↪ path_to_main_latex_file)
57
58     # Get paths to files containing project python code.
59     python_project_code_filepaths = get_filepaths_in_dir(
60         "py", src_dir, ["__init__.py"]
61     )
```

```

49 )
50
51 get_compiled_notebook_paths(script_dir)
52 print(f"python_project_code_filepaths={
53     ↪ python_project_code_filepaths}")
54
55 # Get paths to the files containing the latex export code
56 if include_export_code:
57     python_export_code_filepaths = get_filepaths_in_dir(
58         "py", script_dir, ["__init__.py"]
59     )
60
61 remove_all_auto_generated_appendices(hd)
62
63 # Create appendix file # ensure they are also deleted at the
64 ↪ start of every
65 # run.
66 create_appendix_manager_files(hd)
67
68 # TODO: Sort main files.
69 export_python_project_code(
70     hd,
71     normalised_root_dir,
72     sort_filepaths_by_filename(python_project_code_filepaths),
73 )
74 if include_export_code:
75     export_python_export_code(
76         hd,
77         normalised_root_dir,
78         sort_filepaths_by_filename(python_export_code_filepaths),
79     )
80
81 def get_compiled_notebook_paths(script_dir):
82     """Returns the list of jupyter notebook filepaths that were
83     ↪ compiled
84     successfully and that are included in the same dias this script (
85     ↪ the src
86     directory).
87
88     :param script_dir: absolute path of this file.
89     """
90     notebook_filepaths = get_all_files_in_dir_and_child_dirs(
91         ".ipynb", script_dir
92     )
93     compiled_notebook_filepaths = []
94
95     # check if the jupyter notebooks were compiled
96     for notebook_filepath in notebook_filepaths:
97         # swap file extension
98         notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf
99         ↪ ")
100
101         # check if file exists
102         if os.path.isfile(notebook_filepath):
103             compiled_notebook_filepaths.append(notebook_filepath)
104     return compiled_notebook_filepaths
105
106 def get_script_dir():

```

```
105     """returns the directory of this script regardless of from which
      ↪ level the
106     code is executed."""
107     return os.path.dirname(__file__)
```

---

## S Appendix src/export\_data/plantuml\_compile.py

```
1  """This script automatically compiles the text files representing a
   ↪ PlantUML.
2
3  # diagram into an actual figure.
4
5  # To compile locally manually:
6  # pip install plantuml
7  # export PLANTUML_LIMIT_SIZE=8192
8  # java -jar plantuml.jar -verbose sequenceDiagram.txt
9  """
10
11 import os
12 import subprocess # nsec
13 from os.path import abspath
14
15 from src.export_data.helper_dir_file_edit import (
16     get_dir_filelist_based_on_extension,
17 )
18 from src.export_data.plantuml_get_package import got_java_file
19
20
21 def compile_diagrams_in_dir_relative_to_root(
22     await_compilation,
23     extension,
24     jar_path_relative_from_root,
25     input_dir_relative_to_root,
26     verbose,
27 ):
28     """Loops through the files in a directory and exports them to the
   ↪ latex.
29
30     /Images directory.
31
32     Args:
33     :param await_compilation: Make python wait until the PlantUML
   ↪ compilation
34     is completed. param extension: The filetype of the text file that
   ↪ is
35     converted to image.
36     :param jar_path_relative_from_root: The path as seen from root
   ↪ towards the
37     PlantUML .jar file that compiles .uml files to .png files.
38     :param verbose: True, ensures compilation output is printed to
   ↪ terminal,
39     False means compilation is silent.
40     :param extension: The file extension that is used/searched in
   ↪ this
41     function.
42     :param input_dir_relative_to_root: The directory as seen from
   ↪ root
43     containing files that are modified in this function.
44
45     Returns:
46         Nothing
47
48     Raises:
49         Nothing
50     """
51     # Verify the PlantUML .jar file is gotten.
52     got_java_file(jar_path_relative_from_root)
```



```

53 diagram_text_filenames = get_dir_filelist_based_on_extension(
54     input_dir_relative_to_root, extension
55 )
56
57 for diagram_text_filename in diagram_text_filenames:
58     diagram_text_filepath_relative_from_root = (
59         f"{input_dir_relative_to_root}/{diagram_text_filename}"
60     )
61
62     execute_diagram_compilation_command(
63         await_compilation,
64         jar_path_relative_from_root,
65         diagram_text_filepath_relative_from_root,
66         verbose,
67     )
68
69
70 def execute_diagram_compilation_command(
71     await_compilation,
72     jar_path_relative_from_root,
73     relative_filepath_from_root,
74     verbose,
75 ):
76     """Compiles a .uml/text file containing a PlantUML diagram to a .
77         ↪ png image
78         using the PlantUML .jar file.
79
80     Args:
81         :param await_compilation: Make python wait until the PlantUML
82             ↪ compilation
83             is completed. param extension: The filetype of the text file that
84             ↪ is
85             converted to image.
86         :param jar_path_relative_from_root: The path as seen from root
87             ↪ towards the
88             PlantUML .jar file that compiles .uml files to .png files.
89         :param verbose: True, ensures compilation output is printed to
90             ↪ terminal,
91             False means compilation is silent.
92         :param input_dir_relative_to_root: The directory as seen from
93             ↪ root
94             containing files that are modified in this function.
95     Returns:
96         Nothing
97
98     Raises:
99         Nothing
100     """
101     # Verify the files required for compilation exist, and convert
102     ↪ the paths
103     # into absolute filepaths.
104     (
105         abs_diagram_filepath,
106         abs_jar_path,
107     ) = assert_diagram_compilation_requirements(
108         jar_path_relative_from_root,
109         relative_filepath_from_root,
110     )
111
112     # Generate command to compile the PlantUML diagram locally.
113     print(

```

```

108         f"abs_jar_path={abs_jar_path},"
109         + f" abs_diagram_filepath={abs_diagram_filepath}\n\n"
110     )
111     bash_diagram_compilation_command = (
112         f"java -jar {abs_jar_path} -verbose {abs_diagram_filepath}"
113     )
114     print(
115         f"bash_diagram_compilation_command={
116             ↪ bash_diagram_compilation_command}"
117     )
118     # Generate global variable specifying max image width in pixels,
119     ↪ in the
120     # shell that compiles.
121     os.environ["PLANTUML_LIMIT_SIZE"] = "16192"
122
123     # Perform PlantUML compilation locally.
124     if await_compilation:
125         if verbose:
126             subprocess.call(
127                 bash_diagram_compilation_command, shell=True # nsec
128                 ↪ B602
129             )
130         else:
131             subprocess.call(
132                 bash_diagram_compilation_command,
133                 shell=True, # nsec
134                 stderr=subprocess.DEVNULL,
135                 stdout=subprocess.DEVNULL,
136             ) # nsec
137     else:
138         if verbose:
139             with subprocess.Popen(
140                 bash_diagram_compilation_command,
141                 shell=True, # nsec
142                 stdout=subprocess.PIPE,
143                 stderr=subprocess.STDOUT,
144             ) as process:
145                 get_output_of_bash_command(process)
146         else:
147             with subprocess.Popen(
148                 bash_diagram_compilation_command,
149                 shell=True, # nsec
150                 stderr=subprocess.DEVNULL,
151                 stdout=subprocess.DEVNULL,
152             ) as process:
153                 get_output_of_bash_command(process)
154
155     def get_output_of_bash_command(process, verbose=True):
156         """Returns the output of a bash command."""
157         stdout, stderr = process.communicate()
158         result = stdout.decode("utf-8")
159         if verbose:
160             print(f"result={result}")
161             print(f"stderr={stderr}")
162         return result
163
164     def assert_diagram_compilation_requirements(
165         jar_path_relative_from_root,
166         relative_filepath_from_root,
167     ):

```

```

167 """Asserts that the PlantUML .jar file used for compilation
    ↳ exists, and
168 that the diagram file with the .uml content for the diagram
    ↳ exists. Throws
169 an error if either of two is missing.
170
171 :param relative_filepath_from_root: Relative filepath as seen
    ↳ from root of
172 file that is used in this function.
173 :param output_dir_from_root: Relative directory as seen from root
    ↳ , to which
174 files are outputted.
175 :param jar_path_relative_from_root: The path as seen from root
    ↳ towards the
176 PlantUML .jar file that compiles .uml files to .png files.
177
178 Returns:
179     Nothing
180
181 Raises:
182     Exception if PlantUML .jar file used to compile the .uml to .
    ↳ png files
183     is missing.
184     Exception if the file with the .uml content is missing.
185 """
186 abs_diagram_filepath = abspath(relative_filepath_from_root)
187 abs_jar_path = abspath(jar_path_relative_from_root)
188 if os.path.isfile(abs_diagram_filepath):
189     if os.path.isfile(abs_jar_path):
190         return abs_diagram_filepath, abs_jar_path
191     raise Exception(
192         f"The input diagram file:{abs_diagram_filepath} doesn't
    ↳ exist."
193     )
194 raise Exception(f"The input jar file:{abs_jar_path} does not
    ↳ exist.")

```

---

## T Appendix src/export\_data/plantuml\_generate.py

```
1  """This script generates PlantUML diagrams and outputs them as .uml
   ↪ files."""
2
3  import os
4  from os.path import abspath
5
6  from src.export_data.helper_dir_file_edit import (
7      create_dir_relative_to_root_if_not_exists,
8      dir_relative_to_root_exists,
9  )
10
11
12  def generate_all_dynamic_diagrams(output_dir_relative_to_root):
13      """Manages the generation of all the diagrams created in this
   ↪ file.
14
15      Args:
16      :param output_dir_relative_to_root: Relative path as seen from
   ↪ the root dir
17      of this project, to which modified files are outputted.
18
19      Returns:
20          Nothing
21
22      Raises:
23          """
24      # Create a example Gantt output file.
25      filename_one, lines_one = create_trivial_gantt("trivial_gantt.uml
   ↪ ")
26      output_diagram_text_file(
27          filename_one, lines_one, output_dir_relative_to_root
28      )
29
30      # Create another example Gantt output file.
31      filename_two, lines_two = create_trivial_gantt("
   ↪ another_trivial_gantt.uml")
32      output_diagram_text_file(
33          filename_two,
34          lines_two,
35          output_dir_relative_to_root,
36      )
37
38
39  def output_diagram_text_file(filename, lines,
   ↪ output_dir_relative_to_root):
40      """Gets the filename and lines of an PlantUML diagram, and writes
   ↪ these to
41      a file at the relative output path.
42
43      Args:
44      :param filename: The filename of the PlantUML Gantt file that is
   ↪ being
45      created.
46      :param lines: The lines of the Gantt chart PlantUML code that is
   ↪ being
47      written to file.
48      :param output_dir_relative_to_root: Relative path as seen from
   ↪ the root
49      dir of this project, to which modified files are outputted.
50
```

```

51 Returns:
52     Nothing
53
54 Raises:
55     Exception if input file does not exist.
56     """
57 abs_filepath = abspath(f"{output_dir_relative_to_root}/{filename}"
58     ↪ ")
59
60 # Ensure output directory is created.
61 create_dir_relative_to_root_if_not_exists(
62     ↪ output_dir_relative_to_root)
63 if not dir_relative_to_root_exists(output_dir_relative_to_root):
64     raise Exception(
65         "Error, the output directory relative to root:"
66         + f"{output_dir_relative_to_root} does not exist."
67     )
68
69 # Delete output file if it already exists.
70 if os.path.exists(abs_filepath):
71     os.remove(abs_filepath)
72
73 # Write lines to file.
74 with open(abs_filepath, "w", encoding="utf-8") as f:
75     for line in lines:
76         f.write(line)
77     f.close()
78
79 # Assert output file exists.
80 if not os.path.isfile(abs_filepath):
81     raise Exception(f"The input file:{abs_filepath} does not
82     ↪ exist.")
83
84 def create_trivial_gantt(filename):
85     """Creates a trivial Gantt chart.
86
87     Args:
88         :param filename: The filename of the PlantUML diagram file that
89         ↪ is being
90         created.
91
92     Returns:
93         The filename of the PlantUML diagram, and the lines of the
94         ↪ uml content
95         of the diagram
96
97     Raises:
98         Nothing
99     """
100     lines = []
101     lines.append("@startuml\n")
102     lines.append("[Prototype design] lasts 15 days\n")
103     lines.append("[Test prototype] lasts 10 days\n")
104     lines.append("\n")
105     lines.append("Project starts 2020-07-01\n")
106     lines.append("[Prototype design] starts 2020-07-01\n")
107     lines.append("[Test prototype] starts 2020-07-16\n")
108     lines.append("@enduml\n")
109     return filename, lines

```

```

108 def create_another_trivial_gantt(filename):
109     """Creates a trivial Gantt chart.
110
111     :param filename: The filename of the PlantUML Gantt file that is
112         ↪ being
113         created.
114
115     Returns:
116         The filename of the PlantUML diagram, and the lines of the
117         ↪ uml content
118         of the diagram
119
120     Raises:
121         Nothing
122     """
123     lines = []
124     lines.append("@startuml\n")
125     lines.append("[EXAMPLE SENTENCE] lasts 15 days\n")
126     lines.append("[Test prototype] lasts 10 days\n")
127     lines.append("\n")
128     lines.append("Project starts 2022-07-01\n")
129     lines.append("[Prototype design] starts 2022-07-01\n")
130     lines.append("[Test prototype] starts 2022-07-16\n")
131     lines.append("@enduml\n")
132
133     return filename, lines

```

---

## U Appendix src/export\_data/plantuml\_get\_package.py

```
1 """Downloads the PlantUML package if it does not yet exist."""
2 import os
3 import subprocess # nsec
4
5 import requests
6
7
8 def check_if_java_file_exists(relative_filepath):
9     """Safe check to see if file exists or not.
10
11     Args:
12     :param relative_filepath: Path as seen from root towards a file.
13
14     Returns:
15         True if a file exists.
16         False if a file does not exists.
17
18     Raises:
19         Nothing
20     """
21
22     return os.path.isfile(relative_filepath)
23
24
25 def got_java_file(relative_filepath):
26     """Asserts if PlantUML .jar file exists. Tries to download is one
27     ↪ time if
28     it does not exist at the start of the function.
29
30     Args:
31     :param relative_filepath: Path as seen from root towards a file.
32
33     Returns:
34         True if a file exists.
35
36     Raises:
37         Exception if the PlantUML .jar file does not exist after
38         ↪ downloading
39         it.
40     """
41     # Check if the jar file exists, curl it if not.
42     if not check_if_java_file_exists(relative_filepath):
43         # The java file is not found, curl it
44         request_file(
45             "https://sourceforge.net/projects/"
46             + "plantuml/files/plantuml.jar/download",
47             relative_filepath,
48         )
49     # Check if the jar file exists after curling it. Raise Exception
50     ↪ if it is
51     # not found after curling.
52     if not check_if_java_file_exists(relative_filepath):
53         raise Exception(f"File:{relative_filepath} is not accessible"
54             ↪ )
55     print("Got the PlangUML Java file.")
56     return True
57
58
59 def request_file(url, output_filepath):
60     """Downloads a file or file content.
```

```

57
58 Args:
59 :param url: Url towards a file that will be downloaded.
60 :param relative_filepath: The path as seen from the root of this
    ↪ directory,
61 in which files are outputted.
62
63 Returns:
64     Nothing
65
66 Raises:
67     Nothing
68     """
69
70 # Request the file in the url
71 response = requests.get(url, timeout=20) # seconds
72 with open(output_filepath, "wb", encoding="utf-8") as f:
73     f.write(response.content)
74
75
76 def run_bash_command(bashCommand):
77     """Unused method. TODO: verify it is unused and delete it.
78
79     :param bashCommand: A string containing a bash command that
80     can be executed.
81     """
82     # Verbose call.
83     # subprocess.Popen(bashCommand, shell=True)
84     # Silent call.
85     # subprocess.Popen(bashCommand, shell=True, stderr=subprocess.
    ↪ DEVNULL,
86     # stdout=subprocess.DEVNULL)
87
88     # Await completion:
89     # Verbose call.
90     subprocess.call(bashCommand, shell=True) # nsec
91     # Silent call.
92     # subprocess.call(bashCommand, shell=True, stderr=subprocess.
    ↪ DEVNULL,
93     # stdout=subprocess.DEVNULL)

```

---



## V Appendix src/export\_data/plantuml\_to\_tex.py

```
1  """Exports the generated PlantUML diagrams to the latex Images
   ↳ directory."""
2  import os.path
3  import shutil
4
5  from src.export_data.helper_dir_file_edit import (
6      create_dir_relative_to_root_if_not_exists,
7      get_dir_filelist_based_on_extension,
8  )
9
10
11 def export_diagrams_to_latex(
12     input_dir_relative_to_root, extension,
13     ↳ output_dir_relative_to_root
14 ):
15     """Loops through the files in a directory and exports them to the
16     ↳ latex.
17
18     /Images directory.
19
20     :param dir: The directory in which the Gantt charts are being
21     ↳ searched.
22     :param extension: The file extension that is used/searched in
23     ↳ this
24     function. The filetypes that are being exported.
25     :param input_dir_relative_to_root: Relative path as seen from the
26     ↳ root dir
27     of this project, containing files that modified in this function.
28     :param output_dir_relative_to_root: Relative path as seen from
29     ↳ the root dir
30     of this project, to which modified files are outputted.
31     """
32     diagram_filenames = get_dir_filelist_based_on_extension(
33         input_dir_relative_to_root, extension
34     )
35     if len(diagram_filenames) > 0:
36         # Ensure output directory is created.
37         create_dir_relative_to_root_if_not_exists(
38             ↳ output_dir_relative_to_root)
39
40     for diagram_filename in diagram_filenames:
41         diagram_filepath_relative_from_root = (
42             f"{input_dir_relative_to_root}/{diagram_filename}"
43         )
44
45         export_gantt_to_latex(
46             diagram_filepath_relative_from_root,
47             ↳ output_dir_relative_to_root
48         )
49
50 def export_gantt_to_latex(
51     relative_filepath_from_root, output_dir_relative_to_root
52 ):
53     """Takes an input filepath and an output directory as input and
54     ↳ copies the
55     file towards the output directory.
56
57     :param relative_filepath_from_root: param
58     ↳ output_dir_relative_to_root:
```

```

50 :param output_dir_relative_to_root: Relative path as seen from
51     ↳ the root dir
52 of this project, to which modified files are outputted.
53
54 Returns:
55     Nothing.
56
57 Raises:
58     Exception if the output directory does not exist.
59     Exception if the input file is not found.
60 """
61 if os.path.isfile(relative_filepath_from_root):
62     if os.path.isdir(output_dir_relative_to_root):
63         shutil.copy(
64             relative_filepath_from_root,
65             ↳ output_dir_relative_to_root
66         )
67     else:
68         raise Exception(
69             f"The output directory:{output_dir_relative_to_root}
70             ↳ does"
71             + " not exist."
72         )
73 else:
74     raise Exception(
75         f"The input file:{relative_filepath_from_root} does not
76         ↳ exist."
77     )

```

---