Roadmap: TruCol

A decentralised collaboration protocol for test-driven development

September 6, 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 4. (the Python models themselves are included as appendices.

The results of these models are presented in section 5. 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 6. Next, the results and sensitivity of the models are discussed in section 7 and a conclusion is provided in section 8.

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 models are somewhat of an abstract/subjective task, three different approaches are used in an attempt to establish some reference material with respect to accuracy.

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 Markets

To use the Top Down Model, section 3.1 describes the TAM in which the TruCol company will operate. To this end, section 3.1.1 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 pharmaceutics market in section 3.1.2 to get some insight in their respective market sizes. Furthermore, section 3.1.3 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.

3.1 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 as 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.

3.1.1 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 optimalisation 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 optimalisation 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 McKinseys 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 optimalisation across the entire logistics industry.
- **asu-8:** Companies are willing to pay at least 1 % 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 4.1 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% 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% 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.

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

$$\begin{aligned} net-profit_{DHL+UPS+FedEx} &= 4.1+7.7+1.29 = 13.09 billion \\ \frac{net-profit_{global\_logistics}}{net-profit_{DHL+UPS+FedEx}} &= \frac{0.85}{0.15} \\ net-profit_{global\_logistics} &= net-profit_{DHL+UPS+FedEx}\frac{0.85}{0.15} \\ net-profit_{global\_logistics} &= \frac{13.09\cdot 0.85}{0.15} \\ net-profit_{global\_logistics} &= 74.2 billion \end{aligned} \tag{1}$$

Hence, if each of those companies in the logistics sector could increase their profits on average annually by .1% using algorithmic optimisation, and if they would use the TruCol protocol to do that, and if they would be willing to invest 1% 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 $74.2 \cdot 0.001 \cdot 0.01 = \$0.74 million$

3.1.2 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.

3.1.3 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.

4 Market Analysis Model

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.

4.1 Top Down Model

5 Results

5.1 Top Down Model

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

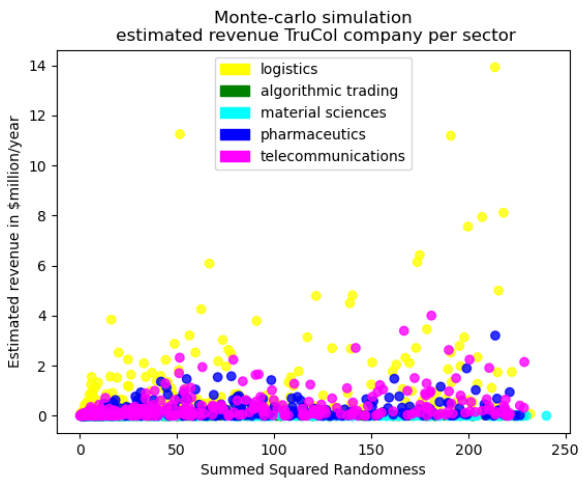

Figure 1: A scatterplot generated by a Monte-Carlo simulation to provide an impression on the estimated projected total addressable market per sector for the TruCol company.

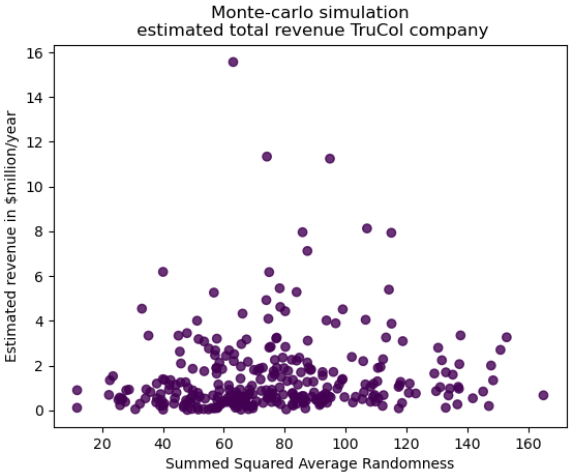

Figure 2: A scatterplot generated by a Monte-Carlo simulation to provide an estimate on the projected total addressable market for the TruCol company.

6 Sensitivity Analysis

6.1 Top Down

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

7 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.

8 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.

A Appendix src/Activity.py

```
"""Object that is used to store the activities that compose a Gantt chart."""
import copy


# pylint: disable=R0902
class Activity:
    """Used to create an activity in a Gantt chart."""

    # pylint: disable=R0913
    def __init__(
        self,
        description,
        duration,
        new_tag,
        colour=None,
        parent=None,
        starts_at_child_nr_start=None,
        starts_at_child_nr_end=None,
        font_size=None,
        hourly_wage=None,
        hours_per_day=None,
        min_parallel_workers=1,
        max_parallel_workers=1,
        milestone=None,
    ):
        self.parent = parent
        self.description = description
        self.duration = duration
        self.children = []
        self.font_size = font_size
        self.starts_at_child_nr_start = starts_at_child_nr_start
        self.starts_at_child_nr_end = starts_at_child_nr_end
        self.hourly_wage = hourly_wage
        self.min_parallel_workers = min_parallel_workers
        self.max_parallel_workers = max_parallel_workers
        self.milestone = milestone

        if hours_per_day is None:
            self.hours_per_day = 8
        else:
            self.hours_per_day = hours_per_day
        if self.parent is None:
            if colour is None:
                raise Exception("Parent activity needs a colour")
            self.colour = colour
            self.tag = []
        else:
            if (self.parent.hourly_wage is not None) and (hourly_wage is None):
                self.hourly_wage = self.parent.hourly_wage
            self.colour = parent.colour
            self.tag = copy.deepcopy(self.parent.tag)

        # create tag
        self.tag.append(new_tag)

    def add_children(self, children):
        """Stores the child activities in an activity.

```

```
        :param children:
        """
        self.children = children

    def get_tag(self):
        """Converts the tags of an activity into a string, with the
           ↪ activity
        indexes separated by underscores."""
        return "_".join([str(tag) for tag in self.tag])

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
           ↪  the
        computation.

        :param x:
        """
        return x + 2
```

B Appendix src/Cost_model.py

```python
"""List of cost model parameters."""

params = {
    "wages": {
        "blockchain_dev": 75 + 1,
        "front_end_dev": 40 + 1,
        "human_resources": 35 + 1,
    },
    # Bounties used to attract initial protocol users.
    "bounty_subsidising": 100000,
    "buffer": 100000,  # Buffer to take unknown costs into account.
    "daily_operational_costs": 200,
    "days_to_operational_break-even": 270,
}


# pylint: disable=R0902
class Cost_model:
    """Computes the total costs."""

    # pylint: disable=R0913
    def __init__(
        self,
        parent_costs: dict,
        the_params: dict,
    ):

        self.parent_costs = parent_costs
        self.operational_costs: int = (
            the_params["daily_operational_costs"]
            * the_params["days_to_operational_break-even"]
        )
        (
            self.total_costs,
            the_params["labour_costs"],
        ) = self.total_costs_to_dict(the_params)

        the_params["total_cost"] = self.compute_total_cost()
        the_params["non_labour_costs"] = (
            the_params["total_cost"] - the_params["labour_costs"]
        )

        dict_to_latex_table(
            "latex/Tables/total_costs_table.tex",
            self.total_costs,
            "Description",
            r"Cost [\euro]",
            "Total Expected Investment Costs",
        )

    def total_costs_to_dict(self, the_params):
        """Generates a dictionary including all expected project
           ↪ costs."""
        total_costs = {}
        labour_costs = 0

        # Get the order of the parent actities.
        order = []
        for key in self.parent_costs.keys():
            order.append(key)
```

```
        # The stored total costs are cumulative costs, so subtract the costs of
        # the previous parent to get the actual costs of the parent.
        for i, key in enumerate(order):
            if i == 0:
                total_costs[key.description] = self.parent_costs[key]
            else:
                total_costs[key.description] = (
                    self.parent_costs[key] - self.parent_costs[order[i - 1]]
                )
            labour_costs = labour_costs + total_costs[key.description]

        total_costs["Bounty Subsidising"] = the_params["bounty_subsidising"]
        total_costs["Buffer"] = the_params["buffer"]
        total_costs["Operational Costs"] = self.operational_costs
        return total_costs, labour_costs

    def compute_total_cost(self):
        """Generates the expected summed total costs."""
        total_cost = 0
        for value in self.total_costs.values():
            if isinstance(value, tuple):
                total_cost = total_cost + int(value[0])
            else:
                total_cost = total_cost + int(value)
        return total_cost


def dict_to_latex_table(
    filepath: str,
    the_params: dict,
    key_header: str,
    value_header: str,
    caption: str,
):
    """Writes a dict to a latex file.

    :param the_params: dict:
    :param key_header: str:
    :param value_header: str:
    :param caption: str:
    """
    tuples = dict_to_latex_tuples(the_params)

    with open(filepath, "w", encoding="utf-8") as f:
        backreturn = "\\\\\n" + " " * 4

        content = backreturn.join(
            [f"{_tuple[0]} & {_tuple[1]}" for _tuple in tuples]
        )

        f.write(
            f"""
\\begin{{longtable}}{{@{{}}cp{{.7\\textwidth}}@{{}}}}
    \\caption{{{caption}\\label{{table:nonlin}}}}\\\\
    \\toprule
    {{\\bfseries {key_header}}} & {{\\bfseries {value_header}}} \\\\
    \\midrule
    \\endfirsthead
```

```
    \\caption{{{caption} (continued)}}\\\\
    \\toprule
    \\multicolumn{{2}}{{l}}{{\\scriptsize\\emph{{\\ldots{{}}
       ↪ continued}}}}\\\\
    {{\\bfseries {key_header}}} & {{\\bfseries {value_header}}} \\\\
       ↪ \\midrule
    \\endhead
    \\multicolumn{{2}}{{r}}{{\\scriptsize\\emph{{to be continued\\
       ↪ ldots}}}}\\\\
    \\bottomrule
    \\endfoot
    \\bottomrule
    \\endlastfoot
    {content}\\\\
\\end{{longtable}}
    """.strip()
        )


def flatten_dict(some_dict: dict):
    """Flattens a dict that contains values and dicts.

    :param some_dict: dict:
    """
    flat_dict = {}
    # Flatten dict
    for key, value in some_dict.items():
        if isinstance(value, dict):
            for newKey, newValue in value.items():
                flat_dict[newKey] = newValue
        else:
            flat_dict[key] = value
    return flat_dict


def dict_to_latex_tuples(some_dict: dict):
    """Converts a dict to a list of key,value tuples without
       ↪ underscores.

    :param some_dict: dict:
    """
    flat_dict = flatten_dict(some_dict)
    tuples = []
    for key, value in flat_dict.items():
        if isinstance(key, str):
            key = key.replace("_", " ")
            key = key.replace("&", r"\&")
        if isinstance(value, str):
            value = value.replace("_", " ")
            value = value.replace("&", r"\&")
        tuples.append((key, value))
    return tuples
```

C Appendix src/Create_python_gantt.py

```
"""This file is used to create a Gantt chart using Python code. It is like a
PlantUML API. The reason to use Python instead of directly specifying a Gantt
in a .uml file, is because it is easier to update/modify the Gantt chart.

For example, if you want to add an activity somewhere, in the .uml you
have to manually update all the tag numbers to preserve the order. (And
do that again for the colour specifications etc.) That is tedious work.
In this Python API, you can simply say: start the activity new activity
after activity X, and make activity Y start at the end of the new
activity. Then it automatically updates all the tags, for all
properties, e.g. activity descriptions, colours, order etc.
"""
from src.Activity import Activity
from src.export_data.Milestone import Milestone


# pylint: disable=R0914
def create_python_gantt(
    wages: dict, milestone_font_size=None, start_date=None
):
    """Specifies the data for a Gantt chart."""
    # Create a list to store the parent activities
    parents = []
    date_milestones = [
        Milestone(
            description="Complete CI deployment",
            font_size=milestone_font_size,
            date=start_date,
        )
    ]
    # parent one
    protocol = Activity(
        description="Develop protocol",
        duration=120,
        new_tag=0,
        colour="Green",
        hourly_wage=0,
    )
    # children
    onchain = Activity(
        description="On-chain: Solidty+VRF",
        duration=60,
        new_tag=0,
        parent=protocol,
        hourly_wage=wages["blockchain_dev"],
    )
    git_tellor = Activity(
        description="Git integration: Tellor",
        duration=90,
        new_tag=1,
        parent=protocol,
        starts_at_child_nr_start=0,
        hourly_wage=wages["blockchain_dev"],
```

```
    )
    git_chainlink = Activity(
        description="Git integration: Chainlink",
        duration=90,
        new_tag=2,
        parent=protocol,
        starts_at_child_nr_start=0,
        hourly_wage=wages["blockchain_dev"],
        milestone=Milestone(
            description="Support all languages",
            font_size=milestone_font_size,
        ),
    )
    alt_chains = Activity(
        description="Alternative Chains",
        duration=90,
        new_tag=3,
        parent=protocol,
        starts_at_child_nr_start=0,
        hourly_wage=wages["blockchain_dev"],
    )

    # grandchildren
    ci = Activity(
        description="(Decentralised) Continuous integration",
        new_tag=0,
        duration=30,
        parent=git_chainlink,
    )
    # git_chainlink.add_children([ci])
    security = Activity(
        description="Security & Robustness",
        duration=60,
        new_tag=1,
        parent=git_chainlink,
    )
    # ci.add_children([security])

    # merge
    protocol.add_children(
        [onchain, git_tellor, git_chainlink, alt_chains, ci, security
           ↪ ]
    )
    parents.append(protocol)

    # parent_two
    platform_eco = Activity(
        description="Platform & ecosystem",
        duration=120,
        new_tag=1,
        colour="DarkOrchid",
        starts_at_child_nr_start=0,
        hourly_wage=wages["front_end_dev"],
    )
    # children
    website = Activity(
        description="Website", duration=50, new_tag=0, parent=
           ↪ platform_eco
    )
    marketing_platf = Activity(
        description="Marketing platform",
        duration=30,
```

```
        new_tag=1,
        parent=platform_eco,
    )
    bounties = Activity(
        description="Subsidize bounties",
        duration=10,
        new_tag=2,
        parent=platform_eco,
    )
    platform_buffer = Activity(
        description="Platform Planning Buffer",
        duration=30,
        new_tag=3,
        parent=platform_eco,
        milestone=Milestone(
            description="First Customer Usage",
            font_size=milestone_font_size,
            # TODO: remove date, make it place on the right position.
            date="2023-03-17",
        ),
    )

    # Grandchildren
    api = Activity(description="API", duration=50, new_tag=0, parent=
    ↪ website)
    gui = Activity(
        description="GUI",
        duration=50,
        new_tag=1,
        parent=website,
        starts_at_child_nr_start=0,
    )
    forum = Activity(
        description="Forum",
        duration=10,
        new_tag=2,
        parent=website,
        starts_at_child_nr_start=0,
    )
    website.add_children([api, gui, forum])

    platform_eco.add_children(
        [website, marketing_platf, bounties, platform_buffer]
    )
    parents.append(platform_eco)

    # parent_three
    company = Activity(
        description="Launch company",
        duration=150,
        new_tag=2,
        colour="Yellow",
        starts_at_child_nr_start=0,
        hourly_wage=wages["human_resources"],
    )
    # children
    partners = Activity(
        description="Qualitative partner research",
        duration=20,
        new_tag=0,
        parent=company,
    )
```

```
    organisation = Activity(
        description="Establish organisation",
        duration=80,
        new_tag=1,
        parent=company,
    )
    marketing_company = Activity(
        description="Marketing", duration=30, new_tag=2, parent=
            ↪ company
    )
    company_buffer = Activity(
        description="Organisation Planning Buffer ",
        duration=20,
        new_tag=3,
        parent=company,
        milestone=Milestone(
            description="Operational Break Even",
            font_size=milestone_font_size,
            # TODO: remove date, make it place on the right position.
            date="2023-06-09",
        ),
    )

    # Grandchildren
    auditing = Activity(
        description="Auditing", duration=10, new_tag=0, parent=
            ↪ organisation
    )
    hiring = Activity(
        description="Hiring", duration=20, new_tag=1, parent=
            ↪ organisation
    )
    administration = Activity(
        description="Administration",
        duration=10,
        new_tag=2,
        parent=organisation,
    )
    legal = Activity(
        description="Legal", duration=20, new_tag=3, parent=
            ↪ organisation
    )
    financial = Activity(
        description="Financial", duration=20, new_tag=4, parent=
            ↪ organisation
    )
    organisation.add_children(
        [auditing, hiring, administration, legal, financial]
    )

    company.add_children(
        [partners, organisation, marketing_company, company_buffer]
    )
    parents.append(company)

    # Create Milestones

    return parents, date_milestones


def addTwo(x):
    """adds two to the incoming integer and returns the result of the
```

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

D Appendix src/Gantt.py

```python
"""Creates the Gantt chart specified in create_python_gantt."""

from src.Create_python_gantt import create_python_gantt
from src.export_data.Milestone import (
    get_milestone_for_date,
    get_milestone_style,
    get_milestone_uml_line,
)


# pylint: disable=R0902
class Gantt:
    """Creates the Gantt chart specified in create_python_gantt."""

    def __init__(self, filepath: str, params: dict):
        self.start_line = "@startgantt"
        self.project_start_date = "2022/10-01"
        self.closed_days = ["saturday", "sunday"]
        self.gantt_font_size = 100
        self.gantt_font_size_line = (
            f"skinparam classFontSize {self.gantt_font_size}"
        )
        self.box_font_size = "30"

        # self.font_size="skinparam defaultFontSize  100"
        self.parents, self.date_milestones = create_python_gantt(
            params["wages"], self.box_font_size, self.
                ↪ project_start_date
        )
        self.end_line = "@endgantt"
        self.lines, self.parent_costs = self.get_plantuml_gantt_lines
            ↪ ()
        self.write_gantt(filepath, self.lines)

        self.costs = None

    def get_plantuml_gantt_lines(self):
        """Gets the list of lines of the UML file to create a Gantt
            ↪ chart."""
        lines = []
        lines.append(self.start_line)
        lines.append(f"project starts the {self.project_start_date}")
        lines = self.add_closed_dates(lines)
        lines.append(self.gantt_font_size_line)
        # Add milestone style here
        lines.append(get_milestone_style("blue", 100, "red", "yellow"
            ↪ ))

        # Include date milestones
        for date_milestone in self.date_milestones:
            lines.append(get_milestone_for_date(date_milestone))

        # Writes acitivity line, appends milestone line below
            ↪ activity.
        lines, parent_costs = self.loop_through_parents_printing(
            ↪ lines)
        lines.append(self.end_line)
        return lines, parent_costs

    def loop_through_parents_printing(self, lines):
```

```
        """Prints all relevant data of the parent activities.

        :param lines:
        """
        # print descriptions
        for i, _ in enumerate(self.parents):
            lines = self.print_parent_descriptions(lines, self.
                ↪ parents[i])

        # print order
        for i, _ in enumerate(self.parents):
            if i > 0:
                if not self.parents[i].starts_at_child_nr_start is
                    ↪ None:
                    lines.append(
                        f"[{self.parents[i].get_tag()}] starts at ["
                        + f"{self.parents[i].starts_at_child_nr_start
                            ↪ }]"
                        + "'s start"
                    )
                else:
                    print(
                        f"parent_descr={self.parents[i].description},
                            ↪  and "
                        + "starts at: "
                        f"{self.parents[i].starts_at_child_nr_start}"
                        + " and  tag="
                        + f"{self.parents[i].get_tag()}, writing end"
                    )
                    lines.append(
                        f"[{self.parents[i].get_tag()}] starts at ["
                        + f"{self.parents[i-1].get_tag()}]'s end"
                    )
            lines = self.print_parent_order(lines, self.parents[i])

        # print colour
        for i, _ in enumerate(self.parents):
            lines = self.print_parent_colour(lines, self.parents[i])

        # compute costs
        total_costs = 0
        parent_costs = {}
        for i, _ in enumerate(self.parents):
            parent, total_costs, lines = self.print_parent_costs(
                total_costs, lines, self.parents[i]
            )
            parent_costs[parent] = total_costs
        return lines, parent_costs

    def print_parent_descriptions(self, lines, parent):
        """Creates the lines with the descriptions of the parents.

        :param lines:
        :param parent:
        """
        lines.append("")
        lines = self.print_descriptions(parent, lines)
        return lines

    def print_parent_order(self, lines, parent):
        """Creates the lines that specify the order of the parent
            ↪ activities.
```

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

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

```
        """Creates the lines that specify the costs of the activities .

        # TODO: output to LaTex variables.

        :param total_costs:
        :param activity:
        :param lines:
        """
        activity_costs = (
            activity.duration * activity.hours_per_day * activity.hourly_wage
        )
        lines.append(
            f"'[{activity.description}]  takes: {activity.duration}[days] "
            + f"equating to:{activity.duration*activity.hours_per_day}[hours]"
            + f" and costs:{activity.hourly_wage} per hour, yielding activity"
            + " costs:"
            + f"{activity_costs}"
            + " Euros."
        )
        total_costs = (
            total_costs
            + activity.duration * activity.hours_per_day * activity.hourly_wage
        )
        for child in activity.children:
            total_costs, lines = self.print_costs(total_costs, child,  lines)
        return total_costs, lines

    def add_closed_dates(self, lines):
        """Adds the recurring days on which no work is performed in the Gantt.

        :param lines:
        """
        for closed_day in self.closed_days:
            lines.append(f"{closed_day} are closed")
        return lines

    def write_gantt(self, filepath, some_list):
        """Writes the lines of a gant to an output file.

        :param filepath:
        :param some_list:
        """
        with open(filepath, "w", encoding="utf-8") as f:
            for item in some_list:
                # pylint: disable=C0209
                f.write("%s\n" % item)
        f.close()

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of  the
        computation.

        :param x:
```

```
274         """
275         return x + 2
```

E Appendix src/__main__.py

```python
"""Entry point for this project, runs the project code and exports data if
export commands are given to the cli command that invokes this script."""


from src.arg_parser import parse_cli_args
from src.Cost_model import params
from src.export_data.export_data import export_data

# Project code imports.
from src.revenue_model.Model_top_down import Model_top_down

# Export data import.


# Parse command line interface arguments to determine what this script does.
args = parse_cli_args()

# run monte-carlo for revenue estimation
model = Model_top_down()


# Run data export code if any argument is given.
if not all(
    arg is None for arg in [args.l, args.dd, args.sd, args.c2l, args.ec2l]
):
    export_data(args, params)
```

F Appendix src/arg_parser.py

```python
"""This is the main code of this project nr, and it manages running the code
and outputting the results to LaTex."""
import argparse


def parse_cli_args():
    """Parses the command line arguments to determine what this code should
    do."""
    # Instantiate the parser
    parser = argparse.ArgumentParser(description="Optional app description")

    # Include argument parsing for data exporting code.
    # Compile LaTex
    parser.add_argument(
        "--l",
        action="store_true",
        help="Boolean indicating if code compiles LaTex",
    )

    # Generate, compile and export Dynamic PlantUML diagrams to LaTex .
    parser.add_argument(
        "--dd",
        action="store_true",
        help=(
            "A boolean indicating if code generated diagrams are compiled"
            + " and exported."
        ),
    )
    # Generate, compile and export Static PlantUML diagrams to LaTex.
    parser.add_argument(
        "--sd",
        action="store_true",
        help=(
            "A boolean indicating if static diagrams are compiled and "
            + " exported."
        ),
    )

    # Export the project code to LaTex.
    parser.add_argument(
        "--c2l",
        action="store_true",
        help="A boolean indicating if project code is exported to LaTex.",
    )

    # Export the exporting code, and the project code to LaTex.
    parser.add_argument(
        "--ec2l",
        action="store_true",
        help=(
            "A boolean indicating if code that exports code is exported"
            + " 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

```
"""Specify hardcoded output data."""


# pylint: disable=R0902
# pylint: disable=R0903
class Hardcoded_data:
    """Contains a list of parameters that are used in this project that
    combines Python code with a LaTex document."""

    def __init__(self):

        # Specify code configuration details
        # TODO: include as optional arguments.
        self.await_compilation = True
        self.verbose = True
        self.gantt_extension = ".uml"
        self.diagram_extension = ".png"

        # Filenames.
        self.main_latex_filename = "main.tex"
        self.export_data_dirname = "export_data"
        self.diagram_dir = "Diagrams"
        self.plant_uml_java_filename = "plantuml.jar"

        # Appendix manager filenames
        self.export_code_appendices_filename = "export_code_appendices.tex"
        self.export_code_appendices_filename_from_root = (
            "export_code_appendices_from_root.tex"
        )
        self.project_code_appendices_filename = "project_code_appendices.tex"
        self.project_code_appendices_filename_from_root = (
            "project_code_appendices_from_root.tex"
        )
        self.automatic_appendices_manager_filenames = [
            self.export_code_appendices_filename,
            self.export_code_appendices_filename_from_root,
            self.project_code_appendices_filename,
            self.project_code_appendices_filename_from_root,
        ]

        self.manual_appendices_filename = "manual_appendices.tex"
        self.manual_appendices_filename_from_root = (
            "manual_appendices_from_root.tex"
        )
        self.manual_appendices_manager_filenames = [
            self.manual_appendices_filename,
            self.manual_appendices_filename_from_root,
        ]
        self.appendix_dir_from_root = "latex/Appendices/"

        # Folder names.
        self.dynamic_diagram_dir = "Dynamic_diagrams"
        self.static_diagram_dir = "Static_diagrams"

        # Specify paths relative to root.
        self.path_to_export_data_from_root = f"src/{self.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_gantts = (
68              f"{self.path_to_export_data_from_root}/"
69              + f"{self.diagram_dir}/{self.dynamic_diagram_dir}"
70          )
71          self.path_to_static_gantts = (
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

```
"""Rpresents a milestone object that can be added to the end of an activity."""


import re


# pylint: disable=R0903
class Milestone:
    """Used to create a milestone at the end or start of an activity in a Gantt
    chart."""

    # pylint: disable=R0913
    def __init__(
        self,
        description,
        font_size,
        at_end=True,
        date=None,
    ):
        self.font_size = font_size
        if font_size is None:
            raise Exception("Error, milestone fontsize not set.")
        self.description = description
        self.at_end = at_end
        self.date = date

        # Create an alphanumeric tag with underscores.
        self.tag = re.sub(r"[^A-Za-z0-9_]+", "_", self.description).lower()
        # Specify milestone stile.


def get_milestone_style(colour, fontsize, backgroundcolour, linecolour):
    """Returns the UML style specification of a milestone.

    :param colour:
    :param fontsize:
    :param backgroundcolour:
    :param linecolour:
    """
    left = """
    <style>
    ganttDiagram {
    milestone {
"""
    middle = f"""
        FontColor {colour}
        FontSize {fontsize}
        FontStyle italic
        BackGroundColor {backgroundcolour}
        LineColor {linecolour}
"""
    end = """
    }
}
</style>
    """
```

```
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,
            ↪ activity_tag)
70      print(f"milestone={milestone}")
71      return milestone
72
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

```
"""File used to export plots to a latex directory.
Call this from another file, for project 11, question 3b:
from Plot_to_tex import Plot_to_tex as plt_tex
multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints), dtype=
    ↪ int);
# actually fill with data
lineLabels = [] # add a label for each dataseries
plt_tex.plotMultipleLines(plt_tex,single_x_series,multiple_y_series,"
    ↪ x-axis
 label [units]","y-axis label [units]",lineLabels,"3b",4,11)
4b=filename
4 = position of legend, e.g. top right.
For a single line, use:
plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),dataseries,"
    ↪ x-axis
label [units]","y-axis label [units]",lineLabel,"3b",4,11)"""

import os

import matplotlib.pyplot as plt
import numpy as np
from matplotlib import lines

# You can also plot a table directly into latex, see
    ↪ example_create_a_table(..)
# Then put it in latex with for example:
# \begin{table}[H]
#     \centering
#     \caption{Results some computation.}\label{tab:some_computation}
#     \begin{tabular}{|c|c|} % remember to update this
# % to show all columns of table
#         \hline
#         \input{latex/project3/tables/q2.txt}
#     \end{tabular}
# \end{table}


class Plot_to_tex:
    """Object used to output plots to latex directory of project."""

    def __init__(self):
        self.script_dir = self.get_script_dir()

    # plot graph (legendPosition = integer 1 to 4)
    def plotSingleLine(
        self,
        x_path,
        y_series,
        x_axis_label,
        y_axis_label,
        label,
        filename,
        legendPosition,
        project_name,
    ):
        """

        :param x_path: param y_series:
        :param x_axis_label: param y_axis_label:
        :param label: param filename:
```

```
        :param legendPosition: param project_name:
        :param y_series: param y_axis_label:
        :param filename: param project_name:
        :param y_axis_label: param project_name:
        :param project_name:

        """
        # pylint: disable=R0913
        # TODO: reduce 9/5 arguments to at most 5/5 arguments.
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(x_path, y_series, c="b", ls="-", label=label,
            ↪ fillstyle="none")
        plt.legend(loc=legendPosition)
        plt.xlabel(x_axis_label)
        plt.ylabel(y_axis_label)
        plt.savefig(
            os.path.dirname(__file__)
            + f"/../../../latex/{project_name}"
            + "/Images/"
            + filename
            + ".png"
        )

    #         plt.show();

    # plot graphs
    def plotMultipleLines(
        self,
        x,
        y_series,
        x_label,
        y_label,
        label,
        filename,
        legendPosition,
        project_name,
    ):
        """

        :param x: param y_series:
        :param x_label: param y_label:
        :param label: param filename:
        :param legendPosition: param project_name:
        :param y_series: param y_label:
        :param filename: param project_name:
        :param y_label: param project_name:
        :param project_name:

        """
        # pylint: disable=R0913
        # TODO: reduce 9/5 arguments to at most 5/5 arguments.
        fig = plt.figure()
        ax = fig.add_subplot(111)

        # generate colours
        cmap = self.get_cmap(len(y_series[:, 0]))

        # generate line types
        lineTypes = self.generateLineTypes(y_series)

        for i in range(0, len(y_series)):
```

```
            # overwrite linetypes to single type
            lineTypes[i] = "-"
            ax.plot(
                x,
                y_series[i, :],
                ls=lineTypes[i],
                label=label[i],
                fillstyle="none",
                c=cmap(i),
            )
            # color

        # configure plot layout
        plt.legend(loc=legendPosition)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.savefig(
            os.path.dirname(__file__)
            + f"/../../../latex/{project_name}"
            + "/Images/"
            + filename
            + ".png"
        )

    # Generate random line colours
    # Source: https://stackoverflow.com/questions/14720331/
    # how-to-generate-random-colors-in-matplotlib
    def get_cmap(self, n, name="hsv"):
        """Returns a function that maps each index in 0, 1, ..., n-1
           ↪ to a
        distinct RGB color; the keyword argument name must be a
           ↪ standard mpl
        colormap name.

        :param n: param name:  (Default value = "hsv")
        :param name: Default value = "hsv")
        """
        return plt.cm.get_cmap(name, n)

    def generateLineTypes(self, y_series):
        """

        :param y_series:

        """
        # generate varying linetypes
        typeOfLines = list(lines.lineStyles.keys())

        while len(y_series) > len(typeOfLines):
            typeOfLines.append("-.")

        # remove void lines
        for i in range(0, len(y_series)):
            if typeOfLines[i] == "None":
                typeOfLines[i] = "-"
            if typeOfLines[i] == "":
                typeOfLines[i] = ":"
            if typeOfLines[i] == " ":
                typeOfLines[i] = "--"
        return typeOfLines
```

```
    # Create a table with: table_matrix = np.zeros((4,4),dtype=object ) and pass
    # it to this object
    def put_table_in_tex(self, table_matrix, filename, project_name):
        """

        :param table_matrix: param filename:
        :param project_name: param filename:
        :param filename:

        """
        cols = np.shape(table_matrix)[1]
        some_format = "%s"
        for _ in range(1, cols):
            some_format = some_format + " & %s"
        some_format = some_format + ""
        # TODO: Change to something else to save as txt.
        np.savetxt(
            os.path.dirname(__file__)
            + f"/../../../latex/{project_name}"
            + "/tables/"
            + filename
            + ".txt",
            table_matrix,
            delimiter=" & ",
            fmt=format,
            newline="  \\\\ \\hline \n",
        )

    # replace this with your own table creation and then pass it to
    # put_table_in_tex(..)
    def example_create_a_table(self):
        """Example on how to create a latex table from Python."""
        project_name = "1"
        table_name = "example_table_name"
        rows = 2
        columns = 4
        table_matrix = np.zeros((rows, columns), dtype=object)
        table_matrix[:, :] = ""  # replace the standard zeros with empty cell
        print(table_matrix)
        for column in range(0, columns):
            for row in range(0, rows):
                table_matrix[row, column] = row + column
        table_matrix[1, 0] = "example"
        table_matrix[0, 1] = "grid sizes"

        self.put_table_in_tex(table_matrix, table_name, project_name)

    def get_script_dir(self):
        """returns the directory of this script regardless of from which level
        the code is executed."""
        return os.path.dirname(__file__)

    def export_plot(self, some_plt, filename):
        """

        :param plt:
        :param filename:

        """
```

```
236         self.create_target_dir_if_not_exists("latex/Images/", "graphs
               ↪ ")
237         some_plt.savefig(
238             "latex/Images/" + "graphs/" + filename + ".png", dpi=200
239         )
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 
255 if __name__ == "__main__":
256     main = Plot_to_tex()
257     main.example_create_a_table()
```

J Appendix src/export_data/create_dynamic_diagrams.py

```python
"""Generates .uml PlantUML files based on Python code.

Typically used for Gantt charts.
"""
from src.export_data.plantuml_compile import (
    compile_diagrams_in_dir_relative_to_root,
)
from src.export_data.plantuml_generate import
    ↪ generate_all_dynamic_diagrams
from src.export_data.plantuml_to_tex import export_diagrams_to_latex


def create_dynamic_diagrams(args, hd):
    """Generates the dynamic diagrams."""
    # Generate PlantUML diagrams dynamically (using code).
    if args.dd:
        generate_all_dynamic_diagrams(
            f"{hd.path_to_export_data_from_root}/Diagrams/
                ↪ Dynamic_diagrams"
        )

        # Compile dynamically generated PlantUML diagrams to images.
        compile_diagrams_in_dir_relative_to_root(
            hd.await_compilation,
            hd.gantt_extension,
            hd.jar_path_relative_from_root,
            hd.path_to_dynamic_gantts,
            hd.verbose,
        )

        # Export dynamic PlantUML text files to LaTex.
        export_diagrams_to_latex(
            hd.path_to_dynamic_gantts,
            hd.gantt_extension,
            hd.diagram_output_dir_relative_to_root,
        )

        # Export dynamic PlantUML diagram images to LaTex.
        export_diagrams_to_latex(
            hd.path_to_dynamic_gantts,
            hd.diagram_extension,
            hd.diagram_output_dir_relative_to_root,
        )
```

K Appendix src/export_data/create_static_diagrams.py

```
"""This file creates the diagrams that are written directly in plain
    ↪ .uml
files."""
from src.export_data.plantuml_compile import (
    compile_diagrams_in_dir_relative_to_root,
)
from src.export_data.plantuml_to_tex import export_diagrams_to_latex


def create_static_diagrams(args, hd):
    """Generates .png images of the static diagram .uml files.

    :param args:
    :param hd:
    """
    # PlantUML
    if args.sd:
        # Compile statically generated PlantUML diagrams to images.
        compile_diagrams_in_dir_relative_to_root(
            hd.await_compilation,
            hd.gantt_extension,
            hd.jar_path_relative_from_root,
            hd.path_to_static_gantts,
            hd.verbose,
        )

        # Export static PlantUML text files to LaTex.
        export_diagrams_to_latex(
            hd.path_to_static_gantts,
            hd.gantt_extension,
            hd.diagram_output_dir_relative_to_root,
        )

        # Export static PlantUML diagram images to LaTex.
        export_diagrams_to_latex(
            hd.path_to_static_gantts,
            hd.diagram_extension,
            hd.diagram_output_dir_relative_to_root,
        )
```

L Appendix src/export_data/export_data.py

```python
"""File used to export data to latex."""
from pprint import pprint

from src.Cost_model import dict_to_latex_table
from src.export_data.create_dynamic_diagrams import
    ↪ create_dynamic_diagrams
from src.export_data.create_static_diagrams import
    ↪ create_static_diagrams
from src.export_data.Hardcoded_data import Hardcoded_data
from src.export_data.helper_dir_file_edit import overwrite_file
from src.export_data.latex_compile import compile_latex
from src.export_data.latex_export_code import export_code_to_latex


def export_data(args, params):
    """Parses the Python arguments that specify what should be
       ↪ compiled.

    :param args:
    """

    hd = Hardcoded_data()

    param_lines = export_latex_params(params)
    print(f"param_lines={param_lines}")

    # Export parameters to file.
    overwrite_file("latex/Tables/params.tex", param_lines)

    # Export model parameters to Latex table:
    dict_to_latex_table(
        "latex/Tables/params_table.tex",
        params,
        "Parameter",
        "Value",
        (
            r"Cost Model Parameters in \euro (/hr or absolute, unless
               ↪ "
            + "specified otherwise)"
        ),
    )

    # Generating PlantUML diagrams
    create_dynamic_diagrams(args, hd)
    create_static_diagrams(args, hd)

    # Plotting graphs using Python code, and export them to latex.
    # Generate plots.
    # Export plots to LaTex.

    # Export code to LaTex.
    if args.c2l:
        # TODO: verify whether the latex/{project_name}/Appendices
           ↪ folder
        # exists before exporting.
        # TODO: verify whether the latex/{project_name}/Images folder
           ↪  exists
        # before exporting.
        export_code_to_latex(hd, False)
    elif args.ec2l:
```

```
55          # TODO: verify whether the latex/{project_name}/Appendices
               ↪ folder
56          # exists before exporting.
57          # TODO: verify whether the latex/{project_name}/Images folder
               ↪  exists
58          # before exporting.
59          export_code_to_latex(hd, True)
60
61      # Compile the accompanying LaTex report.
62      if args.l:
63          compile_latex(True, True)
64          print("")
65      print("\n\nDone exporting data.")
66
67
68  def export_latex_params(params):
69      """Exports the model parameters and computed values to LaTex
           ↪ variables."""
70      # Export model parameters to .tex file with LaTex variables.
71      param_lines = []
72      # TODO: flatten dict
73      # print(f'params={params}')
74      pprint(params)
75
76      for key, value in params.items():
77          if key == "wages":
78              for wages_key, wages_value in params[key].items():
79                  param_lines.append(
80                      "\\newcommand"
81                      + chr(92)
82                      + str(wages_key.replace("_", ""))
83                      + "{"
84                      + str(wages_value)
85                      + "}"
86                  )
87          else:
88              param_lines.append(
89                  "\\newcommand"
90                  + chr(92)
91                  + str(key.replace("_", "").replace("-", ""))
92                  + "{"
93                  + str(value)
94                  + "}"
95              )
96      return param_lines
```

M Appendix src/export_data/helper_bash_commands.py

```python
"""Code used to execute Bash commands from Python."""
import subprocess  # nosec

from src.export_data.plantuml_compile import
    ↪ get_output_of_bash_command


def run_a_bash_command(await_compilation, bash_command, verbose):
    """Runs a bash commands from Python.

    :param await_compilation:
    :param bash_command:
    :param verbose:
    """
    if await_compilation:
        if verbose:
            subprocess.call(
                bash_command,
                shell=True,  # nosec
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
            )
        else:
            # pylint: disable=E1129
            # pylint: disable=R1732
            subprocess.call(
                bash_command,
                shell=True,  # nosec
                stderr=subprocess.DEVNULL,
                stdout=subprocess.DEVNULL,
            )
    else:
        if verbose:
            # pylint: disable=R1732
            subprocess.Popen(
                bash_command,
                shell=True,  # nosec
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
            )
        else:
            with subprocess.Popen(
                bash_command,
                shell=True,  # nosec
                stderr=subprocess.DEVNULL,
                stdout=subprocess.DEVNULL,
            ) as process:
                get_output_of_bash_command(process)
```

N Appendix src/export_data/helper_dir_file_edit.py

```
"""A helper file that is used for directory and file editing."""
import glob
import os
import shutil


def file_contains(filepath, substring):
    """Returns True if a file exists. None otherwise.

    :param filepath:
    :param substring:
    """
    with open(filepath, encoding="utf-8") as f:
        return substring in f.read()


def get_dir_filelist_based_on_extension(dir_relative_to_root,
    ↪ extension):
    """

    :param dir_relative_to_root: A relative directory as
    seen from the root dir of this project.
    :param extension: File extension that is used/searched in this
        ↪ function.

    """
    selected_filenames = []
    # TODO: assert directory exists
    for filename in os.listdir(dir_relative_to_root):
        if filename.endswith(extension):
            selected_filenames.append(filename)
    return selected_filenames


def create_dir_relative_to_root_if_not_exists(dir_relative_to_root):
    """

       :param dir_relative_to_root: A relative directory as
    seen from the root dir of this project.

    """
    if not os.path.exists(dir_relative_to_root):
        os.makedirs(dir_relative_to_root)


def dir_relative_to_root_exists(dir_relative_to_root):
    """

       :param dir_relative_to_root: A relative directory as
    seen from the root dir of this project.

    """
    if not os.path.exists(dir_relative_to_root):
        return False
    if os.path.exists(dir_relative_to_root):
        return True
    raise Exception(
        f"Directory relative to root: {dir_relative_to_root}"
        + " did not exist, nor did it exist."
    )
```

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

```
    """

    :param filepath:
    :param lines:

    """
    with open(filepath, "a", encoding="utf-8") as the_file:
        for line in lines:
            the_file.write(f"{line}\n")


def append_line_to_file(filepath, line):
    """

    :param filepath:
    :param line:

    """
    with open(filepath, "a", encoding="utf-8") as the_file:
        the_file.write(f"{line}\n")
        the_file.close()


def remove_all_auto_generated_appendices(hd):
    """

    :param hd:

    """

    # TODO: move identifier into hardcoded.
    all_appendix_files = get_all_files_in_dir_and_child_dirs(
        ".tex", hd.appendix_dir_from_root, excluded_files=None
    )
    for file in all_appendix_files:
        if "Auto_generated" in file:
            delete_file_if_exists(file)


def delete_dir_relative_to_root_if_not_exists(dir_relative_to_root):
    """

       :param dir_relative_to_root: A relative directory as
    seen from the root dir of this project.

    """
    if os.path.exists(dir_relative_to_root):
        # Remove directory and its content.
        shutil.rmtree(dir_relative_to_root)
```

O Appendix src/export_data/helper_tex_editing.py

```python
"""Helper that modifies LaTex file to allow automatically including Pyathon
code as appendices."""
import os

from src.export_data.helper_dir_file_edit import (
    append_line_to_file,
    append_lines_to_file,
    convert_filepath_to_filepath_from_root,
    delete_file_if_exists,
    get_filename_from_dir,
)


def code_filepath_to_tex_appendix_filename(
    filename, _, is_project_code, is_export_code
):
    """

    :param filename:
    :param from_root:
    :param is_project_code:
    :param is_export_code:

    """

    # TODO: Include assert to verify filename ends at .py.
    # TODO: Include assert to verify filename doesn't end at .py anymore.
    filename_without_extension = os.path.splitext(filename)[0]

    # Create appendix filename identifier segment
    verify_input_code_type(is_export_code, is_project_code)
    if is_project_code:
        identifier = "Auto_generated_project_code_appendix_"
    elif is_export_code:
        identifier = "Auto_generated_export_code_appendix_"

    appendix_filename = f"{identifier}{filename_without_extension}"
    return appendix_filename


def verify_input_code_type(is_export_code, is_project_code):
    """

    :param is_export_code:
    :param is_project_code:

    """
    # Create appendix filename identifier segment
    if is_project_code and is_export_code:
        raise Exception(
            "Error, a file can't be both project code, and export code at"
            + " same time."
        )
    if not is_project_code and not is_export_code:
        raise Exception(
            "Error, don't know what to do with files that are neither  project"
```

```
            + " code, nor export code."
        )


def tex_appendix_filename_to_inclusion_command(appendix_filename,
    ↪ from_root):
    """

    :param appendix_filename:
    :param from_root:

    """
    # Create full appendix filename.
    if from_root:
        # Generate latex inclusion command for latex compilation from
            ↪  root dir.
        appendix_inclusion_command = (
            f"\\input{{latex/Appendices/{appendix_filename}.tex}} \\
                ↪ newpage"
        )
        # \input{latex/Appendices/Auto_generated_py_App8.tex} \
            ↪ newpage
    else:
        # \input{Appendices/Auto_generated_py_App8.tex} \newpage
        appendix_inclusion_command = (
            f"\\input{{Appendices/{appendix_filename}.tex}} \\newpage
                ↪ "
        )
    return appendix_inclusion_command


def create_appendix_filecontent(
    latex_object_name, filename, filepath_from_root, from_root
):
    """

    :param latex_object_name:
    :param filename:
    :param filepath_from_root:
    :param from_root:

    """
    # Latex titles should escape underscores.
    filepath_from_root_without_underscores = filepath_from_root.
        ↪ replace(
        "_", r"\_"
    )
    lines = []
    lines.append(
        rf"\{latex_object_name}{{Appendix "
        + rf"\{filepath_from_root_without_underscores}}}"
        + rf"\label{{app:{filename}}}"
    )
    if from_root:
        lines.append(rf"\pythonexternal{{latex/..{filepath_from_root
            ↪ }}}")
    else:
        lines.append(rf"\pythonexternal{{latex/..{filepath_from_root
            ↪ }}}")
    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.
                ↪ appendix_dir_from_root}{x}"),
128             hd.automatic_appendices_manager_filenames,
129         )
130     )
131
132     # Create new appendix_manager_files
133     list(
134         map(
135             # pylint: disable=R1732
136             lambda x: open(
137                 f"{hd.appendix_dir_from_root}{x}", "a", encoding="utf
                    ↪ -8"
138             ),
139             hd.automatic_appendices_manager_filenames,
140         )
141     )
142
143     # Ensure manual appendix_manager_files are created.
144     list(
145         map(
146             # pylint: disable=R1732
147             lambda x: open(
148                 f"{hd.appendix_dir_from_root}{x}", "a", encoding="utf
                    ↪ -8"
149             ),
150             hd.manual_appendices_manager_filenames,
151         )
152     )
153
154
155 # pylint: disable=R0913
156 def create_appendix_file(
157     hd,
158     filename,
159     filepath_from_root,
160     latex_object_name,
161     is_export_code,
162     is_project_code,
163 ):
164     """
165
166     :param hd:
167     :param filename:
168     :param filepath_from_root:
169     :param latex_object_name:
```

```
    :param is_export_code:
    :param is_project_code:

    """
    verify_input_code_type(is_export_code, is_project_code)
    filename_without_extension = os.path.splitext(filename)[0]
    if is_project_code:
        # Create the appendix for the case the latex is compiled from root.
        appendix_filepath = (
            f"{hd.appendix_dir_from_root}/Auto_generated_proj"
            + f"ect_code_appendix_{filename_without_extension}.tex"
        )

        # Append latex_filepath to appendix manager.
        # append_lines_to_file(
        #     f"{hd.appendix_dir_from_root}{hd.project_code_appendices_filename}",
        #     [tex_appendix_filepath_to_inclusion_command(appendix_filepath)],
        # )

        # Get Appendix .tex content.
        appendix_lines_from_root = create_appendix_filecontent(
            latex_object_name, filename, filepath_from_root, True
        )

        # Write appendix to .tex file.
        append_lines_to_file(appendix_filepath, appendix_lines_from_root)
    elif is_export_code:
        # Create the appendix for the case the latex is compiled from root.
        appendix_filepath = (
            f"{hd.appendix_dir_from_root}/Auto_generated_export"
            + f"_code_appendix_{filename_without_extension}.tex"
        )

        # Append latex_filepath to appendix manager.
        # append_lines_to_file(
        #     f"{hd.appendix_dir_from_root}{hd.export_code_appendices_filename}",
        #     [tex_appendix_filepath_to_inclusion_command(appendix_filepath)],
        # )

        # Get Appendix .tex content.
        appendix_lines_from_root = create_appendix_filecontent(
            latex_object_name, filename, filepath_from_root, True
        )

        # Write appendix to .tex file.
        append_lines_to_file(appendix_filepath, appendix_lines_from_root)
    # TODO: verify files exist


def export_python_project_code(
    hd, normalised_root_dir, python_project_code_filepaths
):
    """

```

```
    :param hd:
    :param normalised_root_dir:
    :param python_project_code_filepaths:

    """
    is_project_code = True
    is_export_code = False
    from_root = False
    for filepath in python_project_code_filepaths:
        create_appendices(
            hd,
            filepath,
            normalised_root_dir,
            from_root,
            is_export_code,
            is_project_code,
        )
        create_appendices(
            hd,
            filepath,
            normalised_root_dir,
            True,
            is_export_code,
            is_project_code,
        )


def export_python_export_code(
    hd, normalised_root_dir, python_export_code_filepaths
):
    """

    :param hd:
    :param normalised_root_dir:
    :param python_export_code_filepaths:

    """
    is_project_code = False
    is_export_code = True
    from_root = False
    for filepath in python_export_code_filepaths:
        create_appendices(
            hd,
            filepath,
            normalised_root_dir,
            from_root,
            is_export_code,
            is_project_code,
        )
        create_appendices(
            hd,
            filepath,
            normalised_root_dir,
            True,
            is_export_code,
            is_project_code,
        )


# pylint: disable=R0913
def create_appendices(
    hd,
```

```
    filepath,
    normalised_root_dir,
    from_root,
    is_export_code,
    is_project_code,
):
    """

    :param hd:
    :param filepath:
    :param normalised_root_dir:
    :param from_root:
    :param is_export_code:
    :param is_project_code:

    """
    # Get the filepath of a python file from the root dir of this project.
    filepath_from_root = convert_filepath_to_filepath_from_root(
        filepath, normalised_root_dir
    )
    print(f"from_root={from_root},filepath_from_root={filepath_from_root}")

    # Get the filename of a python filepath
    filename = get_filename_from_dir(filepath)

    # Get the filename for a latex appendix from a python filename.
    appendix_filename = code_filepath_to_tex_appendix_filename(
        filename, from_root, is_project_code, is_export_code
    )

    # Command to include the appendix in the appendices manager.
    appendix_inclusion_command = tex_appendix_filename_to_inclusion_command(
        appendix_filename, from_root
    )

    append_appendix_to_appendix_managers(
        appendix_inclusion_command,
        from_root,
        hd,
        is_export_code,
        is_project_code,
    )

    # Create the appendix .tex file.
    # TODO: move "section" to hardcoded.
    if from_root:  # Appendix only contains files readable from root.
        create_appendix_file(
            hd,
            filename,
            filepath_from_root,
            "section",
            is_export_code,
            is_project_code,
        )


def append_appendix_to_appendix_managers(
    appendix_inclusion_command, from_root, hd, is_export_code, is_project_code
```

```
):
    """

    :param appendix_inclusion_command:
    :param from_root:
    :param hd:
    :param is_export_code:
    :param is_project_code:

    """
    # Append the appendix .tex file to the appendix manager.
    if is_project_code:
        if from_root:
            # print('from_root={from_root}Append to:
            # +f"{hd.project_code_appendices_filename_from_root}')
            append_line_to_file(
                f"{hd.appendix_dir_from_root}"
                + f"{hd.project_code_appendices_filename_from_root}",
                appendix_inclusion_command,
            )
        else:
            # print(f'from_root={from_root}Append to:"
            # +f"{hd.project_code_appendices_filename}')
            append_line_to_file(
                f"{hd.appendix_dir_from_root}"
                + f"{hd.project_code_appendices_filename}",
                appendix_inclusion_command,
            )

    if is_export_code:
        if from_root:
            append_line_to_file(
                f"{hd.appendix_dir_from_root}"
                + f"{hd.export_code_appendices_filename_from_root}",
                appendix_inclusion_command,
            )
        else:
            append_line_to_file(
                f"{hd.appendix_dir_from_root}"
                + f"{hd.export_code_appendices_filename}",
                appendix_inclusion_command,
            )
```

P Appendix src/export_data/helper_tex_reading.py

```python
"""Helper script to parse Latex files, to support including LaTex appendices."""
from src.export_data.helper_dir_file_edit import file_contains


def verify_latex_supports_auto_generated_appendices(path_to_main_latex_file):
    """Ensures the Latex file supports including automatically appending code
    as appendices.

    :param path_to_main_latex_file:
    """
    # TODO: change verification to complete tex block(s) for appendices.
    # TODO: Also verify related boolean and if statement creations.
    determining_overleaf_home_line = (
        r"\def\overleafhome{/tmp}% change as appropriate"
    )
    begin_apendices_line = "\\begin{appendices}"
    print(f"determining_overleaf_home_line={determining_overleaf_home_line}")
    print(f"begin_apendices_line={begin_apendices_line}")

    if not file_contains(
        path_to_main_latex_file, determining_overleaf_home_line
    ):
        raise Exception(
            f"Error, {path_to_main_latex_file} does not contain:\n\n"
            + f"{determining_overleaf_home_line}\n\n so this Python code "
            + "cannot export the code as latex appendices."
        )
    if not file_contains(
        path_to_main_latex_file, determining_overleaf_home_line
    ):
        raise Exception(
            f"Error, {path_to_main_latex_file} does not contain:\n\n"
            + f"{begin_apendices_line}\n\n so this Python code cannot export"
            "the code as latex appendices."
        )
```

Q Appendix src/export_data/latex_compile.py

```python
"""Compiles the latex report using the compile script."""


from src.export_data.helper_bash_commands import run_a_bash_command


def compile_latex(await_compilation, verbose):
    """Compiles the LaTex report of this project using its compile script.

    :param await_compilation: Make python wait until the PlantUML compilation
    is completed.
    :param project_name: The name of the project that is being executed/ran.
    :param verbose: True, ensures compilation output is printed to terminal,
    False means compilation is silent.

    Returns:
        Nothing.

    Raises:
        Nothing.
    """

    # Ensure compile script is runnable.
    bash_make_compile_script_runnable_command = (
        "chmod +x latex/compile_script.sh"
    )
    run_a_bash_command(
        await_compilation, bash_make_compile_script_runnable_command, verbose
    )

    # Run latex compilation script to compile latex project.
    bash_compilation_command = "latex/compile_script.sh"
    run_a_bash_command(await_compilation, bash_compilation_command, verbose)
    print(f"ran:{bash_compilation_command}")
```

R Appendix src/export_data/latex_export_code.py

```python
"""Exports code to latex appendices."""
import os

from src.export_data.helper_dir_file_edit import (
    get_all_files_in_dir_and_child_dirs,
    get_filepaths_in_dir,
    remove_all_auto_generated_appendices,
    sort_filepaths_by_filename,
)
from src.export_data.helper_tex_editing import (
    create_appendix_manager_files,
    export_python_export_code,
    export_python_project_code,
)
from src.export_data.helper_tex_reading import (
    verify_latex_supports_auto_generated_appendices,
)


def export_code_to_latex(hd, include_export_code):
    """This function exports the python files and compiled pdfs of jupiter
    notebooks into the latex of the same project number. First it scans which
    appendices (without code, without notebooks) are already manually included
    in the main latex code. Next, all appendices that contain the python code
    are either found or created in the following order: First, the __main__.py
    file is included, followed by the main.py file, followed by all python code
    files in alphabetic order. After this, all the pdfs of the compiled
    notebooks are added in alphabetic order of filename. This order of
    appendices is overwritten in the main tex file.

    :param main_latex_filename: Name of the main latex document of this project
    number.
    :param project_name: The name of the project that is being executed/ran.
    The number  indicating which project this code pertains to.
    """
    script_dir = get_script_dir()
    latex_dir = script_dir + "/../../latex/"
    path_to_main_latex_file = f"{latex_dir}{hd.main_latex_filename}"
    root_dir = script_dir + "/../../"
    normalised_root_dir = os.path.normpath(root_dir)
    src_dir = script_dir + "/../"

    # Verify the latex file supports auto-generated python appendices.
    verify_latex_supports_auto_generated_appendices(path_to_main_latex_file)

    # Get paths to files containing project python code.
    python_project_code_filepaths = get_filepaths_in_dir(
        "py", src_dir, ["__init__.py"]
```

```
    )

    get_compiled_notebook_paths(script_dir)
    print(f"python_project_code_filepaths={
        ↪ python_project_code_filepaths}")

    # Get paths to the files containing the latex export code
    if include_export_code:
        python_export_code_filepaths = get_filepaths_in_dir(
            "py", script_dir, ["__init__.py"]
        )

    remove_all_auto_generated_appendices(hd)

    # Create appendix file # ensure they are also deleted at the
        ↪ start of every
    # run.
    create_appendix_manager_files(hd)

    # TODO: Sort main files.
    export_python_project_code(
        hd,
        normalised_root_dir,
        sort_filepaths_by_filename(python_project_code_filepaths),
    )
    if include_export_code:
        export_python_export_code(
            hd,
            normalised_root_dir,
            sort_filepaths_by_filename(python_export_code_filepaths),
        )


def get_compiled_notebook_paths(script_dir):
    """Returns the list of jupiter notebook filepaths that were
        ↪ compiled
    successfully and that are included in the same dias this script (
        ↪ the src
    directory).

    :param script_dir: absolute path of this file.
    """
    notebook_filepaths = get_all_files_in_dir_and_child_dirs(
        ".ipynb", script_dir
    )
    compiled_notebook_filepaths = []

    # check if the jupyter notebooks were compiled
    for notebook_filepath in notebook_filepaths:

        # swap file extension
        notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf
            ↪ ")

        # check if file exists
        if os.path.isfile(notebook_filepath):
            compiled_notebook_filepaths.append(notebook_filepath)
    return compiled_notebook_filepaths


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

```python
"""This script automatically compiles the text files representing a PlantUML.

# diagram into an actual figure.

# To compile locally manually:
# pip install plantuml
# export  PLANTUML_LIMIT_SIZE=8192
# java -jar plantuml.jar -verbose sequenceDiagram.txt
"""

import os
import subprocess  # nosec
from os.path import abspath

from src.export_data.helper_dir_file_edit import (
    get_dir_filelist_based_on_extension,
)
from src.export_data.plantuml_get_package import got_java_file


def compile_diagrams_in_dir_relative_to_root(
    await_compilation,
    extension,
    jar_path_relative_from_root,
    input_dir_relative_to_root,
    verbose,
):
    """Loops through the files in a directory and exports them to the latex.

    /Images directory.

    Args:
    :param await_compilation: Make python wait until the PlantUML compilation
    is completed. param extension: The filetype of the text file that is
    converted to image.
    :param jar_path_relative_from_root: The path as seen from root towards the
    PlantUML .jar file that compiles .uml files to .png files.
    :param verbose: True, ensures compilation output is printed to terminal,
    False means compilation is silent.
    :param extension: The file extension that is used/searched in this
    function.
    :param input_dir_relative_to_root: The directory as seen from root
    containing files that are modified in this function.

    Returns:
        Nothing

    Raises:
        Nothing
    """
    # Verify the PlantUML .jar file is gotten.
    got_java_file(jar_path_relative_from_root)
```

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

```
        f"abs_jar_path={abs_jar_path},"
        + f" abs_diagram_filepath={abs_diagram_filepath}\n\n"
    )
    bash_diagram_compilation_command = (
        f"java -jar {abs_jar_path} -verbose {abs_diagram_filepath}"
    )
    print(
        f"bash_diagram_compilation_command={
            ↪ bash_diagram_compilation_command}"
    )
    # Generate global variable specifying max image width in pixels,
       ↪ in the
    # shell that compiles.
    os.environ["PLANTUML_LIMIT_SIZE"] = "16192"

    # Perform PlantUML compilation locally.
    if await_compilation:
        if verbose:
            subprocess.call(
                bash_diagram_compilation_command, shell=True  # nosec
                    ↪  B602
            )
        else:
            subprocess.call(
                bash_diagram_compilation_command,
                shell=True,  # nosec
                stderr=subprocess.DEVNULL,
                stdout=subprocess.DEVNULL,
            )  # nosec
    else:
        if verbose:
            with subprocess.Popen(
                bash_diagram_compilation_command,
                shell=True,  # nosec
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
            ) as process:
                get_output_of_bash_command(process)
        else:
            with subprocess.Popen(
                bash_diagram_compilation_command,
                shell=True,  # nosec
                stderr=subprocess.DEVNULL,
                stdout=subprocess.DEVNULL,
            ) as process:
                get_output_of_bash_command(process)


def get_output_of_bash_command(process, verbose=True):
    """Returns the output of a bash command."""
    stdout, stderr = process.communicate()
    result = stdout.decode("utf-8")
    if verbose:
        print(f"result={result}")
        print(f"stderr={stderr}")
    return result


def assert_diagram_compilation_requirements(
    jar_path_relative_from_root,
    relative_filepath_from_root,
):
```

```
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

```python
"""This script generates PlantUML diagrams and outputs them as .uml files."""

import os
from os.path import abspath

from src.export_data.helper_dir_file_edit import (
    create_dir_relative_to_root_if_not_exists,
    dir_relative_to_root_exists,
)


def generate_all_dynamic_diagrams(output_dir_relative_to_root):
    """Manages the generation of all the diagrams created in this file.

    Args:
    :param output_dir_relative_to_root: Relative path as seen from the root dir
    of this project, to which modified files are outputted.

    Returns:
        Nothing

    Raises:
    """
    # Create a example Gantt output file.
    filename_one, lines_one = create_trivial_gantt("trivial_gantt.uml")
    output_diagram_text_file(
        filename_one, lines_one, output_dir_relative_to_root
    )

    # Create another example Gantt output file.
    filename_two, lines_two = create_trivial_gantt("another_trivial_gantt.uml")
    output_diagram_text_file(
        filename_two,
        lines_two,
        output_dir_relative_to_root,
    )


def output_diagram_text_file(filename, lines, output_dir_relative_to_root):
    """Gets the filename and lines of an PlantUML diagram, and writes these to
    a file at the relative output path.

    Args:
    :param filename: The filename of the PlantUML Gantt file that is being
    created.
    :param lines: The lines of the Gantt chart PlantUML code that is being
    written to file.
    :param output_dir_relative_to_root: Relative path as seen from the root
    dir of this project, to which modified files are outputted.

```

```
    Returns:
        Nothing

    Raises:
        Exception if input file does not exist.
    """
    abs_filepath = abspath(f"{output_dir_relative_to_root}/{filename}
       ↪ ")

    # Ensure output directory is created.
    create_dir_relative_to_root_if_not_exists(
       ↪ output_dir_relative_to_root)
    if not dir_relative_to_root_exists(output_dir_relative_to_root):
        raise Exception(
            "Error, the output directory relative to root:"
            + f"{output_dir_relative_to_root} does not exist."
        )

    # Delete output file if it already exists.
    if os.path.exists(abs_filepath):
        os.remove(abs_filepath)

    # Write lines to file.
    with open(abs_filepath, "w", encoding="utf-8") as f:
        for line in lines:
            f.write(line)
        f.close()

    # Assert output file exists.
    if not os.path.isfile(abs_filepath):
        raise Exception(f"The input file:{abs_filepath} does not
           ↪ exist.")


def create_trivial_gantt(filename):
    """Creates a trivial Gantt chart.

    Args:
    :param filename: The filename of the PlantUML diagram file that
       ↪ is being
    created.

    Returns:
        The filename of the PlantUML diagram, and the lines of the
           ↪ uml content
        of the diagram

    Raises:
        Nothing
    """
    lines = []
    lines.append("@startuml\n")
    lines.append("[Prototype design] lasts 15 days\n")
    lines.append("[Test prototype] lasts 10 days\n")
    lines.append("\n")
    lines.append("Project starts 2020-07-01\n")
    lines.append("[Prototype design] starts 2020-07-01\n")
    lines.append("[Test prototype] starts 2020-07-16\n")
    lines.append("@enduml\n")
    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
        ↪ being
112     created.
113 
114     Returns:
115         The filename of the PlantUML diagram, and the lines of the
            ↪ uml content
116         of the diagram
117 
118     Raises:
119         Nothing
120     """
121     lines = []
122     lines.append("@startuml\n")
123     lines.append("[EXAMPLE SENTENCE] lasts 15 days\n")
124     lines.append("[Test prototype] lasts 10 days\n")
125     lines.append("\n")
126     lines.append("Project starts 2022-07-01\n")
127     lines.append("[Prototype design] starts 2022-07-01\n")
128     lines.append("[Test prototype] starts 2022-07-16\n")
129     lines.append("@enduml\n")
130 
131     return filename, lines
```

U Appendix src/export_data/plantuml_get_package.py

```python
"""Downloads the PlantUML package if it does not yet exist."""
import os
import subprocess  # nosec

import requests


def check_if_java_file_exists(relative_filepath):
    """Safe check to see if file exists or not.

    Args:
    :param relative_filepath: Path as seen from root towards a file.

    Returns:
        True if a file exists.
        False if a file does not exists.

    Raises:
        Nothing
    """

    return os.path.isfile(relative_filepath)


def got_java_file(relative_filepath):
    """Asserts if PlantUML .jar file exists. Tries to download is one time if
    it does not exist at the start of the function.

    Args:
    :param relative_filepath: Path as seen from root towards a file.

    Returns:
        True if a file exists.

    Raises:
        Exception if the PlantUML .jar file does not exist after downloading
        it.
    """
    # Check if the jar file exists, curl it if not.
    if not check_if_java_file_exists(relative_filepath):
        # The java file is not found, curl it
        request_file(
            "https://sourceforge.net/projects/"
            + "plantuml/files/plantuml.jar/download",
            relative_filepath,
        )
    # Check if the jar file exists after curling it. Raise Exception if it is
    # not found after curling.
    if not check_if_java_file_exists(relative_filepath):
        raise Exception(f"File:{relative_filepath} is not accessible" )
    print("Got the PlangUML Java file.")
    return True


def request_file(url, output_filepath):
    """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)  # nosec
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

```
"""Exports the generated PlantUML diagrams to the latex Images directory."""
import os.path
import shutil

from src.export_data.helper_dir_file_edit import (
    create_dir_relative_to_root_if_not_exists,
    get_dir_filelist_based_on_extension,
)


def export_diagrams_to_latex(
    input_dir_relative_to_root, extension, output_dir_relative_to_root
):
    """Loops through the files in a directory and exports them to the latex.

    /Images directory.

    :param dir: The directory in which the Gantt charts are being searched.
    :param extension: The file extension that is used/searched in this
    function. The filetypes that are being exported.
    :param input_dir_relative_to_root: Relative path as seen from the root dir
    of this project, containing files that modified in this function.
    :param output_dir_relative_to_root: Relative path as seen from the root dir
    of this project, to which modified files are outputted.
    """
    diagram_filenames = get_dir_filelist_based_on_extension(
        input_dir_relative_to_root, extension
    )
    if len(diagram_filenames) > 0:
        # Ensure output directory is created.
        create_dir_relative_to_root_if_not_exists(output_dir_relative_to_root)

    for diagram_filename in diagram_filenames:
        diagram_filepath_relative_from_root = (
            f"{input_dir_relative_to_root}/{diagram_filename}"
        )

        export_gantt_to_latex(
            diagram_filepath_relative_from_root, output_dir_relative_to_root
        )


def export_gantt_to_latex(
    relative_filepath_from_root, output_dir_relative_to_root
):
    """Takes an input filepath and an output directory as input and copies the
    file towards the output directory.

    :param relative_filepath_from_root: param output_dir_relative_to_root:
```

```
    :param output_dir_relative_to_root: Relative path as seen from
       ↪ the root dir
    of this project, to which modified files are outputted.

    Returns:
        Nothing.

    Raises:
        Exception if the output directory does not exist.
        Exception if the input file is not found.
    """
    if os.path.isfile(relative_filepath_from_root):
        if os.path.isdir(output_dir_relative_to_root):
            shutil.copy(
                relative_filepath_from_root,
                   ↪ output_dir_relative_to_root
            )
        else:
            raise Exception(
                f"The output directory:{output_dir_relative_to_root}
                   ↪ does"
                + " not exist."
            )
    else:
        raise Exception(
            f"The input file:{relative_filepath_from_root} does not
               ↪ exist."
        )
```

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. 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. 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. 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. Accessed: 2021-07-11.