

**Campus Meal Ordering System**

**Project Plan**

**By *Team Foodie***

**Lab Group: TS3**

**Date: October 2020**

**Instructors:**

**Bo An**

**Leong Yin Yoke Junie**

**Qiu Wei**

**Wang Rundong**

# **SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**NANYANG TECHNOLOGICAL UNIVERSITY**

[**Introduction**](#_l419sw4ma3hb) **3**

[Project Overview](#_km5rnm7qeu3q) 3

[Project Description and Scope](#_dsy4w51k2vyq) 3

[**Project Organization**](#_kest5w9ut8u1) **3**

[Team Structure](#_c2zll7b8pcmw) 3

[Roles and Responsibilities](#_cq1hy4v2we2j) 3

[Team Communication](#_z1yzxuaikl72) 4

[**Process Definition**](#_k5gwh1puqnu9) **4**

[Lifecycle Model](#_3524iqoara4l) 4

[**Schedule**](#_2b3lffev4g71) **5**

[Activity Dependencies and Schedule](#_b3377ixmjh0k) 5

[Work Breakdown Structure](#_pw7zw3kju1us) 5

[Work Packages](#_uk3qv8hh35y9) 5

[Activity Dependencies](#_5eegohvtv8b) 6

[Work Package Details](#_m778l2mag4d0) 6

[**Project Estimates**](#_cly509hzmcvk) **6**

[Code Size Estimation using Function Points](#_ebpy89nwlnz5) 6

[Unadjusted Function Points](#_c9g0uc4p4fr) 6

[Adjusted Function Points](#_mrdiewj2eflm) 7

[Lines of Code](#_63zfhbw1zm2r) 8

[Efforts, Duration, and Team Size Estimation](#_oam98xprn041) 9

[Distribution of Effort](#_b7p62kxj9xcd) 9

[Cost Estimates](#_nn1pqh3yr65g) 9

[**Product Checklist**](#_4mqotyljktfa) **9**

[**Best Practice Checklist**](#_7pa9xrw4uuyl) **10**

[**Risk Management**](#_kromnuhgb9qm) **11**

[**Quality Assurance**](#_h9epui4h6n7j) **11**

[**Monitoring & Control**](#_1xsjmjep37ex) **12**

# **Introduction**

## **Project Overview**

There are various food delivery applications serving the general public. However, the prices of the food and delivery fees can be prohibitive to students. Therefore, a food delivery application which makes available myriads of foods of the canteens scattered around the campus is needed to improve and enrich the students’ and staff’s dining experience of their daily life.

## **Project Description and Scope**

Campus meal ordering system (CMOS) is a food ordering and delivery application specifically designed to serve the entire campus population. With its mass delivery design in mind, the app provides food ordering and delivery services free of charge. Supported by the Flutter development framework, it is portable on both IOS and Android mobile platforms. Empowered by the firebase backend and database services, it is also highly scalable.

# **Project Organization**

## **Team Structure**

The following is the list of teams

Project Manager: Ma Xiao

Developer Team: He Yu Hao, Han Si Meng, Ma Xiao, Yeoh Jun Yi, Loh Yi Xuan Renice

Quality Assurance Team: Yeoh Jun Yi, Loh Yi Xuan Renice

Release Engineering: Han Si Meng

## **Roles and Responsibilities**

Project Manager: Ma Xiao

* Makes the project proposal, project planning and scheduling.
* Oversees project progress
* Assigns tasks and organize meetings and discussions
* Manages and motivates team members

System Architect / Lead Developer: He Yu Hao

* Analyses requirements and comes up with software architecture.
* Provides detailed design of the system, including Database, APIs, etc.

Frontend Developer: Han Si Meng, Ma Xiao, Loh Yi Xuan Renice

* Work on the client side development using Flutter.

Backend Developer: He Yu Hao, Yeoh Jun Yi

* Implement product based on detailed design document

Quality Assurance Manager: Loh Yi Xuan Renice

* Helps with the software quality assurance plan and risk management plan
* Ensures acceptable software quality
* Designs testing strategies
* Executes the test procedures

Quality Assurance Engineer: Yeoh Jun Yi

* Makes the software quality assurance plan
* Makes the risk management plan Ensures acceptable software quality
* Creates and manages test plan
* Verifies software requirements

Release Engineer: Han Si Meng

* Makes software release plan and configurations

## **Team Communication**

Team “Foodie” communication channels include the following:

* Weekly meetings are held on Monday.
* Group announcements are sent through project WhatsApp Group
* Online Zoom discussions or face to face meetings are held as necessary.
* Split up into subgroups as necessary, in order to work more cooperatively on specific problems.

# **Process Definition**

## **Lifecycle Model**

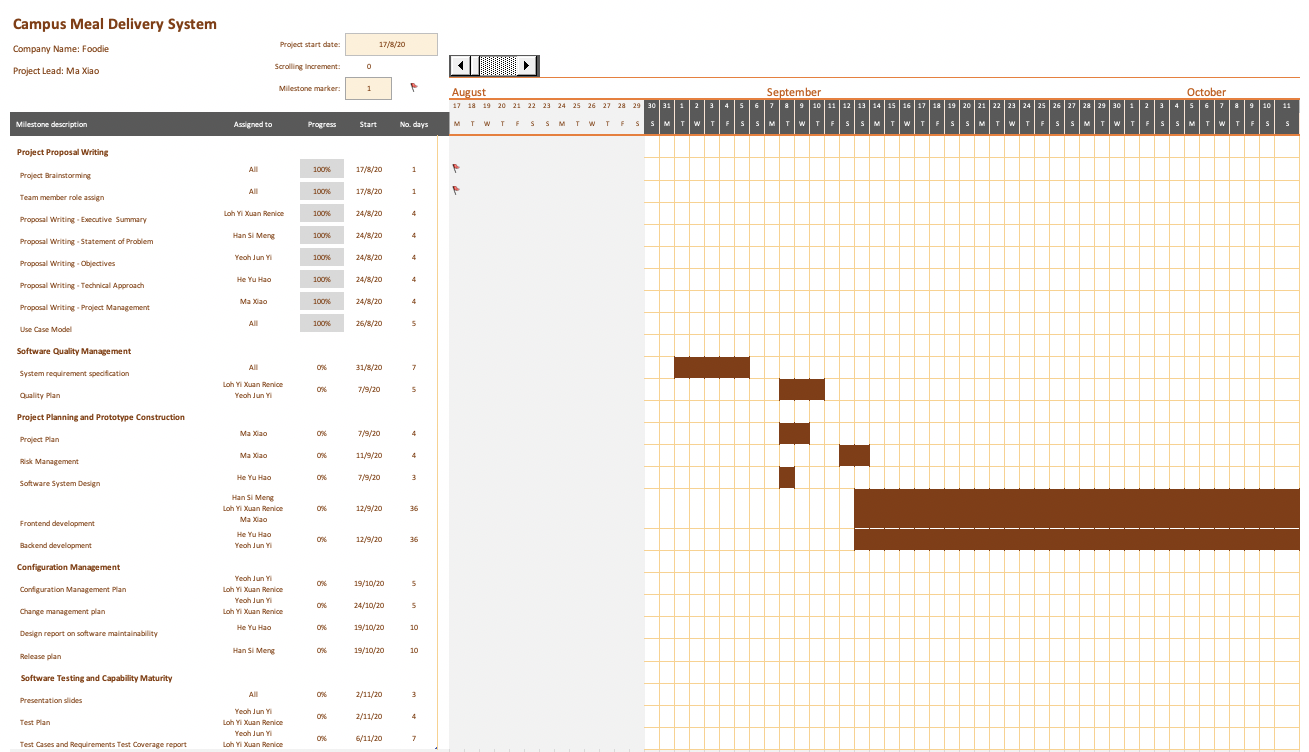
Since our application is small and the complexity is relatively low, and the technology used is well developed and up-to-date, the agile lifecycle is a good choice. The adaptive software development (ASD), a model from agile, is used throughout this project. Three important cycles in ASD are speculate, collaborate and learn. We need to consider system requirements, objective, scope and more, which speculate means, and do teamwork and make efforts together, which collaborate means, and also learn and pick up knowledge along the way, which learn means.

ASD intends to use iterative development, which is more flexible than waterfall. In every iteration of ASD, new design, coding, unit testing, integration and quality assurance are implemented. This is more tolerant to changes and ensures good progress. It can deliver new features and functions at each iteration, and revisit and refine at later rounds.

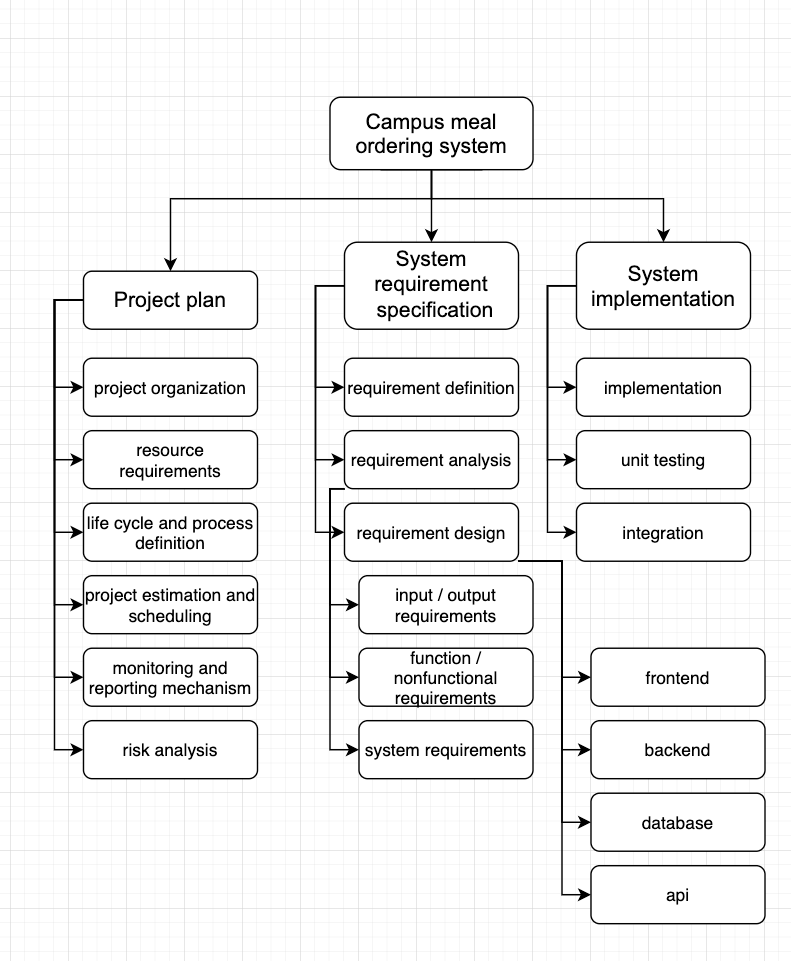
ASD allows us to learn throughout this project, which is important for students like us to pick up knowledge on both theoretical parts and implementation parts. Learning curves is more flat as we can learn each iteration.

# **Schedule**

## **Activity Dependencies and Schedule**



## **Work Breakdown Structure**

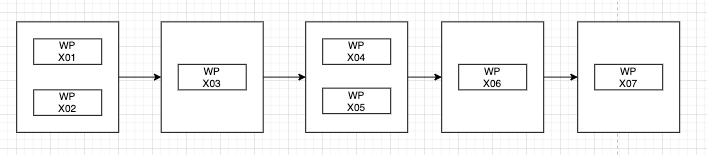


## **Work Packages**

* Project Proposal
* Requirement Specification
* Software System Design
* Frontend Services
* Backend Services
* Management Plan
* Test & Integration

## **Activity Dependencies**

| Work Package # | Work Package Description | Duration | Dependencies |
| --- | --- | --- | --- |
| X01 | Project Proposal | 14 days | - |
| X02 | Requirement Specification | 7 days | - |
| X03 | Software System Design | 5 days | X01, X02 |
| X04 | Frontend Services | 36 days | X03 |
| X05 | Backend Services | 36 days | X03 |
| X06 | Management Plan | 4 days | X04, X05 |
| X07 | Test & Integration | 11 days | X06 |



## **Work Package Details**

| Project | Campus meal ordering system | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Work Package | X01 | X02 | X03 | X04 | X05 | X06 | X07 |
| Assigned To | All | All | He Yuhao, Ma Xiao, Yeoh Jun Yi | Han Simeng, Loh Yi Xuan Renice, Ma Xiao | He Yuhao, Ma Xiao, Yeoh Jun Yi | All | Han Simeng, Loh Yi Xuan Renice |
| Effort | 14 PD | 7 PD | 5 PD | 36 PD | 36 PD | 4 PD | 11 PD |
| Start Date | 17/8/20 | 31/8/20 | 7/9/20 | 12/9/20 | 12/9/20 | 19/10/20 | 2/11/20 |
| Purpose | Determine overview | Elicitate requirements | Build technical architecture | Build frontend | Build backend | Plan process management | Test components and integrate |
| Inputs | None | Requirements | Requirements document | Software architecture | Software architecture | Management plan and software | Software |
| Activities | Provide a brief overview of the project, its objectives and proposed project deliverables, and scopes are defined. | Get requirements and analyse them. Provide features or functions that need to be implemented. | Build overall architecture, including frontend, backend, database, and apis for communications. | Design user interfaces and provide need for the backend process. | Design backend service apis to fulfill frontend needs. Save and provide data. | Check if software built is going well and fulfill expected needs. Feedbacks are used to update management plans. | Create unit testing for each component, and integrate the frontend and backend. |
| Outputs | Proposal document | Requirements document | Software architecture | User interface | Backend services | None | Integrated software |

# **Project Estimates**

## **Code Size Estimation using Function Points**

We calculated unadjusted function points based on the complexity of functions provided by this system. Code size is then estimated by the adjusted function point.

### **Unadjusted Function Points**

CMOS contains 3 applications and each supports the following proposed functions:

Client Application:

* Registration and login
* Browse promotion
* Browse restaurants
* Browse restaurant
* Browse dish
* Add dish to cart
* Browse cart and check out

Vendor Application:

* Registration and login
* Browse placed orders
* Confirm/reject an order

Deliveryman Application:

* Registration and login
* Browse orders to pick up
* Browse orders to deliver
* Browse orders delivered
* Update orders status

The measure of unadjusted function points is based on five primary component elements of these functions: Inputs, Outputs, Inquiries, Logical Files, and Interfaces. Each element ranges from Low Complexity, Medium Complexity to High Complexity. The detailed evaluation of the complexity is as follows:

Inputs:

| Item | Complexity |
| --- | --- |
| Gathering order information (Selected restaurant, selected dish, quantity, delivery time, delivery location) | High |

Outputs:

| Item | Complexity |
| --- | --- |
| Display a list of restaurants | Medium |
| Display a list of dishes in a restaurant | Medium |
| Display detailed information of a dish | Low |
| Display order history | Medium |
| Display cart | High |
| Display promotions | Low |
| Display a list of requested orders to vendor | Low |
| Display a list of orders to pickup to deliveryman | Low |
| Display a list of orders to deliver to deliveryman | Low |
| Display a list of orders delivered to deliveryman | Low |

Inquires:

| Item | Complexity |
| --- | --- |
| Registration and login | Medium |
| Add item to cart | Medium |
| Vendor confirm or reject an order | Low |
| Delivery man updates the delivery status of the order to picked up | Low |
| Delivery man updates the delivery status of the order to delivered | Low |
| Display a list of orders delivered to deliveryman | Low |

Logical Files:

| Item | Complexity |
| --- | --- |
| Handling registration and login, access control | Medium |
| Handling order placement | High |
| Handling order status update | Medium |

Interfaces:

| Item | Complexity |
| --- | --- |
| Google OAuth2 authentication | Medium |

Calculation of Unadjusted Function Points:

| **Characteristic** | **Low** | | **Medium** | | **High** | |
| --- | --- | --- | --- | --- | --- | --- |
| Inputs | 0 | × 3 | 0 | × 4 | 1 | × 6 |
| Outputs | 6 | × 4 | 3 | × 5 | 1 | × 7 |
| Inquiries | 4 | × 3 | 2 | × 4 | 0 | × 6 |
| Logical Files | 0 | × 7 | 2 | × 10 | 1 | × 15 |
| Interfaces | 0 | × 5 | 1 | × 7 | 0 | × 10 |
| **Unadjusted FP** | 36 |  | 50 |  | 28 |  |
| **Total=L+M+H** | 116 | | | | | |

### **Adjusted Function Points**

| **Influence Factors** | **Score** | **Detail** |
| --- | --- | --- |
| Data Communications | 5 | Application is more than a front-end, but a lot of data will be processed through the backend. There are multiple frontend communicate and processing the same data |
| Distributed Functions | 4 | Distributed processing and data transfer are online and in both directions. |
| Performance | 5 | Response time and throughput is critical during all business hours. It is directly related to the user experience. |
| Heavily used | 3 | The system will only be heavily used during the meal ordering time |
| Transaction rate | 3 | Daily peak transaction period can be anticipated. |
| On-line data entry | 1 | Seldom online data entry in the system |
| End-user efficiency | 4 | The system is to be designed to be used efficiently |
| On-line data update | 1 | Seldom online data update |
| Complex processing | 1 | Seldom complex processing work in the system |
| Reusability | 4 | The application was specifically packaged and/or documented to ease re-use, and the application is customized by the user at source code level. |
| Installation Ease | 0 | The application installation will be taken care of by Google Play store and Apple Store |
| Operational Ease | 0 | No requirements on operational ease |
| Multiple sites | 5 | User may need install multiple different clients version to experience different roles in the system, like client, vendor and delivery man |
| Facilitate change | 0 | No requirement on facilitate change |
| Total score | 36 | |
| **Influence Multiplier** = Total score × 0.01 + 0.65 = 36 × 0.01 + 0.65 = 1.01 | | |
| **Adjusted FP** = Unadjusted FP × Influence Multiplier = 116 × 1.02 = 117.16 | | |

Note the meaning of the scoring:

| Score | Meaning |
| --- | --- |
| 0 | No influence |
| 1 | Insignificant influence |
| 2 | Moderate influence |
| 3 | Average influence |
| 4 | Significant influence |
| 5 | Strong influence |

### **Lines of Code**

We use Dart language as the project’s programming language.

According to Capers Jones statistics, there is no data for language Dart.

Dart has a very similar complexity as Java, which is 53 lines of code per function point.

Therefore, we have: Lines of Code = 117.16 FP × 53 LOC/FP = **6209.48 LOC**

## **Efforts, Duration, and Team Size Estimation**

We use the Functional Point (FP) model to estimate the effort, duration and team size required for the project.

* Working days include 5 days in a week.
* Effort = Size / Production Rate = (6209.48 LOC) / (120 LOC/PD) = 51.74 PD
* Duration = 3 × (Effort) 1/3 = 3 × (51.74) 1/3 = 11.17 Days
* Team size = 51.74 PD / 11.17 D = 4.63 P = 5 Persons

## **Cost Estimates**

| **Category** | **Item** | **Supplier** | **Quantity** | **Unit Price** | **Total** |
| --- | --- | --- | --- | --- | --- |
| Personnel | Project manager | N.A. | 1 | $15,000.00 | **$15,000.00** |
| Project team members | 4 | $5,000.00 | **$20,000.00** |
| Equipment | Computers | Apple | 5 | $3,499.00 | **$17,495.00** |
| Technologies | Firebase Service | Google | 1 | $5,000.00 | **$5,000.00** |
| Android Developer Account | Google | 1 | $25.00 | **$25.00** |
| Apple Developer Account | Apple | 1 | $299.00 | **$299.00** |
| Utility costs | Office rental | NTU | 1 | $6,000.00 | **$6,000.00** |
| Transportation | Grab | 1 | $1,000.00 | **$1,000.00** |
| **TOTAL** | | | | | **$64,819.00** |

# **Product Checklist**

The plan is that the items listed below will be delivered on the stated deadlines.

| **Project Deliverable** | **Estimated Deadline** |
| --- | --- |
| Project Proposal | 31 Aug 2020 |
| System Requirements Specifications | 14 Sep 2020 |
| Quality Plan | 14 Sep 2020 |
| Project Plan | 19 Oct 2020 |
| Risk Management | 19 Oct 2020 |
| Design report on software maintainability | 19 Oct 2020 |
| Configuration Management Plan | 2 Nov 2020 |
| Change management plan | 2 Nov 2020 |
| Release plan | 2 Nov 2020 |
| Entire software product | 2 Nov 2020 |
| Test Plan | 16 Nov 2020 |
| Test Cases and Requirements Test Coverage report | 16 Nov 2020 |
| CMMI level 2 definition | 16 Nov 2020 |

# **Best Practice Checklist**

* Documentation on which the project is based must be recorded
  + Requirements should be thoroughly checked for completeness, accuracy, and consistency. Software requirement specification should be documented in detail.
  + The requirements should be checked by the manager to ensure it aligns with the stakeholder requirements.
* The software design should take into consideration complexity management.
  + Coupling and interfaces between the modules, procedures and data should be reduced.
  + Reduce repetitive code or functionality to reduce the code size.
* Budgets and resources should be estimated reasonably.
* Don’t under-estimate the time. Effort on integration, testing, documentation and maintenance should be estimated realistically.
* Management of source code should be on a version control collaboration platform to ensure effective collaboration among team members and maintain traceability.
  + The code should be tested before making a commit.
  + Every commit should contain only one modification.
  + Every commit should contain a description.
  + A full working instance can be made by checking out and following the steps in README.md
  + Code review should be done between the team members to find errors effectively.
* A dashboard should be maintained to display a high-level overview of the task progress and project milestones.
  + This should be visible to the developers and managers.
  + The managers should check the dashboard to review design appropriateness and communicate with the team frequently.
* A plan on quality control should be made.
  + Metrics should be established to evaluate the quality and determine if a milestone has been met or the project is ready to be released.
* Testing should be done rigorously
  + Testing plan should be done as the project development proceeds.
  + Software testing includes black-box and white-box testing, which involves unit testing, integration testing, acceptance testing, functional testing, system testing, and performance testing.
  + An automated testing tool should be developed and run regularly.
* Operations and maintenance should be executed when the software is deployed.
  + An automated monitoring system should be in place to report errors and notify the developers.
  + Developers should be assigned for recording and fixing errors.
  + Modifications to the code and configuration should be recorded.

# **Risk Management**

The process below is followed accordingly to manage risk in the project. The specific procedures and details shall be provided in the **Risk Management Plan**.

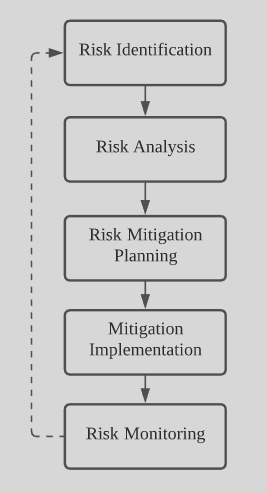


Figure: Risk Management Process

# **Quality Assurance**

The project will achieve the quality assurance by following the standard set by the company. The specific and details shall be provided in the **Quality Plan**.

Specific test procedures and details will be provided in the **System Test Plan**.

The CMOS project will utilise the following testing methodologies:

**Functional Testing**

* Unit Testing
  + Involves testing system components individually to ensure that they are working as required
* Integration Testing
  + Involves testing simultaneously and continuously when components are combined or new components are added
* In-Place Testing
  + Involves testing of whole system as a unit

**Non-Functional Testing**

* Acceptance/Beta Testing
  + Conducted to verify CMOS meets client’s specific requirements
  + Requires client to perform Software Testing in real environment to ensure that there are no major failures and it satisfies the business requirements from an end-user perspective
* Performance Testing
  + Involves determining CMOS’s behaviour and performance in terms of responsiveness, speed and stability under a workload
  + Allow the team to determine the maximum workload a system can handle without performance degradation

These methodologies will be used to test two important aspects of the CMOS:

* System Function
  + To ensure that functional requirements are met and software flaws are eliminated
* Algorithm Function
  + To ensure that heuristic aspects of the application such as preference ranking and filter performs accurately

# **Monitoring & Control**

Here are some of the procedures need to take to successfully monitor the progress of a software project:

**Monitoring resource consumption**: Monitoring the project team’s resource usage in terms of human resources, cloud computation resources usage. These will provide a quantitative measurement of the progress of the project.

**Project risk identification**: Early identification of major risks to the project allows for placement of preventative measures before problems can develop. Major risks have been identified in the Risk Management section of this document, along with the measures being taken to avoid them.

**Regular reviews of project progress**: During the development of the Campus Meal Ordering System, the development team shall meet weekly to review the progress of all project tasks, including project management, progress analysis, development issues, and testing.

**Timeline planning and task decomposition**: Estimated timeline of the project is outlined in this document. A more detailed timeline can be obtained by aggregating the subcomponents decompositions hierarchically. Meanwhile, the decomposition will help in adjustment of task assignment.