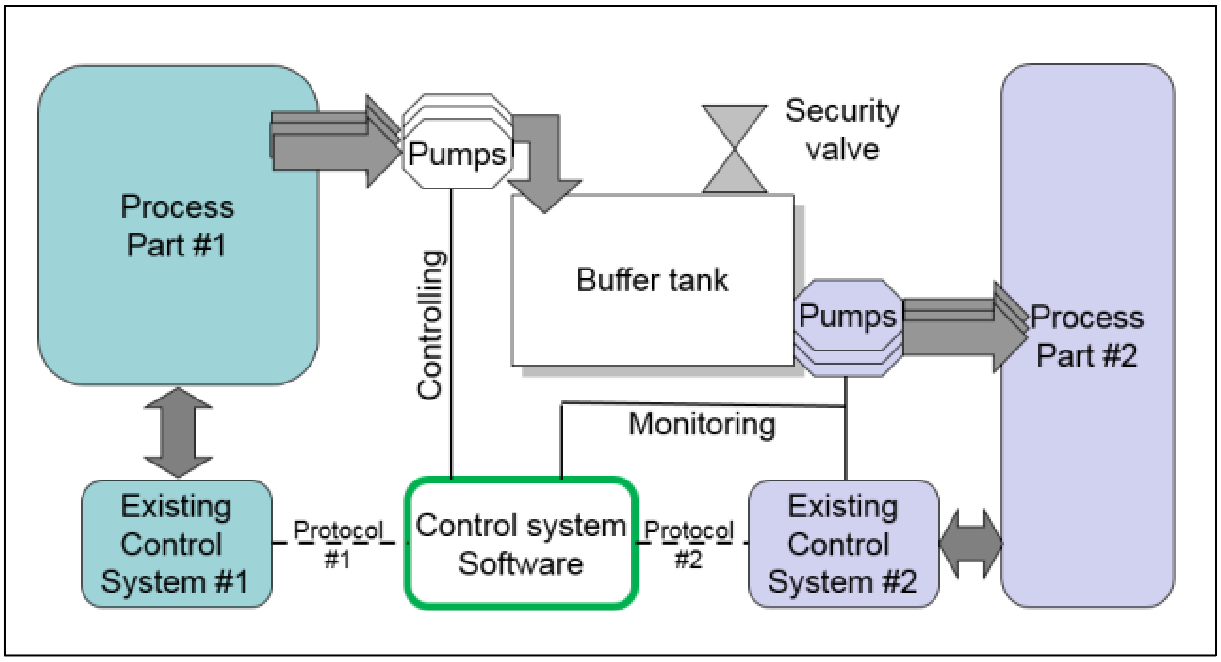
Control System

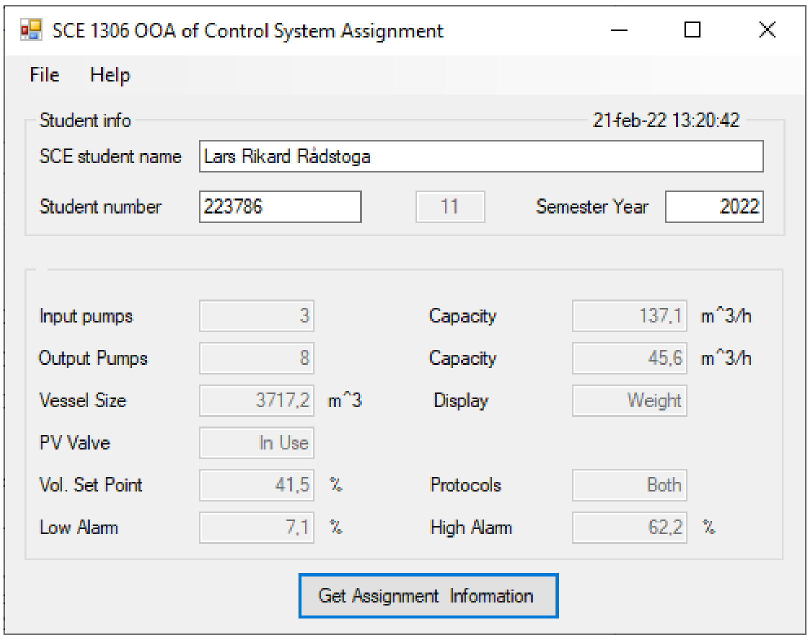
Lars Rikard Rådstoga | 223786

2022-02-22



1 Introduction

The following report contains analysis work for a software system of a control system.
A specification in addition to some generated problem details, given in Figure 1-1, are used as the basis for the report. The system in question will control input pumps to a buffer tank, as seen in Figure 1-2, to steer the volume to a set value. The system also has information available from the existing control system #2 about the output pumps, but no level sensor is available in the buffer tank itself. Requirements will be gathered to analyze the system and then a series of UML diagrams will be created along a fully dressed use case document. Performing such analysis can be very useful as communication tools between the developers and the product owner to understand the needs better.


SCE 1306 OOA of Control System Assignment
File
Help
Get Assignment Information

Student info — 21-feb-22 13:20:42

SCE student name	Lars Rikard Rådstoga				
Student number	223786	11		Semester Year	2022

Input pumps	3		Capacity	137,1	m^3/h
Output Pumps	8		Capacity	45,6	m^3/h
Vessel Size	3717,2	m^3	Display	Weight	
PV Valve	In Use				
Vol. Set Point	41,5	%	Protocols	Both	
Low Alarm	7,1	%	High Alarm	62,2	%

Figure 1-1 Generated details about the assignment.

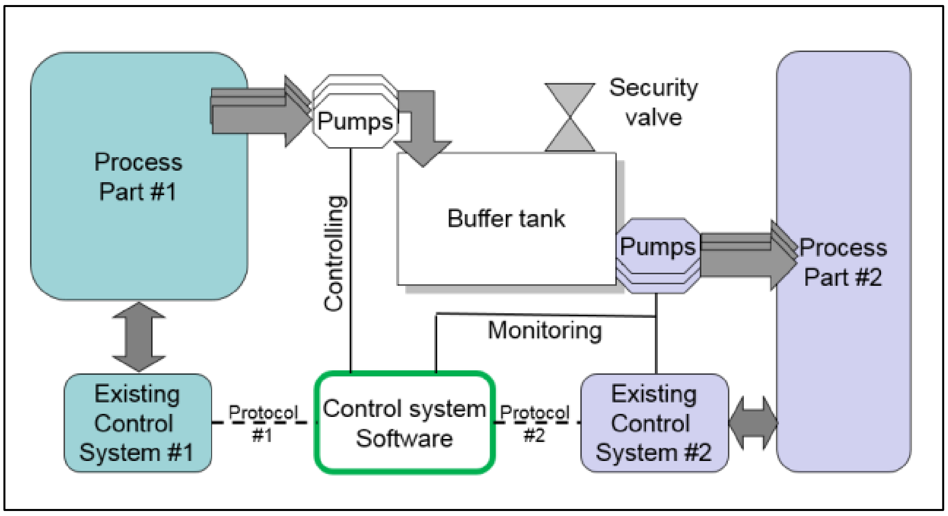


Figure 1-2 System sketch of the control system in its integrated setting. The sub-system concerned by this document are made up of the components with white backgrounds.

2 Requirements

Functional requirements:

- Input pumps shall be controlled.
- A setpoint to the buffer tank volume percentage shall be configurable.
- The system shall communicate with both existing controllers.
- Alarms shall be triggered on defined thresholds.
- Alarms shall be communicated on both protocols.
- The current contents of the tank shall be displayed.
- Alarm outputs for volume extrema shall be displayed.
- Constants such as alarm thresholds, pump capacities, vessel sizes, display units, etc. shall be set by configuring an XML file that is read by the program on startup.

Non-functional requirements:

- Input pump capacity shall be proportional to the control signal to the pumps.
- The output pumps shall not be monitored as their activity is communicated by control system #2.
- The current contents of the tank shall be displayed as the weight.
- The system shall support multiple (3) input pumps.

2.1 Use case

A use case diagram can be generated by the requirements as seen in Figure 2-1. The model displays a relation between which components relates to which actions, this will give a general overview of the system.

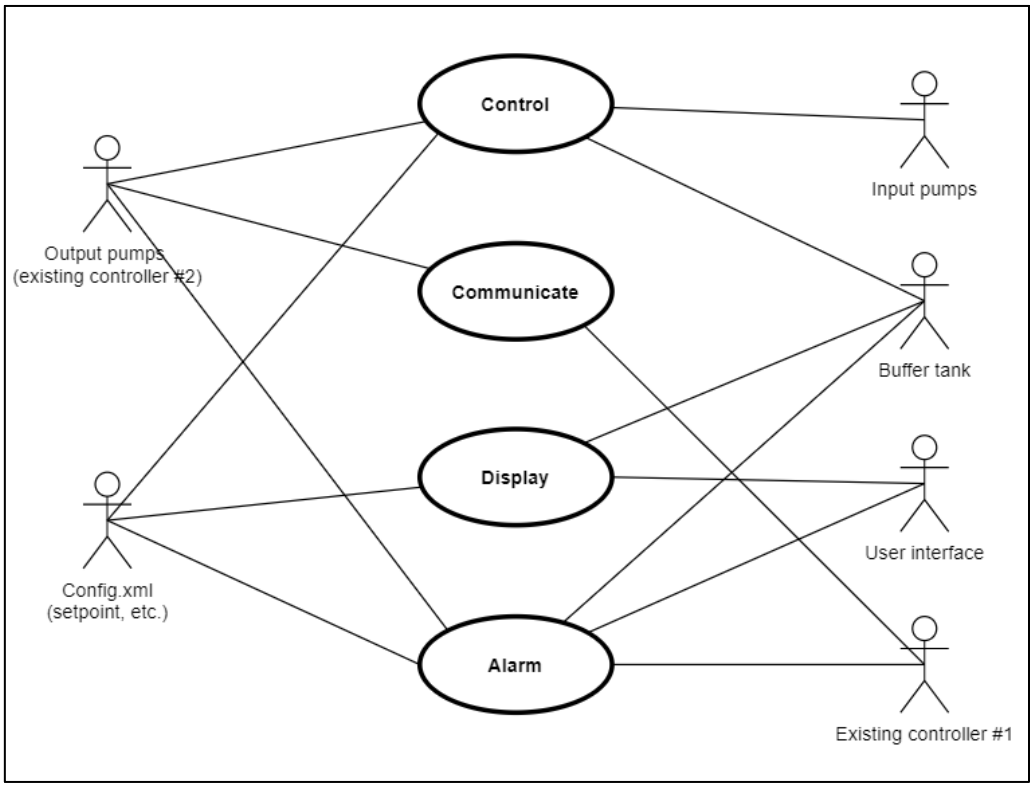


Figure 2-1 Use case diagram showing actors and actions of the system.

2.2 Domain model

A domain model places the real components of the system into a sort of map. This gives an idea of how the components of the system can be represented as classes in the program and how they will relate to each other. The diagram in Figure 2-2 is an attempt at such a diagram but contains perhaps a higher level of detail than required.

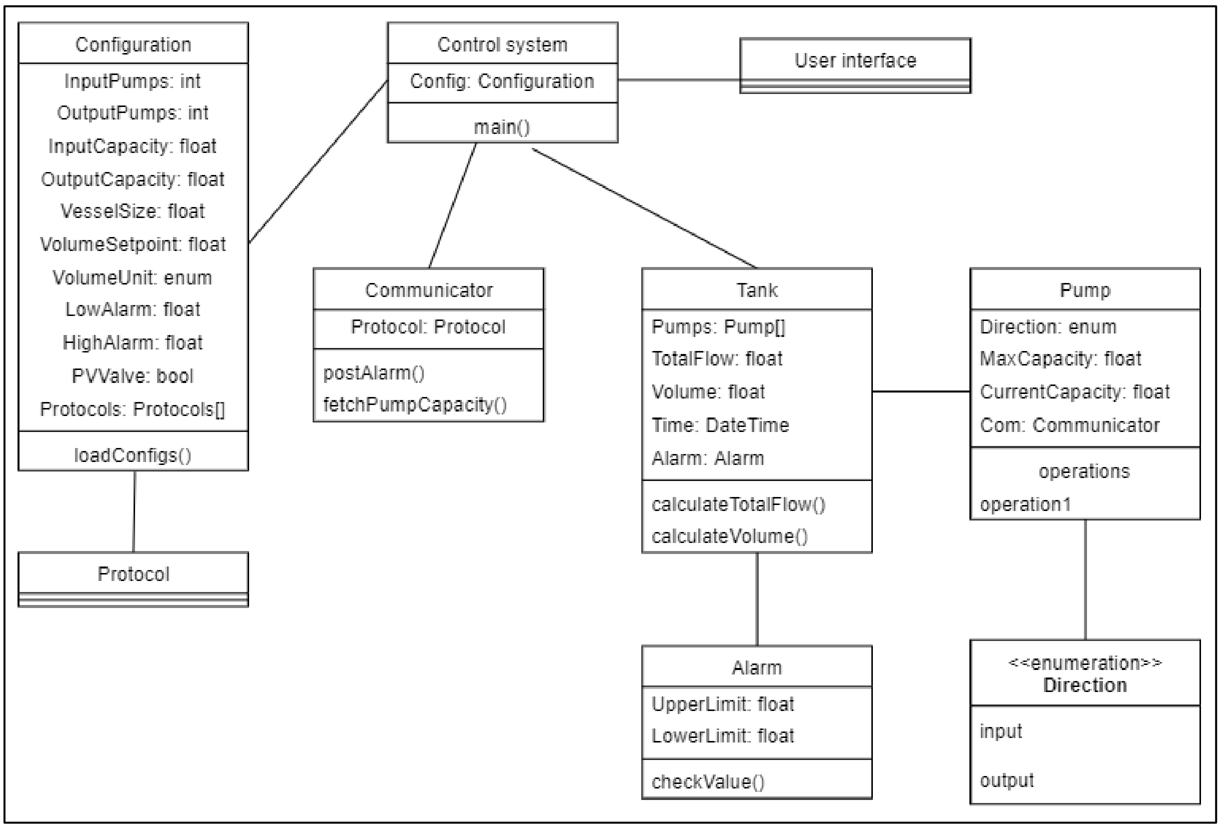


Figure 2-2 A domain model of the system displaying a relation between components of the system.

2.3 Use case analysis

A fully dressed use case document describing the volume control can be found in Table 2-1.

Table 2-1 A fully dressed use case document.

Use case section	*Comment*
Use case name	Control volume.
Scope	Vessel and input pump.
Level	
Primary actor	Computer/Control system.

Stakeholders and interests	Process part #2.
Preconditions	Knowing vessel size, maximum capacities of the pumps, and establishing communication with preexisting control system #2.
Success guarantee	Reading outflow pump signals from preexisting control system #2 and setting capacity of input pumps.
Extension points	Startup.
Main success scenario	1. Get initial volume by user entry, no sensor exists 2. Control loop: 3. Get outflow from control system #2. 4. Calculate error from setpoint. 5. Set inflow on input pumps. 6. wait a short amount of time.
Extensions	3a. Activate an alarm if the outflow isn't communicated. 5a. Activate an alarm if the pump doesn't respond to signal change.
Special requirements	
Technology list	
Frequency of occurrence	Not specified, *should ask the process engineer for the time requirements!*
Miscellaneous	Inaccuracies in the flow measurements/estimations can cause the de facto error to deviate from the estimated error and cause the vessel content volume to become too high/low without the system knowing.

2.4 System sequence diagram

A sequence diagram describing the volume control, visually presenting the volume control use case can be seen in Figure 2-3.

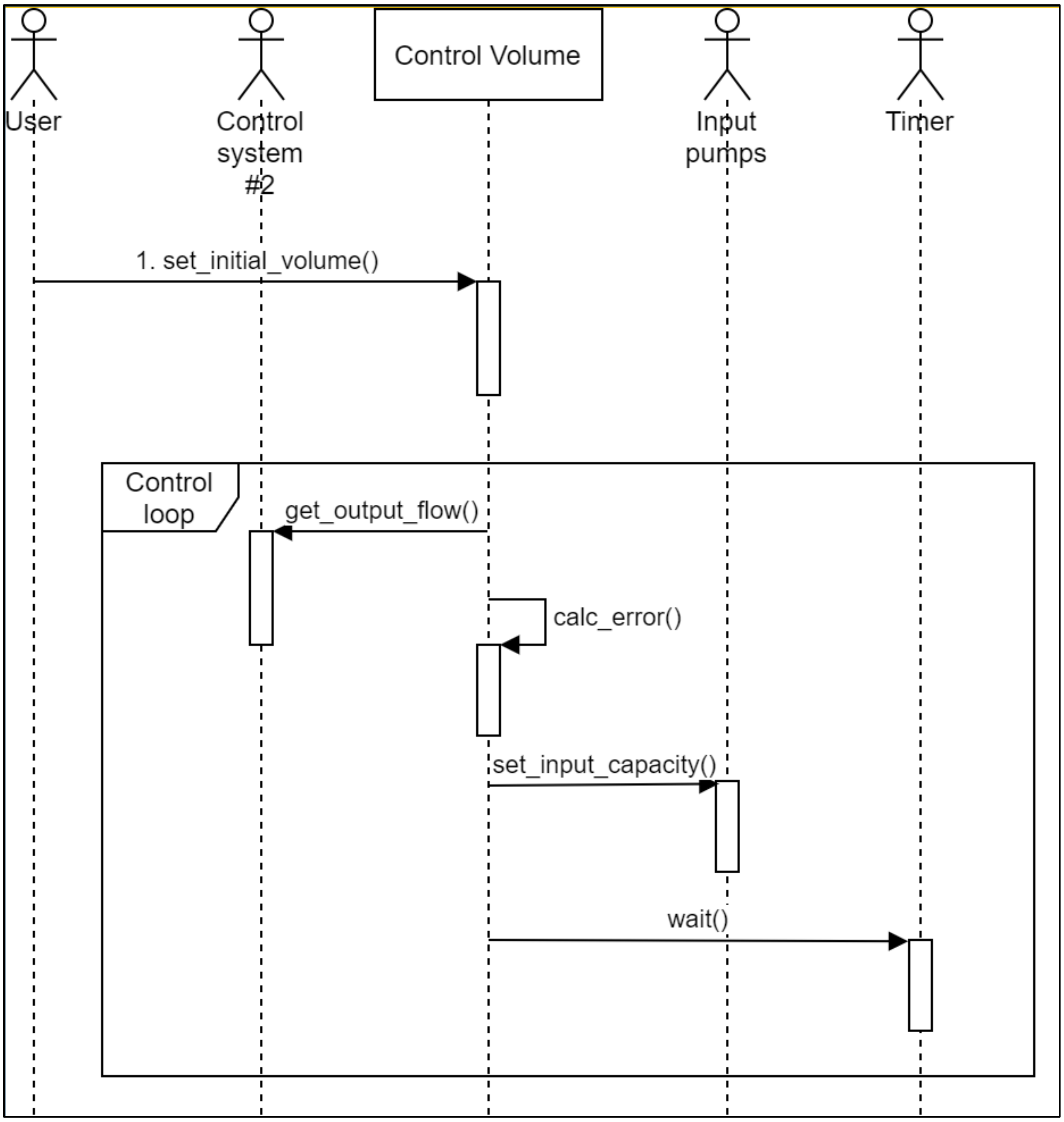


Figure 2-3 A system sequence diagram of the volume control use case.

3 Development process

The following sections will discuss topics regarding the development process: how UP will be used during development, how the contents of the requirements chapter relate to phases of UP, and a time estimation for the project and a discussion on test documents.

3.1 Unified process

The unified process is a constructive and orderly way to progress the development of a system. Dividing the workflow into the four phases: inception, elaboration, construction, and transition. While also working in iterations prioritizing important and risky parts of the system.

Currently the project is in the elaboration phase and is lacking a few details from the inception phase such as the estimated schedule. Following are few sentences regarding which activities that still needs to be or should be done for each phase:

The **inception** phase needs to estimate the schedule and the cost estimate for the project.

The **elaboration** phase still needs to decide on the technology stacks for both the development and production environments. And a bit more on the system architecture: should OPC technology be used for communication or something else. And very importantly the diagrams and documents should be created for every use case.

The **construction** phase should be used to fine tune specifications and implement them. Scrum-like sprint planning meetings should be used to discuss what use cases to work on, the product owner should make the final decisions; The use cases should be discussed using the UML diagrams, confusions should be cleared up in case the developer’s explanations don’t meet the product owner’s specifications. Design, implementation, and testing follows as the main portion of an increment during this phase. At the end of an increment, it is useful to hold a scrum-like sprint review meeting where the use case is tested by the product owner.

In the **transition** phase there should be a verification process indicating that the requirements for the system are fulfilled (acceptance criteria). There should also be documentation for the end user and the administrator of the system: user, installation, and maintenance manuals.

3.2 In relation to the elaboration and construction phases

The work carried out in the requirements chapter of this document relates to both the elaboration and the construction phases. Creating the list of requirements from the specifications is part of the elaboration phase. But as the unified process allows work to be carried out for both requirements gathering and analysis & design in both phases, one could argue that the use case, domain, sequence, class, and object diagrams can all be part of both phases. Therefore, it is safe to create first drafts of these in the elaboration phase and finish them during the construction phase.

3.3 Development time estimation

It is hard to estimate the time of the project when information is still missing, i.e.: What type of computer will the control system run on? What technology stacks are supported by that hardware? How many developers will work on the project, and how much of their work week will be allocated to this project? Do the developers have experience with the technology stack from earlier experience or will there be a bit of a learning period?

Anyway, if we assume that the involved technology is familiar to the developer then we can estimate about two weeks per use case, so a minimum of eight more weeks.

3.4 Indication of test documents

There are multiple ways to test a system, some of them are:

Unit testing, check if the small parts of the system (like a single function) produce the expected outputs matching the inputs it is given, this can be automated. **Integration testing**, check if small parts of the system, when combined produce expected/wanted behavior. **Requirements testing**, check if the produced functionality in the system matches the requirements. **Stress testing**, check how the system behaves to a big user load or otherwise high activity. This type of testing does not suit all systems. **Performance testing**, investigate if the system satisfies the time requirements. **User testing**, test how a typical user of the program interacts with it. Is it easy to learn and use? Is the system behaving as they expected? **Edge case testing**, how does the system handle abnormal inputs like undefined, nulls, etc. Are there ways to crash the system that is still not handled? This can be combined with unit and integration testing, or it can be a separate test if those are automated. [1]

When creating unit tests, which can also be created before developing the actual feature, it can be useful to use the main success scenario in the fully dressed use case document as a guideline as to what to test. And depending on how much you plan to test it can be useful to also address the extensions portion of that document.

When requirements testing it is very handy to have the list of functional and non-functional requirements to compare the solution with. Updating such a list when change of requirements happens is also good practice, as to not cause confusions if requirements are defined at multiple points in time. Testing requirements with an outdated list can cause trouble if multiple people are involved with the project.

4 Summary

Converting specifications of a desired system into requirements is a great way to systematically list the needs of a solution. Such a list is great because it contains testable goals for the system, testing which can be done in both early and late stages of a project. Early in terms of confirming with the product owner that the requirements are understood and late in terms of testing if the product satisfies the set requirements. Creating a use case diagram highlights what actors/parts of the system are involved with which functionalities and the domain model shows how the real-world things can be represented inside of a program and how they relate to each other. These diagrams can quickly be developed from the list of requirements and can be used to further communicate the needs of the solution with the product owner and the development team. The fully dressed use case document and the sequence diagram focus a lot on how the use cases will function in the system, creating these are great planning tools for the developer(s) and function as documentation to investigate if the system is to be improved or expanded at later points in time.

5 References

[1] [Online]. Available: https://www.geeksforgeeks.org/types-software-testing/.