

Process report



Final version

Nikola Nikushev

Bilger Yahov

Lyubomir Dimov

Tao Hua

Table of Contents

[Overall process 2](#_Toc440802671)

[Understandings made with the group and the teacher 5](#_Toc440802672)

[Separation of work 7](#_Toc440802673)

[Choices and problems 9](#_Toc440802674)

[Personal Evaluations 13](#_Toc440802675)

# Overall process

#### Week 1

In week 1, the project was completely new area for the group members. We (the group members) started working on first version of the URS in which we had to present the functional, nonfunctional Requirements and the User Interface. We had our first meeting on Thursday week 1. The teacher gave us feedback on the use cases and nonfunctional requirements. We deleted some of the use cases we had made, combined some of them and worked on improving the document.

For week 1 we had:

* Group meeting (only with the group members) – discussing Use cases and nonfunctional requirements
* Individual work (work was separated between group members)
* Meeting with the teacher presenting First version of URS
* Feedback
* Plan for next week based on feedback
  + UI was needed in the document
  + Some Use-cases had to be combined, some shortened and some deleted. There should be consistency between the final project and the Use-cases
  + Wrong usage of Nonfunctional requirements (expressing qualities with wrong words)

#### Week 2

In week 2, second version of the URS had to be finished. We had big problems in deciding what will be the process of placing the element on the grid. Other problems in this week were in making the User Interface with all the images and it had to be user friendly. Fortunately, we had some knowledge of UI, because we had followed UID course in which we learned how to make it clean and easy to use, so we applied our skills. On Thursday we had a meeting. The teacher looked satisfied but we also had many things to do better. The way we expressed the use cases was not always correct. We had to be consistent in the using of words (for example not to use item instead of component).

For week 2 we had:

* Group meeting – User Interface was made
* Individual work – Separated use-cases and nonfunctional requirements between group members
* Meeting with the teacher presenting Second version of URS
* Feedback
* Plan for next week based on feedback:
  + UI was good.
  + Nonfunctional requirements were good
  + Use cases had some inconsistencies and some were too extended, and some were too short and unclear

#### Week 3

In week 3, The URS was almost done and we had to deliver the final version by Friday. We started working on the Design document to give a first version by the end of the week so we get feedback. The URS was finished on the first group meeting (only with group members) we had during the week. The work on the design document was separated.

For week 3 we had:

* Group meeting – URS was finished and sent to the teacher. Discussion about Design Document what classes are we going to have and how to make connection between them.
* Individual work – Class diagram and sequence diagrams
* Meeting with the teacher
* Feedback
* Plan for next week based on feedback
  + URS was finished, mark was to be given till the end of the week
  + Some classes were not needed and the class diagram in general needed some shrinking
  + In general, the teacher liked the class diagram. It had to be implemented in Visual Studio, to generate code. Minor changes on the classes had to be made

#### Week 4

In week 4, Design document was on the go. We had made class diagram, the only things were to implement it in Visual Studio so it can generate code, and to make sequence diagrams. We had some issues in implementing it in Visual Studio, some bugs were encountered. Final version of the Design Document was structured and we were ready to present it on the meeting with the teacher on Thursday.

For week 4 we had:

* Group meeting – Discussed class diagram and sequence diagrams
* Individual work – Implementation of Class diagram in Visual Studio, Sequence diagrams made with some nice tool and documentation
* Meeting with the teacher presenting final version of Design Document
* For next week we had to start implementing the project

#### Week 5,6,7

In week 5, 6, 7 we were busy doing the implementation of the project. Many problems were discovered. We didn’t need some of the classes, most of the classes were incomplete and the design needed changes. It was hard also to be consistent with the use cases, but solutions were found. We added many extras to the project, except the ones that were documented.

For week 5, 6, 7 we had:

* Group meeting each week where we plan our work and give solution to issues which couldn’t be solved individually.
* Individual work – implementation of the project
* Skype meetings and chat discussions – those were very effective, because we constantly had issues during implementation and communication in the group was vital

# Understandings made with the group and the teacher

#### URS

During the work on URS, we had many problems, we received feedback from the teacher and that gave us new level of knowledge for functional and nonfunctional requirements.

Understandings made during URS:

* How to properly write Use-cases, so they be consistent and good to read
* How to combine Use-cases
* How to separate Use-cases
* How to expand Use-cases
* What is the goal of nonfunctional requirements and how to reach it
* How to separate work in doing Functional and Nonfunctional requirements
* What are the main qualities of the User Interface and how to make it easy to use
* How to communicate and discuss problems about functional and nonfunctional requirements

#### Design Document

During the work on Design Document, we had to make a class diagram which took us a lot of time and needed a lot of communication in the group.

Understandings made during Design Document:

* How to separate the project into classes
* What major issues might occur when implementing the project (the class diagram might need changes during implementation)
* Where we need to use Generalization
* What has to be handled by SystemFlowly class
* How to make proper Sequence diagram and what are the things about sequence diagrams that we have to keep in mind
* How will be handled mistakes made in the initial class diagram during implementation
* How to separate work and communicate during work on Class diagrams and sequence diagrams
* How to generate documentation automatically from Visual Studio

#### Implementation of Flowly

The implementation was the hardest part of the project. Many problems occurred and difficult situations had to be handled. First thing we had to do is to place our project on stable environment where everybody could work – we used GitHub for that. We needed improvement of communication, so we were doing some scrum meetings and we discussed changes in communication applications. Changes on the initial version of the class diagram were made, some entities were deleted, some were added and most were expanded. The separation of the work was uneven, some members took initiative and did more than other members. The final version of the project was a big achievement for everybody of the group. The project was successful and gave a lot of knowledge to everyone in the group.

Understandings made during Implementation of Flowly:

* How to use GitHub (using Git Shell)
* Communicating with group members during implementation of a project
* Scrum meetings
* Changes on initial version of class diagram. How to expand the classes
* Using graphics and drawings in C#
* Creating components and display them using their coordinates
* Making connections between objects using another object
* Separating the application into parts (grid, toolbox, properties, menu)
* Not losing graphics on events like minimize application
* Editing objects after their initialization
* Removing objects from the system
* Serialization, Saving, Loading files
* Making undo functionality of application
* Calculating and implementing collision of objects and lines
* Calculating transition of values (flows) between objects using another object (pipe)
* Separating the object output (flow)
* Multithreading
* Being consistent with URS/Design Document
* Using the use-cases for the application functionality
* Finding solutions working as a group
* Documenting and commenting the project

# Separation of work

During URS and Design Document preparation, the work was spread even among the group members. During the implementation two of the Group members Bilger and Nikola took the initiative and worked harder than the rest. A lot of the work was done in group meetings where everybody participated. During the meetings we summed up what was done and what had to be done. Plan and separation of work was done on each meeting.

Bilger Yahov and Nikola Nikushev did most of the implementation.

Individual involvement in the project:

#### Bilger Yahov involved in:

* Use-cases
* Nonfunctional Requirements
* Major creation of Class diagram
* Sequence diagrams
* Serialization
* Saving/Loading/Clearing grid
* Implementation of Component classes
* Major Implementation of ComponentDrawn class
* Implementation of Form
* Implementation of SystemFlowly class
* Implementation of Grid class
* Implementation of Undo Functionality and Change class
* Work on design features of the application
* Transition of flow through the system
* Implementation of minor elements of the project
* Implementation of ConnectionPoint class
* Implementation of Edit functionality
* Fixing bugs in the system
* Major involvement in implementation of the project

#### Lyubomir Dimov involved in:

* Documentation of URS, Design Document and Process Report
* Major creation of Use-cases
* Class Diagram
* Sequence diagrams
* Major Implementation of Edit functionality
* Transition of flow through the system
* Assistance in implementing Component classes and ComponentDrawn class
* Implementation of ConnectionPoint class
* Overall assistance in implementation

#### Tao Hua involved in:

* Use-cases
* Designing the form and User Interface
* Class Diagram
* Assistance in implementation of the project

#### Nikola Nikushev involved in:

* Use-cases
* User Interface
* Class Diagram
* Implementation of SystemFlowly
* Major Implementation of the From
* Major Implementation of Grid class
* Major Implementation of Creating, Removing Component
* Major Implementation of Pipeline and connection of components
* Implementation of collisions of objects and pipelines
* Implementation of 2D world in which the pipes cannot collide with themselves, with other pipelines or with other objects
* Implementation of Component classes
* Implementation of ComponentDrawn and ConnectionPoint
* Implementation of Enumerations
* Multithreading
* Implementation of minor elements of the project
* Transition of flow through the system
* Implementation of Drawings and Graphics
* Images on the form
* Work on design features of the application
* Fixing bugs in the system
* Major involvement in implementation of the project

# Choices and problems

#### Use cases and nonfunctional requirements

For our use cases we picked scenarios in which the main functionality of the application will be expressed. The names of the use-cases were short, giving clear definition what is the use-case all about. We were consistent in using words in our functional requirements.

We first met to discuss what will be the functionality of the system. Most of the functionality was already defined in the document given by the teacher, so it was not a hard task to define the functionality. We also added some other requirements like Undo, Saving/Loading/Clearing Grid, and Edit of component. The second step was to make the use-cases.

For nonfunctional requirements we were thinking of how to make the application Easy to use, with good performance, reliable. We came up with ideas of how to make the user experience good and we wrote our nonfunctional requirements in the URS document.

#### User Interface

For the UI we sketched it on paper and we discussed different solutions. We came up with idea of having toolbox on the top of the grid or to have toolbox on the right. We finally decided to have Grid as around 80 % of the whole application screen and the toolbox will be placed on the left of the grid giving options for Creating, Removing or Editing component.

#### Modes

We decided to make our application in a way that the user gets into modes of Creating, Editing or Removing component. In this way the user will be able to specify properties of the component that is to be placed on the grid. Also, the user can put many components with same properties and not entering the properties every time he places a component. Modes give the user a lot flexibility.

There was the problem that modes might be confusing for the user because he might want to click on the grid while he’s on specific mode and to not understand what is the reason for the action the application is taking. So we found a solution for it. When mode is used, its toolbox component is highlighted and this makes it very clear that user is in specific mode.

#### Class diagram

For the class diagram we had a lot of discussions about how it should look like. We didn’t know if it will be correct to use Generalization and Inheritance for the components. We had hard time with deciding how we are going to make a connection from one component to another using pipeline.

So we had group meetings and discussions in which we found solutions for our problems. First we decided that our components will have Connection point which can be output or input. We made a SystemFlowly class in which most of the objects will be initialized and we built the connection between those objects.

We used Inheritance for our Components classes. Pump, Sink, Splitter, Merger, Pipe inherit from ComponentDrawn. The hold information about themselves as well as their connection points, which represent the input/output connections they have.

Together with all the faced problems around pipe-lines, algorithms and ideas how to make everything run smoothly, we faced also a problem during implementation of the saving functionality. Unfortunately Visual Studio and more precisely C# has some restrictions on serializing objects from built-in classes. The problem we encountered was related with serializing our grid which in fact is a PictureBox. Every time we tried to serialize it the environment was giving an error that the graphics object of PictureBox cannot be serialized. For that reason we came up with an idea of creating a new help – class. The idea of the class was to store information about the grid like – the name of the grid, destination of the grid and components placed on the grid. We were of course able to serialize objects from the newly created class.

ConnectionPoint class allows us to properly distinguish the connection a component has with another by a pipe line. It allows one pipe to be connected to it and depending on whether it is an input or output it will either receive flow or pass on/receive flow.

The class Change allows us to store unique changes made to the form, such as adding a component, removing a component, updating the properties of a component, adding a pipeline. We store a change in a file which later on we can read and use as a Undo/Change state function.

#### Creating component

The application’s main functionality is to create components on a grid. So this was the first thing we started implementing.

We made the creating of component in a way that when the user clicks some of the toolbox components, the application goes into specific WorkingMode. Тhe working mode indicates which type of component is to be drawn, it can also mean removing or editing component. When the user clicks on working mode for creating specific component he/she can choose properties from the properties list and afterwards place the component on the grid.

We encountered some problems while implementing the functionality, one of which was the collision of components on the grid. So a solution was an algorithm which checks if the component is placed over another component. With this algorithm we solved our problem.

We faced some problems while implementing the part where the newly created component had to get the properties specified below in the properties section. We had to keep in mind that every time a new component is created the old values from the properties windows had to be refreshed and depending on the type of component which properties should be enabled and which not.

#### Pipeline 2D world and connecting components using pipe

We wanted our application more realistic, so we needed a solution for making our pipelines in a way that they don’t intersect. A solution is found by making an algorithm for creating a pipeline. When pipeline is created by the user the algorithm checks next action the user and calculates if this action will intersect the pipe with itself with other pipe or component.

After we had the solution of how make proper pipe we had to make methods of connecting the components. For this we use Connection point, which are objects inside the component object. They can be output or input, they also have coordinates. So a pipeline begins in an output connection point of component and finishes in input connection point of other component.

#### Transition of the flow

We were in a situation where we have made methods for connection components and now we had to transfer flow from one component to another when they are connected. We found a solution and the application transfers flow from output component to input component but it checks the capacity of the input component. If the component capacity is smaller than the flow it is receiving the pipe connecting it is colored red. This only effects the sink, as it only has a capacity for its input. If the output flow is lower than the input capacity, then the input component gets the flow from the output component and the pipe color is gray.

While working with this we had problems distinguishing the main object and from where to get the flow that is being transferred and how to update the outputs of the component. We found a solution to this problem by registering a pipe connection to a connection point, making it easier to check if the connection point is connected or not. This allowed us to easily check the flow and update it on the component.

#### Edit component

We were in a situation where we have made methods for connection components and now we had to transfer flow from one component to another when they are connected. We found a solution and the application transfers flow from output component to input component but it checks the capacity of the input component. If the component capacity is smaller than the flow it is receiving the pipe connecting it is colored red. This only effects the sink, as it only has a capacity for its input. If the output flow is lower than the input capacity, then the input component gets the flow from the output component and the pipe color is gray.

While working with this we had problems distinguishing the main object and from where to get the flow that is being transferred and how to update the outputs of the component. We found a solution to this problem by registering a pipe connection to a connection point, making it easier to check if the connection point is connected or not. This allowed us to easily check the flow and update it on the component.

#### Managing the working grid

One of the most important features of the application – to manage the working grid, took us a lot efforts to implement. There are a couple of functionalities that can be applied on the working grid:

1. New grid
2. Save grid
3. Save as grid
4. Open grid
5. Clear grid
6. Close grid

We faced some problems implementing them and we gained a lot new knowledge as well. We had to make a clear difference between “Save” and “Save as”. Of course the difference was that when the user chooses “Save as”, our application has to give the opportunity to the user to choose specific path and name. While “Save” functionality is meant to save the grid on the same place with the same name if it is already saved before or to do exactly the same steps as “Save as” if it is not saved before.

We managed to implement all the functional options mentioned above in the way that is best suitable for the user. For example if there are changes made on the grid and the user wants to close the grid or open a new one, our application always asks the user if he/she wants to save the changes before closing.

#### Undo and choose previous state functionality

Thinking about this challenge first we found out that a stack needs to be implemented and also an opposite action of each change that is made. This seemed to us as a lot work and of course we tried to come up with something smarter.

Trying different solutions we found that we can actually serialize each step of the grid in a separate folder. We implemented it and it seemed really successful. That way we can undo a change going to a previous state moreover we extended the functionality in a way that the user can undo a couple of changes, not only one!

In a way this makes our Flowly application unique.

#### Graphics cleared after minimizing application

When the application is minimized the content would not be re-drawn to the form, even after the Paint event would be called as normally intended by the form. The paint event would be called a little before the form itself would be fully maximized. We had to find a walk around with a small delay to the visualization of the components on the grid.

By adding a thread to draw the objects after the form has been fully maximized we managed to create a way to visualize the content of the grid after a minimize event.

#### Multithreading

Using multi-threading we re-draw the components after the form has been minimized and been fully maximized.

We attempted to use multi-threading to speed up the process of saving a change but it turned out to be very difficult because it required synchronization between the graphics engine, form components and data that is inside our system. The same variable would be used by multiple threads and there would be conflicts or some data could get lost in the process.

Instead we optimized the process so it would store information in a way that is easy and fast to be saved to the files.

#### // If you can think of anything

# Personal Evaluations