Product planning

The First Order

Chris Berg 4216776
Martin Koster 4371011
Hung Nguyen 4232410
Christian aan de Wiel 4396286
Ruben Wiersma 4214250

04 may 2016

Contents

1 Introduction

The Multimedia context project 2016 is an assignment for the company PolyCast. This company records classical concerts for orchestras (PolyCast, 2016a). There are many aspects to such a big event and every aspect has its struggles and problems. First, there is the question of what to record at a given moment. To answer this question, a script is written in advance. A problem that shows up here this is that you cannot use the same camera for two consecutive cues and that sometimes, it's hard to visualize the concert before it actually happens. Another aspect is to control the cameras during the performance. A problem in this aspect is, for example, a view from a camera which is different than expected. Finally, you have the editing after the performance to fine tune the audio and video. As you can see, there are a lot of places where things could go wrong or where work is done that can be done more efficiently, like a camera with a wrong view. Automation can help take away human error and do some repetitive or 'boring' work so that the people at Polycast can focus on creative tasks. Software can help visualize different scenarios for the polycast team and can put those into effect at the actual concert.

At this moment the company uses software to create scripts (filemaker), control the cameras, and edit their recordings, but none of these are incorporated in an overarching workflow where all their programs work together. The software to control the cameras does a decent job at controlling the cameras, but someone has to find the right cue at the right time by listening to the scorereader and the script is made in software that is not specifically made for writing scripts. We will propose a product that can serve as a thread throughout the entire workflow, starting with script creation.

The next section will explain what the vision is of our project and which problems we would like to solve. In section three, we explain how we want to accomplish this during our sprints. To do this optimally we have to set priorities which are stated in section four. Finally to check whether we have done a feature we must define what we define by done, this is done in section five.

2 Priorities

We use the MoSCoW-method (Clegg and Barker, 2004) in order to prioritize the features we want to embed in the application. This is a good practice to see what have to be done first to get a working version of the system and what can be done later which isn't necessary for a working version. It is also good to have everything in a list for a good overview of a features which have to be done. The MoSCoW-method consists of four principles, which are understood as follows:

Must Requirements which have to be contained in the project, else we have to consider the project a failure.

Should Requirements which are important, but do not have to be in the project.

Could Requirements which are not important, but may be included in the project.

Won't Requirements which should not be included in the project at this time.

Prioritization was made in the following way: First, we decided what the core product would be. The core product consists of the score view, a map view and the ability to add cues to that score view by selecting a camera and an instrument in the map. Actions can be added to these cues and they can be saved to a database. All these features for the core product ended up in the 'must have' section. We then went on to define features that we'd like to have, but would not see in the first version of the product, like adding bar numbers to a cue or a preview of the camera view. These ended up in the 'should' section. Finally we went on to think of features that would be cool to have in the product but not feasible for the scope of this project. These ended up in the 'could have' and 'won't have' sections appropriate to the feasibility of the features themselves.

2.1 Must have

- The application shows a view of the score
- The application shows a view of the map
- The application shows a summary of the created camera action
- A user must be able to add a cue to a timeline view of a score
- A cue is added to a point given by its x coordinate in the score
- The x coordinate determines the order of the cues

- There is a map of the concert hall that displays camera positions and instrument positions
- A user is able to enter camera positions into the system
- A user is able to enter instrument positions into the system
- A user can set up cues by selecting a camera, and an instrument in the map view
- A user can set the zoom of the camera
- A user can set the action of the camera
- A user can set the duration of a camera action
- A user can see the availability of a camera
- Cameras cannot be selected when they are not available
- Camera actions are saved in a script to a database
- The database is separate from the application
- The application can export a script to PDF
- The application can export a script to XML
- The application runs in a web environment
- The interface works asynchronously

2.2 Should have

- A preview of the camera action is shown in the map (viewlines from camera)
- A cue is stored with the bar that it's placed in
- A user can appoint bars in the score so the script can link cues to bar numbers
- Camera and instrument positions can be saved
- Camera and instrument positions can be restored from a file
- When a instrument position is added to the map, the name of the instrument should be auto-completed
- The application has a view for live performances where camera operators can see the upcoming shots and prepare them for recording

- In the live view, the score reader can progress the cue by clicking a next cue button
- The operator view shows the operator which cue they have next

2.3 Could have

- The user can see a preview of the camera on the instrument
- The operator view is different for each operator based on the cameras they operate
- The director can add constraints to the use of the camera

2.4 Won't have

- Bars are automatically detected in the score
- Notes are automatically detected in the score
- In a live performance the script is automatically progressed
- Camera's are automatically adjusted to what's in the script
- Scripts are automatically generated

3 Road map

In the road map, we discuss the planning done throughout the project. The road map consists of iterations called sprints, in which we complete certain tasks. The way we have constructed this road map is to go through the list of musts in the priority list and build a core product.

Sprint 1 We lay the foundation of the project. We decide which elements are important in the application and what they are going to look like.
Tasks: GUI, dataframework, vision.

Sprint 2 We focus on the basic features of the application: building scripts and the visualization of the concert hall.
Tasks: Script creation, interactive map of the hall, interactive score

Sprint 3 We finish the core product that was started in sprint 2.
Tasks: Script creation, interactive map of the hall, interactive score

Sprint 4 We focus on how new maps of the hall are constructed and how camera positions can be changed in a quick and easy way.
Tasks: UI tweaks (e.g. dragging camera's and Map editing), Map making

Sprint 5 We take care of saving and loading projects, and exporting the created schedule.
Tasks: Data export, Saving and loading projects, Saving and loading maps

Sprint 6 We focus on the part of the program which is used in live mode. We create the different formats the schedule can be presented in to the people involved in the process of recording.
Tasks: Creating different views used when in live mode

Sprint 7 We add extra features of the application, in order to enlarge the abilities of the application.
Tasks: Being able to export an XML file which links footage to queues for a video editor project file

Sprint 8 We get the application ready for delivery by running acceptance testing and fixing the software accordingly.
Tasks: Testing the software, Fixing bugs

4 Definition of done

We finish this report by giving a definition of done. We check every feature against this definition to see if it can be merged with the stable branch.

Every feature which is implemented has to be tested for at least 75% with unit test. Of course the preferable amount of tests is 100% but this is almost never possible. Also the code has to be documented fully and has to be coded by the standard of Checkstyle, FindBugs and PMD. However, if a tool gives an error that we have good reason to ignore, we will. Finally Travis CI build must not have any errors.

For every sprint, we have to check if the desired features have actually been implemented. This is done by doing user tests to check if the system is working and works according to the expectations of the user.

The end product will have the same definition of done as the sprints. For the end product we have some additional tests. We will ask the users to use the program to get their opinion of the application. This will be done before the deadline so we can change the things they would prefer to see different. The software is done when all the *must have* requirements have been implemented. These requirements are necessary to get a working end version. For the *should haves* we would like to implement at least 50% of the requirements but this is not a strict threshold. We chose this percentage because we can't do all should haves but we want to have a high standard for the application to make it as interesting for the user as possible.

5 References

- Adobe (2016). Video editing software. Retrieval at 4 May, 2016 from http://www.adobe.com/products/premiere.html
- AHK (2016). Regie documentaire. Retrieval at 4 May, 2016 from http://www.ahk.nl/filmacademie/opleidingen/regie-documentaire/
- Apple (2016). Final Cut Pro X. Retrieved at 4 May, 2016 from http://www.apple.com/final-cut-pro/
- Hanjalic, A.; Bacchelli, A. (2016). Context Project Guidelines. Retrieved at 4 May 2016 from https://docs.google.com/document/d/1Xb4vrF9V78ge_vq4VLrl3YZ2Qq0daCL
- Celtx (2016). Celtx - Free Scriptwriting & All-In-One Production Studios. Retrieved at 4 May, 2016 from https://www.celtx.com/index.html
- Clegg, D.; Barker, R. (2004). Case Method Fast-Track: A RAD Approach. Boston, MA: Addison-Wesley. ISBN 978-0-201-62432-8.
- Final Draft (2016). Official Final Draft — #1-Selling Screenwriting Software —Final Draft. Retrieval at 4 may, 2016 from http://www.finaldraft.com
- Gram, C.; Cockton, G. (1996). Design Principles for Interactive Software. London, UK: Chapman & Hall.
- PolyCast (2016a). PolyCast. Retrieval at 2 May, 2016 from http://www.polycast.nl/
- PolyCast (2016b). PolyCast Our Team. Retrieval at 4 May, 2016 from http://www.polycast.nl/medewerkers
- Trelby (2016). Trelby. Retrieval at 4 May, 2016 from http://www.trelby.org

A User stories

The user stories which describe the actions of an end-user are derived from the features described in section 2. We only describe the end-user stories concerning features with very high (must have) or high (should have) priorities, because features with lower priorities are subject to change and unlikely to be implemented. The team user stories describe what our goals are as a team. We are at the beginning of our project, so user stories that cover the defects and technical improvements will be included later.

A.1 End-user user stories

A.1.1 Very high priority user stories

basics

- As a user, I want to be able to start the application and go through different screens easily.

score

- As a scheduler, I can see the score so I know what kind of instruments there are and what they are playing at each moment.
- As a scheduler, I can click on the score to add a cue so I have a timeframe to which I can assign an active camera.
- As a scheduler, I can easily change the position of a cue so I can change my mind.
- As a scheduler, I can easily delete a cue if I change my mind.

map

- As a scheduler, I can see the map so I know the positions of the musicians and cameras.
- As a scheduler, I can set up a cue by first selecting a camera, and then a musician. This enables me to create camera actions.
- As a scheduler, I can easily set the zoom factor of a camera action so I can order a camera operator to zoom in.
- As a scheduler, I can easily set the begin and end point of interest of a camera so I can order a camera operator to pan.
- As a scheduler, I can see whether a camera is available or not so I know which cameras I can use.

- As a scheduler, I can't select a camera which is not available, preventing me from assigning camera actions to cameras which are already in use.

map editing

- As a scheduler, I can edit the map, This enables me to set the instrument and camera positons to the way they are in the orchestra.
- As a scheduler, when I am editing the map, I can easily enter a new instrument. This enables me to recreate an orchestra in the application.
- As a scheduler, when I am editing the map, I can easily change the position of an instrument. This enables me adapt to changes in the orchestra.
- As a scheduler, when I am editing the map, I can enter a new camera position to the map which represents a camera in the hall.
- As a scheduler, when I am editing the map, I can easily change the position of a camera. This enables me adapt to changes in the camera position setup.

camera actions

- As a scheduler, I can see a summary of the created camera actions so I have an overview of the actions I created.
- As a scheduler, I can easily change a camera action so I can change my mind.
- As a scheduler, I can easily delete a camera action if I change my mind.

export

- As a scheduler, I can view the application on a large screen. This enables me to view large pieces of the score at once and to create large maps.
- As a scheduler, I can export the created script and hand it over to e.g. a camera operator. This enables me to inform the operator about the camera actions he should perform during a certain cue.
- As a camera operator, I can view the application on any device I like. This makes me mobile.

A.1.2 High priority user stories

project

- As a scheduler, I can load a map, containing instrument and camera positions. This enables me to reuse a map a created in a different project.
- As a scheduler, I can load a score into the project which I can see in the application.
- As a scheduler, I can store a project which contains camera actions, map and score. This way, I will not lose my data.
- As a scheduler, I can load a project so I can continue from where I left.

extensions

- As a scheduler, I can easily link the cue position to a bar in the score. This saves time, because this is the place I want to position a cue most often.
- As a scheduler, when I am adding an instrument, I can choose to auto-complete the name of that instrument. This lets the process of editing maps faster.
- As a scheduler, when I am adding a camera action, I can preview the view lines of that camera. This allows me to have a clearer image of what will be in the shot.

live mode

- As a user, I can enter the live mode of the program. This enables me to view only the parts which are necessary during the recordings.
- As a camera operator, I can see the upcoming shots so I can prepare them for recording.
- As a score reader, I can progress the cue by clicking a next cue button. This enables me to keep track of the events happening during the recordings.

A.2 Team user stories

- As a team, we want to have a good overview of tasks that need to be done and we want to reflect on work from previous sprints to improve. This enables us to learn from mistakes from the past and reduces the productivity lost due to faulty process.
- As a team, we want to have a good overview of tasks that need to be done and we want to reflect on work from previous sprints to improve.
- As a team member, I want to be able to see the big picture of the software, so I know how to implement a new functionality.
- As a team member, I want to know the vision of the product, how it will be made and what is important, so I know what to expect from the product.
- As a programmer, I want to have models and controllers ready to use for future developments.