Product Planning

Team: goto fail;

Alex Geenen
Martijn Janssen
Mark van de Ruit
Bart de Jonge
Menno Oudshoorn

Table of Contents

Introduction

In this document, we shall give a broad overview of how we plan to develop our product over the next 9 weeks. We will discuss which features will, and will not be implemented. Also, we will give a schedule of what features will be released in which week. We will also have a look at some user stories of users that will be using our product, and how they affect the product backlog. Also, we shall be looking into a definition of done, to make sure every project member knows exactly when they can call a feature, sprint, or release, “Done”.

Analysis

In this section, a thorough analysis of the context, problem and stakeholder inputs will be given. This is done so that, afterwards, the steps needed to complete the project are clearly defined

Context

The context is Multimedia Services, and it is especially focussed on the company PolyCast BV. This company is specialized in high-definition audio-visual recording and production of concerts. The company operates internationally with many important venues and organizations, and has an own studio in the Concertgebouw in Amsterdam.

During such a recording, there is a lot going on. There are multiple cameras used which are operated by several camera operators. Before this, a script is created. This script is planed out by the director before the recording, and the camera operators will follow this script during the recording. However, nothing always goes fully as expected, so during the recording, the director can also make manual changes by telling the camera operators what to do. This can be a pretty stressful procedure.

The company is using new cameras that can be controlled through IP commands, which opens up a lot of possibilities to improve and automate the workflow at PolyCast, and possibly other similar companies.

Problem

The main problem at PolyCast at this moment is that the workflow still is a very manual procedure. It involves paper scripts, remote operation of cameras with joysticks, and, as already said before, a good dosis of chaos.

The script creation process is still very manual. Everything is still put in one timeline, and there is no support for collision detection or separate camera timelines. Collision detection has to be done manually be the director during or after creating the script, and this can take up quite some time. Also, the director might miss something which can create trouble during live recording

The people at PolyCast have recently started using a simple app that stores camera presets. However, this still has to be done manually, which means a camera operator can only operate a limited amount of cameras, and has to focus a lot on what is coming next rather than focus on the present moment and try to make the most beautiful shots. An app that automatically recalls presets would be very helpful here.

Finally, during recordings there is a score caller who calls out at which point we are and what shots are coming up. This can become quite chaotic, especially if the director is shouting in between as well. It would be much better if this can become digital. Then, highlighting the shots for certain camera operators is also possible, which makes it much clearer for a camera operator what they have to focus on.

Stakeholder input

During a meeting with the PolyCast team, it became clear that they are really looking for something that can make their work less stressful and time-consuming. This corresponds with the problem analysis, and this is also what the final product will focus on.

High-level product backlog

We will use the MoSCoW method to define the features we do and do not implement. The MoSCoW method divides features into four categories:

1. Must haves. These are features that must be implemented for the product to work be the right product for the client. Without these features, there will be no release
2. Should haves: These are features that are not directly fundamental to the product working, but enhance the quality of the product greatly and therefore, a project team should aim to implement as much of these features as possible
3. Could haves: These are features that are not something that one directly thinks of as needed when thinking about the product, but are nice enhancements to possibly make in the future, if the time allows it
4. Won't haves: These are features that will not be implemented, or at least not in the first version of the product

We create a product that covers the whole process of a concert recording. First of all, a script needs to be made by the director. At the moment, this is done almost fully manual, and our product will automate and ease this process. The output of the script creator will be linked to an interface for automatic camera preset recalling. This is done so that the camera operators have to focus less on what is coming next, and more on what is happening at the moment. The script will also be linked to different views for during the recording, in which camera operators, the score caller and the director all see the information in the timeline that is relevant for them. The most important thing here that everyone sees is the progress on the script during the play. This progress is managed by the score caller, and used by the camera operators to see what is coming up. Together, this should ease and speed up the workflow of a concert recording significantly.

The three main sub-products we will be creating are a script creator, an interface for automatic camera preset recalling, and interfaces to show the timeline during actual recording. We will give MoSCoW features for each of these products separately.

Script creator

- Must haves
    - Loading and saving of created scripts
    - Timeline interface to show all camera shots in order, and per camera
    - Adding and deleting of camera shots.
    - Moving camera shots around through a drag-and-drop interface
    - Highlighting of shots that collide with other shots
    - Director timeline that shows a summary of camera shots.
    - Ability to change the size of a timeline, relative to number of counts
- Should haves
    - Automatically generated camera timelines from constraints specified by the director.
        - Constraints such as: I want a shot of the violin from 0 to 10 with either camera 4 or camera 5. I want a conductor shot from 2 to 15 with camera 4. The program should then select camera 5 for the violin shot to avoid collisions.
    - Ability to add a camera to multiple timelines at the same time.
    - Editing of camera shots

- Could haves
    - Modularity and customizability of GUI
        - Modularity means that elements created for the GUI can be reused in different positions, sizes and other configurations, throughout the GUI. Customizability means that the user can choose where and how to resuse these elements. For example: the ability to snap a pane to the left, right, top or bottom of an application, depending on user preference.
    - Post-production editing
- Won't haves
    - Support for transitions between shots.

Camera preset recalling

- Must haves
    - Interface to store and recall camera presets.
        - These are actual camera "settings", like pan, tilt, zoom, etc.
    - A web server used to operate the cameras
    - Automatic preset recalling during recordings
    - Possibility to switch off automatic recall and take over manual control
- Shoud haves
    - Live low-quality preview of upcoming cameras
- Could haves
    - Possibility to move/zoom the cameras through the preset interface.
- Won't haves
    - Live high-quality preview of upcoming cameras in our application

Timeline interfaces during recording

- Must haves
    - Different views for different people, such as the director, a camera operator, the score caller.
        - A score caller is someone who calls out to the camera operators where in the script they are that the moment. This should mostly be replaced by this product, but the score caller can still manage the speed of the timeline and take over manual control if something goes wrong.
    - Ability to easily move ahead through the timeline
    - Ability for the director to edit the timeline during play
    - Ability for the score caller to change the speed at which the timeline progresses
- Should haves
    - Ability for the score caller to choose between manual and automatic timeline speed, and change this within a recording.
    - Highlighting those cameras that a certain camera operator is operating
    - Warnings for possible problems when the director edits the timeline during recording, such as collisions
- Could haves
    - Ability to tap the timeline to see more information about a shot
- Won't haves
    - None for now

Roadmap

In the roadmap, we will give a brief overview of the features we want to implement every week. It is based on scrum principles, and the focus lies on having a working version every week. This means that every week's sprint builds on the working version from the week before. The roadmap is as follows:

Week	Features
4.2	Create mockup of GUI of script creator GUI first version with drag and dropping Product vision and product planning Initial data model and I/O Basic collision detection algorithm for shots.
4.3	Basic GUI styling Saving and loading projects Collisions in GUI. Detection and styling Adding and deleting shots Editing shots (length, description, etc.) Settings screen Basic director's timeline
4.4	Director able to set up basic ideas and constraints Automatic creation of camera timelines from director constraints Setting up web server for camera control
4.5	Experimenting with IP commands to control cameras Creating mock/initial interface for creating/storing presets Create mocks for views during recording
4.6	GUI for live timeline views (score caller, director, camera operators) Initial automatic preset recall algorithm.
4.7	Live camera preview in preset interface Allow live timeline changes and propagate these to preset recaller.
4.8	Backup time: Time to finish everything that couldn't get done in previous weeks. User testing with PolyCast
4.9	Finalize features, fix possible bugs. Final design changes Fixing all static analysis warnings Exporting product

User stories

In this section, we will give some user stories that show the preferences of the user. These preferences should align with the features given earlier. We will divide the user stories into three categories: director, camera operator and score caller. These all have different needs of our product.

Director

As a director, I would like to be able to easily create scripts using a graphical interface. This interface should allow me to save and load scripts with ease.

As a director, I would like to be able to easily modify scripts using a graphical interface. The interface should allow me to place, move and remove shots with ease.

As a director, I would like to be able to easily modify shots in a script using a graphical interface. The interface should allow me to change types, lengths and other variables of shots.

As a director, I would like to be able to specify a set of constraints with respect to which cameras could be used in a possible shot. The separate camera timelines should then be created automatically.

As a director, I would like to be visually notified when I make a script that has collisions or other potential problems. A collision would happen when

As a director, I would like to be able to change the script on the fly, to adapt to unexpected situations.

Camera operator

As a camera operator, I would like to be able to easily see what shots I have to take for the next number of counts, for all my cameras, in one small interface.

As a camera operator, I would like to have more time to focus on my actual shots, so I want my next presets to be loaded automatically.

As a camera operator, I would like to be able to see a live preview of my upcoming shots, so I can already think about possible adjustments.

As a camera operator, I would like to be able to take over manual control at all times, for when an unexpected situation arises.

Score caller

As a score caller, I would like to be able to easily see what shots are coming up and how long it takes before they come up.

As a score caller, I would like to be able to choose between automatic timeline speed, or a manual timeline in which I have to tap a button for each count.

As a score caller, I would like to be able to take over manual control at all times, for when an unexpected situation arises.

Definition of done

In this section, we will show what is considered as the definition of done within our project team. This is done so that every project member has the same idea about when something can be called “Done”, and this should make sure that no unfinished features or products are handled as if they are done.

The definition of done is divided into three parts: features, sprints and release. We chose to divide into these three categories because they are the three main building blocks of our product. A feature is the smallest unit, a sprint consists of features and the release consists of sprints. We will discuss the definition of done for each of these three elements

Features

- Code is written according to standards, this will be checked by checkstyle
- Unit tests have been written for the feature, and pass. Branch coverage > 80% in non-GUI classes.
- Acceptance criteria of the user story are met
- Functional tests are performed by a team member who did not work on the feature
- All existing tests still pass after merging with the stable branch

Sprints

- All the items in the sprint backlog have been implemented, and all these items are done as specified in above definition
- All code is written according to standards
- Sprint plan, Sprint retrospective and architecture design have been created or updated.
- Branch coverage > 80% in non-GUI classes.
- Code has been merged to the master branch

Release

- All the specified Must have features have been implemented
- A significant amount of Should have features have been implemented
- All tests pass. This means unit tests, integration test and, if possible, user tests by the end-user
- All code is written according to standards
- Final product is well documented. This means that the code is documented using the JavaDoc standards, the architecture is documented in the emergent architecture design and the full product is documented in the final report.
- Customer must agree on the final result