Simple software project management for artists

Version 1.0

by Nye Thompson

Introduction

Artists are increasingly working with software, where these technologies may form the subject or the medium of their artwork – or maybe both. Unless you are an artist who writes your own code then you will need to collaborate with a software developer, maybe a whole team of engineers.

Talking to other artists I know that this kind of collaboration although exciting can also be problematic, as people from very different domains can sometimes struggle to communicate effectively.

As a practising artist with a past life as a software project manager I wanted to share some of the ideas that I initially learned while working in software development and which I have modified to help me deliver software-based artwork. I hope they will be useful to you and maybe facilitate your own art/software projects.

So here is a typical situation: you the artist are working with one or more software developers, maybe you're working with a team of people with different skills. Maybe (probably) you're not all based in the same studio or even the same country. How do you communicate clearly and effectively what you need from each other to deliver your project with the minimum stress and rework?

About these methodologies

The methodologies suggested in this guide are based on a system called Agile Development. Agile is a popular software development system which is designed to be responsive to changing project needs while providing individual team members with a high level of autonomy. It provides a good framework for allowing people from different disciplines to work together effectively. I have used key concepts from Agile, but simplified and adapted them for the requirements of artistic projects. Another advantage of this approach is that any software developers you work will are likely to be familiar with this way of working, and this should give them a higher degree of confidence in the project overall.

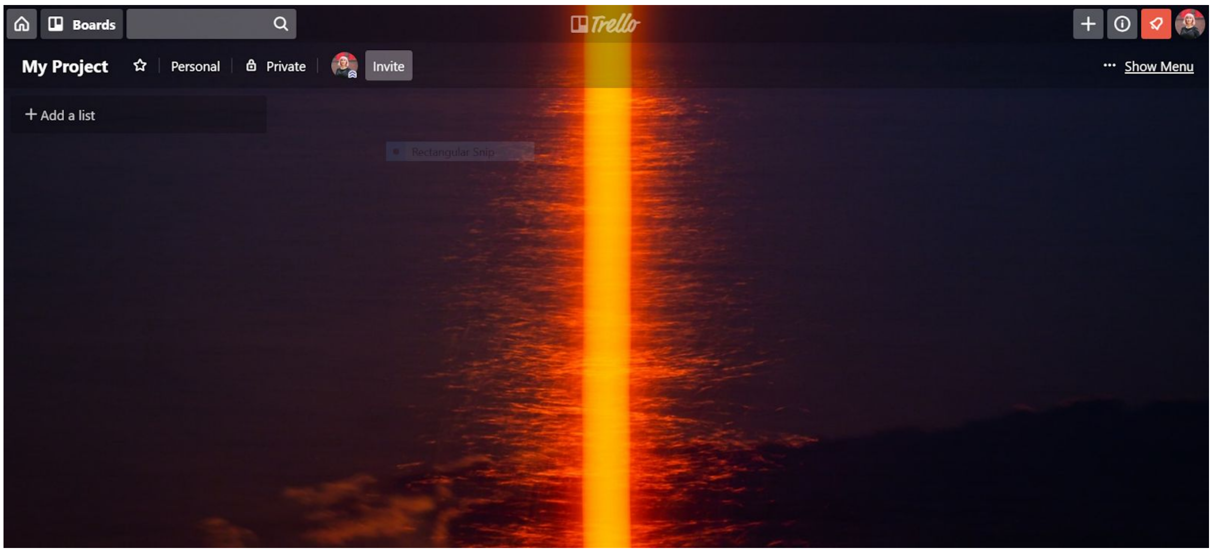
Use project management software

Firstly, you should set up a shared access software system in order to track your project progress and to define and allocate the work that needs to be done. It's really important that all team members should be able to have a shared view of project status and information.

If you're working in a software development environment you would probably use enterprise software like JIRA to manage your projects. Luckily there are similar online systems that you can use for free. I use Trello which nice and flexible, and can be accessed online and as a phone app. I will use Trello here to demonstrate my methodologies but the ideas can be easily adapted to other similar systems.

Setting up a project board

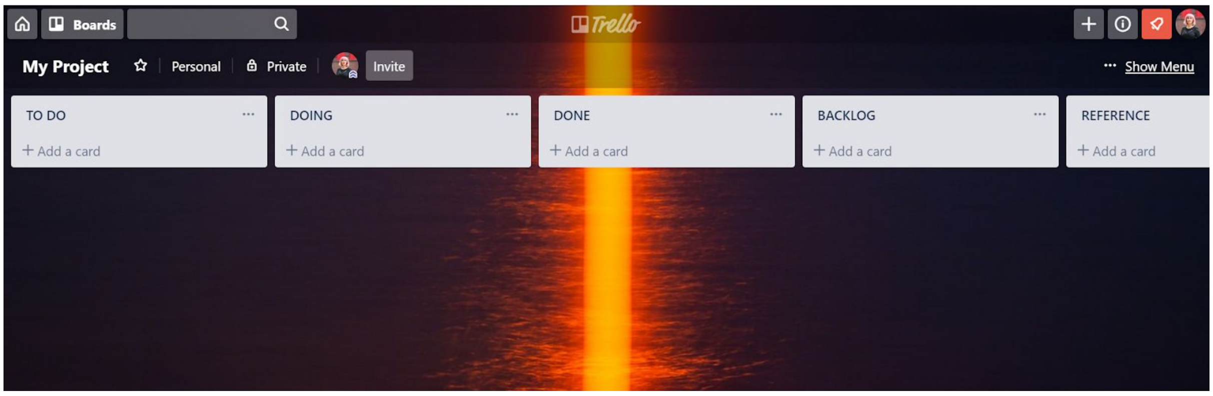
The first step is to set up a Board that will represent your project. When you are ready you can invite all your team members to it. This will hold all of your project tasks and other key project information.



An empty project board

Creating 'Lists'

Your project board will allow you to share information but should also show project progress. You can do this by creating a series Lists that team members can progress their task cards through. I use the following List structure for project tasks.



Project board with Lists set up

List Name	Purpose
TO DO	Tasks which need doing soon e.g. In the next 2 weeks.
DOING	Tasks actively in progress.
DONE	Completed tasks. Leave these here until all relevant stakeholders agree that they are complete, then archive them.
BACKLOG	For tasks with no urgency or a distant due date. People can grab or assign them when they have time or the due date approaches. Using a BACKLOG prevents your TO DO List getting too cluttered which can be demotivating and also lead to the wrong things being prioritised.
REFERENCE	I like to use this to contain key project information at a glance e.g. lists of key dates and links to things like schematics and project proposals. Not for things which require actioning.

Creating task cards

Within your Lists you will create task cards which represent the individual tasks or collections of tasks required to deliver your project. It's good to think carefully about the granularity of these tasks. If you make these tasks too small then you will soon be drowning under cards, but if you make them too big then they will take too long to complete which can get demotivating. Ideally a card shouldn't represent more than about 4-5 days worth of work at most, but smaller than that is better.

Clearly defining software tasks

Each card should contain a clear definition of what the piece of work entails. This is crucial if you are setting a task for someone else - unclear task definition is a major source of frustration in any project. Agile has a great framework for doing this called the *Agile Story*.

You may find this approach overkill for simple tasks, but if you are specifying a software task then I strongly recommend taking the time to do this. Apart from anything it is a good discipline for helping you clarify to yourself exactly what you are trying to achieve with the software you are having developed. This will then help you explain it clearly to the developer.

The Agile Story lays out exactly what the task is designed to achieve, for whom and why. It doesn't need to be technical, if you do this clearly your developers then have the freedom to use their expertise to choose the best way to achieve your stated goal. It is structured as follows:

As a [person or thing that wants the outcome of the task under development] I want [the thing they want] so that I can [the reason they want it, i.e. what will it allow them to do?]

In order to further support your developers you should then state how they will know when the task has been achieved, e.g. what properties and characteristics the software should exhibit. These are known as *Acceptance Criteria* and you should make these as explicit and detailed as you can. Getting these two things right will ensure that you and the developer have a shared understanding of exactly what needs to be developed, why it is being developed and how it should behave. This way you will avoid wasted work or ending up with something which is not what you actually wanted.

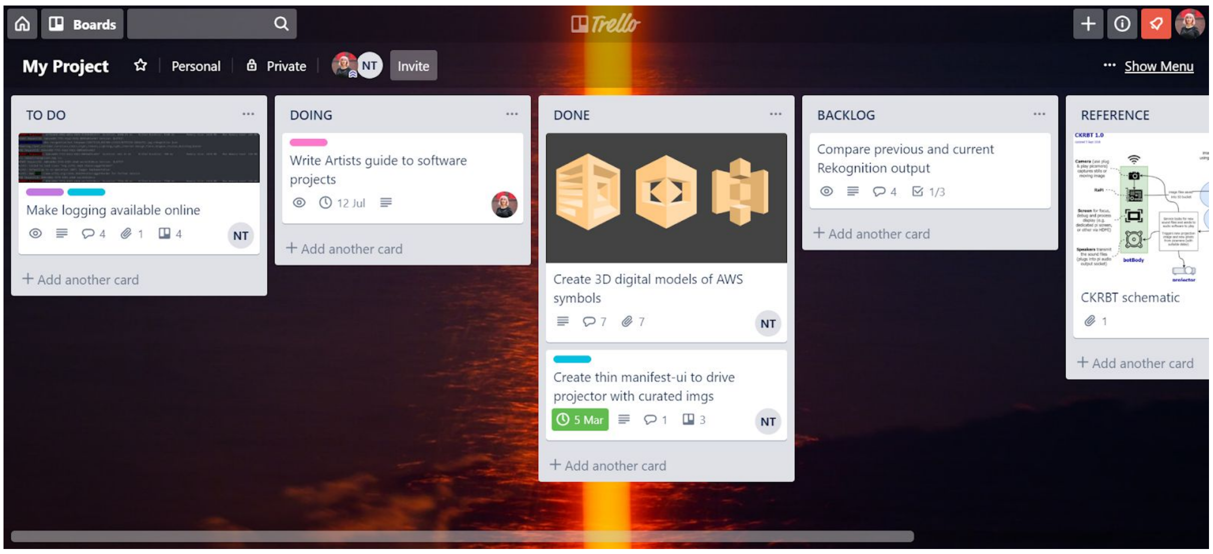
It's a really good idea to write these task cards (especially the Acceptance Criteria section) in collaboration with your developer team mates. You will end up with a much clearer and fuller task specification.

```
As a [CKRBT robot] I want [a curated slideshow of images to be displayed in
the gallery] so that I can [analyse their contents].

Acceptance criteria:
    ●  slideshow can be accessed through a browser
    ●  image transitions can be set globally e.g. fade between images
    ●  slideshow will continue to loop until manually stopped
    ●  images in the slideshow can be changed without requiring software
       changes
    ●  the timing of the transitions should be manually adjustable
    ●  there is some mechanism whereby the order of the images can be
       controlled (e.g. the images are displayed according to filename)
```

```
As an [artist] I would like [detailed live logging of the system activity
to be made available] so that I can [stream it in the gallery and on a
website].

Acceptance criteria:
    ●  logging should include the label and confidence rating
    ●  logging should include date and time of each action
    ●  logging presentation should follow the design mock-up attached
    ●  there should be separate logging for each CKRBT robot
```
Example software development task cards

Using task cards to show progress

Cards should be dragged from TO DO into DOING when someone starts work on them. Then they can be moved into DONE when the task is considered complete by the person working on it. Leave the card in DONE until all relevant stakeholders agree that it is complete, then you can archive it if you like, to keep the board clean.



TO DO
- Make logging available online — 4 · 1 · 4 · NT
- + Add another card

DOING
- Write Artists guide to software projects — 12 Jul
- + Add another card

DONE
- Create 3D digital models of AWS symbols — 7 · 7 · NT
- Create thin manifest-ui to drive projector with curated imgs — 5 Mar · 1 · 3 · NT
- + Add another card

BACKLOG
- Compare previous and current Rekognition output — 4 · 1/3
- + Add another card

REFERENCE
- CKRBT schematic — 1
- + Add another card

.The project board should be an "information radiator" so anyone on the project can tell at a glance the status and ownership of each task/card.

Some tips on effective task card management

- When you take on a task (or assign it to someone else) select them as Member. Their avatar will be displayed on the task card making ownership and responsibility for this task clear.
- All tasks moved to the TO DO List should be assigned to someone (or they may just not get done!)
- Avoid having multiple people assigned to a task where possible, as this makes overall responsibility for completion unclear.
- Use the due date to show urgency, you may also want to drag the most urgent tasks up to the top of the lists to maximise visibility.

Other key considerations

Create a roadmap

Start out with a general idea of the major tasks and the order in which they will need to be performed. This doesn't have to be too detailed, and it will almost certainly change but it's important for everyone to have a shared understanding of your project timeline and any hard deadlines involved.

Talk to each other

Set up a regular recurring meeting time for team members to talk face to face. Be mindful of everyone's time (especially if it's a big team) and set a cadence - weekly/fortnightly/monthly - appropriate to the intensity of the project.

Have places for shared project assets

You will probably want to set up a GitHub repo for the software, and a shared cloud drive such as Dropbox or Google Drive for other assets like documents or images. All these can be linked to directly from your task cards.

Allow plenty of time for software testing

Someone (make sure you have agreed who) will need to test that the code actually works and does what you want. If there are multiple software components, do they work successfully together? Do they work in the environment and on the hardware that you will be using. If you have code which needs to run for long periods of time, e.g. in a gallery you should also consider Soak testing - this means testing in the live context (or as close as possible) for as long as you can manage. Your system may run fine for an hour or so, but what happens if you leave it overnight? For 24 or 48 hours?

Thanks for reading!

I hope that some of this will be useful to you in your own projects. As I'm sure you've realised, these methodologies also work well for any collaborative art project, or even just to organise your own activities (which is what I do). If you have any questions or feedback I'd love to hear them.

Nye Thompson
studio@nyethompson.co.uk