Clarifying the Goals of our CI/CD Pipeline for Video Processing

Mark Bools

Jun 7, 2021

Last Modified: 2022-02-01

Abstract

In this article we take a step back, clear our heads, and start over focussing on the motivations behind the video processor application.

In the last article I vomited my initial thoughts about a ‘video processing’ pipeline onto your screen. This was a cathartic process (for me) but not a particularly useful one for the project as a whole. In this article we start behaving a little more professionally.

We have a very broad definition of what we want to achieve; given some video sources, some target upload environments, and some details about how to assemble videos and upload details (descriptions, titles, etc.) magically transform the bits and upload them. How can we refine this very nebulous idea into actionable requirements.

If I were working solo and the project were only every to be used by me, I’d most likely start hacking away. Of course, a few weeks from now future me would curse past me as an idiot. The code would be questionable, in all likelihood I’d have only a few (if any) tests, the system would most likely be a freakish Frankenstein’s monster. It most likely would do the job but would be a bitch to change and when things go wrong (and they will) future me will mutter that past me was a crappy coder and should be fired immediately.

Is this because I’m a shitty coder? Maybe. I think I’m a fairly average coder. I try to write clear code but invariably when I ‘just hack out’ a working solution I end up regretting it in the long run.

The following may seem overkill for a one man project but the truth is one man projects that are worthwhile will most likely be used by more people than you guess, and once the project becomes more widely used you’re probably going to regret skipping these early steps. Besides there are always at least two people who use your project; you now and future you. If you now avoids work then future you will suffer.

Figuring our what to make

Before hacking out code it's probably a good idea to figure our what we want to make. Not in the vague way we did in the last article but in more specific detail, detail sufficient that we *could* involve others in implementing a solution.

Defining 'what to make' is an ongoing process not a step. It is a simple truth that whenever you think you've specified everything your system should do, you find something new. This is particularly true as you start implementing the system. In fact anyone who has ever worked on system development will know that the final system seldom implements the initial requirements and only those initial requirements.

This said, we need to start somewhere, so we need some initial requirements to get the ball rolling.

From whence requirements?

Someone must have an itch to scratch. If no one has a problem, or at least a want, then no project would exist. Step one is therefore to identify stakeholders from whom we will extract some initial requirements.

Stakeholders

Much has been written on stakeholders. Here's my simple method for identifying stakeholders.

- Who is going to be pissed off when your system fails their expectations?
- What external systems are going to break when your system fails?

We have one clear stakeholder on this project 'me' but to make the following process clearer I will wear a number of hats and break 'me' into multiple stakeholders according to those hats. Here are my initial thoughts on the stakeholder roles.

sponsor Typically the person fighting for the project (and funding); the person with a problem to solve, in this case me.

customer Typically the person paying for the project. Again, in this case, me.

videographer Films videos.

video editor Take videographer's output and cuts it into one or more 'finished' videos.

copywriter Writes ancillary materials (e.g. video titles, descriptions, etc.) intended for publishing with the videos.

publisher Takes finished videos and ancillary materials and uploads them to target platforms.

Are these *all* the stakeholders? Probably not. Do you agree with the stakeholders and their descriptions? Probably not. Does this matter? Nope.

As with everything in system development stakeholder identification is iterative and contingent (change depending on current circumstance).

Basic truth

> It is easier to criticize than to create.

This notion is exemplified by Cunningham's Law:

> The best way to get the right answer on the internet is not to ask a question; it's to post the wrong answer.

For our purposes this means the most effective way to solicit opinions it not to ask 'so, what do you want from the system?' The best thing to do is say, 'we're going to. . . ' and wait for the opinions on why you're wrong to flood in.
So, presented with a list of stakeholders you will rapidly be told of all the people you missed (or why stakeholder X is wrong).

The important point at this stage is to identify 'some' stakeholders and start the rest of our process. Generally this initial list of stakeholders is obvious. A good starting list will include:

sponsor The person(s) who fought for the project.

customer The person(s) who are paying the bills.

users This one is more tricky. Users are typically the people who interact directly with the system, but sometimes you want to include indirect users who interact with the direct users. For example, the direct user might be the help desk operator, the indirect user might be someone who calls the help desk. The point here being that the needs of the indirect user might influence the direct user's needs so including the indirect user in conversations may prompt the direct user into an 'aha, oh yeah we sometimes need that' moment.

Goals

Hang on, didn't we say we wanted to identify stakeholders so we could extract some initial requirements? Why is this section title 'Goals' and not 'Requirements'?

Stakeholders are motivated by their goals and the system we are creating will satisfy those goals. Apart from getting stakeholders to identify the problem to be solved this goal process starts to set the boundaries of our system. Goals provide the 'why' for our system. If there is not 'why' for your system then there is no reason to build it.

Once we have some goals we can start eliciting requirements and relating those requirements to the goals. Any requirement that we cannot trace back to a goal is suspect as it probably isn't solving any problem stakeholders identified and is therefore either unproductive work or, worse, it will detrimental.

How are goals different from requirements? One of my goals is 'earn income from online videos'. I can achieve this goal many different ways, I may even be doing this without the new system but the goal expresses a motivation. How might I achieve this goal? One way is to include advertisements in my videos. 'Insert advertisement into video' is something I want to do in order to achieve my goal 'earn income from video'. Goals are 'why', requirements are 'what'.

Hurrah! We have a requirement! Well. . . No. 'Insert advertisement into video' is a capability we want the system to have but not yet a requirement. Capabilities express business impact, the 'how' our system will satisfy the 'why'. Requirements express 'what' our system must do to deliver the capability 'how', to satisfy the stakeholder's ('who') goals ('why'). This is the impact mapping concept defined described Gojko Adzik[Adž12].

Anyway, back to goals. Having established our current list of stakeholders our first task is to extract some initial goals (again, as with getting our stakeholder list, these goals are only our 'current goals list'—they will change over time).

Extracting goals is a bit like being a small child. We constantly ask, 'Why?' until we start to get silly answers.

Suppose I offer 'add adverts to my videos' as a goal.

'Why do you want to add adverts?'

'Advertisers pay for the ads'

'So you insert adverts to make money? Why?'

'Income from video adverts allows us to make more videos'

'Why make more videos?'

'Making videos generates income and allows me to pay bills and make profits'

Okay, we're now heading into 'silly' territory. Is our goal 'pay bills and make profits'. Well, yes, with my business owner hat on this is the basic goal, 'make profit'. What about as a video editor? Creating multiple versions of a final video (one for each target platform) each differing only by the specific advert shown is a pain in the ass so my goal conversation is a bit different with the video editor hat on. Once again, starting with a stated goal, 'add adverts to videos'.

'Why do you want to add adverts?'

'Well, I don't *want* to add them, but given the need to add them I want to make the process simple'

'Why?'

'Because it's a fairly mechanical, non-creative part of my job. It's boring.'

'So it would be fair to say your goal, for this system, is to offload boring mechanical tasks?'

'Yeah, free up more creative time by eliminating the mechanical tasks associated with inserting ads.'

Bingo! We have a winner. The video editor's goal is to 'free up creative time by eliminating the mechanical tasks associated with inserting tasks'.

Will our system do this? Well, it better, at least in part. The final system may not eliminate *all* the editor's mechanical tasks but it should eliminate at least *some* and it certainly must not add more manual tasks.

Why not continue asking why? 'Why do you want to free up creative time?' seems silly to me. The video editors job is adding value to the output using their creative skills. Asking 'why?' here will only lead us down the 'I make money being creative' line and if you pursue any line of reasoning to the extreme you will end up with one goal 'make money', or you pass out the other side into why people exchange their time and skills for money; 'pay the mortgage', 'pay food bills', etc. and this is obviously not going to help developing the system.

We are seeking the goals relevant to this stakeholder concerning this system.

Is this mechanical 'why?' asking approach going to extract all the good goals? Nope. As with everything else these goals are just our starting point. As we proceed we will find some goals are not useful, some goals are actually not the base goal and we need to mine deeper, and we will almost certainly add more goals. Your current list of stakeholder goals provides a scaffold for discovering more detail about the system you are to develop.

Don't get hung up trying to find *all* the goals (this is as futile as trying to list all the requirements up front). Get a list of goals sufficient for the next steps and move on, there will be ample opportunity (indeed need) to change this list as the project proceeds. As a general rule, the bigger the project, the more time you will spend identifying the initial stakeholder and goals lists.

If you discover a stakeholder with no goals for this system, they're not a stakeholder.

You may uncover a goal that reveals additional stakeholders. 'graphic designer' is not on my initial list of stakeholders. When I uncovered 'free up creative time' in conversation with the video editor this reminded me that there are other creatives involved, one of whom is the graphic designer (they create various elements of the final video but specific to this project they design the title cards to be inserted). A new stakeholder means another round of goal elicitation.

At this point you may be wondering, 'hang on, given that this process is trying to establish requirements for a system that does not exist yet, how can I ask stakeholders about there goals for this system? You know, the system that does not exist?' Well, recall that in the first article I came up with a rough idea for a new system? That's the basis on which I'm building these initial stakeholder and goal lists. We're saying, 'if a system like this existed which of your goals would it help solve?'

If your stakeholder mentions one of their goals but it is not relevant to this project, note it down anyway as it may well spin off another project. Similarly you will often hear stakeholders saying, 'oh, it would be great to have a system that X' but X is not related to the current project. While it may not be useful at this point to track this back to a goal it is certainly worthwhile making a note of it as it may lead to other projects.

Projects get started for all sorts of reasons but one thing is certain, they never start with a full list of requirements. The project sponsor is typically the person who has a problem to be addressed (the person with the itch) and they must articulate that itch to the people involved in solving the problem (the stakeholders) in such a way that the stakeholders can express their goals for the solution.

Capture

We're doing good work here. We have some stakeholders and some goals. One of the things we need to do is capture this information. We need to capture it so that we can keep everyone involved (for now the stakeholders and team extracting the goals) up to speed. The format needs to be accessible to everyone involved (it's pointless to create a document that only your team understands, the document needs to be understood by all stakeholders too).

This common understanding raises an important point; common language. If I say "module" what springs to mind? If you are a programmer you may have several mental models for "module", if you manufacture spacecraft your mental model(s) will be entirely different. If we are working with an organisation developing software to support designing and building spacecraft we need to be clear on what "module" means in any given context. To do this we start a glossary and share this with all involved in the project so we all have a common reference on the meaning of words.

We now have three things to capture.

- Stakeholders
- Goals
- Glossary

What format should be use? I always favour simple text or text markup because my developer brain sees this as the easiest for me to mangle into different output formats but I am in the minority on most projects and we tend to use other formats such as spreadsheets (MSExcel) or other Microsoft proprietary formats. Bottom line, try to capture all project information in as flexible a format as possible while making it available in formats that all stakeholders find acceptable[1].

Of course we might also keep documentation in a Content Management System (CMS)[2] of some sort, even a wiki[3], with the caveat that we must be cautious about versioning and expectations around control of content. This is a popular choice for technical documentation, less so for other forms of documentation.

[1]That said, I've seen far too many consultant documents produced in PowerPoint. There is a special place reserved in Hell for these people.

[2]A system for managing content, typically presented online as a web site.

[3]A form of CMS with a more open editing structure.

That said, this is my project, so we'll use text files. Specifically we will use LaTeX markup for general documentation (I use it for this website too). These text documents will be processed into various forms for presentation. LaTeX format is generally simple enough for anyone to edit but I acknowledge it can be complex at times and off putting for some (then again few will want to edit documents directly[4]).

Housekeeping

Another thing we will start immediately is capturing these documents in a version control tool. Again, there are many to choose from but I'm going to be a boring conformist and use Git. This is good for you too as you can access the documents at `https://gitlab.com/python-utils2/vproc`[5].

Keeping project assets in a version control tool is important but so too is ensuring each document has it's release version identified. This is something of a rabbit-hole, to avoid overcomplicating this article I will discuss document control elsewhere. For now I will rely on the document's title and last modified date and time as a unique identity.

References

[Adž12] Gojko Adžić. *Impact Mapping: Making a Big Impact with Software Products and Projects*. Ed. by Marjory Bisset. Provoking Thoughts, 2012. ISBN: 0955683645.

[4]In my experience most stakeholders are content to scan documentation in order to pass comment but rarely do they want to edit it directly and even those who insist on an editable format don't contribute to the actual edit.

[5]Yeah, I know, `vproc` is a shitty name but this can be changed later.