

Cementing Your Duct Tape

Turning Hacks Into Tools



Hi everyone, welcome to this talk on turning hacks into tools, thanks for joining!

I just want to say I'm very excited to be here, it's my first time at GDC so I hope you'll enjoy the talk.

Before we start, please remember to leave feedback in the app at the end of the talk, I'm sure whoever has to listen to me next time will thank you.

Introduction – Who Am I?

- Currently – Senior Technical Artist on *Total War*
- 4th year at The Creative Assembly
- 7th year in the industry



First, who am I?

My name is Mattias Van Camp, and I am currently a Senior Technical Artist working on the Total War franchise at Creative Assembly.

This is my fourth year at the studio, making it my 7th in the industry; I've previously worked in Belgium (my country of origin) on the first Divinity: original sin and an educational game for kids in grade school called kweetet. After that, I moved to Zurich, Switzerland to work on the Farming Simulator games, before I moved to Horsham here in England to join the Technical Art team at CA.

Overview

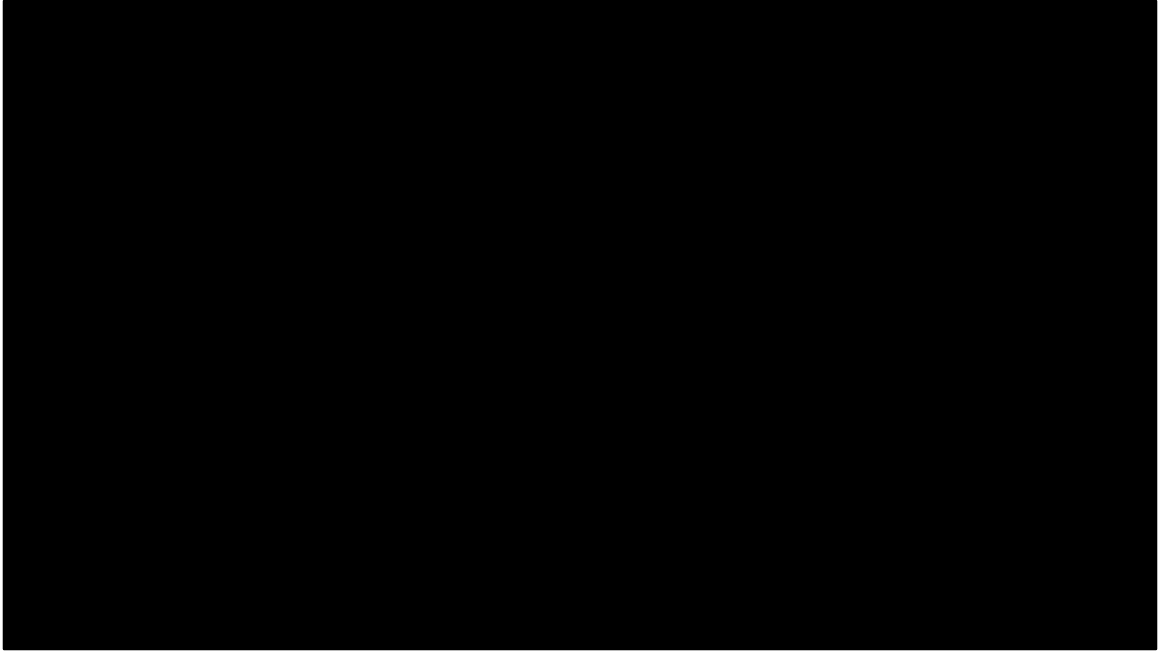
- Technical art on *Total War*
- Examples
- Lessons
- Takeaways



A quick overview of the presentation then, we'll start by quickly explaining what the words "technical art" mean at Creative Assembly on *Total War*, specifically.

Next, we'll throw up some caveats that have to be mentioned before we dive into examples, which will form the meat of the presentation.

Lastly we'll look at lessons and takeaways, at which point we should have some time for questions.



So what is total war? For those of you who are unfamiliar with the franchise or maybe have heard of it but never played it, I've brought a video that illustrates the title I'm here to talk about today: Total War: ARENA.

Technical Art on *Total War*



So now that you know what Total War is, I will very quickly break down what the words “technical art” mean for us specifically.

Our Team

- Service team
 - **Centralized** – we work on every project
 - Most tools and pipelines need to work on **every project**
 - Some tools are project-specific



On Total War, the Technical Art team is a service team – this means we are not dedicated to a single project as a team, and our tools and pipelines are shared globally.

Of course, there are always cases where projects need specific solutions to the problems only they have, but even in those cases we always attempt to make the technology work, at least theoretically, for every project.

Our Team

- Small team
 - 80+ clients across *Total War*
 - ~6 concurrent game projects



We are also a fairly small team; with just 4 tech artists, we service over 80 developers across all Total War games, which span on average 6 projects in concurrent development, at varying stages of production.

We've grown over time, but just to put things into perspective, when I started at CA, the Technical Art department had only existed for about 3 months, and it was just three of us.

Our Role

- Art pipelines and tools
- Conventions and standards
- **Interface** between art and programming



Our role in this picture is to build art pipelines and tools – a lot of our time in this goes to establishing conventions and standards, like folder structures and naming conventions, but we also play the role of interpreter between the art and programming departments.

As you're all no doubt aware, as art for games has become more and more specialized, and programming more complex, the art and programming departments can at times have a challenging time effectively communicating with each other.

As technical artists, it is our job to facilitate this communication; we talk to programmers to establish what it is they need from the art department for their systems to work, and we talk to artists to find out what we can do to make their lives easier.

This is a crucial part of what we do, as it can either save or cost hundreds of man-hours depending on how right or wrong this communication goes.

Before we continue, out of curiosity I would like to ask everyone in the audience to put their hand up if they feel this definition of tech art applies to them or their studio.

Cementing Your Duct Tape



Diving in then, I'm going to start by laying some groundwork to get everyone on the same page.

Duct Tape?

- Quickly implemented solutions
- Function over form
- Plug the leak vs. replace the pipe



What do I mean exactly when I say “duct tape”?

When I use this term in the context of this presentation, I will be talking about solutions that:

- Are usually quickly implemented
- Prioritize function over form, meaning they are more concerned with doing something at all than with how they do it, and in other words...
- Serve to “plug the leak” instead of replacing the pipe.

This last point is key: these types of solutions should never serve for a long time in the shape they usually have at the beginning of their life cycle.

They are explorative experiments, meant to quickly solve an urgent problem, while teaching you about the solution you will actually need further down the line.

Examples



On to what you're all here to see then, some examples illustrating all of this in a way you can visualize!



Examples from *Total War: ARENA*

How a single data change led to a host of improvements

How we made the majority of UI production automatic

To do that, I've brought along two examples from the production of Total War: ARENA, the second of which was only made possible because we did the first; something we didn't actually know beforehand.

The first example will cover how we changed how we deal with getting data into our game; how we determined the nature of the problem, prototyped a solution and eventually fully implemented the change.

The second example covers how we automated the majority of our UI production; this will also show how this change was only possible because the first change had unforeseen advantages.

The “Variant Clean-up”



First up, the example we commonly referred to during production as the “Variant clean-up”.

One single, **small data change** we made resulted in a big production speed increase, better tools, and eventually, automatic UI generation.



In essence, this term, “variant clean-up”, refers to a single change in how we create our units and store their data, meant to simplify and unify where that data was stored. We did this change initially to make it easier for artists to interact with the information that goes into the game, but it turned out this simplification made our data a lot more accessible to tools as well – directly leading to the automated UI pipeline shown in the second example I brought with me today.

The “Variant Clean-up”

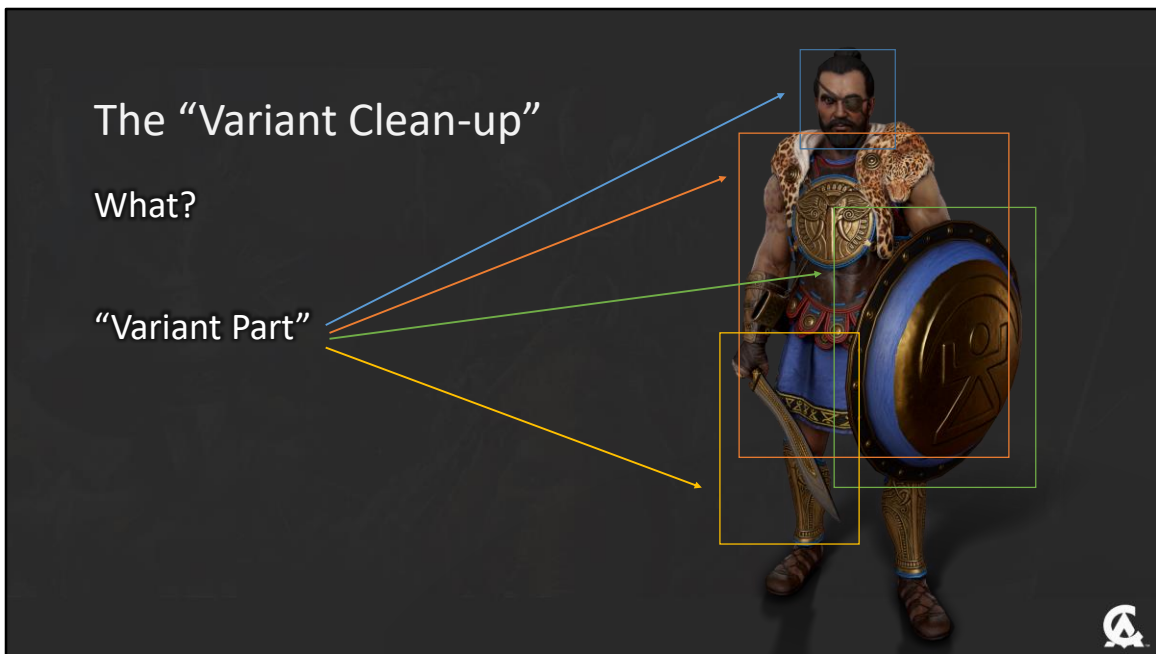
What?

“Variant” (unit)



Before we begin explaining what it involved, a quick vocabulary session.

In Total War, we refer to our units as “variants”. This is a term that stems largely from code, and points to how the system is used to put units on the players’ screens that are composed of parts, each of which can be varied to prevent visual repetition. Hence, in this context, the word “variant” can be substituted for “unit”, if that’s easier.

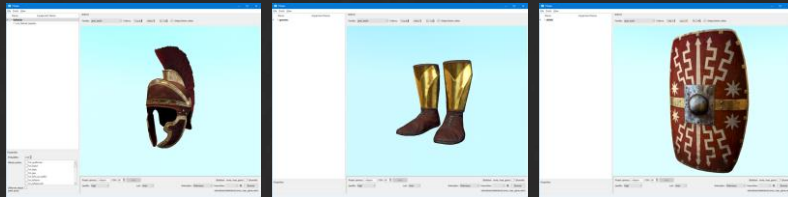


Going further down into the system, each “variant”, then, is constructed of “variant parts”.

In this illustration, I’ve marked parts that might be considered or varied together in the same colour. Here you can see that in this unit’s case (the Carthaginian commander Hannibal), we would separate his outfit from his skin, so we can swap the outfit out easily for things like cosmetic skins. Commonly, things like shields, weapons and heads are separated as well, allowing us to swap those parts out when the unit acquires different weapons or gear.

Step 1: How does it work *now*?

- Multiple systems were added over the years
- Limited tools to deal with the new data

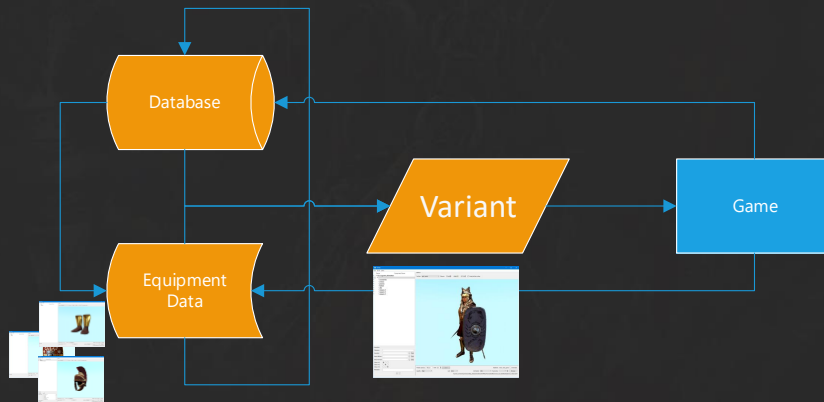


All of this information, things like damage, purchase price etc., gets stored in our database – the big data structure that drives how the game works and behaves. It was therefore logical to also include a reference to the variant parts each equipment references in this database.

The problem this presented over time, was that our in-house unit editor didn't actually then include this information – in simple terms, artists were not able to easily preview the equipments they were creating on the units the equipments were for. Instead, we were storing equipments in separate files, for the game to assemble at run-time.

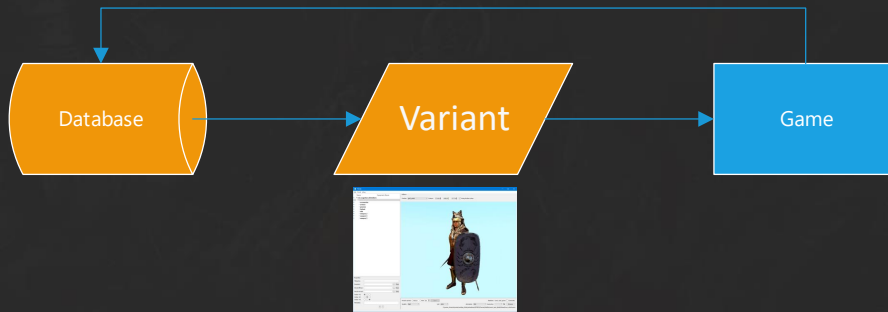
The source of this problem was that data was being stored in media that were not built to handle it – in this case, visual information was being stored in a database designed to hold simple strings and numbers.

Step 1: How does it work *now*?



Thrown into a flow-graph, it starts to become clear what the problem is – there are various points at which multiple tools are needed during the pipeline, and no one step in the process is clearly responsible for a single thing. This makes it harder to work with the data affected by these processes.

Step 2: Can we store it in one place?



So step 2 then becomes to answer the question of how we simplify this. We did some R&D, and based on this flow-graph, displaying how we would *like* things to look, we came up with a prototype.

Step 2: Can we store it in one place?

- 1 day of R&D
- +/- 3 days of work
- Lots of **redundant code was removed**



All in all, this prototype took about a day to put together, followed by 3 days of work to fully implement the code change.

This change, since there were fewer systems to deal with now, was very well-received by programmers: it actually removed more lines than it added, including a big chunk of now-redundant code.

Step 3: What we patch vs. what we automate

- Initial data change was done **entirely through a one-off script**
- This later became a **passive process**
- By then, we fully understood the problem space

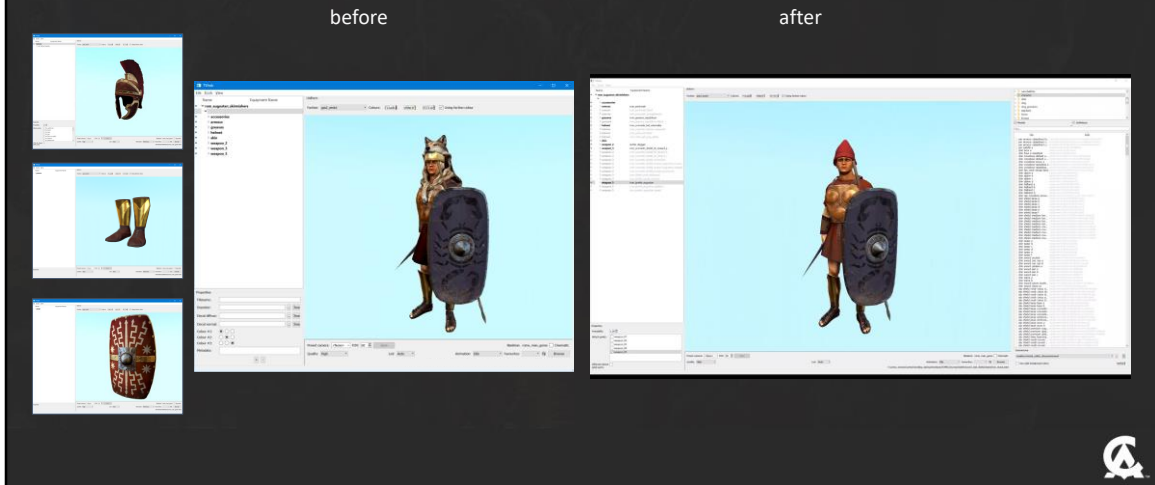


The last step was deciding which bits to patch and which bits to automate – a fair bit of thought actually went into this;

Initially, given the size of the data change required as a result of the code change, the entire change was done automatically through a one-off script that I wrote. Because I kept that script local for a long time, I was able to really understand the problem space, and in that way was able to refine this process over time.

Eventually, once we were confident that the process was safe enough to run automatically, we turned it into a passive one, at which point the script was almost entirely re-written, but this time with full understanding of what problems it was supposed to solve.

The “Variant Clean-up”



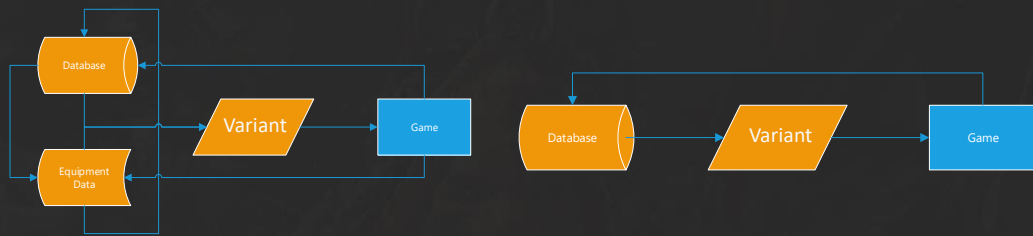
Illustrating that with an example, on the left you see a return of the screenshots shown earlier, with an example of how we used to compose units: each unit had its parts stored in separate variant files, which the game engine then assembled later. This made it very hard to preview what a unit might look like before viewing it in game, making artists' and designers' lives harder.

On the right you can see what our files looked like *after* the change. Instead of storing everything separately, each unit file now contains every equipment it can ever have. Since this is a distinctly defined set of equipments, based on our game design, this is only a limited set per unit. In this way, artists can now view every visual combination a unit can have, without having to start the full game. This directly resulted in a nearly 50% increase of production time for artists when working with our unit editor tools.

The “Variant Clean-up”

before

after



Just to re-iterate, if you visualize this change in a flow-graph, we haven't actually done a big change – in fact, in terms of database changes, this change only represented the removal of a single column from a single table. But the result was a much clearer data flow, better and simpler data, and happier developers.

The “Variant Clean-up”

This one change led to a host of later improvements:

- On-demand unit part loading
 - Start-up time improvements
 - Powerful metrics gathering
 - Ideas for future performance improvement systems
- ... and our entire UI pipeline



This all then later led to a whole host of improvements we didn’t foresee beforehand,

Like on-demand unit part loading, which improved our start-up time quite dramatically,

Powerful metrics,

Ideas for performance-improving systems, and more importantly, it indirectly led to the automation of most of our UI production, but more on that in the next example.

The “Variant Clean-up”

This is an example of an “accelerator”

- Simplification of data led to a deeper understanding of it
- Which led to better processes



All of this boils down to the fact that this change was an example of an “accelerator”; it’s a change that, through simplification of data, leads to a deeper understanding of not just the data itself, but of the processes affecting it. That then in turn leads to better processes overall, which is in everyone’s favour.

Total War: ARENA's UI pipeline



Going on to the second example I've mentioned a few times now, let's talk about ARENA's UI pipeline.

ARENA's UI pipeline

Problem: UI production is expensive

- Lots of icons (... hundreds)
- Content changes incur additional overhead
- Not scalable

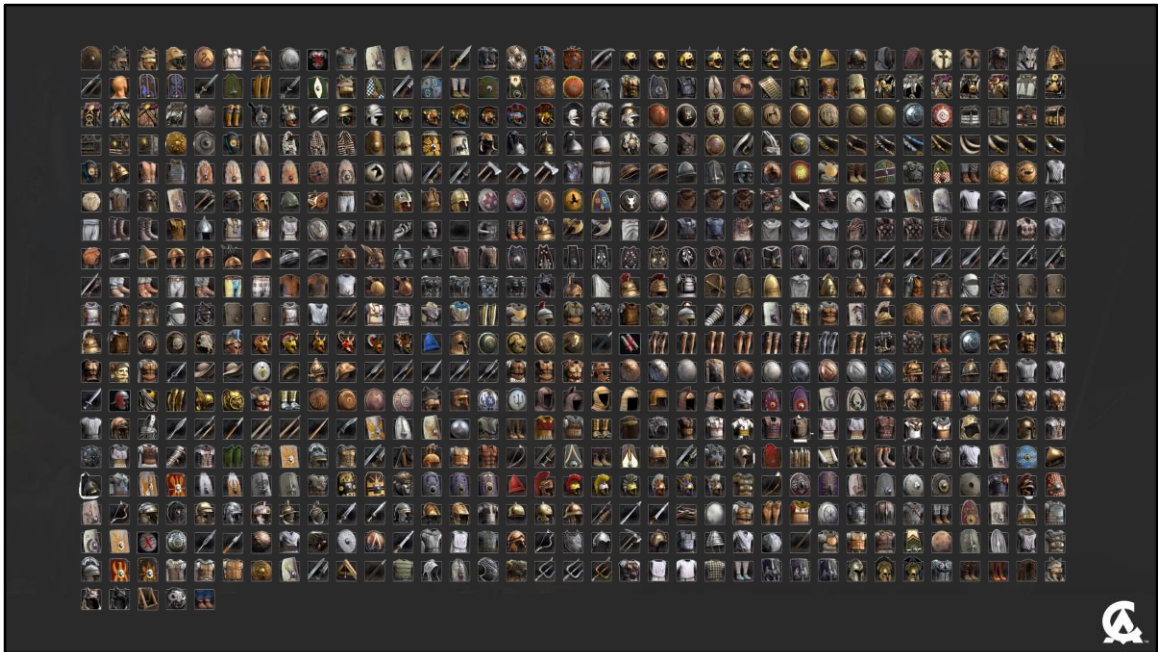


During production, we very quickly noticed that UI work was becoming very expensive.

In our game, we have a LOT of icons, covering everything from achievements to skills, commander portraits, unit cards, equipment icons and unit tree icons.

If we zoom in on equipment icons for a second, the biggest problem they caused was that the icon needed to represent the asset it actually equipped when the player clicked the button.

This means that these icons are subject to change whenever the asset they represent changes – given the fact that they were being produced manually, this is hardly ideal, and certainly not scalable.

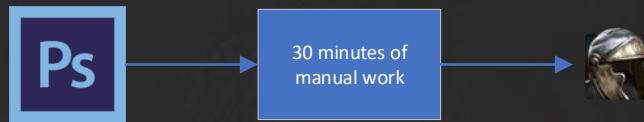


Just to illustrate the scale of the problem, this is just a small sampling of the icons we use in ARENA.

At a certain point, we found ourselves in a situation where production required several hundred of these in a short amount of time. At the time, the UI team was stretched thin, so this requirement didn't really fit into the scope of what we were doing.

Instead, we decided to take a look at tackling this problem using automation.

Step 1: How is the data maintained?



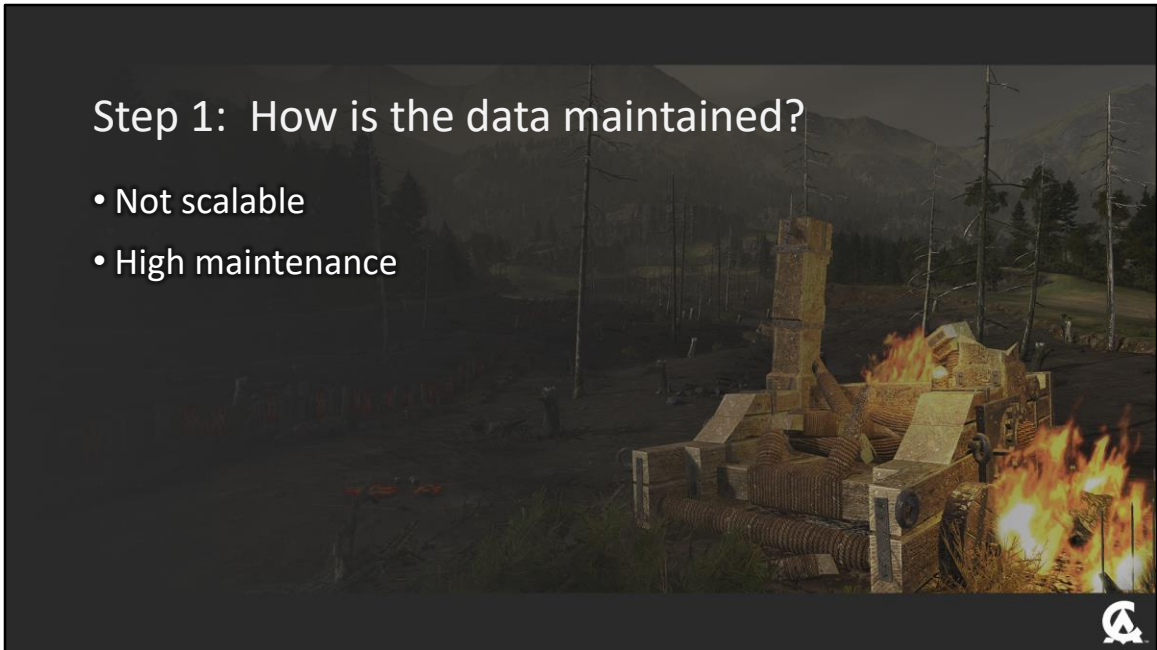
The first step in this process was to establish whether we even could automate this process. To do this, we took a hard look at the existing process... which didn't take very long, as it was very straightforward.

The pipeline, then, looked a bit like this. **(fade in image)**

a UI artist would get a request for an icon, and then produce the final image in whatever way they choose, exporting the final result to the game.

Step 1: How is the data maintained?

- Not scalable
- High maintenance



It was quickly apparent to us that this situation was not very future-proof; given that no external systems were being used other than Photoshop and maybe our engine for the odd screenshot.

This meant that we couldn't really scale up – producing hundreds of icons on demand was simply not feasible. Additionally, any kind of design overhaul of the UI would require recreating every icon in the game, not to mention the overhead caused by changes in the content the icons were based on. After all, the icons are meant to actually show the asset your unit gets equipped.

Step 2: What tools exist that we can use?

- Existing Python code
- Marmoset Toolbag 3 (supports Python 3.6)
- Substance Batch Tools

Step 2 was to examine the existing toolbox: what did we have that we could use to build a quick prototype that might be able to deliver the content we need?

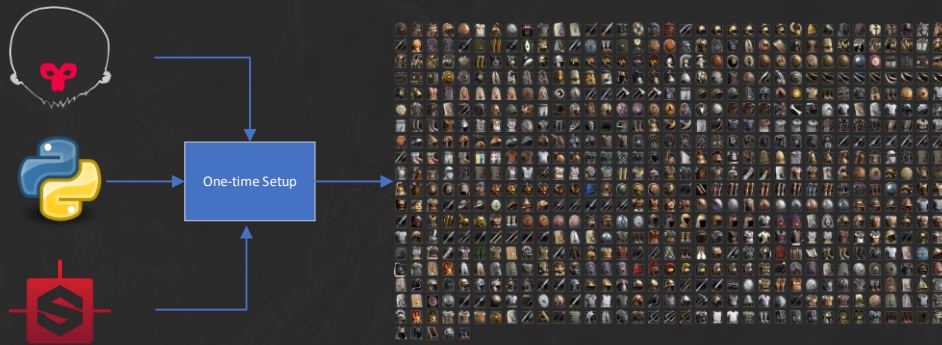
The answer to that question actually turned out to be Marmoset Toolbag – in the normal process of submitting and delivering content to our repository, character artists regularly created marmoset toolbag files that presented the content they created in a nice lighting setup, so external viewers can have an easy way to browse through our catalogue of assets and immediately interact with our content without needing the game engine to do so.

This presented us with an established data set we could use – all we really needed to do was to take a screenshot of each equipment, format it into the right resolution, and export it to the game!

This is where Marmoset really came to the rescue – not one month before we found ourselves in this situation, they shipped Toolbag 303 – which added Python 3.6 support. This allowed us to use our already-established code base and put it to work to do most of the work for us. Combined with our expertise using the Substance Batch Tools, we found we had most of the components we needed ready to go – all

we really needed to do was to hook them up to work together.

Step 3: Proof of concept



The next step was our proof of concept – hook these tools together, and create a way to produce lots of icons.



ARENA's UI pipeline

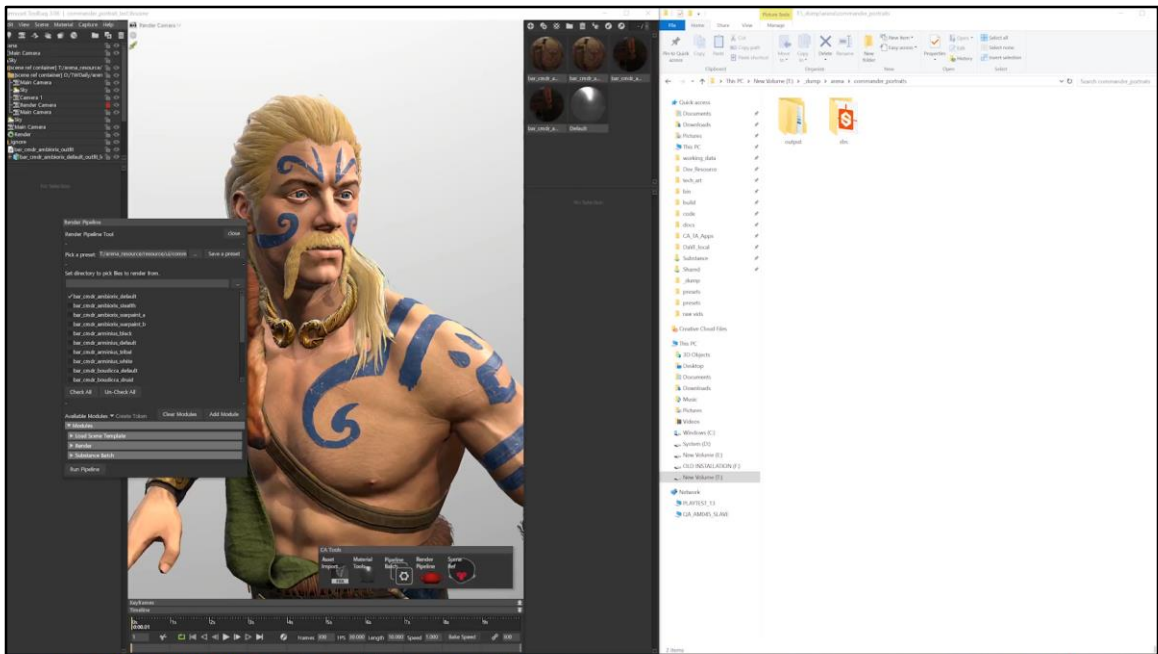
Proof of concept

- Working in less than an hour
- Production-capable in under a day



Because we were not actively trying to make a sustainable pipeline from the get-go, we were actually able to get something working in under an hour, with a production-capable pipeline up and running in under a day.

Again, none of this was about building a sustainable pipeline – it was only about building something that would work well enough, and for long enough, for us to get our content out.



Why is this duct tape?

Proof of concept created and working in two days

- Quick turnaround
- Single use
- Not designed for other work
- Only possible because of data work from previous example



These next few slides will go into why this counts as duct tape – this is important to really drive home the point of this being an ad-hoc solution: it worked once, and so we kept chipping away at it to make it keep working while improving its features and stability.

First and foremost, it's duct tape because it had a very quick turnaround, focused on single use, therefore sacrificing form over function. The goal was to do *something*, not to do it well or in a particularly elegant way.

Given that it was built this way, using this pipeline for any purpose other than what it was designed for would have been very challenging.

Even with this all being said, none of this would have been possible without the data change from the previous example – since we actually had to construct our marmoset scenes based on the equipments our units were wearing, we needed our data set to be easily accessible. Since we standardized the data set, this bit was easy, and we were able to focus on the image creation part, rather than on getting the content into Marmoset at all.

Why is this duct tape?

Used in production while prototyping

- Production work with the tool was mostly done by Tech Art
- Actually working with the tool led to more improvements

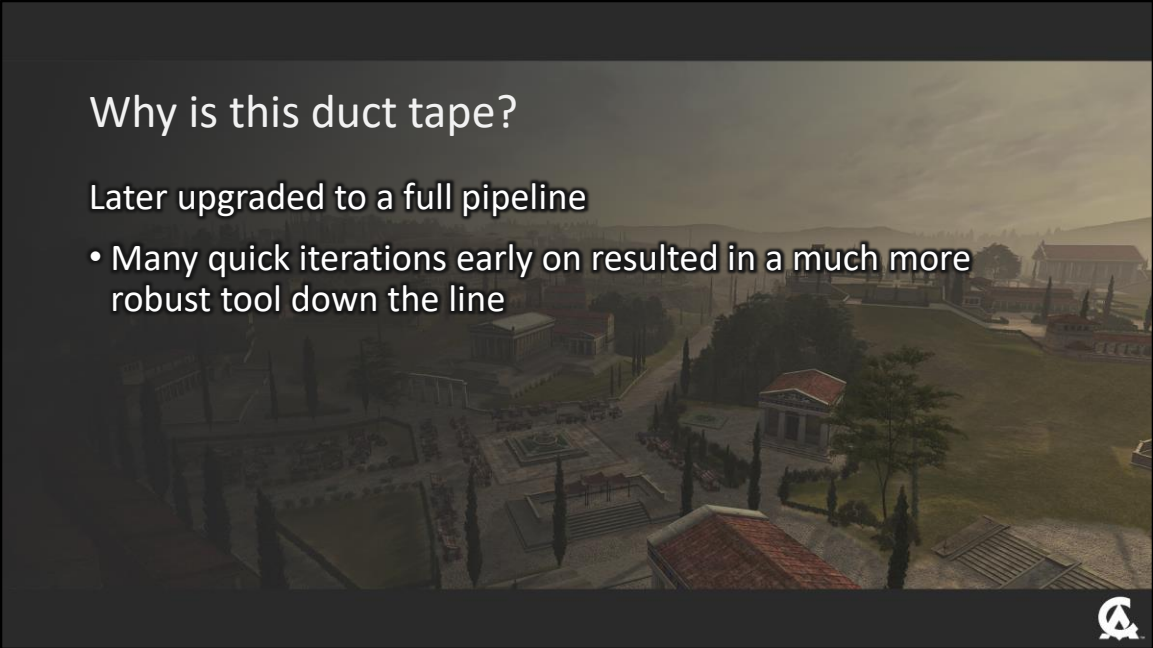


This temporary pipeline then was actually used in production – I actually had our one UI artist sit down next to me while I fiddled with the code on the fly and worked with the primitive UI. This meant that while I was working on the tool, I was able to resolve a lot of user experience issues, since I was working with it myself. A lot of the features that ended up in the final version were born during this stage, and I might never have conceived of them otherwise.

Why is this duct tape?

Later upgraded to a full pipeline

- Many quick iterations early on resulted in a much more robust tool down the line

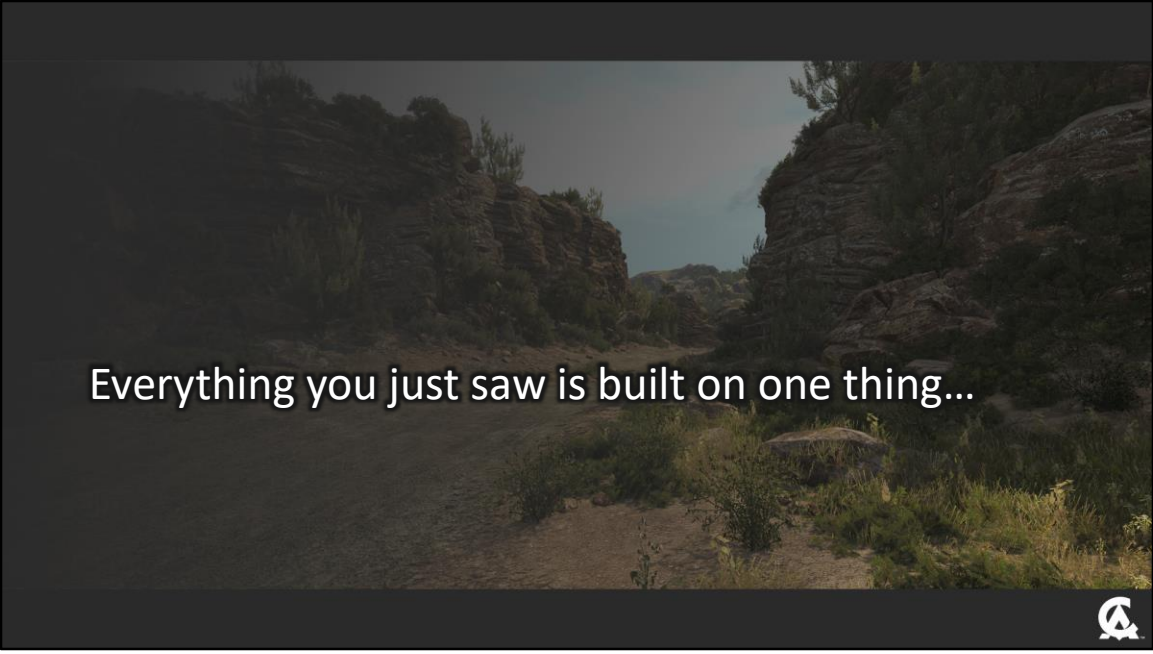


Most importantly, illustrating the flow from duct tape to foundation in this case, this pipeline, while implemented as a temporary solution at first, was later rebuilt from the ground up into a fully sustainable pipeline, and is now slated for use on three different projects. As such, the many iterations done early on and the extended prototype stage during which tech art had full control not just of the tool but the content produced with it, resulted in what turned into a very robust and versatile pipeline; something that might not have happened had we not gone through this process.

The Catch



With that, we come to the crux of it all... the catch, if you will.



Everything you just saw is built on one thing...

Basically everything we said before this point has had a catch...

File Conventions



It all starts at the foundation

Good conventions

- Make tool development easier
- Make prototyping faster
- Make tools more reliable



Basically, none of what I showed here today would have been possible without the conventions that I spend most of my days enforcing – all based on a single underpinning philosophy:

Good conventions make tool development easier,
Prototyping faster and
Tools more reliable.

To anyone in the room that has ever developed a data manipulation tool, I'm sure this isn't news – if you have a strong set of conventions and standards, it means you have to spend way less time accounting for edge cases and exceptions, and more time on actually implementing the functionality you're after. What's more, with strong standards that account for a variety of use cases, you sometimes don't even need artist time to help create the data you need for your pipelines; this last bit is why, with the help of just one UI artist, we were able between the two of us to generate over 260 icons in less than a day using the pipeline presented here – something that would not have been even remotely possible had our conventions not been as strong as they were.

It all starts at the foundation

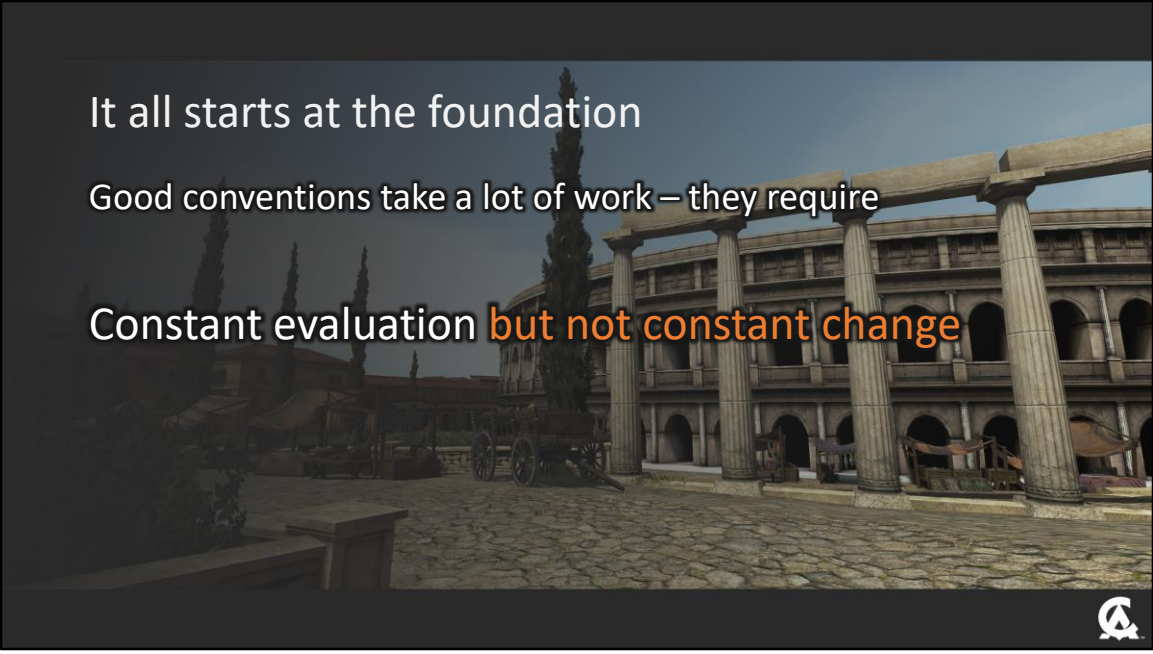
Our conventions in this case

- Meant validation was easier
- Made the use of “duct tape” safer
- Served as a safety net independent of any one tool



The message here is clear: when working with hastily implemented solutions that sacrifice form over function, strong standards mean that your code can make more assumptions, making data validation a lot easier, and the use of duct tape safer, as unpredictable scenarios are much less likely to occur.

In fact, conventions in a lot of cases can serve as safety nets; when all other systems fail, your conventions serve as a safety net with no moving parts: they're still a part of a process, but they form an immovable, stable foundation to build your tools on.



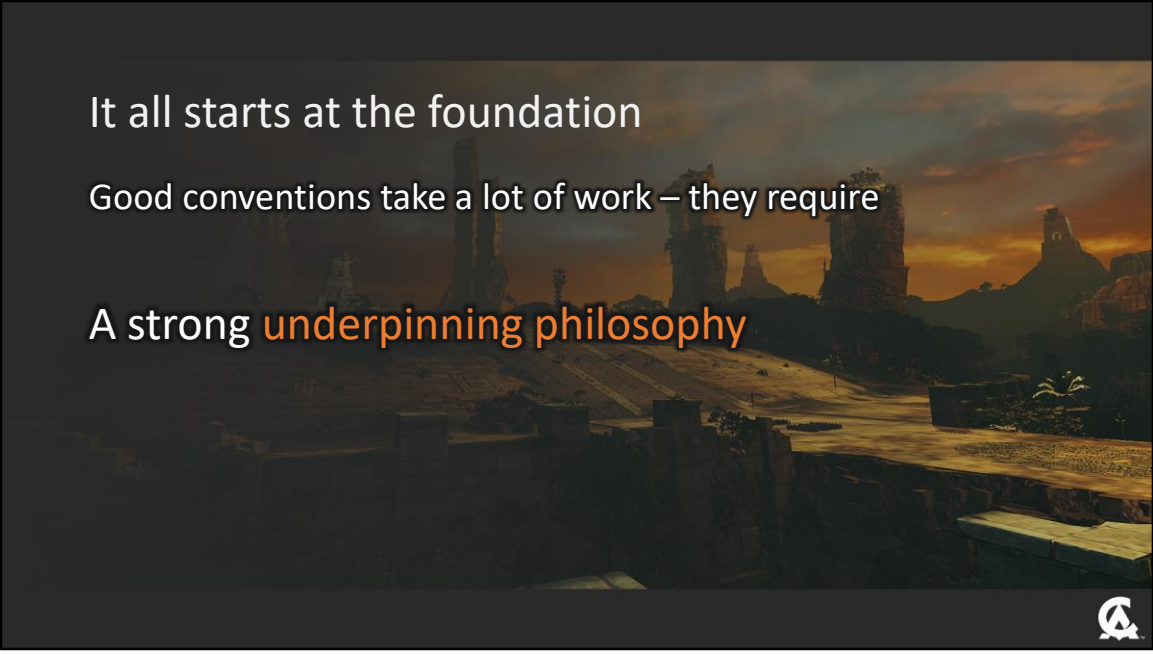
It all starts at the foundation

Good conventions take a lot of work – they require

Constant evaluation **but not constant change**

Of course all of that is easier said than done; strong conventions are hard. Very hard. But as we developed ours over the course of the last few years, and iterated upon variations of the same basic ideas, we found that there are generally three key aspects that all strong conventions share.

The first, is that good conventions require constant evaluation, but extremely importantly, they do NOT require constant change. Treat your folder structures as sacrosanct; only change them when you *_really_* have to, no matter the scope of your project. And when you do, make sure to do your research. To really push that point home, we changed our folder structure 4 times over the course of 2 years, and every time, it took about a month's worth of deliberation before we would build a case strong enough to justify incurring that kind of technical debt.



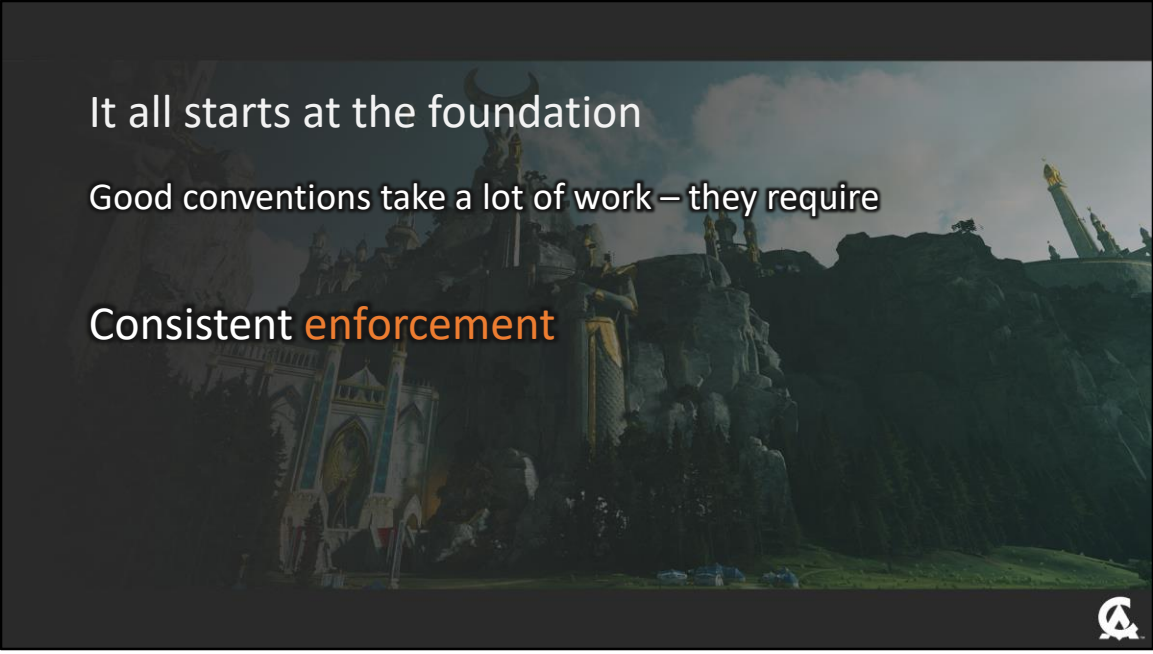
It all starts at the foundation

Good conventions take a lot of work – they require

A strong **underpinning philosophy**



The second key aspect is a strong underpinning philosophy that helps you keep track of things. This philosophy is a set of core principles that anyone can use to take an asset from an unnamed whitebox to a fully labelled catalogue entry. It should answer the question of *how* to name your assets. *How* you get to that highly specific name. More than anything, this set of rules must be made as simple as possible, and as rigid as possible. Adapt your assets to *it*, don't adapt it to your assets.



It all starts at the foundation

Good conventions take a lot of work – they require

Consistent **enforcement**

The last aspect is obvious; it can not be overstated how important it is to enforce conventions. Everyone's strategy is their own to choose in this, but we found periodical, pre-scheduled evaluations of our depot to be the most reliable way of keeping track of this. Over time, the frequency of these evaluations can decrease if no major defects are found, but it is important to keep them up, especially if problems are encountered. More than anything, people who break conventions must be forced to clean up their own mess. This is without a doubt the best way to teach people how to do it the right way, and it's rare for this kind of thing to occur more than a handful of times with the same person if they're forced to clean it up themselves every time.

What we learned



Closing off then, I want to quickly touch on what we, ourselves, learned from the various processes we examined here, and from the solutions we came up with.

It is after all important to recognize that no solution is perfect.

All about the big picture

Beware of tunnel vision

- Do regular **follow-up** evaluations

- Does the solution still fit the project?
- Have the project goals changed?
- Has the scope changed?
- Is the scope likely to change?



First and foremost, everything is about the big picture – you are working to create a game after all, not just the tools that make up its creation process.

The solutions you create only hold value in the context of the project whose problems they solve, and as such creating them for their own sake is a bad idea.

To make sure not to fall into this trap, the first thing you can do is to regularly evaluate your tools.

In doing this, it is key that you continuously consider whether the solution still fits the project, whether project goals or scope have changed, or are likely to change.

All about the big picture

Beware of tunnel vision

- Evaluate your tools in the context of the **whole project**
 - Can the need for this tool **be eliminated**?
 - Is **another tool** more appropriate?



Keeping that in mind, remember that tunnel vision is the enemy of all.

Above all else your solutions need to make sense in the context of the project as a whole.

The goal should **always** be to **eliminate the need** for these solutions, rather than to create better ones. After all, a chain with fewer links has fewer chances for failure.

Keep in mind the way your data flows between different parts of your pipeline, and ask yourself whether you can make a change in how the data is structured that can eliminate or circumvent the need for entire portions of the pipeline, making it shorter and therefore more stable.

What we learned

Lots of things *do* require a proper solution *now*

- Not always obvious
- Errors in judgement can happen



First and foremost, I want to stress that duct tape should NOT be your go-to solution for everything. It is a technique like any other, and it should only be used in cases where it's just the best way to deliver content on time.

Lots of things DO require a proper solution right away, but that's not always obvious. In those cases, the standard method of prototype – development – deployment is very much called for.

What we learned

- Transition duct tape into tools *faster*
 - Patched tech like this *accumulates technical debt very quickly*
- NEVER build anything on top of duct-taped tech!



The second valuable lesson is that the nature of development is such that tech tends to be forgotten about – if it works, it's usually left until it breaks. This can be dangerous if there is duct-taped tech somewhere in the pipeline, because you run the risk of building something on top of it.

Because this is the case, if you find yourself shipping your game using this technique, your highest priority should be to get it out as fast as possible, and replace it with a proper implementation.

Going forward

- Work out when to remove the duct tape...

- When is duct tape appropriate?
- When will production require proper tools?

- ... and stick to it

Going forward then, one of the bigger challenges we've faced using this technique is that you need to plan in advance when you'll remove the duct tape. In our case, we had the first scenario that required the pipeline we patched together, but when production months later required the pipeline again, we had planned in advance to have fully implemented a stable, future-proof implementation of the concept.

The trickiest part of that of course is to stick to the plan. Production realities can mean shifting priorities, but don't underestimate how dangerous it is to leave tech like this in. It works in the short term, but you wouldn't see an engineer closing up a gas pipe with some duct tape from his back pocket and then just walk away, either.

The best way in these situations to prevent further damage, is to just rip the band aid off – if you know you're going to have to do it eventually, you might as well do it now and save yourself from the pain of having to rip it open again when you don't remember how you built it, or worse still, if someone else has to.

Takeaways



With that, we've arrived at the final portion of this talk: takeaways.

The benefit of temporary solutions

- Not every problem requires the “proper” solution *now*
- *Managed* duct tape can be very useful
- Changing production requirements might mean a solution is best left as a temporary fix until a bigger system replaces it



The first thing to take away is that there is a benefit to using temporary solutions, as long as it is fully understood that it should be treated as temporary.

When deciding on whether to use a “proper” fix, in this context it’s always best to just sit down and talk with people to see if you need a full solution *now*, or whether you might be able to come up with something bigger that circumvents the problem entirely.

Every solution is a learning opportunity

- Everything can be done better...
... but not everything has to
- Every step clarifies the problem space and helps you understand a better solution



Because as everyone knows, everything can always be done better,
But not everything has to.

Every step you take towards solving your problem will clarify the problem space – it helps you understand and get to the core of the problem, which inevitably leads to better and more comprehensive solutions. Prototypes and quickly implemented, function-over-form-style tools are great at helping you really create an understanding of the data you're working with.



I gave this statement its own slide to really emphasize just how important this is – it can **not** be overstated how important it is to keep the number of systems that interact with your pipeline to a minimum.

Every tool you need to create content or to get it from artists' brains into the game, is an additional link in the chain that can fail. Really try to identify all the parts of your chain, and trim where possible. This is a matter of constant evaluation and attention, but it is well worth it when that inevitable production crisis hits and you suddenly find yourself debugging three different tools just to identify which one is screwing up the textures.

Pick your targets

- Work to *eliminate* tools, not build new ones
 - A shorter pipeline is always better
 - A data change can be *more effective than any tool* (see: example)
- Build safety nets
 - Build system-independent structures that don't fail, such as good folder structures and naming conventions



The last takeaway is perhaps the most important one:

When creating solutions, your goal should always be to eliminate the need for tools, not to create new ones. A shorter pipeline is always, without exception, more stable, and most of the time, better.

To that effect, as illustrated in the first example, a data change can be more effective than any tool in optimizing your process. And if that change is one-off, you might as well use short-lived code to do it.

Which leads me to the concluding point of this talk, which is that safety nets are incredibly important.

Never build your systems on other systems if you can help it – find something that doesn't move or moves very slowly, like a naming convention or folder structure, and use that as your foundation. When all else fails, and it will, these immutable parts of your pipeline will act as safety nets you can use to catch yourself. After all, everything starts at the foundation.

Great resources

- GDC 2018 talk: Michael Malinowski, “A Practical Approach to Developing Forward-Facing Rigs, Tools and Pipelines” (free on the Vault)
- Wikipedia definition on Technical Debt
- <https://www.breakingthewheel.com/video-game-art-pipelines/>



Before I go

Contact me:

mattias.vancamp@creative-assembly.com

Questions?



Before we go to questions then, if you wish to contact me you can do so through this e-mail, I'd be more than happy to answer any questions we can't get to today. Thank you for listening, I hope you enjoyed the talk, and please do leave feedback in the GDC app.

[applause, hopefully]

[questions]

Potential question: isn't this presentation basically just saying you should prototype your tools before using them in production?

Answer: it is, but there's a subtle difference, I think: part of the message I was trying to convey here today is that it's okay to use said prototypes in production directly, and that using your own tools can be a great experience that really allows you to work out the kinks in the process you're building.

Potential question: where would you say the balance is between using hacks, and building tools?

Answer: I've found it generally depends on the production pressure. Duct tape, or

hacks if you want to call them that, happen naturally when time pressure is high and the issue just needs to get solved. However, if you do find yourself in a high-pressure environment like that, it's easy to get tunnel vision and not see what might be a way around the problem entirely. I suppose the real answer to your question is that it's up to the situation, and your best judgement at the time, taking into account as many factors as possible.

From Amanda:

Isn't ARENA offline now? / How is ARENA doing?

Yes the game servers are offline, live service ended on 22nd Feb.

Why did the game have to shut down?

Unfortunately it wasn't meeting business expectations but the level of community passion for the game was incredible and inspiring.

Is it coming back one day? / On Steam?

At this point we are focusing on taking learnings and experience from ARENA into other TW projects.

What are you working on now?

I'm working across Total War projects.

BEST
STRATEGY GAME
GAME CRITICS AWARD
E3 2018

BEST
STRATEGY GAME
gamescom
2018

NOMINEE
BEST GAME
BRITISH ACADEMY
CREATIVE AWARD
2018



STUDIO
OF THE YEAR
DEVELOP
AWARDS 2018



BEST STRATEGY GAME
TIGA GAMES INDUSTRY
AWARDS 2018

CREATIVE-ASSEMBLY.COM

@CAGAMES