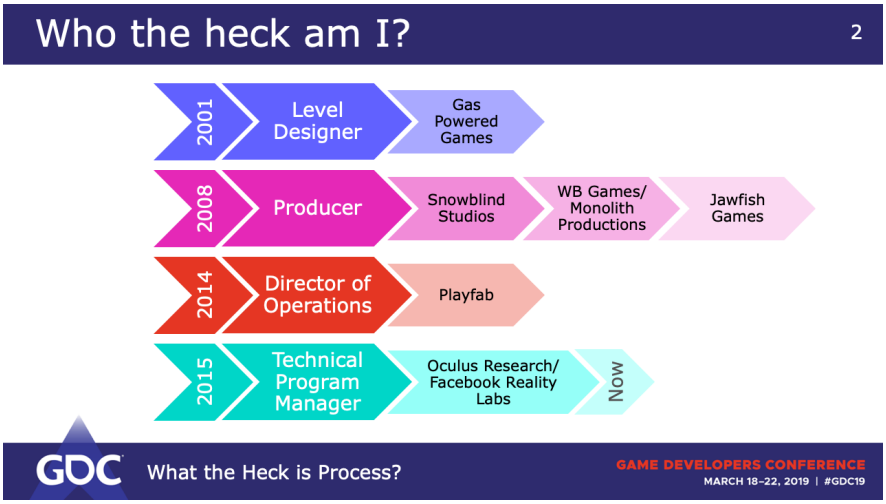

Hello, and welcome to Producer Bootcamp!

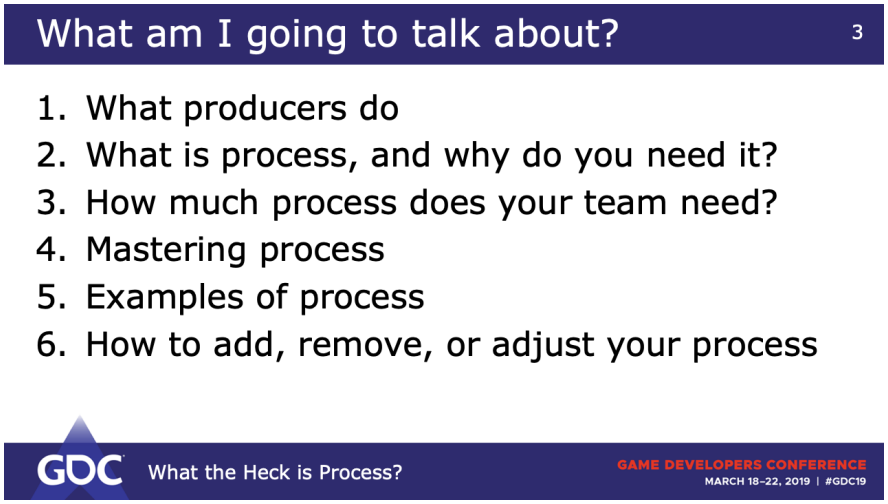
Today I'm going to try to answer the question "What the heck is process?" because it's a question I've been asked, it's a question I've asked, and I think knowing the answer is important for growing as a producer.




First of all, who the heck am I and why should you listen to me talk about process for an hour? My name is Ruth Tomandl and I've been a producer/PM for 11 years (and 2 days!), on 8 different teams at 5 different companies.

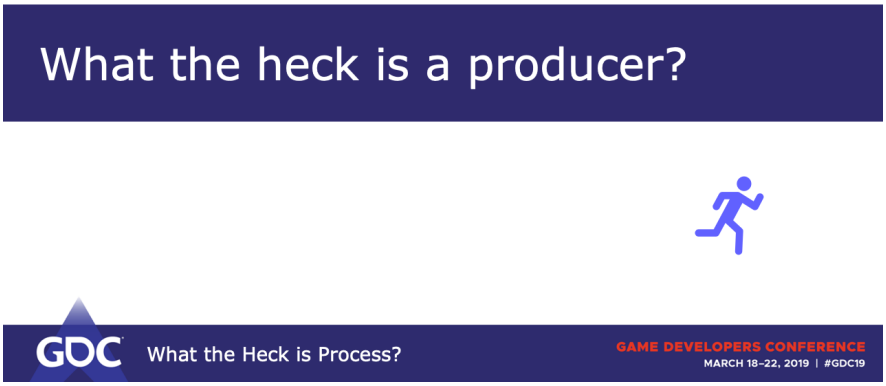
Before I was a producer, I got my start in the games industry at Gas Powered Games as a Level Designer working on the Dungeon Siege games. I found that I really liked scheduling and coordination, so I looked for a Producer job. I worked at a big first party studio for a while, and then at a couple of startups.

I'm currently at Facebook Reality Labs (formerly Oculus Research) managing an AR and VR research team. Now, other than my stint as a level designer I've been essentially doing the same job for my whole career, though the titles have been different. But I've found that whatever the project, whether you're early in prototyping or 3 days from shipping, the job of getting the right work done is pretty much the same everywhere. And getting the right work done is what producers do.




So in this talk, I'm going to talk about:

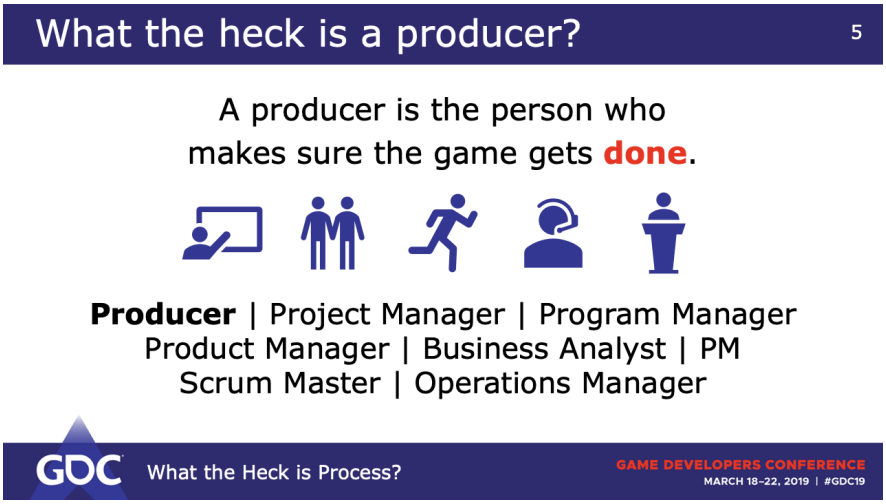
1. That job; what do producers actually do?
2. What is process, and why do you need it?
3. How much process does your team need? And how can you tell?
4. Some ways to become a process master and build processes that your team will use and love! Or if not love, at least tolerate.
5. Some examples of processes that I've used that worked really well, and some that were terrible, and why.
6. And last, some practical advice on how you can add, remove, or adjust your team's process.




First, let's talk about what producers actually do.

How many of you are currently producers on a game team? How many are game developers but not producers? Students? People who aren't producers yet, but want to be?

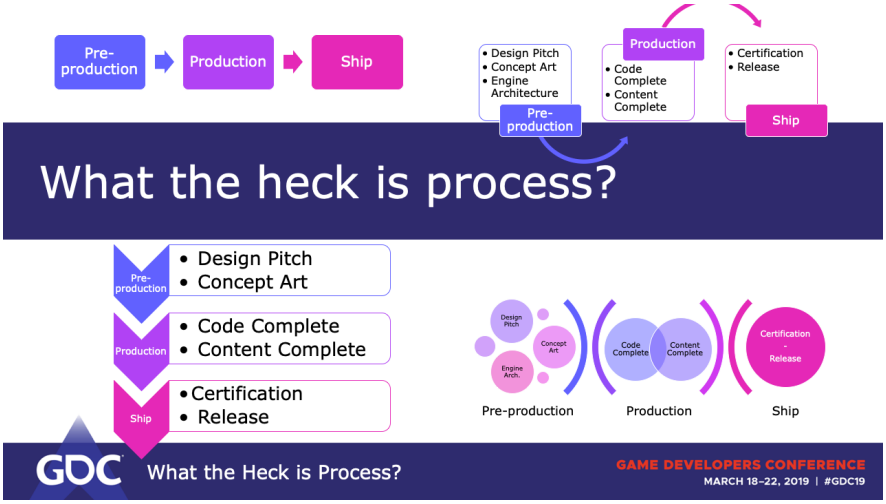
When I was a level designer, I kind of knew what producers did. I mean, they were all over the place. They kept track of things. They shielded us from the publisher, and told the publisher what they needed to know. But it wasn't until I had been a producer for a couple of years that I started to really understand what the job is.




A producer is the person who is responsible for making sure the game gets done. And by done, I mean *done*-done, shipped and available for players to play. So your job, day to day, is to make sure that everyone on your team is doing the right work, and that that work is getting done to the point it can be shipped as a game. You are the person who makes sure the people on your team know what they're supposed to do, and you check to see if they've done it, and then you communicate that it's done to the people who need to know.

This job has many names, depending on the company. Some of these titles describe jobs that are slightly different: for example, a business analyst is often more focused on market requirements, and an operations manager might be more focused on ongoing team needs than on a specific game project. Every team defines this job slightly differently, but overall "make sure the game gets done" is pretty universal.

Process is all the ways that you do that job, and all the tools that you use to do it. As a producer, process is your toolbox, the way programming languages and dev environments are an engineer's toolbox or shader techniques and UV mapping tools are an artist's toolbox.




If it's our main toolbox, we should all know what it is, right? Well,this is what PowerPoint thinks process is. I look at these diagrams and just feel lost; how can you represent something as complicated as making a video game this simply? And then I remember: I've seen tons of producers present their game projects this way, in a way that's beautiful and even makes sense. I've done it myself. So why is it always so intimidating? Why does it always feel like there's some secret that everyone knows but me?

Last year I did a GDC talk about Producer Skills, and I referred to Process quite a lot in that talk. Afterwards, I met up with a production student, and we talked for a while about what he was studying, and about my talk, and he asked me, "So...what *is* process, anyway?"

And it reminded me that when I was a new producer, I had the same question.

I had been a producer for a couple of years, and I was in a big planning meeting where one of the more senior producers, who I looked up to, said that our process was bad.

And I realized I had no idea what the word 'process' meant, though I had heard it used before. I couldn't even agree or disagree with her; was our process bad? Or is it good? What does that even mean? I felt like everyone else had some secret knowledge that, apparently, was really important to know if I was going to be good at this job.

But the thing is, I did know what process was, and I was using it. I just didn't realize it. You literally can't be a producer without it. Process is how you make games.


Because your team's process is just all the ways the different people on your team communicate, interact, hand off work to each other, and make sure things get done.

Let's look at what exactly you need to make and ship a game.

First, you need a green light; some kind of decision that yes, we're going to make a video game. This could be a thumbs-up emoji from your college friend in a chat channel, or it could be a hundred-page publisher contract to kick off preproduction. But a project always starts somewhere. Often specific people on your team will write a pitch document or make a prototype or concept, and then your team will agree yes, we're making this.

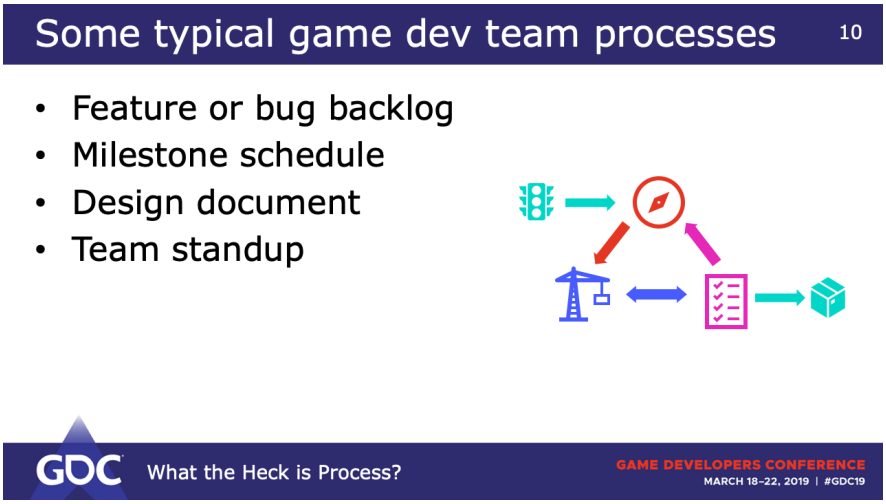
Second, you need a vision for your game. What game are we making? You can't actually make anything if you don't have some kind of direction. Again, this could be a vague idea or an insanely detailed design document, but it needs to exist. The people on your team in charge of direction are usually designers, especially the lead designer or creative director. At a larger company or when working with a publisher, you could also have executives, marketing people or executive producers, providing direction on what features your game needs to have or what setting it needs to have.


Third, you make some stuff: art assets, code, levels, achievements, etc etc. This is what most people think of as 'game development', and it might take the most time and effort, but it's only part of what you need. And all of this work is informed by the Game Vision that we made in the last step.

And you need project management. This often gets dismissed as just overhead, but it's as critical to making your game as the other pieces. You're making a bunch of things; great. Are they the right things? Are they done? Do we all agree on what 'done' means? Did we make everything we need to make? Are we ready to ship this game or not?

Because the last thing you need to make a game is: you need to ship. It really doesn't matter how good your design was, or how awesome your character art was, or any of those other things, if your game doesn't ship.


The interactions between all those different roles, the way you communicate your game vision and control your execution and manage your efforts, is process. Process is the arrows between those roles, the way information moves around your team. On game teams, producers usually end up owning the process for the whole team, as well as owning the project management, because there's a ton of overlap between those responsibilities. But you have process whether you have any producers on your team or not.




Let's go over some common processes that your team is probably using right now.

A feature backlog or bug backlog. Effectively this is a list of all the things your team has to do. I think this is the most common process in game development; I can't even conceive of making a game without it.

A milestone schedule: if your game development has multiple phases, especially if you're working with a publisher or doing work-for-hire, you'll have a list of milestones and what your team needs to accomplish by each one, with delivery dates.

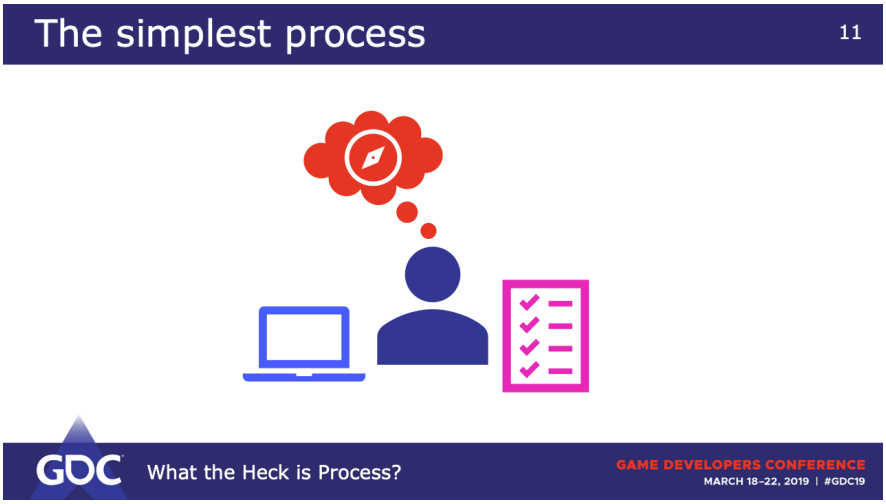
A design document: this is often a primary way that the game vision is communicated to the execution team, and often how a design team aligns on the game vision. It's a key process in greenlighting your game, as well.

A daily or weekly team standup: this is a common process for managing what work is being done, especially on small, experienced, or self-managed teams. And it's a key process in Scrum, which is a codified system of processes.

Now, there's one thing these processes all have in common: you would use them even if you didn't have any producers, or if your producer is terrible at their job. And they would exist in some form even if you didn't write them down or call them process!

If you have a good producer on your team, they can own these and make them better. And if you don't have producers on your team, what will happen is that your designers, artists, and engineers will just spend time maintaining these processes themselves.

You might think your team has no process, or you might have heard of a mythical studio where they don't have any process; I've certainly heard this legend from some of the anti-process developers I've worked with. But a team with no process at all is just a myth, and I'll prove it.




Let's look at the simplest development team possible. One a one-person team, you only need a tiny amount of process, because all the roles are filled by the same person, so communication and alignment between them are as good as communication and alignment can be. And look, not a producer to be seen!

But you do still have process: The direction is your ideas, the execution is you making all the things yourself, and management could your personal to-do list. Adding things to that checklist, thinking twice to make sure they fit your design ideas, building them, and deciding they're done and it's time to ship is the simplest process possible.

Of course, solo developers that make successful games usually do think a lot about process, and write about it, and use process that works well for them. I'll link a couple of dev logs that talk about this in the notes. It's easier to find the right process when you just have to think about yourself.

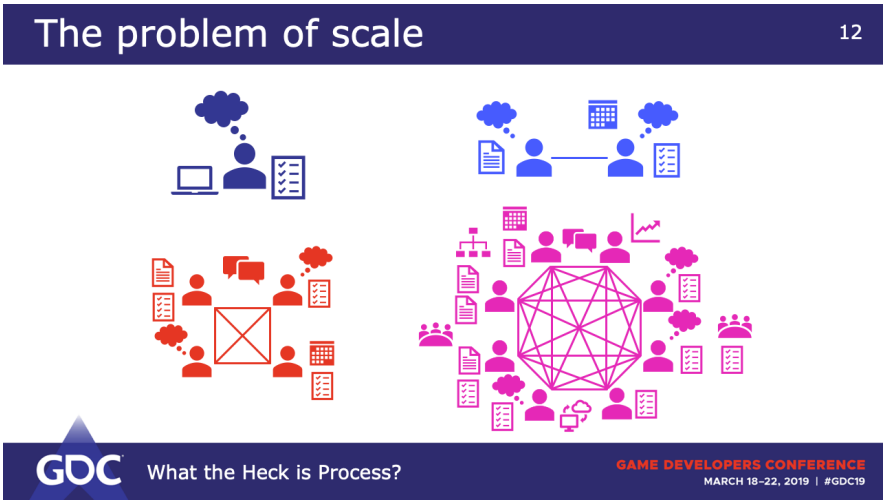
But as your team grows, you run into the problem of scale.

https://en.wikipedia.org/wiki/Metcalfe%27s_law

https://www.pentadact.com/2013-10-15-gunpoint-development-breakdown/

https://www.reddit.com/r/IAmA/comments/71urxt/hi_im_a_solo_indie_game_developer_who_just_spent/dnqyi5r

https://electromagneticproject.wordpress.com/2012/03/27/thinking-with-platforms/




A team of one only has one information channel; they have to keep track of their own work.

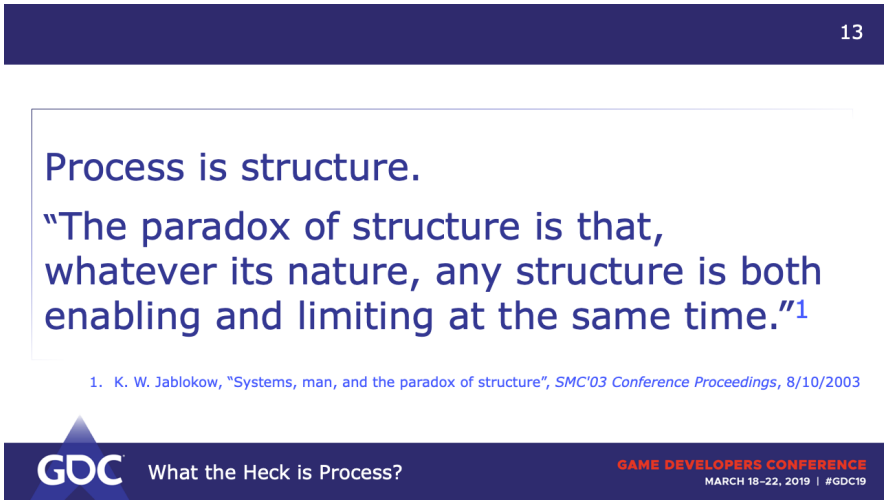
Once you have more than one person on your team, those people have to communicate information to each other. They have to make sure they're on the same page, they have to deliver partially-done work to each other, and they have to review each other's work to agree that it's done.

So a team of two has 4 information channels: Person A and Person B each keep track of their own work, Person A tells things to Person B, and Person B tells things to Person A. The network complexity increases as a square of the number of people in it.

Now add two more people. Maybe two people are sharing a task list, so now they need a way to divide up tasks between the two of them. If two people on the team make a decision, they now have to communicate that information to the other two people on the team. There are more opportunities for information to be lost, or for misunderstandings.

With 8 people, now you have sub-teams, with team meetings, org charts, areas of ownership, bug lists. How many of you work on a team larger than 8 people? Every one of those communication channels is an opportunity for the wrong thing to happen. You have to add more process to prevent that, to keep track of decisions made, add ways to communicate that information, and so on forever.

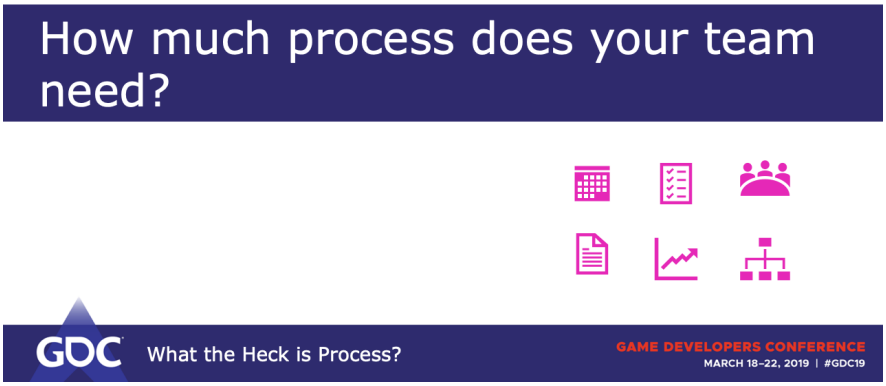
https://stackoverflow.com/questions/984885/how-do-i-explain-the-overhead-of-communication-between-developers-in-a-team
https://en.wikipedia.org/wiki/The_Mythical_Man-Month




Remember that process is structure. And the paradox of structure is that, whatever it's nature, it is inherently enabling and limiting at the same time.

Process can let your team do more, but it also places limitations on them. Some teams will really need the benefits of more structure, while other teams will suffer too much from the limitations for more structure to be worth it.

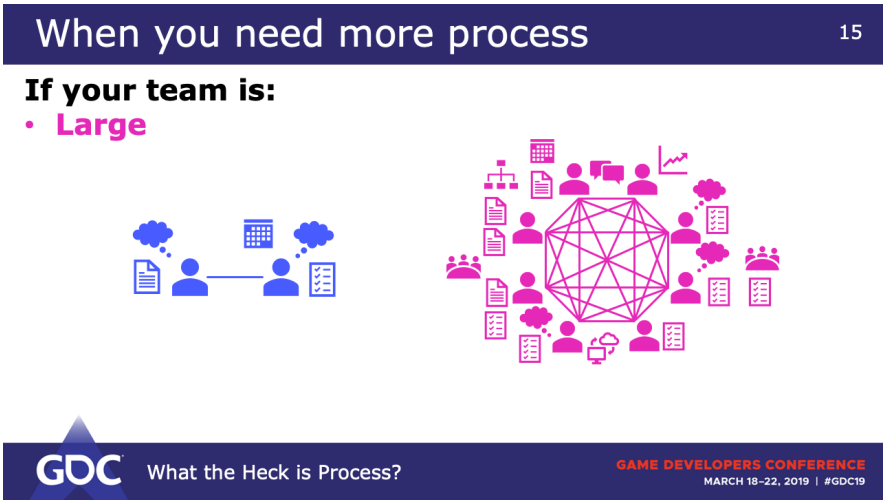
https://ieeexplore.ieee.org/document/1244239




So how much process does *your* team need?

I urge everyone to go read Gamasutra's list of 10 seminal game postmortems; I'll link to it in the notes. All 10 of the postmortems on that list are about big, complex games, and all of them talk about process a *lot*. In most of them, process is at the core of the 'what went wrong' list; in some of them, it's high on the list of what went right. But I've never seen a postmortem that didn't have a big section on communication or process; it's often what makes or breaks your game.

[http://www.gamasutra.com/view/news/238773/10_seminal_game_postmortems_every_developer_should_read.php]

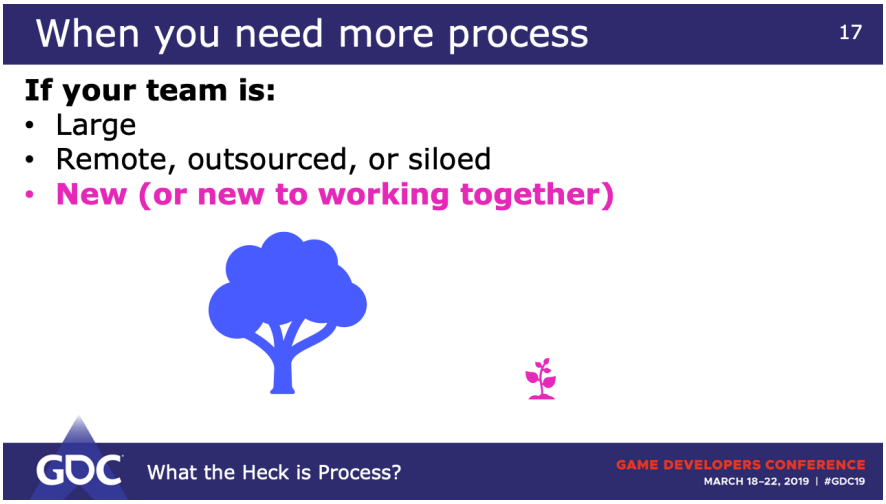



And some types of teams need more process than others.

As we talked about earlier, Metcalf's law means that the larger your team, the more information channels there are, so the more opportunities there are for miscommunication and misalignment. So if your team is large, you need more process to manage your communication and keep track of who's doing what.

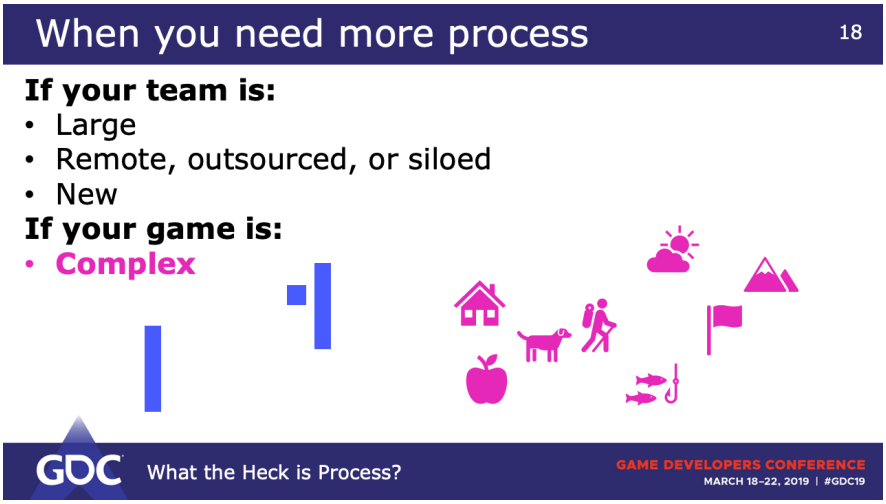

Similarly, if your team is geographically separated, or if teams are structured by discipline so people who work together don't talk to each other very much, that will also increase the complexity of communication and add communication barriers. And the more barriers to communication and coordination you have, the more process you need.

If you've ever worked with a remote team, or outsourced work to another company, you know how critical it is to keep precise track of what that team is doing, when they're ready to deliver, what feedback is being communicated to them, etc. Most teams that I've worked on that have outsourced have a producer just to manage that work, and the teams that don't have a dedicated outsourcing producer wish they did.




If your team hasn't worked together before, or you have less experienced people on your team, you'll need more process.

People who have been making games for a long time, or especially teams who have worked together for a long time, have good instincts for what's going to go wrong, when things will be done, and the best way to communicate with each other. If your team is brand new or made up of mostly inexperienced developers, you'll need your process to handle more of that work.

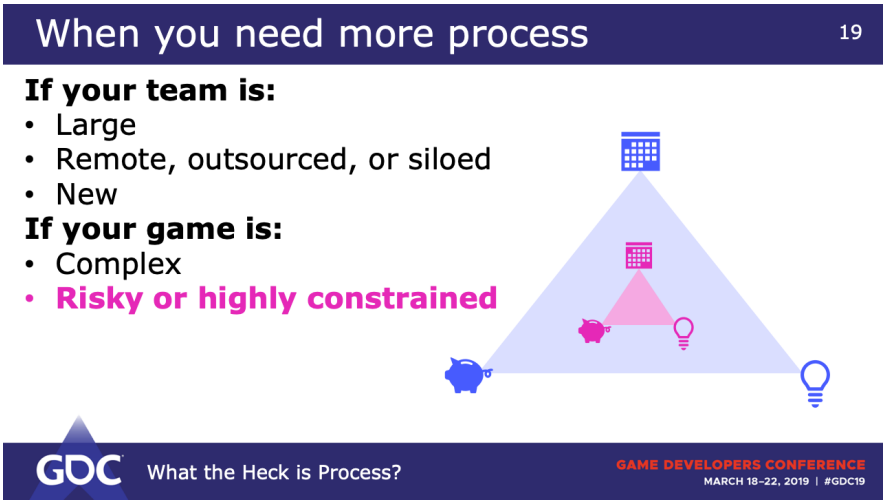



Another factor is the complexity of your game. The more complex your game, the more information each person will need to do the right work, which means there's more information to move around to make sure the direction is clear and everyone knows what they need to do.

If you're making pong, it's pretty easy to decide what everyone needs to do, communicate and agree, and then go execute.

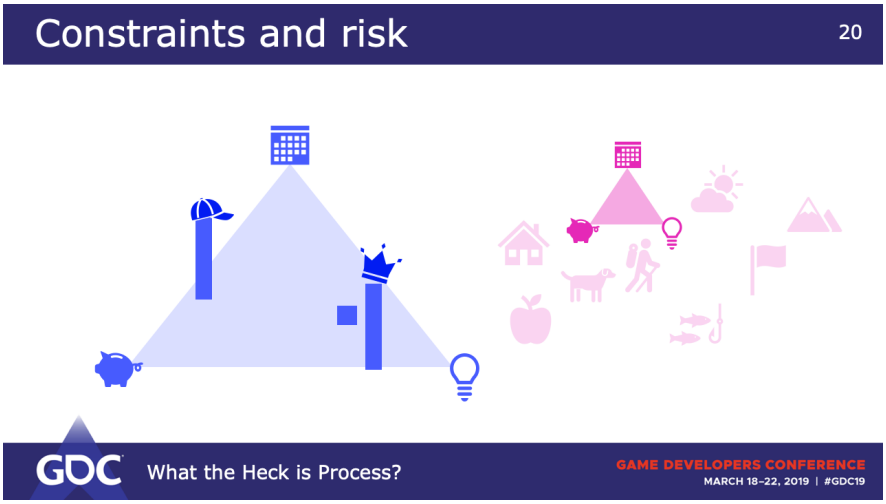
If you're making an open world exploration game where you can tame wild dogs, build and decorate your house, and fish, with weather, hunger, and cooking…you're going to need process to keep track of all the things you need to do, how all those systems will interact, who's working on what, who's testing what, etc. etc.

Remember how all 10 of those seminal Gamasutra postmortems were for big, complex games? They tend to be the most educational because those projects have the most things that can go wrong. Which leads us to…




Constraints and risk. This is the Iron Triangle: it's a representation of the constraints on your game: budget, schedule, and scope. You (presumably) have a limited amount of money and headcount, a schedule you have to meet, and certain features that must be included in your game. Sometimes the constraints are loose and allow your team a lot of creative freedom and exploration.

Sometimes, they're super tight and you have to count every day and every dollar. You have to think: what do you absolutely have to deliver? What can you cut to meet the schedule, or stay in budget? These constraints determine the flexibility you have, and dictate how tightly you have to control the work done by your team with more process.

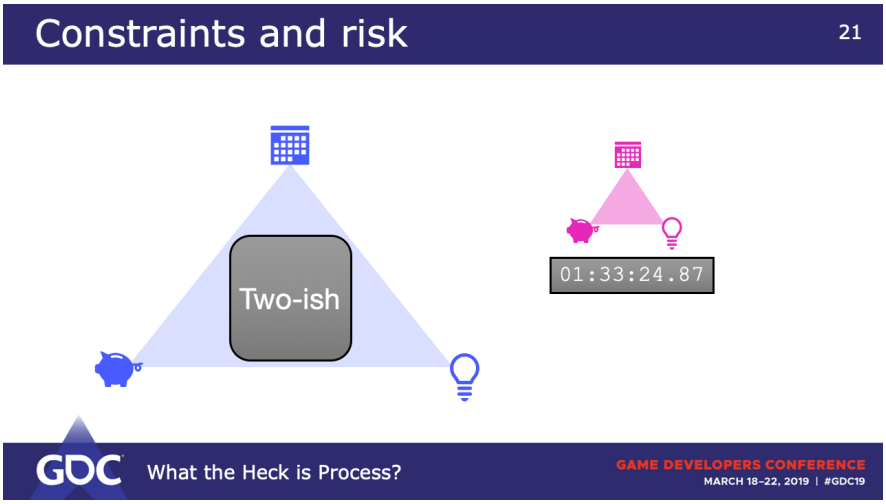



Let's say you're making your Pong game; that's a very limited scope. If you have a bunch of money and time, you can adda feature where you can buy hats and put them on your Pong paddles, without a lot of risk. You have to ship eventually, but go nuts; maybe paddle hats will be the coolest thing in the game; let's try it.

But if you're making your open world exploration game, that's a ton of scope. Maybe it's based on a popular movie, so you have to adhere to the style guide for character art, world building, and dialogue. The game has to ship 2 weeks before the movie comes out or your studio will get shut down. In this situation you need to keep a very close eye on every single thing that anyone on your team is doing. You can't waste a single day, and you can't let anyone build anything that doesn't closely adhere to the design documents and IP style guide, because that would lead to total disaster.

Process is how you move information around your team, and with more constraints, you need more information. Think of it this way: when do you need to know exactly what time it is?

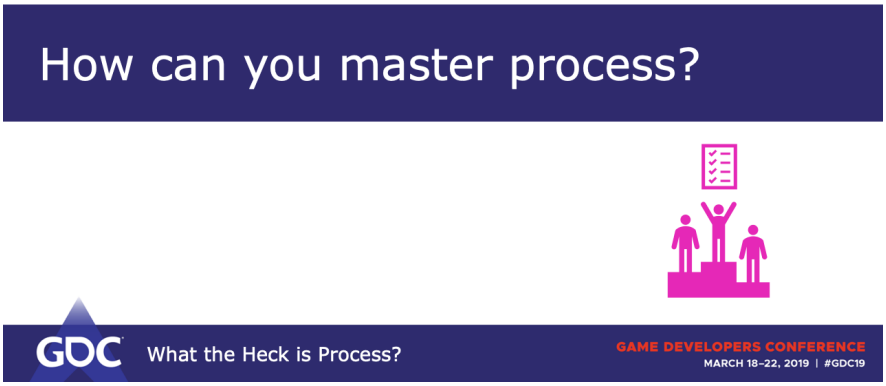
[http://www.ambysoft.com/essays/brokenTriangle.html]



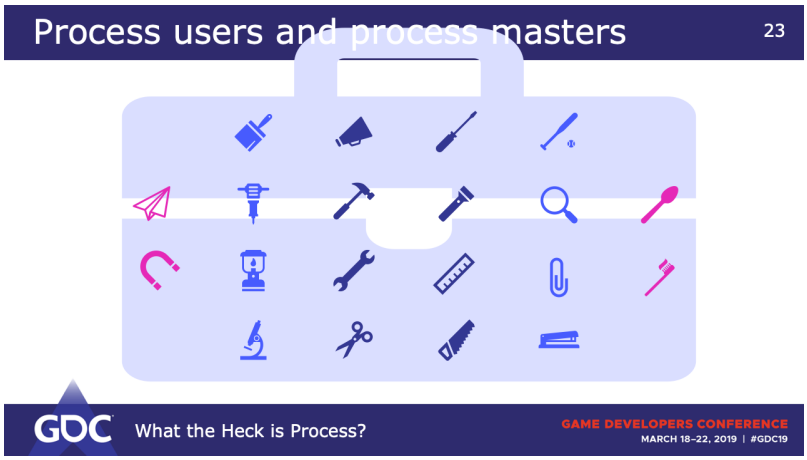

A lot of smartwatches have these faces that estimate the time; they'll say "Two-ish" or "Almost four" instead of giving you the actual time. And honestly, if I'm on vacation and don't have anywhere I have to be, that's good enough.

But if I have a critical job interview in an hour, and traffic is pretty bad,I want to know *exactly* what time it is.

The tighter your constraints, and the greater the penalty for failure, the greater your need for information precision. And you get precise information from structured processes.




Ok, we've talked about what process is and why you need it. So how do you actually do it? How can you, the producer, become a process master and build amazing processes for your team that they will understand and find useful?




First, there are two types of producers: process users and process masters. Process users are good at maintaining and running existing processes: they join a team, learn that team's processes, and keep them running.

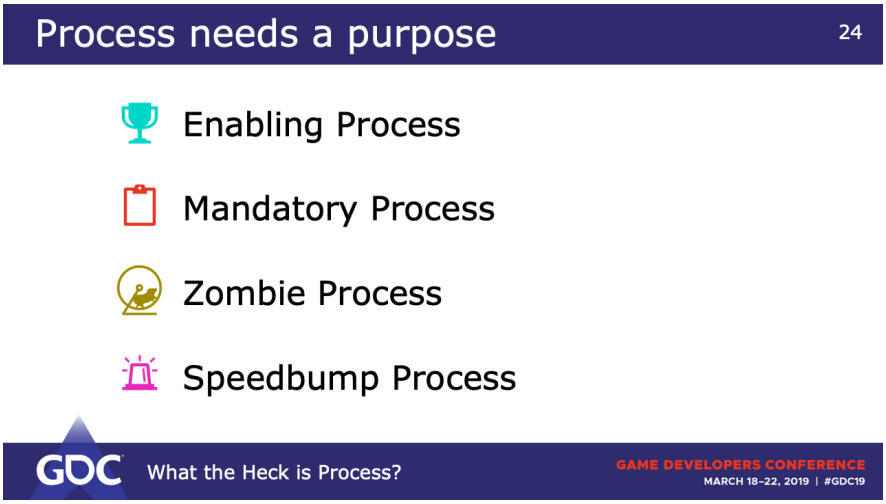
You start, like I did, by learning one or two tools and get good at using them under supervision. For example, on my first producer job, my manager showed me how to use the team's bug database and had me enter bugs, and she had me track art assets in a spreadsheet the team was already using.

Then you'll work on a different game, maybe on a different team, and you'll learn a few more tools. This is around when I learned how to make Gantt charts, and learned that they're good for first-pass planning but *not* good for tracking.

The more varied your experience, the more varied the tools you know how to use, and you'll start to build some of your own. I made a pretty amazing excel sheet that tracked what level each environment artist was working on, and that would output how many person-weeks we had spent on each level and estimate how many more weeks were needed.

Over time, you'll start having some go-to tools that you use all the time, and some weird ones that you use occasionally for specific circumstances. On one game project, I was having trouble reminding my team when the next milestone was, so I set up a TV in the studio lobby to show a powerpoint of screenshots of the weirdest bugs I could find in the game to catch people's attention, and every other slide was a summary of the next milestone with the date. That's a process I'll probably never use again, but next week I might have to build or use something just as weird.

This is what process masters do: They build up a robust and varied process toolbox that lets them apply different processes to different situations, and they apply those different tools not just based on what they've seen work well in the past, or what they personally like using, but by really understanding what the need is, and what process could fill that need. You have to build that toolbox by shipping games. And you learn how to use those tools by understanding the needs of your team.




Whenever you're adding a new process for your team, and even when you're using existing processes, you should ask why. Why do we need this process? What is it for?

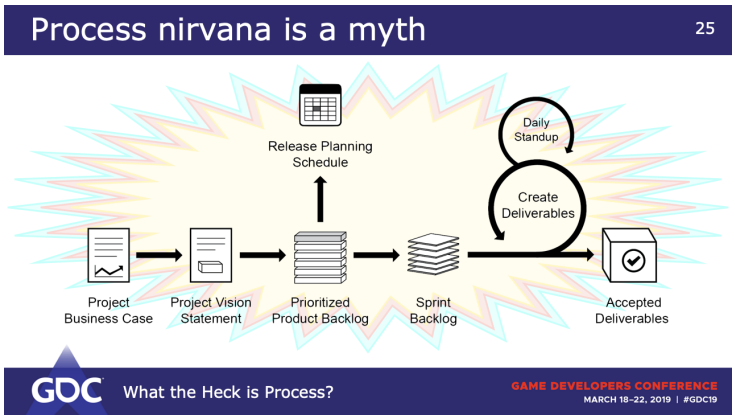
There are many processes that your team gets clear value from, actually likes, and uses willingly; they fit some need that your team has. These are how your team knows what work they need to do. It's obvious why these processes are useful, and they're usually straightforward to implement and use.

And there are mandatory processes. These are processes that your team doesn't get much value from, but you still have to do them for some other reason. Some of these will be things that are kind of useful, but you wouldn't do them if nobody was forcing you. Others are just…well, we have to do this thing or we won't get paid. Ugh.

Zombie processes are the worst ones, and good producers will identify and get rid of these. These are processes that used to be useful, or that someone thought would be useful, but they aren't, and everyone still does them either out of habit or because they think they're supposed to. I've never been on a team that didn't have at least one or two of these. Sometimes they don't take much effort, but they're just kind of sad and don't help anything, and they add to the list of reasons people say they hate process.

Last is a type that doesn't sound useful, but it is. Speedbump processes exist to keep people from doing things they shouldn't do, or to slow down things that really should take more time. Remember that structure is inherently limiting: limitation isn't always a bad thing. There are some things that you should limit, and speedbump processes let you do that.

I'm going to talk about some examples of each of these types, how to identify them, and why you need the good ones and need to fix the bad ones.




One note before I get to the examples: I think producers, in the back of our minds, carry a beautiful dream of process nirvana. I certainly do: when I first learned about Scrum, I thought it was the answer to all my problems. Finally, a perfect way to communicate the exact information my team needs and no more! Everyone on my team will be able to do exactly the right work with zero wasted effort. Of course, when I came back from my Scrum class and told my boss about the Nirvana I had discovered, he laughed at me and said I should try it and see what happens. What happened was: I added a couple of tools to my box, but mostly my team was as effective as they had been before, and slightly annoyed that they had to change how they tracked their work.

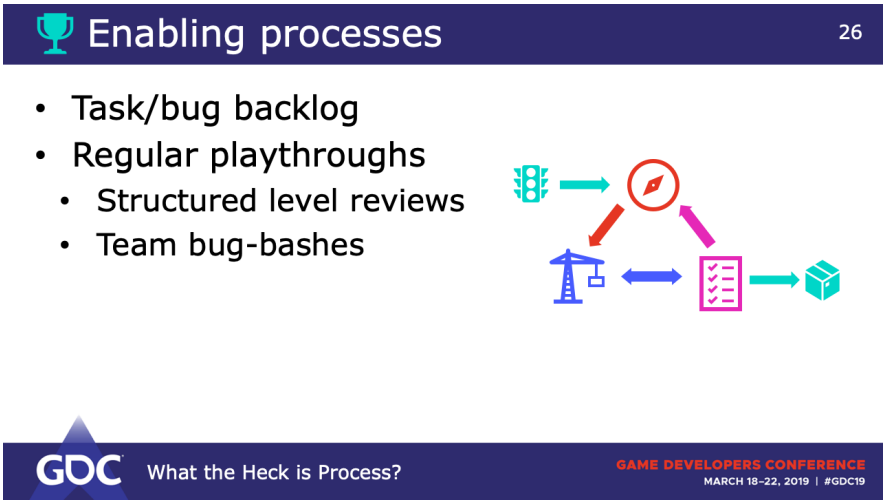
Of course, this beautiful dream of perfection does not exist. And it's also really harmful: it makes producers strive for perfection when they should be trying for 'good enough'. I have spent hours of my life I'll never get back on trying to build a Project Gantt Chart that would simultaneously convey every piece of information about a project while also being easy for anyone on the team to understand at first glance. I no longer believe it's possible to do that; if it is, I don't know how to do it. But I do still sometimes feel the urge to try it again.

And this dream makes producers feel like failures if they aren't achieving this mythical perfect state where all their processes are perfect, decision makers always have exactly the information they need, and everyone goes home at 6:30 every day while also hitting beta 2 months ahead of schedule.

Not achieving this mythical state does not mean you're a failure. But you do have to think about what's working and what's not working for your team, and what they need that they're not getting.

So I'm going to talk about examples of successful and unsuccessful processes I've used, and specifically about what need they were fulfilling or trying to fulfill. I will not be able to tell you how to achieve perfection, because you can't, but I hope to help you get closer to 'pretty good'.

[Scrum image from Wikimedia Commons:
https://commons.wikimedia.org/wiki/File:Scrum_Flow_for_one_Sprint.png]

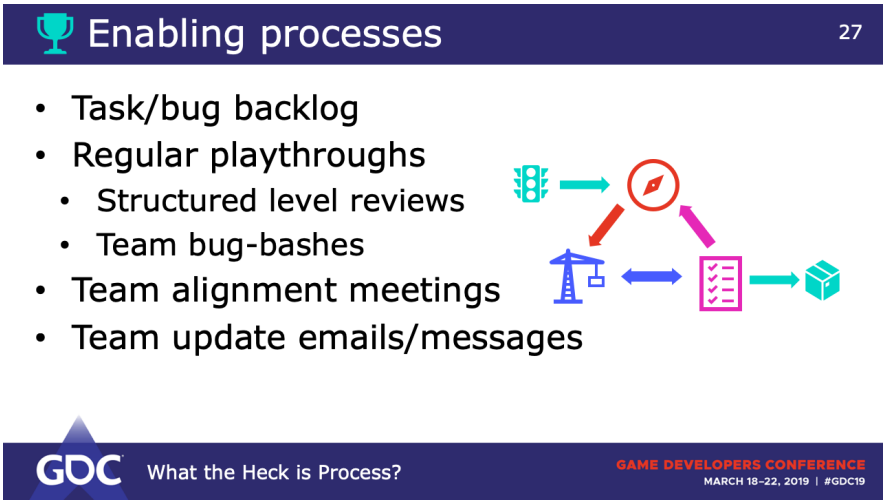



First, let's talk about the processes that are genuinely useful to your team.

On any team that's functioning pretty well, most of your processes will be good, effective ones. So I had a lot of examples to choose from here.

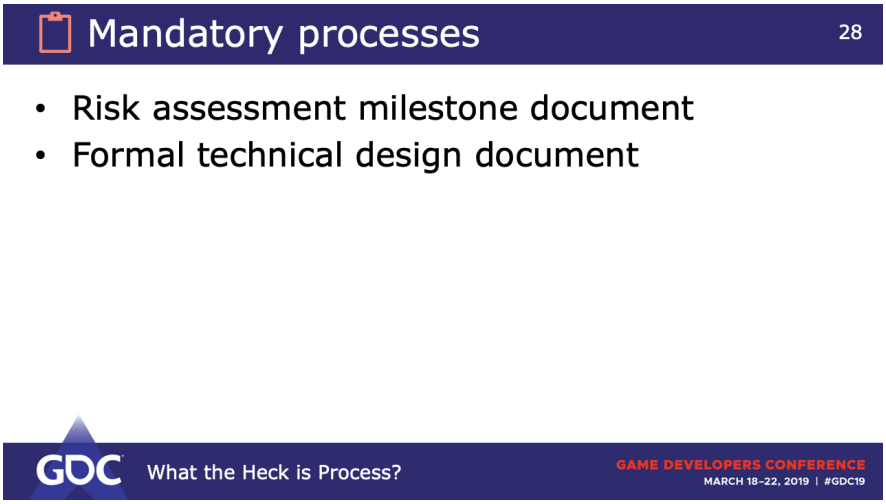
First, as discussed earlier, a task or bug backlog is a pretty universal, extremely useful process. Because these are so critical, they're usually set up pretty well and are usable by everyone on the team.This process is a great way to manage the work being executed, and measure how much more needs to be done.

Second, regular playthroughs of your game. I've seen a few different types of these: for example, as you're building your game content you might do regular level reviews where the team gets in a room and watches someone play through a level, and discusses what works and what doesn't as a team, or decides what needs to be changed. Later in the project, a lot of teams will do bug-bashes where everyone plays the game and finds and logs as many bugs as they can into the database.This process evaluates the current state of the game and gets critical information to decision-makers about what still needs to be done.




Team alignment meetings are especially useful on complex projects with large teams.This is a useful process for communicating the game vision to the team regularly, to make sure that everyone knows what the vision is and that they're working toward it. Most teams I've been on have done these monthly, and sometimes more frequently at especially chaotic phases of the project.

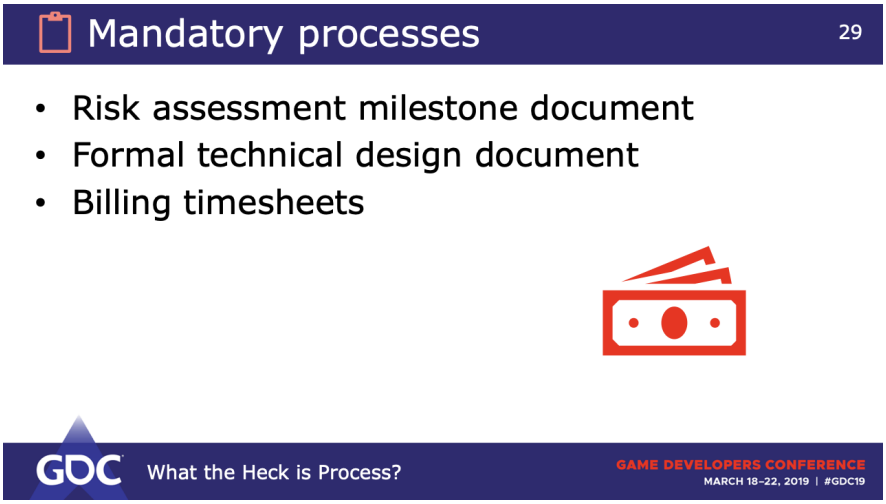
And one of my favorite model processes: you, the producer, send out regular emails, posts, or chat blasts to the team updating them on important information.Sometimes this is more to keep the decisionmakers updated on what's happening on the team, and sometimes it's more about keeping the team aligned on the current status, risks, and constraints. I've usually done these at a weekly cadence, but sometimes when approaching an important milestone, I'd send these out daily, with a big red countdown at the top. You might have heard "you have to communicate something 7 different ways before your whole team knows something". In cases where your team doesn't feel like they know what's going on, this type of regular communication can be a big help, and then you only have to find 6 more ways to tell everyone what's happening.




Ok, those were the nice processes everyone likes. Here are some processes that I've just had to use in the past, though they didn't directly benefit my team very much.

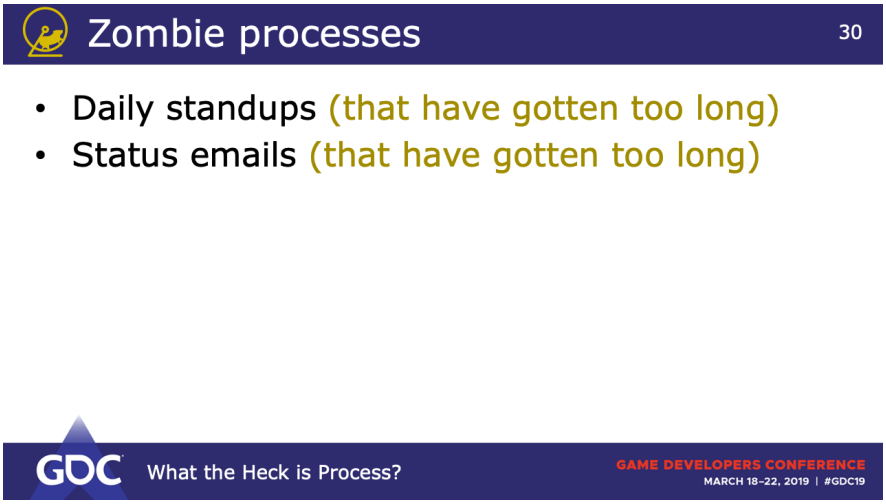
Every publisher I've worked with has required specific milestone documents, often providing templates to show exactly what they want. For example, one large publisher required us to create a risk assessment document for project greenlight, and for us to update it for every milestone. This was a list of major risks to the project, how likely those risks were, how bad an affect they'd have if they occurred, and what we could do to mitigate them. Now: Risk assessments are very useful, and it's a great habit to do one regularly for your game, especially if you're doing something risky. But the team I was on was pretty overloaded and probably wouldn't have made that document, or updated it as often, if it wasn't required.

The same publisher required us to write a formal technical design document and keep it updated at each milestone. This document was almost entirely useless for our team; it was probably helpful to some degree for the engineering leads to sit down and write out their plan for the game engine and features, but it definitely wasn't worth the time it took them. At least, not to us. What this document was really for was to prove to the publisher that we knew what we were doing and that we weren't going to throw their money away. So this process benefited my team indirectly, in that it allowed us all to get paid, but we definitely wouldn't have done this if it wasn't required.




Billing timesheets are not very common in the games industry but some teams use them, especially if they're doing work for hire or if they have another reason they need to closely track how each person on the team is spending each hour. We used timesheets for a while on one of the Dungeon Siege games when I was a level designer, and I actually really liked them; they let me plan my work better and really reinforced that I was working on the right things. But I think I was the only person on the team that felt that way; everyone else hated them and we didn't use them on the next project.

You'll notice all three of these have one thing in common: they are about money. Remember that tighter constraints require more process, and if you're spending someone else's money to make your game, that adds budget constraints because your partner will want you to keep close track of how you're spending their money.




Zombie processes were either useful at some point and your team keeps doing them out of habit, or someone with authority *thought* they'd be useful and nobody has the power or will to tell them no.

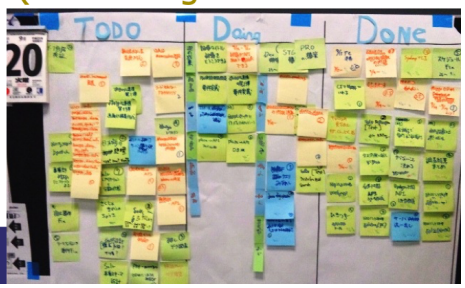
Team stand-ups are usually a great process, especially on more mature, self-managed teams. Everyone on the team says what they're working on, what's blocking them, and what they've gotten done recently. But I know of teams where stand-ups have gotten out of hand, started taking an hour each day, and either some people go on and on about what they're working on and everyone tunes out, or every person on the team tries to keep their update so brief that they don't communicate any information. But because people have gotten used to the communication advantage of the standup, they keep going, and participating, even as it starts taking up a huge chunk of their time for very little value. If your standups are taking up more than 10% of your team's time, fix them. Even 10% is really high, unless everyone is getting a ton of value from that time spent.

On Dungeon Siege, the producer sent out weekly status emails that listed what each person was working on and what they had just finished. This is a tough one, because it sounds really useful (and occasionally on some teams it is!) but on that team, they got so long that people stopped reading them. One designer kept putting "tell me if you read this!" in his section, and nobody ever did. But again, the team had gotten used to it and it *seemed* valuable, but it wasn't actually valuable.

## Zombie processes



- Daily standups (that have gotten too long)
- Status emails (that have gotten too long)
- Design Kanban board (that has gotten too messy)



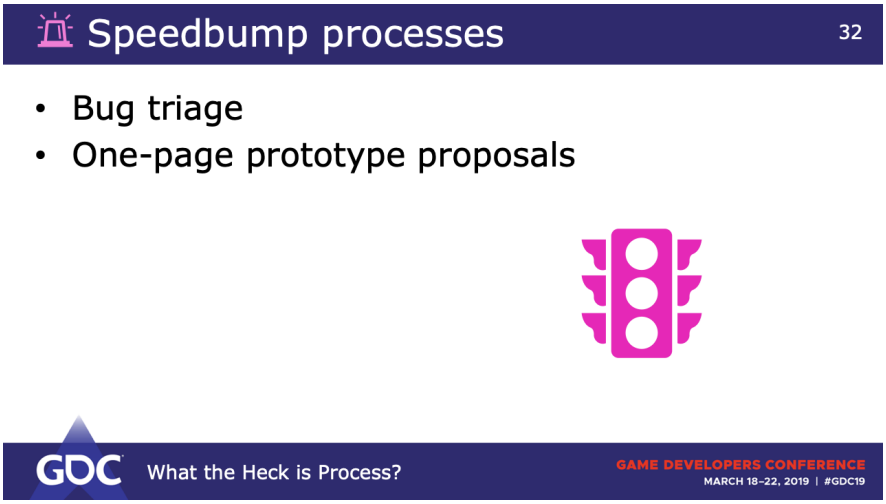



On another project, the designers had started using a Kanban board to track their work. Now, a Kanban board is often a great process: here's an example. You grab a whiteboard, make three columns on it: To-do, In progress, and Done, and you use a sticky note to represent each task. A lot of people really like the physicality, and it forces you to keep the process simple.

Usually. On the project I was on, the Kanban board had gotten way out of control. I really regret not taking a picture of it: sticky notes were piled up on the board, stuck on top of each other in huge stacks, impossible to read, and yet every few days a designer would walk up to the board, try to find the note for the task they had just done, pull it off the board, watch silently as a stack of sticky notes fluttered to the floor, stick their task in the new column, sigh, and walk back to their desk. It was the saddest whiteboard I had ever seen. And it took a while to kill that process, because everyone assumed that someone was getting valuable information from it. But nobody was, so it was just a waste.

Now, note that all three of these processes are bad because the noise drowned out the signal. Each of these *could* be great, under different circumstances. Daily standups, status emails, and Kanban boards are all awesome processes if you do them right. But if you let them grow to the point that they're too unwieldy to be useful any longer, you need to kill them or change them to be more useful. Maybe the tool that your team is trying to use isn't the right one for them. Maybe if this Kanban board had been a digital task board, it would have worked better.

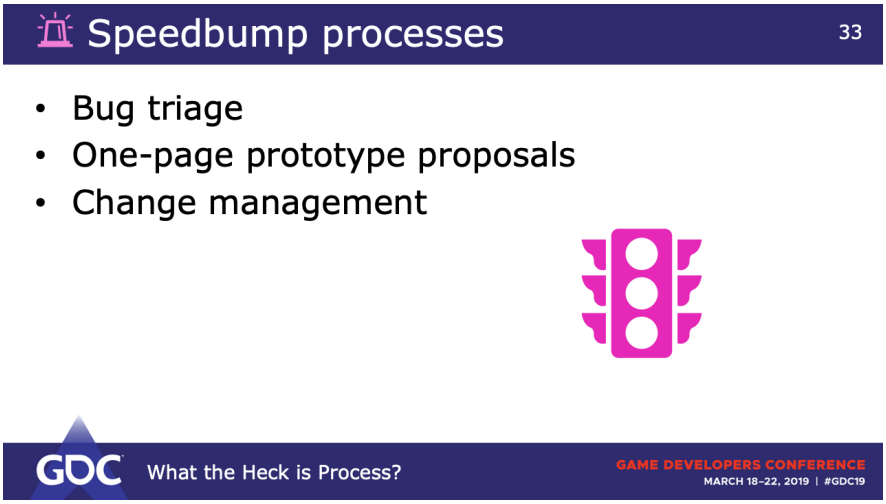
Kanban photo from Wikimedia Commons:
https://commons.wikimedia.org/wiki/File:Simple_Task_Kanban.jpg



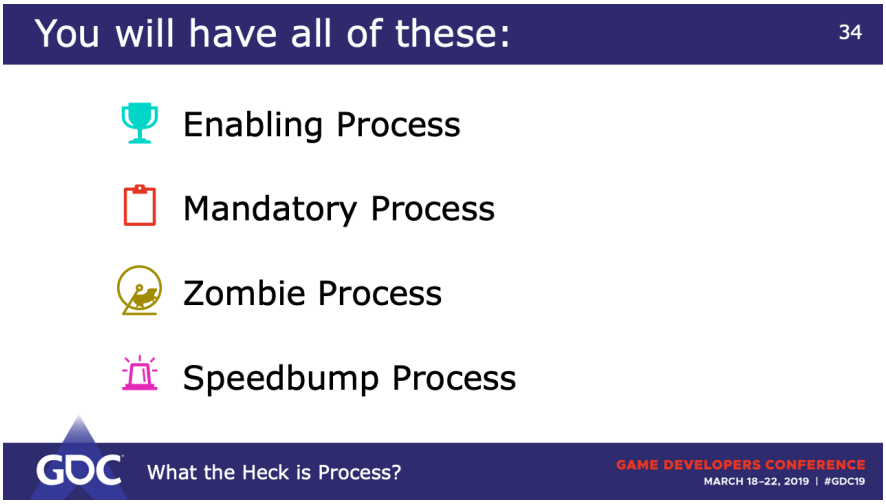

And last, speedbump processes. You will probably not use these very often, but it's good to remember that they exist. Remember that process is both enabling and limiting at the same time. Sometimes you need to keep people from doing things, and that's when you pull these out of your toolbox.

First, bug triage. I've never shipped a bug-free game. The closer you get to shipping, the more selective you have to be about what you'll fix so you don't add new bugs with the fixes. Meeting regularly with your test team and deciding no, we're not going to fix certain bugs, is a speedbump process. You're limiting what work you will do, and deciding not to do certain things.

One-pagers: this is something we do on my current team; it started as a speedbump process but over time it's become the main way we agree on new projects. We were in a prototyping phase, and because we have a lot of smart, proactive people on the team they all wanted to go build prototypes. And while it's really easy for a single person to start a prototype, at some point they'll need help to finish it. And we were having trouble tracking all of these prototype projects. So we added a speedbump: before starting work on a prototype, the developer has to write a one-page document describing what they want to build, how long it will take, what features the prototype will have, and who else they'll need to work on it. Then that one-pager has to be approved by the team leads before they can start working. One page is not much to ask for, and it's been a super useful way for us to keep track of who's working on what. And it introduces just enough of a speedbump that if the prototype isn't really a good idea, often the one-pager doesn't even get written.

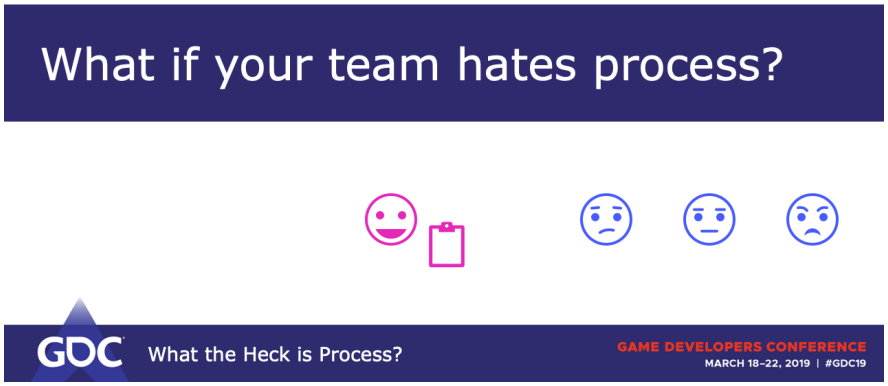



And both of these are really examples of change management: change management is a fundamental process in many other industries, like architecture, where changes have huge real-world consequences and have to be budgeted for. It's less common in game development, but it does get used: in addition to the first two examples, code reviews are a process where an engineer can't check in a code change unless someone else reviews and signs off on it. Or even level review meetings, where proposed changes to a level need to be signed off on by the design and art leads. Again, these are processes to keep people from doing things, to limit work that could cause problems or that maybe you shouldn't do. Changes aren't bad, but there are phases in your project where you need the changes to slow way down or stop completely. If you keep making changes, you won't be able to ship.




Realistically, you will always have a mix of all four of these. You will never make the same game twice, and your team will never stay exactly the same. Even if you get your processes really refined, if you make a second game with the same exact people, those people will all have learned from the development of the first game, and a more experienced team needs different process than a less experienced one. So good, enabling processes stop being quite as useful and become zombie processes, or documents that your publisher forced you to deliver become more useful to your team over time, or a speedbump process you put in place becomes a model process that's really useful across your whole team.

Remember that process is never totally perfect, so you will always have this mix. That doesn't mean you're doing something wrong if you have a couple of zombies hanging around that you haven't quite killed yet, or if you have some mandatory processes that your team hates but you can't figure out how to get rid of.



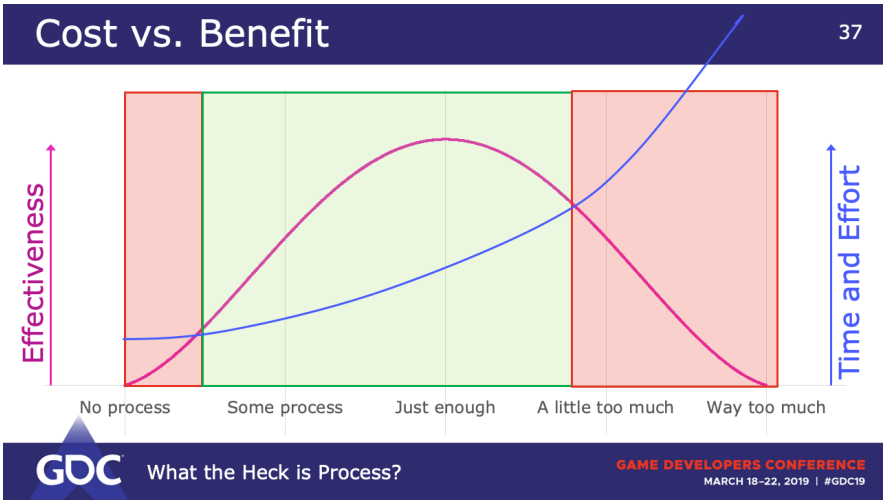

But what if your team just hates process? How can you give your team the structure they need without them making these faces every time you ask them for task updates?


If you have way more or way less process than your team needs, it won't be as effective.

And if your team is like most of the ones I've worked with, they're afraid that you'll add so much process that they won't be able to do good work.

This is the nightmare scenario for a lot of game developers: that they'll be expected to spend half their day updating tasks, writing documents, or going to meetings, and they won't be able to spend their time doing what they love, which is: making awesome games.

But the truth is, just about every game dev team has too little process, and would be more effective if they had more. I've never read a post-mortem where one of the problems listed was "too much structure", but I've read dozens that call out problems resulting from not enough. If you read any game post-mortem, go to the "What went wrong" section, and I'll bet money that at least one of them will be a problem resulting from not enough structure. [http://www.gamasutra.com/features/postmortem/]




But sometimes it can be hard to tell whether you have too much structure, or not enough.

The more process you add, and have to maintain, the more time and effort it will take; not just from you, but from your team. You have to keep an eye on how effective your process is vs. how much effort you're putting in:

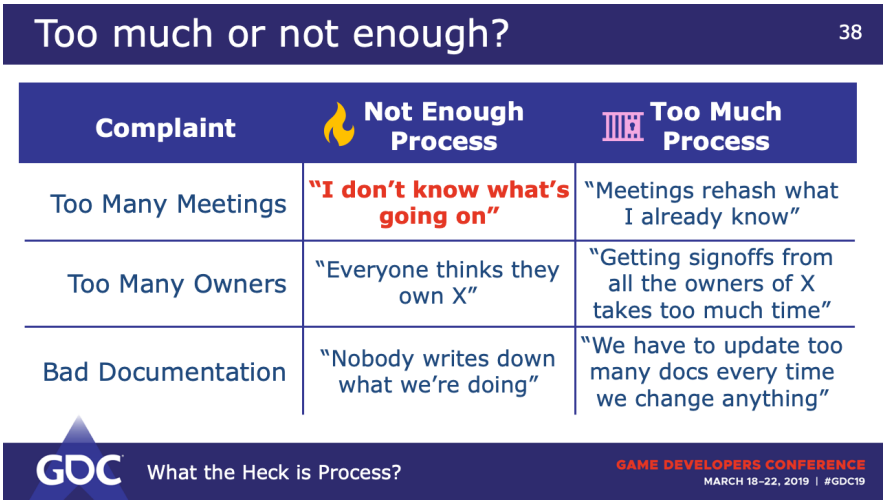
First of all, you can never really put zero effort into your process. You can put very little effort into it and let chaos reign, but in that case your team will be very ineffective.

If you're putting enough effort into your process that it helps enable your team, great! Your process is good enough. Things will continually change, so you'll need to work to keep your team in this green zone, but generally this is the right place to be.

But imagine what it would look like if you had way too much process on your team. Everyone had to track their work in 15-minute increments on time sheets. They had to get a form signed off by every lead before checking in an asset or code change. Every meeting needed a pre-approved agenda. If you tried, you could probably implement so much process that your team would grind to a halt. And I have heard of teams where this happened.

So in both worst-case scenarios, you're spending more time on process than it's giving you in benefit.

You can fix either of these problems, but you have to know which problem you're fixing: too much process and you need to lessen it, or not enough and you need to add more?



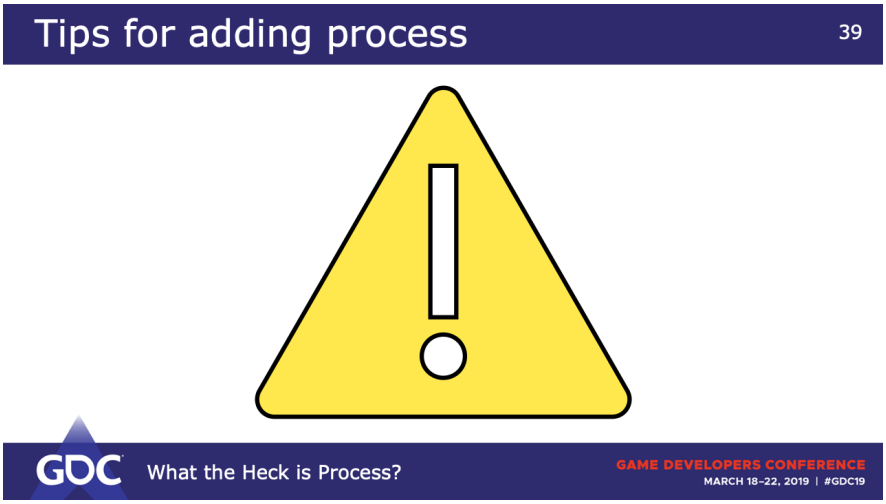

It's not always clear; sometimes the complaints sound the same in both cases.

For example: Too many meetings. I've never been on a team that didn't complain about too many meetings. But the underlying problem might be that your team doesn't know what's going on, so they hold more meetings to try to get the information they need. Orit could be that your team does know what's going on so they don't learn anything new in the meetings they have, and they're redundant.

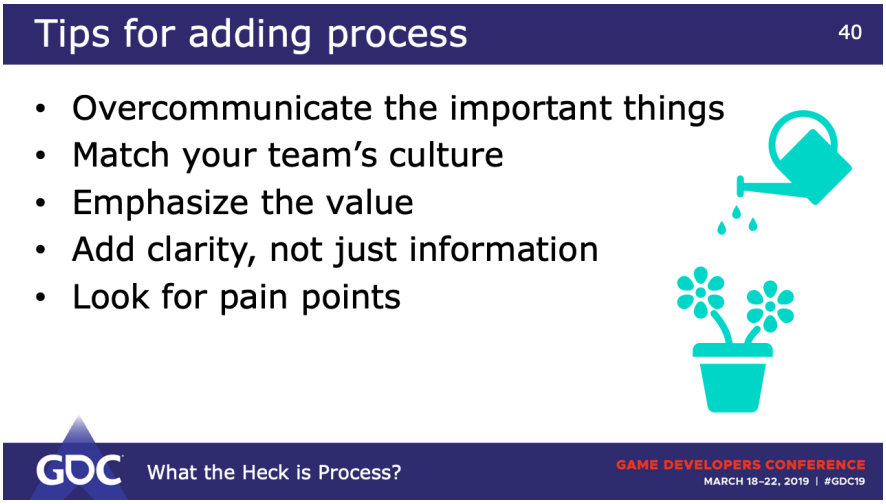
If your team complains that there are too many owners for each feature, that might mean that too many people think they own the feature because there isn't a process that clearly defines ownership. Or they might be complaining that the process requires them to get too many signoffs before doing work, because you have too much required process in place.

If your documentation is bad, is it because you don't have any, because nobody writes down what you're doing? Or is it because you have so much documentation that it takes too long to update it when things change?

"I don't know what's going on" is the key phrase to listen for. Does your team know what's going on, or not? If they don't, you need to add process. So let's talk about some tips for how to do that.




First of all, I want to say: adding process is really hard. My team is currently working on adding a project tracking process, and it's a struggle. I asked one of our experienced PMs for his tips on adding a new process, and he said, "I don't know, I'm still learning. Tell them it's really hard." So: warning: it's really hard. There's no magic answer. But here are some tips that have come in handy for me.



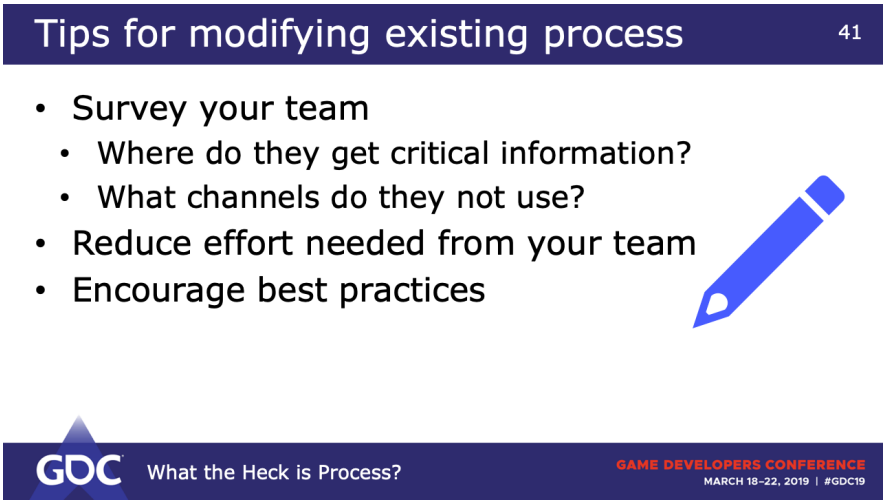

Overcommunicate the important things. Like: what is your game vision? What are you trying to make? Everyone on your team needs to have a really clear understanding of your direction. I've always heard, "you need to tell someone something 7 times before it really sinks in." But then someone told me the other day that it's actually 16 times. Whatever it is, it's more than you think it is. So focus on making sure people can find the most important information.

Second. if you're adding a new process, match it to your team's culture. How do they like to work? I've been on teams that loved daily standups, and I've been on teams that hated them and saw them as a total waste of time. The people on your team probably have a good idea of what will work for them, and they'll gravitate towards new process that fits their style.

Focus on the need that the process is intended to fill. If your team really hates process, don't say "What process will we add for approvals?", say, "How will we decide whether or not to add a feature?"

Make sure your process always adds clarity, not just information. The teams I've been on that had 50 top-priority goals, the problem wasn't that we needed more information on all of those goals. The problem was, we needed to know which one was actually the most important, and why.

Look for the pain points on your team: if you don't have a process in place where it's needed, someone has figured out a way to work around it. The people on your team are smart. So look for where people are spending time on moving information or direction around your team, and find a way to make it more efficient. Or if someone on your team needs so much help that they actually come to you and ask for a process; that makes it pretty easy, though it probably means you should have added that process earlier!

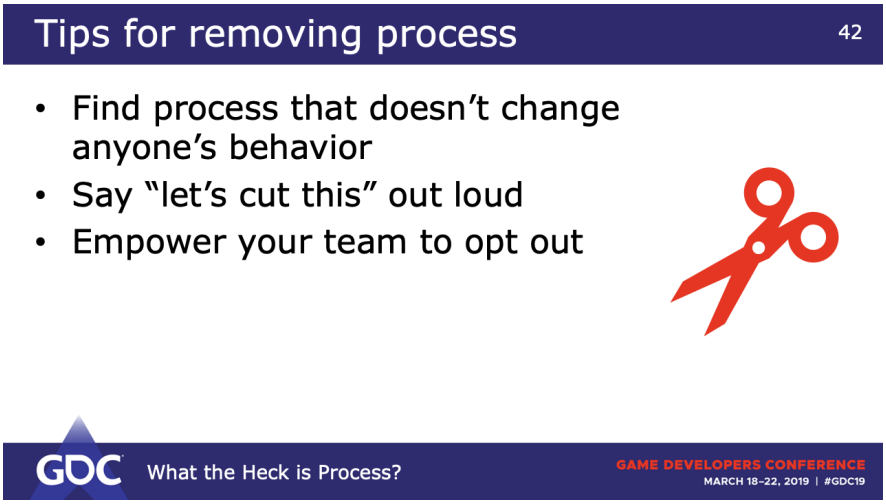



Now again, it's not possible to have zero process. So usually you'll be tweaking process that already exists, but maybe needs some improvement.

Again, your team is smart. Ask them what works for them now. I'm lucky enough to have someone on my team who is really into documentation and communication processes, and he did a survey of our team, asking them where they get the information they need to do their work, which processes they use and like, and what they are missing. Not only did this help us improve our processes, but it also helped the team feel more ownership because they had input into which processes we kept and which ones we changed.

Look at the processes your team already uses, and see if you can reduce the effort needed from your team to keep those processes running. Can you build a dashboard that can be a one-stop shop for your team to check their bug list? Can you create a web form for tasks that fills in required information automatically? Would a virtual Kanban board be easier for your team to use? Reduce friction where you can.

Encourage best practices across your project. I'm also lucky to have someone on my team that's passionate about saving people time, and she recently wrote a meeting best practices document and shared it with the team. Remind people what the purpose of your process is, what need it's filling, and how to keep it efficient, and then see what needs to change to fit those best practices.

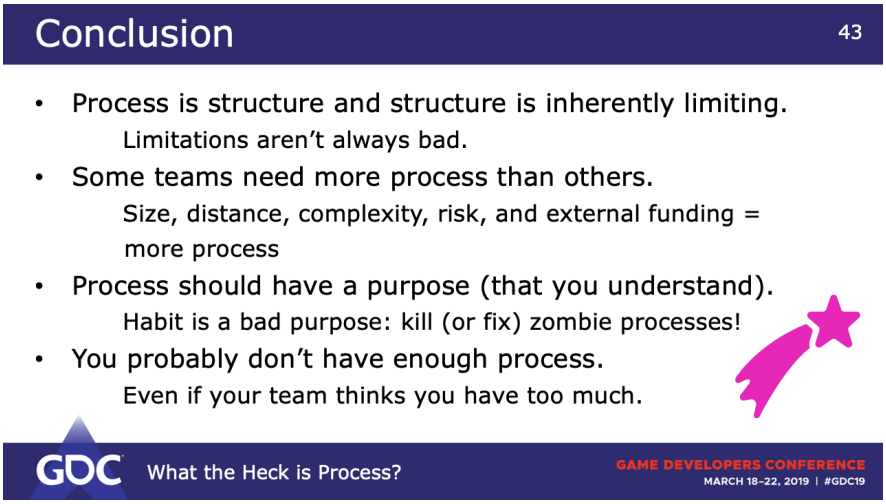



And finally, how do you remove the processes that aren't working? Generally, these will be those zombie processes have gotten too unwieldy or that your team just doesn't need any more. Occasionally they'll be processes that are actively bad. In either case:

If a process doesn't change anyone's behavior, then you do not need it. Only keep processes that change the decisions people make, the work they do, or what you prioritize. Anything else is extra work. Remember the long status emails that nobody read? Since nobody read them, they had no effect and might as well not exist. And in fact, it's better that they don't exist because they were taking up our time. Put that time and effort towards something actually important.

One of my favorite tricks is to just propose out loud that your team gets rid of something. If nobody argues with you, and especially if you hear sighs of relief when you say out loud that you should cut something, get rid of it! If nobody fights for it, nobody needs it.

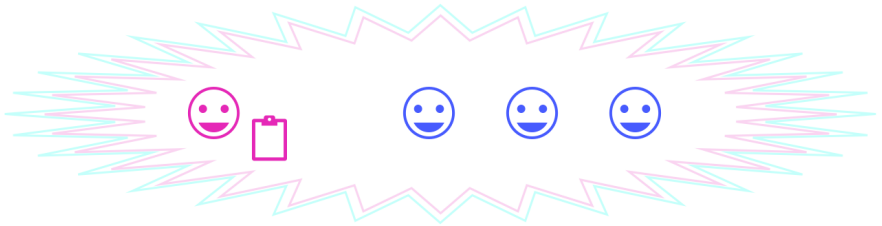
And last, if you can, empower your team to opt out of processes that don't work for them. Now, this won't always work: some processes require everyone to participate for them to work and some processes just need to happen so you get paid. But for example, you can probably remind people on your team that they can decline meetings that aren't useful. Or you can ask at the end of a meeting whether you should keep having it, what value the people in the meeting got from it. Some teams have an authoritative lead that really loves tons of meetings, but usually you have too many meetings because some of them are zombies that people go to out of habit, and you can kill them by empowering people to not go to meetings that they get no value from.




In conclusion:

- Process is structure and structure is inherently limiting.
  - Limitations aren't always bad.
- Some teams need more process than others.
  - Size, distance, complexity, risk, and external funding = more process
- Process should have a purpose (that you understand).
  - Habit is a bad purpose: kill (or fix) zombie processes!
- You probably don't have enough process.
  - Even if your team thinks you have too much.

## Thanks!



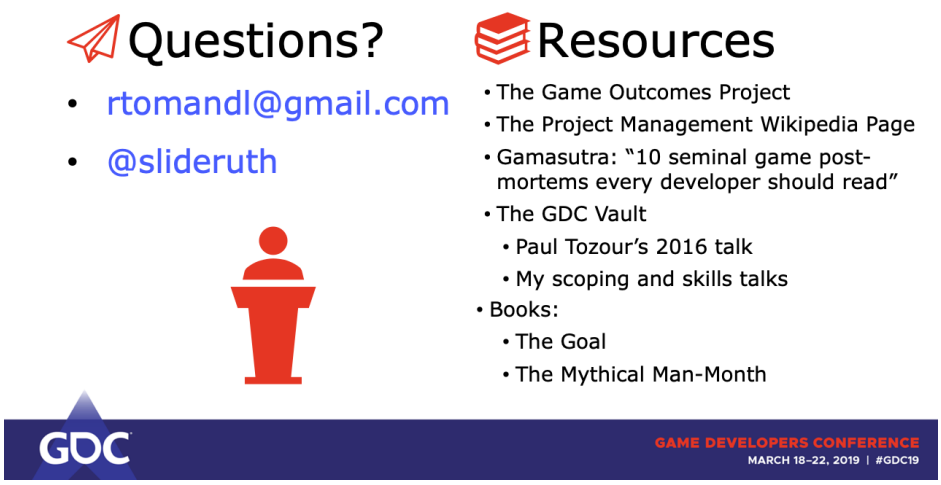
Scrum Flow for one Sprint.png, by Satpathyt. Image available at https://commons.wikimedia.org/wiki/File:Scrum_Flow_for_one_Sprint.png, licensed via CC BY-SA 4.0.

Simple_Task_Kanban.jpg, by Rakuten, Inc. Image available at https://commons.wikimedia.org/wiki/File:Simple_Task_Kanban.jpg, licensed via CC BY SA-3.0.

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Thank you!




Email me for slides! Also I'll post a link on Twitter, @slideruth.

The Game Outcomes Project: http://intelligenceengine.blogspot.com/2014/12/the-game-outcomes-project-part-1-best.html

Literally the Project Management Wikipedia page: https://en.wikipedia.org/wiki/Project_management

10 seminal game postmortems every developer should read: http://www.gamasutra.com/view/news/238773/10_seminal_game_postmortems_every_developer_should_read.php

GDC Vault: http://www.gdcvault.com/
Especially Paul Tozour's talk: https://www.gdcvault.com/play/1023258/The-Game-Outcomes-Project-How
And my 2016 talk about scoping your game: http://www.gdcvault.com/play/1023138/Producer-Bootcamp-How-Saying-No
My 2018 talk about producer skills: https://www.gdcvault.com/play/1024798/Producer-Bootcamp-Be-the-Best

The Goal: http://www.amazon.com/Goal-Process-Ongoing-Improvement/dp/0884271951 a parable about process and project management.

The Mythical Man-Month: https://en.wikipedia.org/wiki/The_Mythical_Man-Month about process, its limitations, and its benefits.