Hello everyone, and welcome to Agile Spikes in Game Dev

Where I’m going to try and cross-pollinate an idea from the rest of the Agile development world into game development.

It’s me!

Indie Dev

Uni Lecturer

EA Melbourne

PlaySide

…Independent!

But first, a little about me – Vikram!

1. I started my games career as an engine and tools programmer at a small indie studio in Melbourne

2. Taught at a university for a couple of years, getting better at communication and transferring knowledge

3. Worked on engine tech at EA for about 5 years

4. Since then, I moved to PlaySide, I was Senior Lead Engineer and 2IC of Engineering, working on improving the Engineering Department, especially training leads in Technical Production, in order to improve integration with Production

5. And now, I’m getting ready to start my next role!

Overview

1. What are Spikes?
2. Spike Goals
3. Game Dev Spikes
4. Practical Examples

30s → 1m

Okay, so today I'm introducing how to use Agile Spikes in Game Dev

1) And to make this talk as accessible as possible, I'm going to first introduce what they are

2) Then we're going set some goals for what a game dev spike should

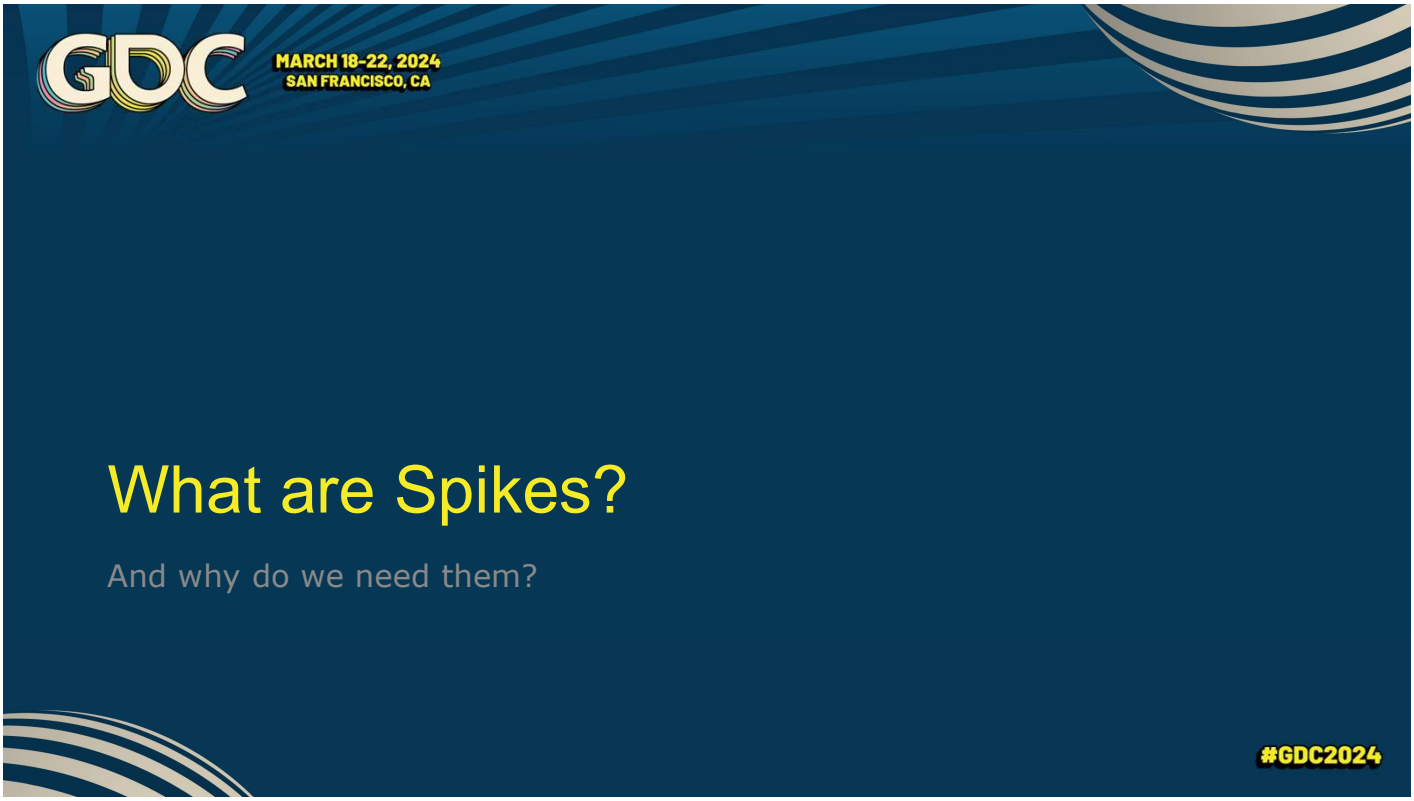
look like

3) After which, I'll introduce a framework for you to use, so you know how you can use them for yourself and your team

4) And finally, we'll dive into some practical examples.

Hopefully, this talk will prove to be useful for Producers, Leads, and anyone interested in improving their ability to learn

Now as a note, this talk will *mostly* talk about Engineering as a basis for spikes, because I want to draw on personal experience as much as possible, but other disciplines can absolutely make use of this too



0.5m → 1.5m

So firstly, hands up how many of us are in Production? Whether as a Lead Engineer or as a Producer or whatever your title is?

- Right, keep your hand up if you've ever heard of Spikes?

Cool! It's pretty much what I

expected.

Let’s learn something new, and hopefully I can give you all a new tool in your kit!

A spike is a short, reproducible, exploratory task

to gain the knowledge you need to **reduce the risk** of a technical approach,

better understand a requirement,

or **increase the reliability** of a story estimate.

1.0m → 2.5m

My definition of a spike is thus:

1) It is a short, reproducible, exploratory task

2) And it either reduces risk

3) helps understand requirements

4) or reduces estimate variability

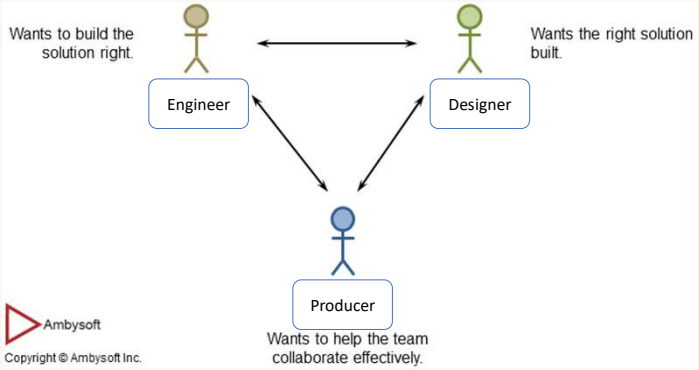
This definition is the combination of several Agile frameworks' opinions on what a spike is – such SAFe, LESS, XP, etc. – blended with some of my own experience with using Spikes as a knowledge base builder and teaching tool.

The only novel thing in my definition, is the term **reproducible**, and I'll show you how to make that work for you.

Let's map these 3 potential benefits onto your team to see how this can work

Bridging the Gap

- Validate assumptions
- Clarify requirements
- Facilitate and plan



0.5m → 3.0m

The titles on this slide are from Agile Modelling – let's convert them to game development roles.

1) We want to bridge the gap between these three parties.

2) The Engineer wants to **validate assumptions** and make sure they're building the solution well – after all,

they will be maintaining it!

3) The Designers wants to be clear with their needs and make sure their **requirements are understood**

4) The Producer wants to **facilitate** things, and also gain the information needed to **roadmap** and ensure that the team **stays on schedule**

But how does it work?

""[if] you don't know how some domain works […] build a minuscule, core solution [and] discover what the domain really consists of""

- Alistair Cockburn

0.5m → 3.5m

Spikes are an **old** concept – they originate in the late 90's with eXtreme Programming.

One of the co-signers of the Agile Manifesto, Alistair Cockburn, says "[if] you don't know how some domain works […] build a minuscule, core solution [and] discover what the domain really consists of"

This might sound familiar to a lot of us – especially those with experience in ***design*** – isn't this just a prototype?

Well… kind of! Alistair calls Prototypes "the more general version of [a spike]"

So, Spikes are just Prototypes?

- Experiment
- Time boxed
- Communicated

0.5m → 4.0m

Let’s dive into that - what separates Spikes from prototypes, mood boards, mock ups, etc.?

They **are** related – but let’s bring it back to my own department…

1) does your *engineering* department do the same kind of prototyping (aka **experimentation**) as Art, UI, or

Design?

2) And are these activities **time boxed**?

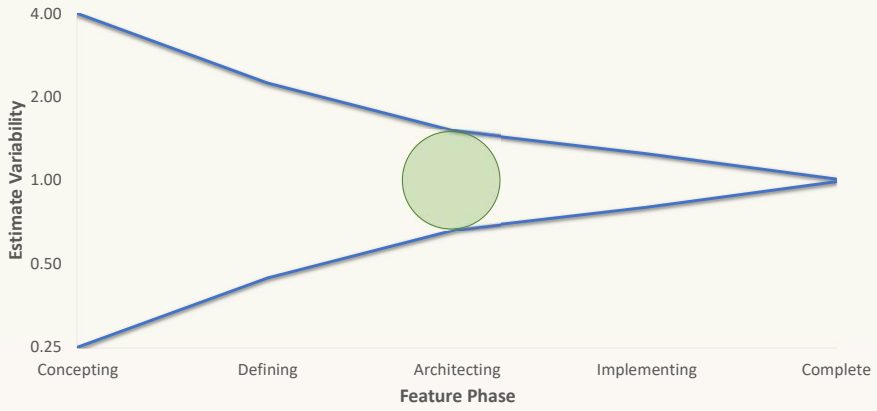
3) And how well are they **communicated**, not just at the moment of completion, but *forward in time*?

Have you ever had a situation where a technical decision was made, that person left the company, and a year later you have a room full of engineers scratching their heads as to **why on earth** would anyone build that thing this way?!

Even if we're already *kind of* doing spikes, surely there are things we can learn from the wider software development industry!?

So, let's dive into a very specific reason to use Spikes:

Cone of Uncertainty



1m → 5m

It's likely that most of you have heard of the Cone of Uncertainty, but let's briefly recap

The Cone of Uncertainty represents the **best-case** accuracy it's possible to have in software estimates at different points in a project or feature delivery.

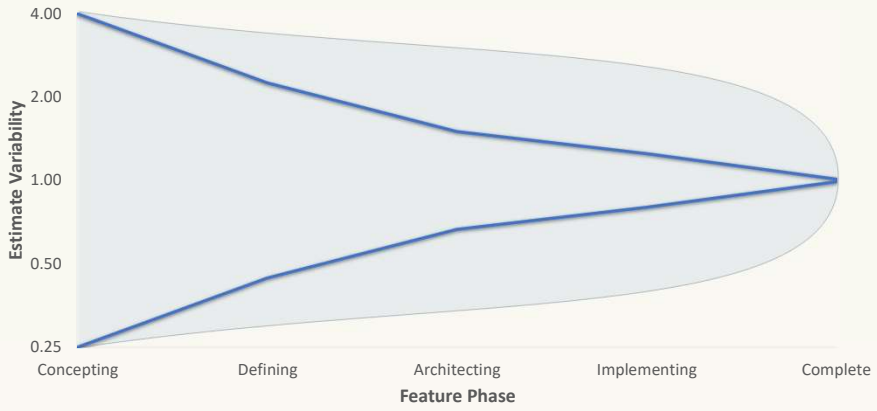
1) The Cone is generally used to highlight that you don't make commitments to delivery too early, you want to ensure your estimate variability is as low as possible before committing.

One important thing to note, however, is that the Cone represents the error, or variability, in estimates created by **skilled** estimators.

It's easily possible to do worse – but it is **not** possible to be more accurate; it's only possible to be lucky.

Let's discuss an adjacent concept to the Cone – the **cloud** of uncertainty

Cloud of Uncertainty



1m → 6.5m

When a project is not conducted in a way that reduces variability—the uncertainty isn't a Cone, but rather a Cloud that persists to the end of the project.

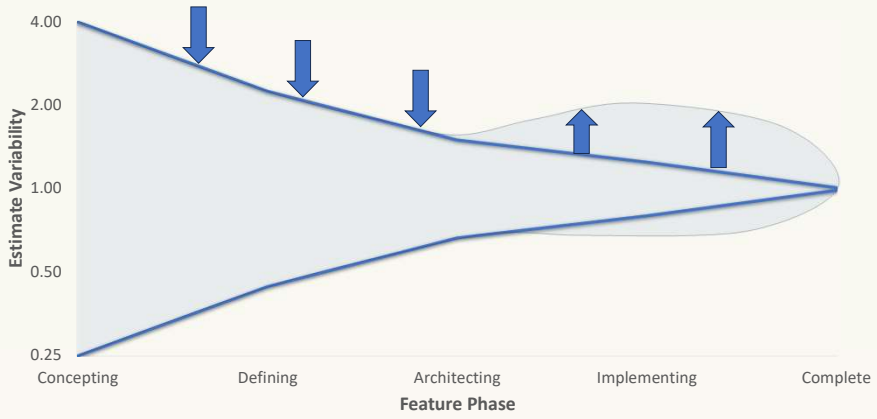
This can be because a project is not well controlled, or where the estimators are not very skilled.

The issue isn't really that the estimates don't converge; the issue

is that the **project itself** doesn't converge, that is, it doesn't drive out enough variability to support more accurate estimates.

The Cone narrows only as you make decisions that eliminate variability

Narrowing the Cloud



1.0m → 7.5m

There are a few things that are recommended to reduce the cloud of uncertainty and bring it closer to the cone:

In regular Software Engineering they talk about

1) creating the product vision,

2) defining the requirements,

3) designing the user interface and making architecture diagrams...

But you and I all know these documents don't always **directly** translate to games, we are a lot more iterative. So, what's the common thread here?

These decision-making processes define what **is** *and* is **not** in scope for your work – and then **document** that decision – *that* is how you narrow the cloud of uncertainty.

So, in game development, we have equivalent decision-making processes about what is and is not in scope for our work

Art mood boards combined with hardware limits makes art probably one of the least troubled departments

Prototypes and design documents can

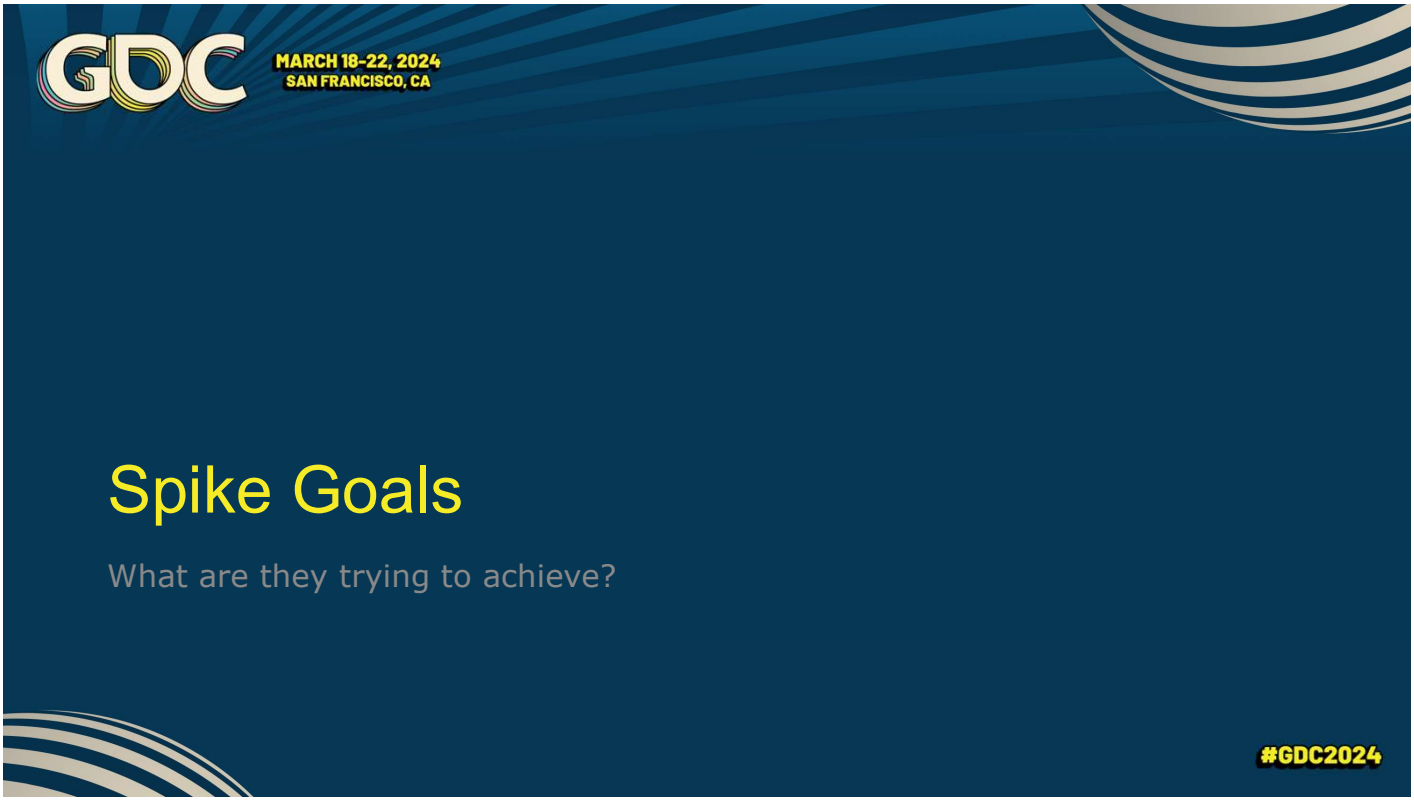
work for design

4) but what about scope creep?

And what about your engineers? How detailed are your technical design documents, really?

5) And how often are your engineers flying by the seat of their pants as you iterate further?

Using spikes effectively should be able to help you make the kinds of decisions that keep narrowing the cloud of uncertainty.



→ 7.5m

Okay, so let’s take all this background and knowledge and spin it into a set of requirements more directly applicable to game development… what should we include?

Requirements of a Spike

1. They must rapidly answer a question
2. Communicate learnings and create alignment
3. Must reduce estimate variability
4. (and/or) must increase confidence in a solution

1.5m → 9m

1) A spike should answer a question – and do it *quickly* – quicker than just trying to do the full solution in the first place, and with a return on investment for doing so
2) These answers should be documented in a way that shares the understanding, not just to the immediate team in the immediate now, but also for future team

members and to inform future decisions

3) The Spike should reduce estimate **variability** for the task it is looking at – we should be able to **better predict** how long it will take to make the full solution
4) and/or it should increase our **confidence** that a given solution is the right one – we need to **know** that the solution we're about to spend weeks or months or years building will **solve the problem**

A Spike must... reduce Project Risk

… now how on Earth do we do that?

→ 9m

If we can make a Spike format that can do this, I think we can all see how this would…

1) reduce project risk!

2) Now how are we going to achieve our 4 goals?

Answer a question rapidly

- Define the question to minimise time spent
- Timebox the amount of time allowed for the Spike
- Close the loop by allowing the answer to raise questions

1m → 10m

To answer a question rapidly…

1) We must first define the question – the more narrowly defined, the faster the answer can be generated!

The question should also define what kind of solution we are expecting – this this Spike designed to reduce estimate variability, or increase

confidence in a solution?

2) Secondly, timebox it – don’t let your engineer build an ivory tower solution

3) Thirdly, isolate any follow up work by allowing the initial answer to raise new questions in a way that can be prioritised by the Product Owner!

Communicate learnings

- Produce something that we can look at
- Standardise a report and keep them accessible
- Review the findings

1m → 11m

How do we make sure that this is communicated and created alignment?

1) The developer must **produce something** – they can't just "do research" and then spit out an answer.

2) We're going to ensure the answer

is **documented**, and this documentation is going to be **standardised** – could be google docs, markdown files committed into your repo, or confluence pages – whatever works for you!

Specifically, we're going to use a spike report document format that was created by academics, and tweak it for a professional environment

3) Finally, we'll have a **review session**.

This will be where the developer presents their findings to their lead, and optionally a small number of their peers, and gathers feedback and signoff.

Reduce Estimate Variability

- You know what you need to do, but not how long it will take
- Convert large inestimable tickets into smaller well-defined ones
- Prototype or whiteboard a solution
- Create a step-by-step breakdown of how you build a solution

1m → 12m

If the Spike is to reduce estimate variability, then:

1) You know what needs to happen, but not how long it will take

2) You might have, for example, an Epic, and you want to convert it into User Stories to enable better estimation

Or it should be able to take a complicated User Story, improve its definition, and break it down into Tasks

3) There are a few ways to do this, but the classic option is to prototype a solution with no care as to quality, just bash it out!

You could make a prototype in another game engine, or just lock an engineer into a room with a whiteboard, or any other way that you need to get the information to break things down!

4) Then, you should be able to use that to break down the solution

For engineers, it’s common that once we complete something the first time, we immediately have some strong ideas about how we could have done it better

So, get the developer to write down the order they would do it in if they had to do it a second time!

But keep in mind that modern Agile frameworks state that the way you answer the question is a lot less important than answering it rationally!

Increase Solution Confidence

- Understand the Requirements
- Draft multiple possible solutions
- Evaluate each solution

1.0m → 13m

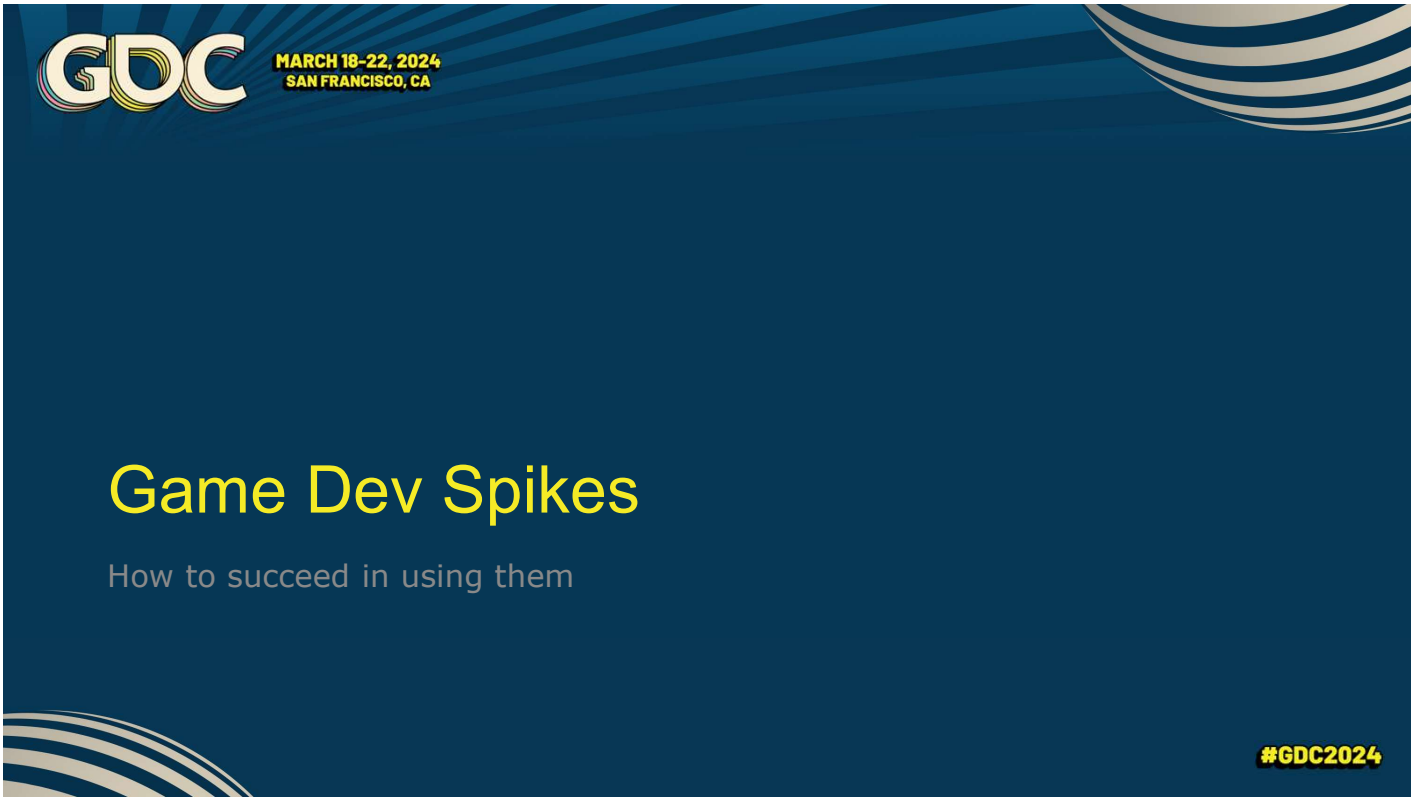
However, if the spike is designed to increase confidence in the solution, my recommendation is to:

1) Understand the Functional and Technical Requirements (You can use the IEEE System Requirements Specification non-functional requirements to help drive this)

2) Ask the engineer to propose multiple solutions, not just the first thing that comes to mind.

And they must be **communicable** – so it’s likely you want to use some form of architecture diagramming

3) And then you want to Compare and Contrast the possible solutions, using the requirements that you’ve specified



→ 13m

So, practically, how can we use them for game development?

I’m going to provide you a framework now.

The Game Development Spike - Plan

- Provide Context
- Specify the Question
- Set Acceptance Criteria
- Timebox the investigation
- (Optionally) Provide Resources

1m → 14m

This will come in three parts - firstly, the Spike Plan

Have the Lead, Production, or a Senior Engineer write this, before you start the investigation

1) The Spike needs to have context, why is there a problem?

2) The Spike needs to ask a very

specific question

3) The Spike needs very specific acceptance criteria

4) The Spike needs a deadline or a timebox. How long are we allowed to spend on the investigation?

5) If you’re asking a Junior to do the Spike, give them some help. Lay out some high-level steps that you would take, based on your experience, to answer the question.

The Game Development Spike - Answer

- Generate artefacts needed to answer the question
  - Prototype
  - Work breakdown structure
  - Feature technical designs
  - Anything else

1m → 15m

Then, the actual work to answer the question

This depends on the actual question being asked here, and we'll dive into a few possible ways to do this later on, depending on the type of spike.

1) But ultimately, the developer

needs to **generate artefacts** which answer the question.

These could be:

2) A prototype, either mocked up in your existing game project, or a very rough mock up in a quick-to-use engine like Unity or Unreal

3) A work breakdown structure, indicating how a particular epic can be broken down, including dependencies and estimates

4) A full feature technical design document, or just Architecture Diagrams, even if they’re just drawn on a whiteboard and you snap a photo

5) Or absolutely anything else the developer needs to answer the question

The Game Development Spike - Report

- What did they do?
- What did they find out?
- What would they recommend?
- Highlight any open concerns
- Present the report

1m → 16m

Finally, within the timeboxed period, the developer must write a Spike Report to communicate their findings

Technically, this **is** an artefact that is generated by the Spike process, but it is more of a meta-artefact, that brings them all and in the confluence binds them

1) It needs to include a brief recollection of the steps that the developer undertook to answer the question

2) It needs to include a section on what the developer learned, and it should focus on the **facts**

3) Then, separately, they can weigh in with opinions – e.g. telling us **which** solution they think we should use

4) If there are any open risks, concerns, or recommendations, the developer should highlight them

5) And the developer must *present* the report, to their lead and/or a **small** group of peers (to avoid bike-shedding), and get it signed off

The theory here, is that if someone else was to repeat the spike, they should be able to do it **much faster** by following in their shoes, and get a similar result

Types of Spikes

1. Functional Spikes
2. Technical Spikes

0.5m → 16.5m

Okay, now we know the framework… let's get even deeper into the techniques to answer different types of questions.

So, if some spikes Reduce Estimate Variability and others improve your confidence in a given Solution, can we break this into two kinds of spikes? Of course we can!

We’re going to describe two types of spikes: Functional and Technical Spikes.

For this part, I can’t find a true origin, but it has been codified in a few places.

The definitions here are my interpretation of a combination of definitions from SAFe, Microsoft’s Code with Engineering Playbook, IBM, and others.

1) Firstly, we have Functional Spikes – They are used to make sure we’re on the right track – aka “Analyse a known Solution” – which should reduce estimate variability

2) Secondly, we have Technical Spikes - Figure out which way to go – aka “Research possible Solutions” – which should increase your confidence in a solution

Sometimes, a Spike might require **both** styles of discovery.

The spike’s acceptance criteria will define its true purpose, and knowing its particular type doesn’t add much value beyond what we’ll find in the acceptance criteria.

Functional Spike Indicators

- Do we understand what the user needs/designer intends?
- How much work is there and how do we break it down?
- It should be possible, but we don't know how
- *Reduce Estimate Variability*

1.0m → 17.5m

Functional Spikes are used to "unsure" questions like:

- We're unsure of the requirements
- Unsure of how long something will take
- Unsure of how to implement this

These spikes often **validate** solutions that you already have in mind, and they **often** involve a prototype or a work breakdown structure, or both.

If you create a prototype, you **must** discard it – prototypes entering production code is often cited as a **risk** introduced by using Spikes

1) These spikes, generally, **reduce estimate variability**

Technical Spike Indicators

- Should we build our own solution or buy something?
- How much will this feature impact performance?
- How should we architect the implementation?
- *Increase Solution Confidence*

1m → 18.5m

Whereas Technical Spikes are used in these situations:

- They are about **finding** the right solution
- These **often** output some form of technical design documentation

1) These spikes, generally, **increase**

confidence in a solution

Elements of a Successful Spike

To remember the key elements of an effective technical spike, use the STREET acronym:

- Stakeholder: **Identify the stakeholder.** This person might be the technical lead who needs to select the most suitable programming framework or tool, or the solution architect who is seeking detailed information to make a high-level decision.
- Timebox: **Agree on a timebox.** The timebox can vary from hours to days based on the scope of the investigation. It normally falls within the bounds of an iteration. Discuss undertaking another spike if the timebox doesn't provide enough time to address the criteria.
- Review: **Review as a team.** At the end of the timebox, document the outcome of each option and review the outcomes as a collective. Invariably, the discussion of each participant's experiences in building the spike solution leads to the collective decision.
- Expected outcome: **Clearly define the expected outcome** and value of the spike. The outcome might be a decision that needs to be made or a risk that is mitigated, and the impact on the project.
- Evaluation criteria: **Establish the evaluation criteria.** What must each option accomplish from a functional and nonfunctional perspective? Nonfunctional criteria can include ease of adoption, comprehensive documentation, and active ecosystem.
- Team: **Establish the team.** Each option under evaluation must have a defined and equal effort assignment to ensure that comparisons are sensible and appropriate.

IBM's Street Acronym

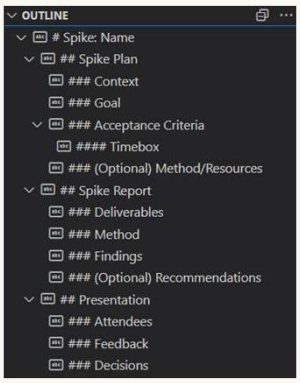
1.0m → 19.5m

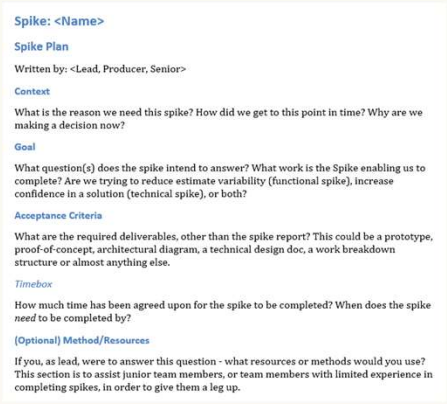
Finally, to back me up, I'm going to refer to IBM's acronym, STREET, which it uses to ensure spikes are effective

- Identifying the **Stakeholder**
- Agreeing on a **Timebox**
- **Reviewing** as a team
- Defining the **Expected** Outcome

- Establishing the **Evaluation** Criteria
- And establishing the **Team**.

Spike Report Template





Spike: <Name>

Spike Plan

Written by: <Lead, Producer, Senior>

Context

What is the reason we need this spike? How did we get to this point in time? Why are we making a decision now?

Goal

What question(s) does the spike intend to answer? What work is the Spike enabling us to complete? Are we trying to reduce estimate variability (functional spike), increase confidence in a solution (technical spike), or both?

Acceptance Criteria

What are the required deliverables, other than the spike report? This could be a prototype, proof-of-concept, architectural diagram, a technical design doc, a work breakdown structure or almost anything else.

Timebox

How much time has been agreed upon for the spike to be completed? When does the spike *need* to be completed by?

(Optional) Method/Resources

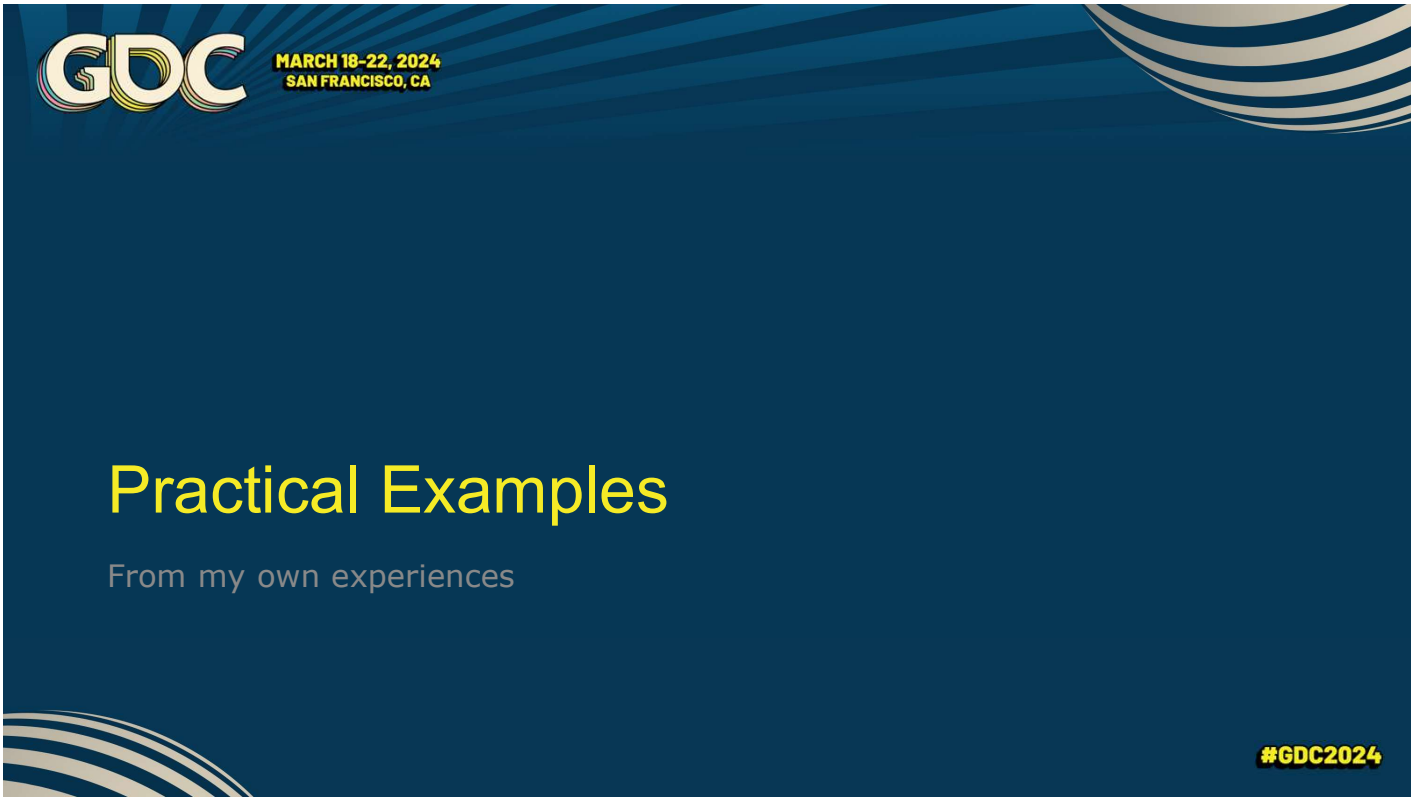
If you, as lead, were to answer this question - what resources or methods would you use? This section is to assist junior team members, or team members with limited experience in completing spikes, in order to give them a leg up.

vikram.codes/spikes

0.5m → 20m

Alright, and to help you all out, I've uploaded a template for the Spike Plan and Report onto my website!

It comes in markdown, as well as word doc, so you can make it fit whatever documentation platform you use yourselves!



→ 20m

Okay so let’s dive into a few examples from my own history

I’m going to keep these quick and snappy, as well as anonymised to avoid running into NDAs or legal problems, but there’s enough about how and why the spikes reduced risk

Functional Spike – Roadmap

- Background:
  - Large feature
  - Solution decided early
  - Senior with information
  - Inability to roadmap
  - Perpetually “two months away”
- Cost: 8 engineers, 2 weeks

- Outputs:
  - Prototype of alt. solution
  - Architecture diagram
  - Estimated WBS
- Benefits:
  - Changed solution direction
  - Prioritised another feature instead

You can use this to extract ideas from an expert, break down a big problem into an estimable solution runway, and make pivot decisions

30s → 20m30s

Functional Spike – Knowledge Base

- Background:
  - Work for hire project
  - Custom engine
  - Institutional knowledge
  - Need to do a “simple task”
- Cost: 1 engineer, 2 days

- Outputs:
  - New material shader
  - Confluence page
- Benefits:
  - Upstream refactored the system
  - Future shaders became quicker

You can use this to generate documentation and improve developer experience

30s → 21m

Technical Spike – Performance

- Background:
  - Performance Issues
  - 1000’s of objects on screen
  - Technical Art solution
  - Multiple ways to redo impl.
- Cost: 1 TA & 1 eng., 1 week

- Outputs:
  - Feature Technical Design Doc
  - Comparison of 3 solutions
- Benefits:
  - Reduced perf impact by ~30%
  - Taught TA complexity analysis

You can use this to teach people things, and it’s not just for engineers

30s → 21m30s

Technical Spike – Architecture

- Background:
  - Custom Engine
  - Porting to VR
  - Need to redesign UI systems
  - Eng. is certain of a solution
- Cost: 1 engineer, 2 weeks

- Outputs:
  - Feature Technical Design Doc
  - Comparison of 3 solutions
- Benefits:
  - Validated Requirements
  - *Eng.* chose different solution
  - Avoided impl. that didn't meet needs

Use it to validate how you build something, and test an engineer's assertions

30s → 22m

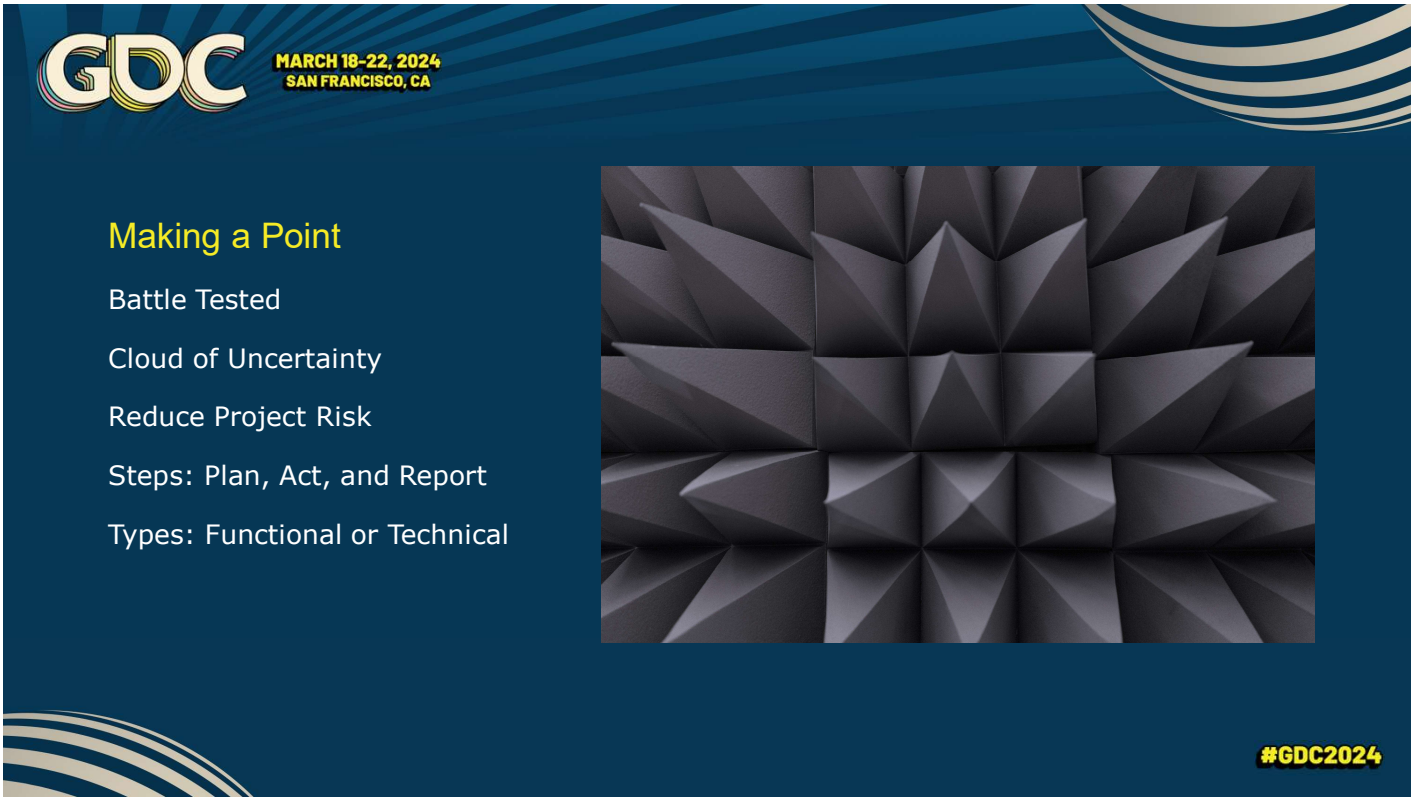
Failed Spike – Testing

- Background:
  - Unreal Engine Project
  - No automated testing
  - Constantly breaking custom tooling
  - Nobody has exp. with UE testing
- Cost: 1 engineer, 4 weeks

- Outputs
  - A report on all UE testing abilities
  - But no actual tests implemented?!
- Benefits:
  - I learned to set narrower goals
    - "how can we implement tests?"
    - "implement a single test"

This shows what **not** to do

1m → 23m



1m → 24m

So, as a summary:

1) Spikes are a battle tested Agile tool dating back to Extreme Programming

2) They're a way to narrow the Cloud of Uncertainty

3) Ultimately, therefore, their goal is to Reduce Project Risk

4) The three steps of my framework are to: Plan, Act, and Report

5) And there are, broadly speaking, two types of Spikes: Functional or Technical

Call to Action

- Download the template
- Try it *once!*
- "That'll take a [month]" - Functional Spike
- "My solution is obviously the best" - Technical Spike

1m → 25m

1) I want everyone to go download the spike template

2) And in one place in the next couple of months just try it out!

Fully expect that the first spike/report is not going to be much better than your current process

It takes a couple of iterations to build

the skill up to really maximise the benefit

But just try it **once** and see if it doesn't give you *any* benefit

Because if it can give you benefits after your first attempt, you know for sure it will give you even more after you get better at the process and customise it to suit your team.

3) If an engineer says, "that'll take a long time", do a functional spike

4) Or if an engineer says, "Oh I just **know** my solution is the best", make them prove it with a technical spike!

Oh! Also, please don't assume that spikes apply **only** to your engineering team!

Your design team or art teams may also find value in having a standardised report!

Especially for generating documentation when trying out new technology or processes.

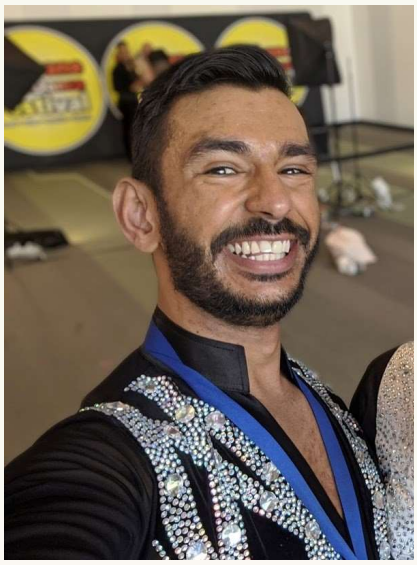
Finally – a lot of what I've discussed here actually has a historic context within Agile, as well as an academic and pedagogical backing – so if you'd like to download the slides from GDC Vault and check out the Appendices, please do so when you can!

Get in touch

contact@vikram.codes

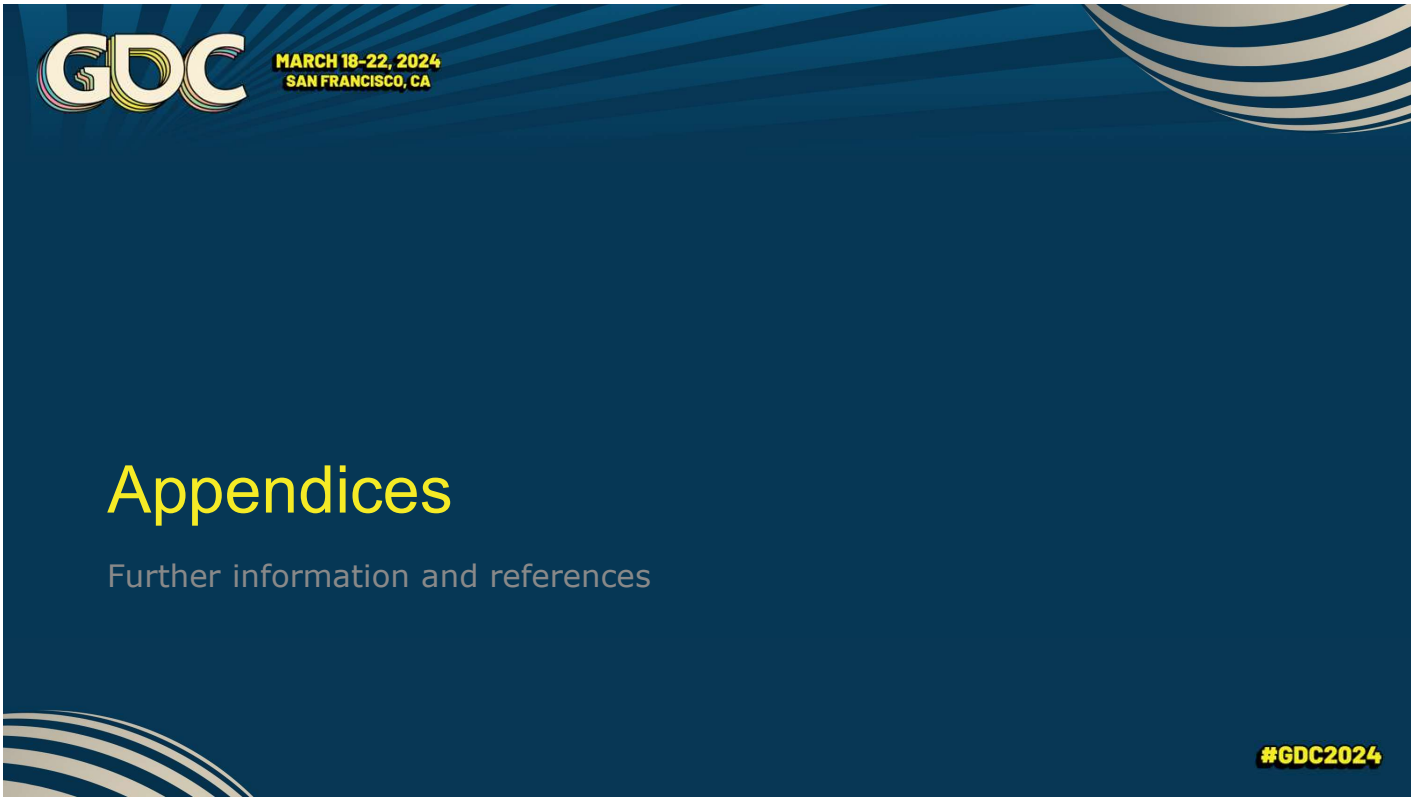
Download the template:

vikram.codes/spikes



→ 25m

And that’s it! I’ve been Vikram, please get in touch if you have questions, and uh… any questions now?



This is a set of references and slides that were originally considered for inclusion – before being cut in order to make the time limit!

References

- http://www.extremeprogramming.org/
- https://wiki.c2.com/?SpikeSolution
- https://scaledagileframework.com/spikes/
- Agile Development Spikes Applied to Computer Science Education, Woodward et al.
- https://agiledictionary.com/209/spike/
- https://web.archive.org/web/20180712125321/https://scrumalliance.org/learn-about-scrum/agile-atlas/agile-atlas-commentaries/may-2014/spikes-in-scrum-the-exception,-not-the-rule
- https://microsoft.github.io/code-with-engineering-playbook/design/design-reviews/recipes/sprint-spike-template/
- https://microsoft.github.io/code-with-engineering-playbook/design/design-reviews/recipes/technical-spike/
- https://microsoft.github.io/code-with-engineering-playbook/design/design-reviews/recipes/engineering-feasibility-spikes/
- https://www.ibm.com/garage/method/practices/discover/value-from-technical-spikes/
- http://www.jamesshore.com/v2/books/aoad1/spike_solutions
- https://www.researchgate.net/publication/323829859_Practices_for_Achieving_Accuracy_in_Software_Costing_and_Estimation
- https://www.mdpi.com/1999-5903/11/9/196
- https://www.researchgate.net/profile/Luigi-Lavazza/publication/337832147_ICSEA_2019_-_The_Fourteenth_International_Conference_on_Software_Engineering_Advances/links/5dee23d392851c83646e6420/ICSEA-2019-The-Fourteenth-International-Conference-on-Software-Engineering-Advances.pdf#page=167

--- TODO: make sure this includes everything I reference

--- TODO: make sure everything included in this is referenced in various slides

The most obvious way to start extreme programming (XP) is with a new project. Start out collecting user stories and conducting spike solutions for things that seem risky.

http://www.extremeprogramming.org/start.html

A spike is

Spikes are *ancient* – well, sort of!

They were, to my knowledge, first documented in **1999** as part of eXtreme Programming (XP) – that is a couple of years *before* the Agile Manifesto was published

So yes, although the title of this talk is “Agile Spikes for Game Development” – Spikes pre-date agile!

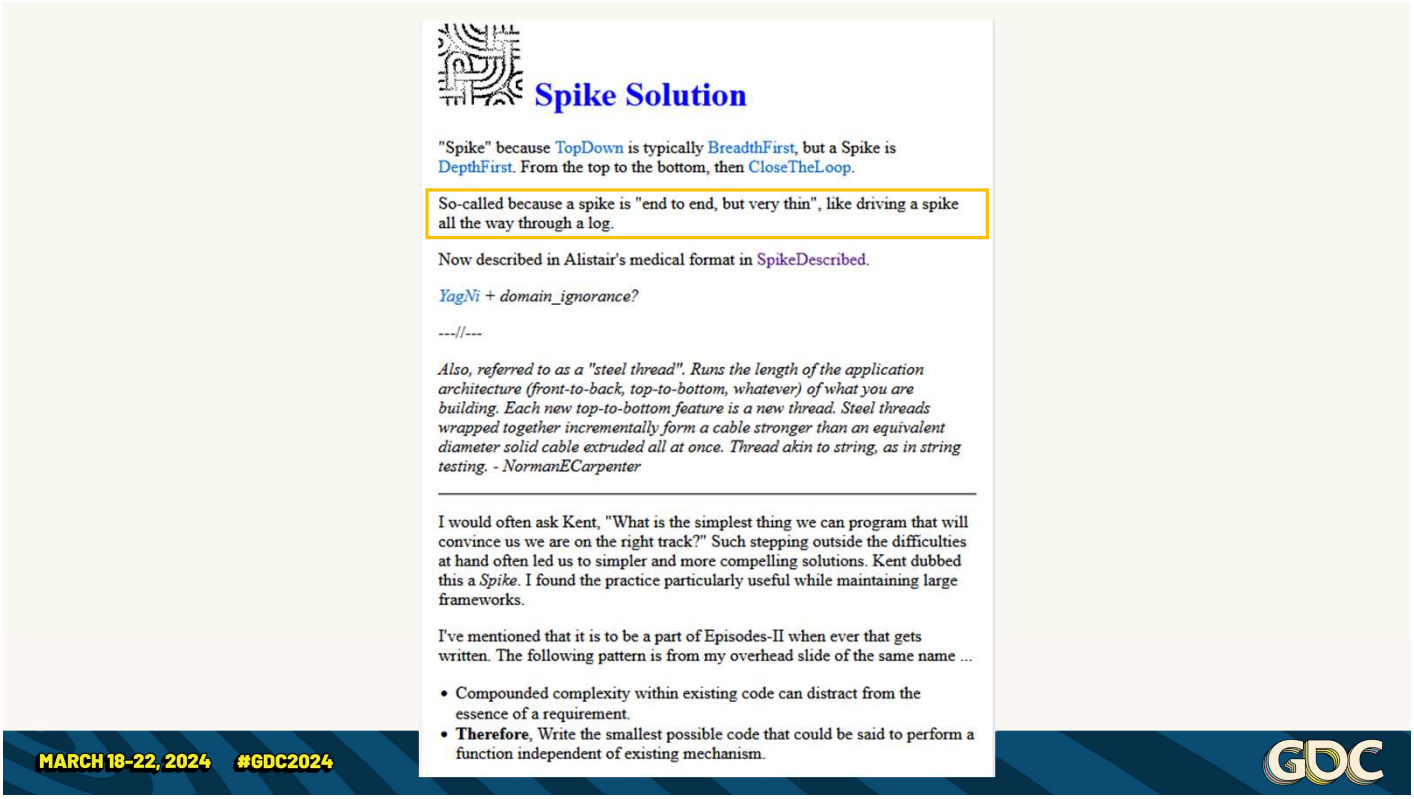
Now, XP advises people to start new projects by conducting “spike solutions” for *things that seem risky*

But what **is** a spike?

"What is the simplest thing we can program that will convince us we are on the right track?"

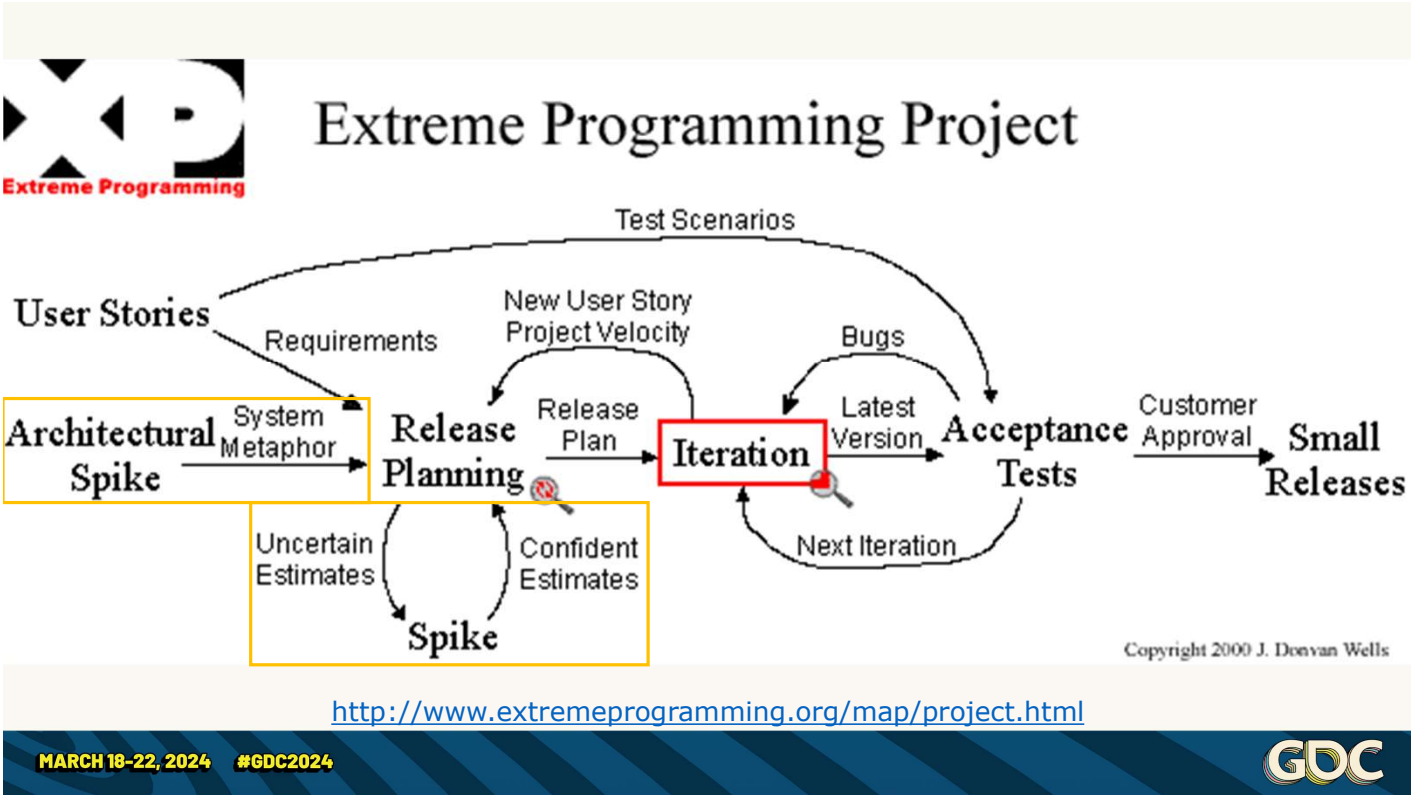
- Ward Cunningham

Ward Cunningham (pioneer of XP and co-signer of the original Agile Manifesto) says **this** about them on the world's first wiki (wikiwikiweb, aka wiki.c2.com, aka "the wiki")



Why it’s called a spike is hotly contested, but wikiwikiweb says…

…it is because “a spike is "end to end, but very thin", like driving a spike all the way through a log. “



Within the project map of XP, there are actually *two* references to Spikes (http://www.extremeprogramming.org/rules/spike.html):

1. Architectural Spikes, which output a System Metaphor to **inform** Release Planning (what we might call pre-production in the games industry)
2. And "regular" Spikes, which take *uncertain* estimates and return

confident estimates

So, leaving the realm of… wow, okay, 25 years ago (at the time of presenting)…

Extreme Programming helped birth Agile, Agile has been implemented and reinterpreted dozens of times.

The most common flavour, arguably, being Scrum.

One noteworthy work, in 2007, was a

book called "**The Art of Agile Development**" – which had a solid section on Spikes

And then we have variations on Scrum and Agile to raise the bar for larger teams and larger projects!

SoS (Scrum of Scrums), LESS (Large Scale Scrum), DAD (Disciplined Agile Delivery), SAFe (Scaled Agile Framework)

I don't, as it currently stand, have any strong preference for which way to scale Agile is best – especially within games.

But we're not here to answer that question, we're here to talk about Spikes!

Let's examine what a couple of the most common modern frameworks have to say!

Slide 43

VS0 If you're reading this, I'm not actually going TODO this... it was for when this slide was originally in the presentation
Vikram Saran, 2024-03-15T02:41:55.825

Architects clarify by programming spike solutions

[…] encourage master-programmer architects to first refine and discover ideas through *programming* a *spike solution* […]

https://less.works/less/technical-excellence/architecture-design

LESS says:

Your architect should **refine and discover ideas** through programming

Architects clarify by programming spike solutions

[follow that by] conveying the insights to other developers or by documenting the discoveries in an agile documentation workshop.

https://less.works/less/technical-excellence/architecture-design

LESS says:

And then **convey the insights** – possibly by *documention*

[Spikes] purpose is to gain the knowledge necessary to **reduce the risk** of a technical approach, **better understand** a requirement, or **increase the reliability** of a story estimate.

https://scaledagileframework.com/spikes/

In SAFe, they call Spikes an “Enabler Story” and say that: …

Note something important here: Once it left the realm of XP, the **format** of the answer started to matter less and less.

The more important thing is that it achieves its goal of **reducing uncertainty**

Agile Development Spikes Applied to Computer Science Education

- Published at the 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)
- A combined effort from researchers at Swinburne University of Technology and the Australian National University

http://users.cecs.anu.edu.au/~ejmontgomery/publications/2013-08_spikes_tale.pdf

Clinton Woodward, James Montgomery, Rajesh Vasa and Andrew Cain published a paper on using Agile Spikes to improve learning outcomes in Computer Science Education in 2013.

(SAFe 1.0 was first released in 2011, and the 1.0 Big Picture does in fact

mention Spikes, so it’s likely this was a direct inspiration)

For the record, I had the great opportunity of learning under this model, and not only were the semesters in Games Programming and AI for Games learning from Dr. Clinton Woodward the most engaging, motivating, and informative classes I had in my undergraduate degree, they also propelled the start of my career in the industry, as well fueling some of my own post-graduate research.

The paper codified and introduced a few things that we’re going to need later, let’s have a look at them:

KoST Model

- Knowledge gaps – missing domain knowledge
- Skill gaps – a lack of practice or experience, leading to poor estimation
- Technology gaps – does the technology exist to create the solution needed, given the budget and timeline

Firstly, they created the KoST model, to identify “Knowledge, Skill, or Technical” gaps

Knowledge gaps represent what a person does not know about the domain for which they are developing.

Skill gaps exist where a person knows what is required and broadly how to do it but requires more practice or experience. A skill gap

prevents accurate estimates of how long a task will take.

Technology gaps address the question of whether, given a budget and timeline, there exists the technology to assemble the solution required

Note that the gaps are very specific to the teams' unique experience and skill; another development team given the same tasks would be expected to identify very different gaps

Spikes for CS Education

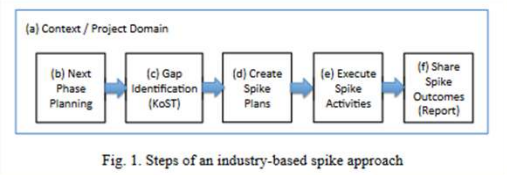


Fig. 1. Steps of an industry-based spike approach

Name:
A meaningful and unique name for this spike plan
Context:
Outline the reason and context for the spike
Knowledge Gap:
Skill Gap:
Technology Gap:
State one or more specific gaps to be addressed
Goals/Deliverables:
List the specific goals and deliverables of this spike.
Describe a tangible and minimal end-to-end solution.
Start date:
Target date:
Good dates are needed for a controlled process.
Planning notes:
Outline a proposed plan of how this spike can be undertaken.
Capture as much team knowledge here as possible to make the work as effective as possible.

Fig. 2. Spike Plan Template

Name:
Should match the original spike plan name
Goals:
Restate the original spike plan goals (outcomes)
Personnel:
State who the contact person is and any backup.
Technologies, Tools and Resources Used:
Provide a specific list of useful resources for other team members to learn from this spike. Specific books, software (versions) and URLs are common.
Tasks Undertaken:
List key tasks likely to be needed to help another developer
What We Found Out:
Concise evidence: graphs, screen shots, list of outcomes, data, notes.
Open Issues / Risk (optional):
List any issues and risks that were not resolved. This may include a new range of risks previously known.
Recommendations (optional):
This might include suggesting additional spikes are needed, or that the gap has been completely solved and the team should move on.

Fig. 3. Spike Outcome Template

Next, they introduced a closed activity loop model, which is somewhat akin to a sprint, and added two specific steps – Creating a Spike **Plan**, and sharing the Spike's **Outcomes** via a *Report*

They note that although there is some overlap between the report and the plan, "there is […] value [in] productivity-by-convention where having a standard format can make

the process of creating outcome reports, and understanding them, quicker and easier.”

After that, the rest of the paper discusses their practical experience in running courses using Spikes as assessments, pedagogical benefits and challenges, and potential avenues to expand this style of teaching.

Even after a little bit of searching academic databases, to my knowledge, only one other University has then implemented this style of teaching in their advanced programming courses – the University of the Sunshine Coast, while I was the Program Coordinator for Games Programming there!

If there are any educators in the crowd, I’d love to see more use of this model. I speak as both a student *and* as an educator!