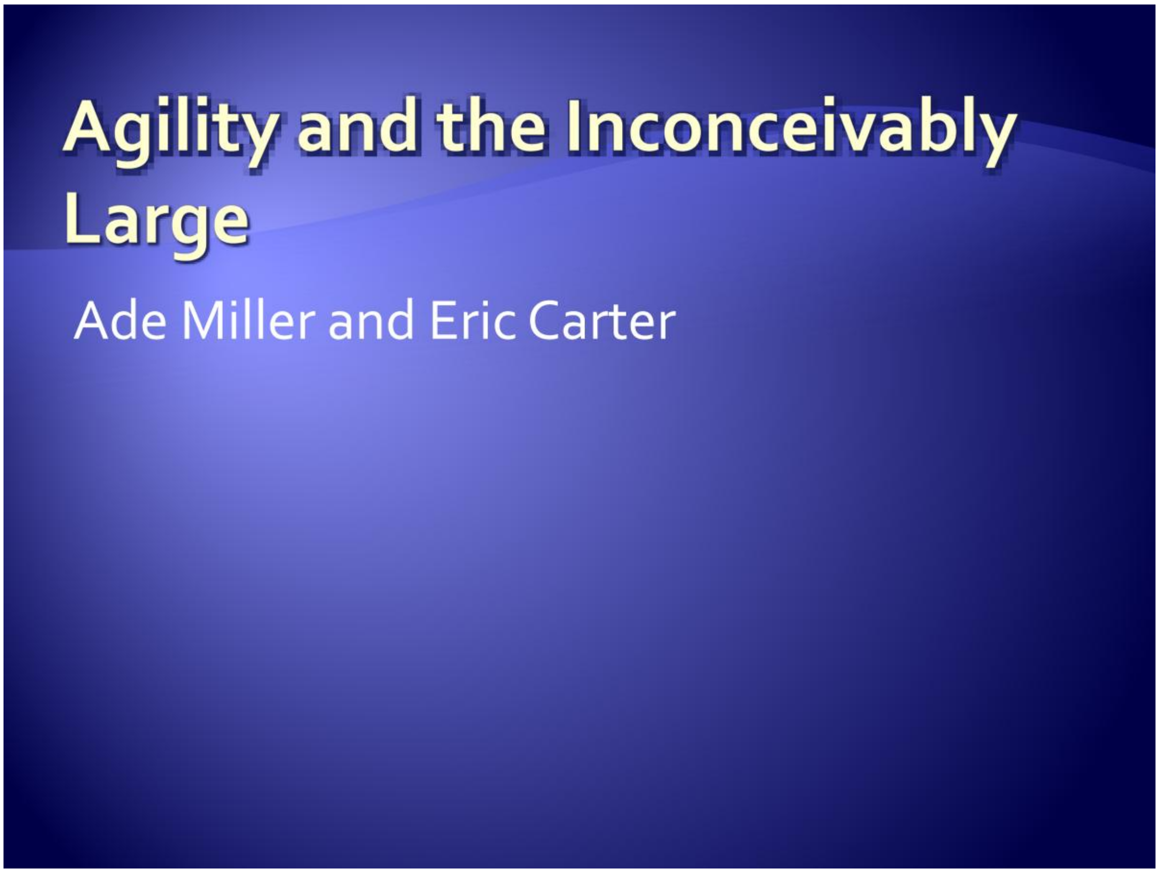


Agility and the Inconceivably Large

Ade Miller and Eric Carter

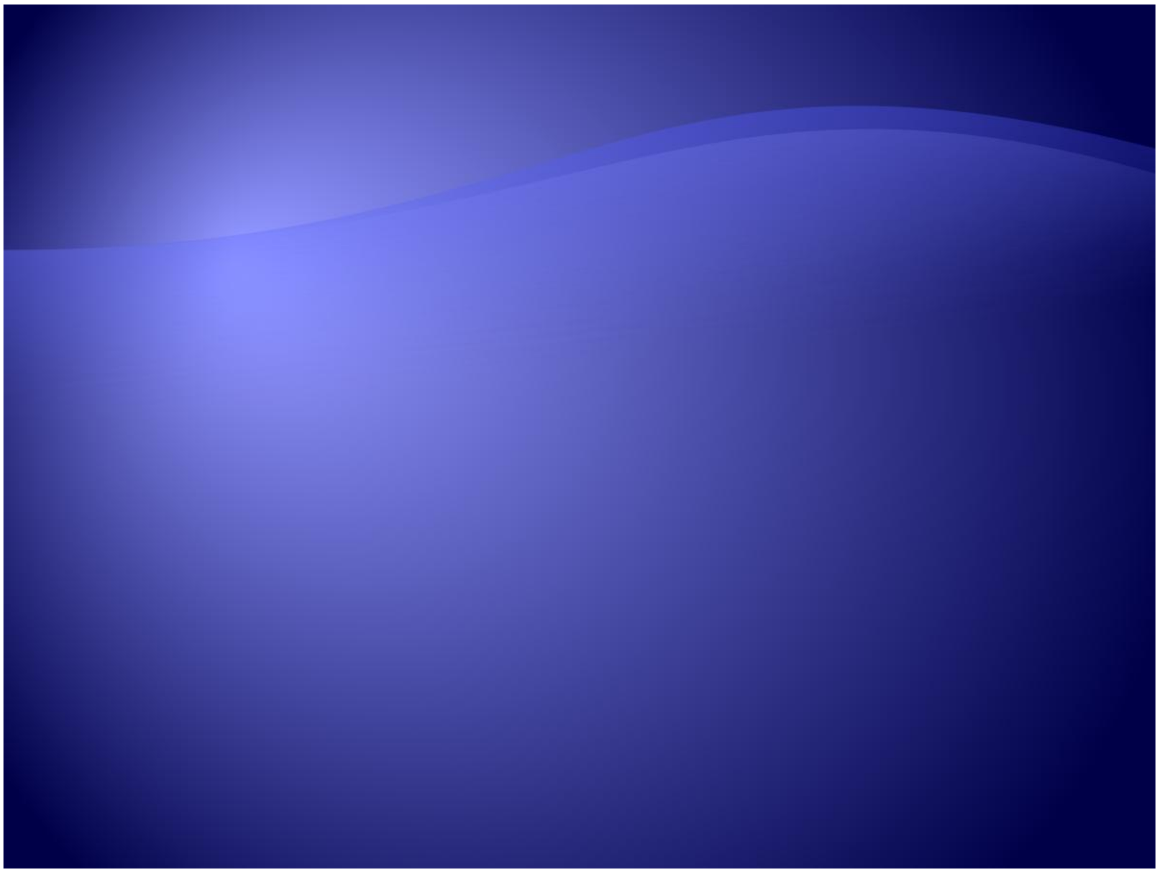
Notes

This deck contains some speaker notes. Most of the real content can be found in the paper that accompanied this deck. You can find a link to the paper on my talks page here:

http://www.ademiller.com/blogs/tech/talks/

The speaker notes for this deck ran off the slide in several places. This PDF includes additional blank slides that were not in the talk to allow you to read the notes.

OK… So here goes!



Let's go!...

INTRODUCTION… TWENTY MINUTES… THE TITLE

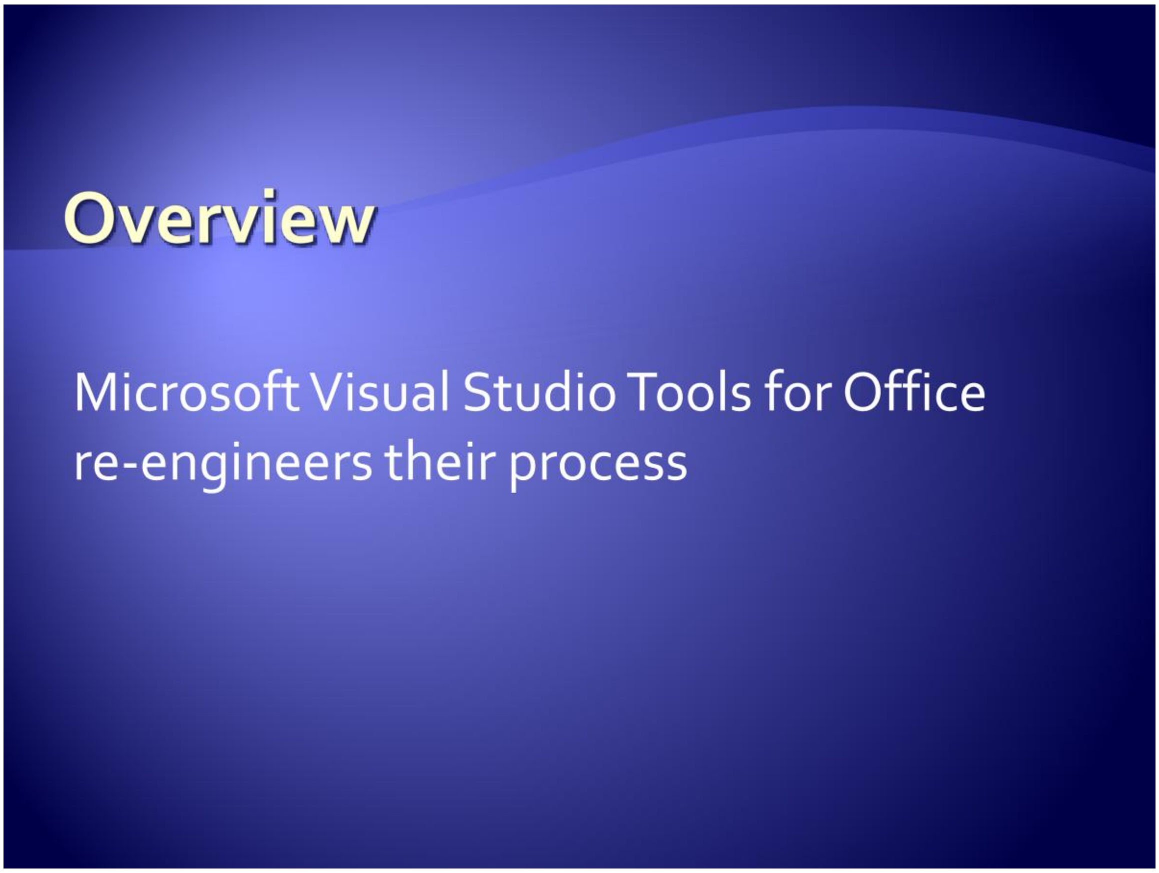
Introduce the authors and apologies for non attendees.

This is a **twenty minute talk** with ten minutes for questions at the end. If you have questions please try and save them up!

I'd just like to say a few words about **the title**…

This isn't about "our software is bigger than your software". It's about the feeling I had when I first joined Developer Division and realized that I was working on a product that included the .NET runtime, several languages and dozens of tools all built around a common IDE in addition to my team taking a additional dependency on Office. As a developer I would get a new drop of *everything* every week or so.

For a while this made my head spin.



Overview

Microsoft Visual Studio Tools for Office re-engineers their process

ONE OF 15 TEAMS… 75 PEOPLE… 43 MILLION LOC… REALLY ABOUT HOW…

This talk is really about how Developer Division rolled out a new software development process across the whole division and how our Product Unit was able to leverage that to enable further agility within that larger process.

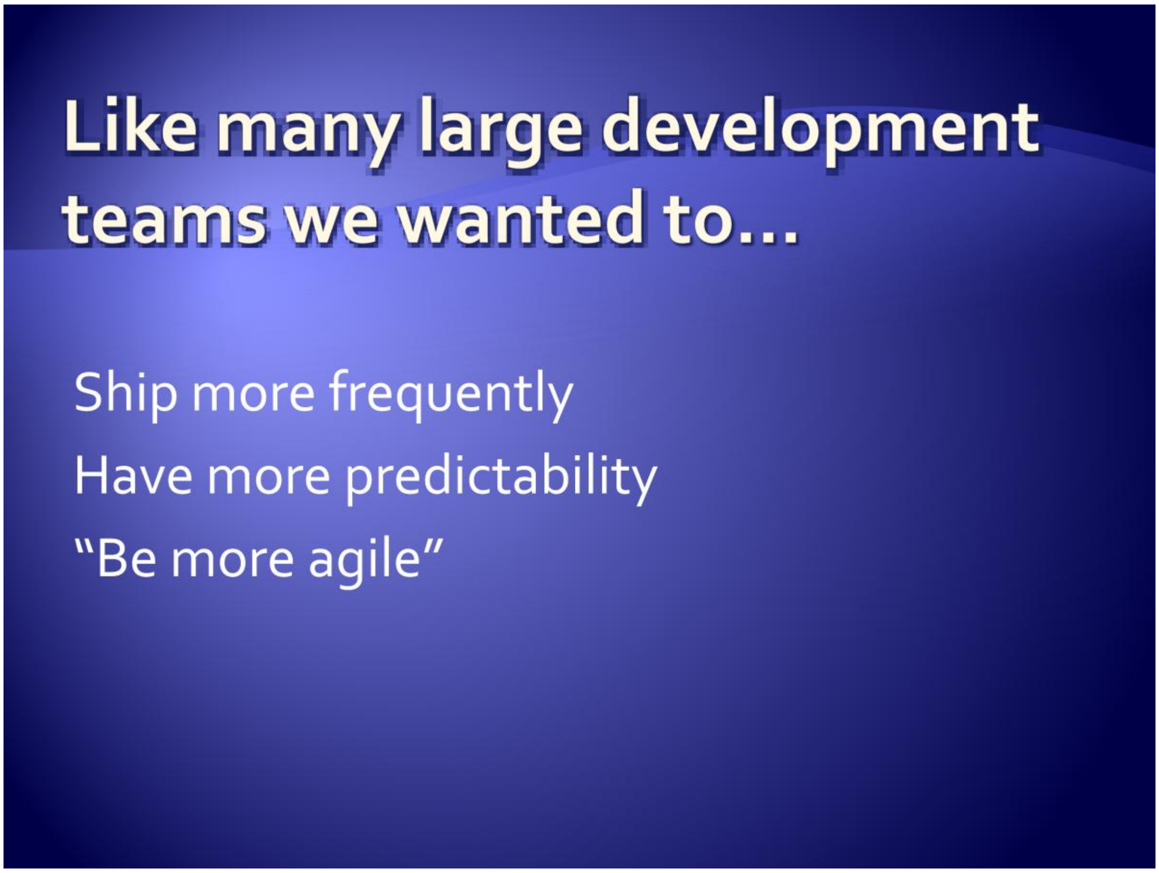
Visual Studio Tools for Office product unit, henceforth referred to at "the PU" – because the full name is waaay too long - is just **one of 15 teams** that comprise Visual Studio.

This product unit develops the toolset within Visual Studio that allows developers to **customize Office applications** using .NET code. For example; custom behavior behind and Excel worksheet written in C#.

The PU consists of **75 people**;

- 30 developers
- 30 testers
- A further offshore test team of 6
- Management comprising the remainder

Visual Studio as a whole is around **1200 people** working on a product containing **43 million** lines of code. Both these numbers are **growing**.



Like many large development teams we wanted to…

Ship more frequently

Have more predictability

“Be more agile”

WHO USES VS… SHORTEN GAPS… DEPENDENCIES ARE CHALLENGING…

Developer Division used the Feature Crew Model to attempt to address these issues across the whole Division. Our PU took advantage of this and did a lot more work to be even more agile inside this existing model.

How many people here use Visual Studio?

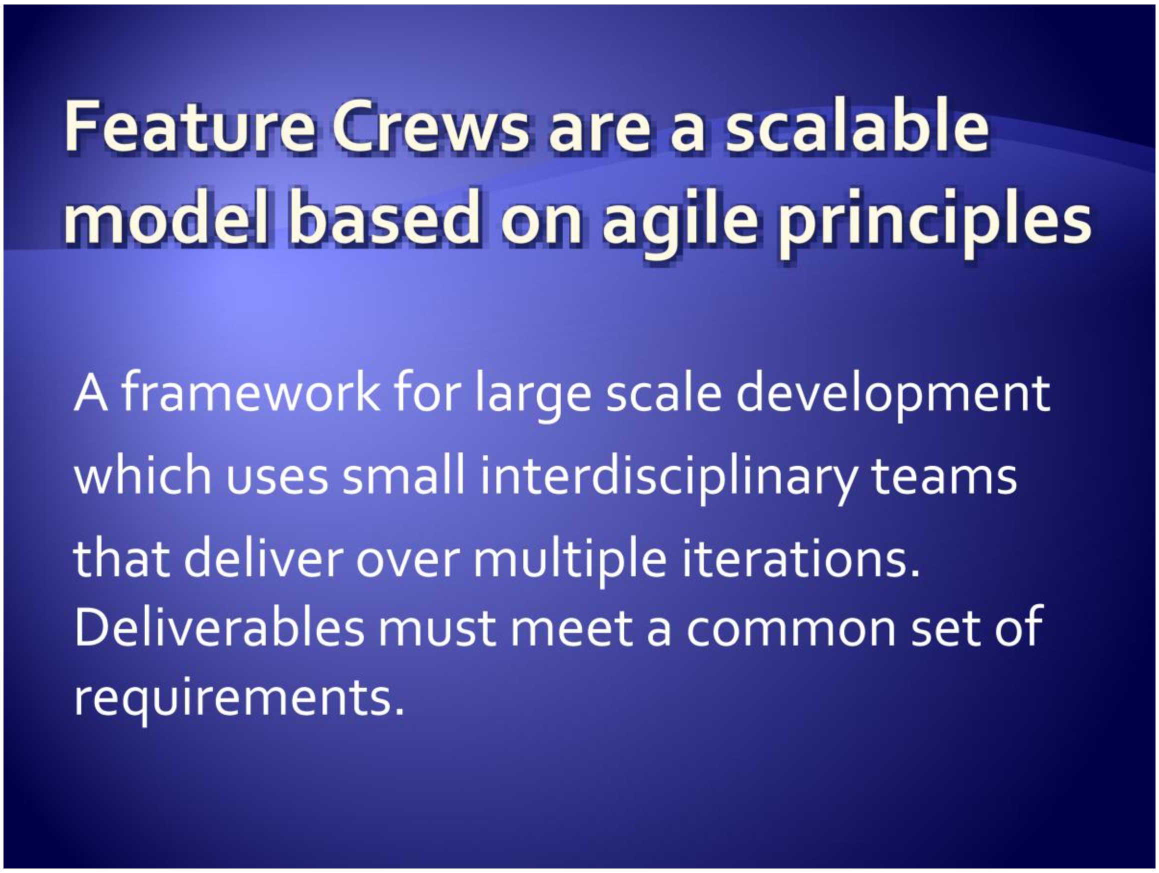
How many of you were OK with **long gaps** between releases?

One of the key drivers was **start to shorten the gap** between releases…

During development we also wanted to ship Customer Technology Previews (CTPs) to customers more **frequently** so that we could get **feedback** and potentially **alter the direction** of product development earlier and more often during the product cycle.

We wanted to **shorten the gap** between features being **implemented**, tested and their **delivery to customers** (in the form of a CTP, betas or the final product). We also wanted to increase the **quality and usability of the CTPs** making sure that features in a CTP were clearly defined.

As wanted to deliver a continuous stream of value to the customer.



Feature Crews are a scalable model based on agile principles

A framework for large scale development which uses small interdisciplinary teams that deliver over multiple iterations. Deliverables must meet a common set of requirements.

FRAMEWORK… SMALL TEAMS… NOT VERY PRESCRIPTIVE… COMMON SET OF REQUIREMENTS

A the feature crew model is really a **framework for large scale development** that seek to address these issues. It was originally developed and used by the **Microsoft Office** teams.

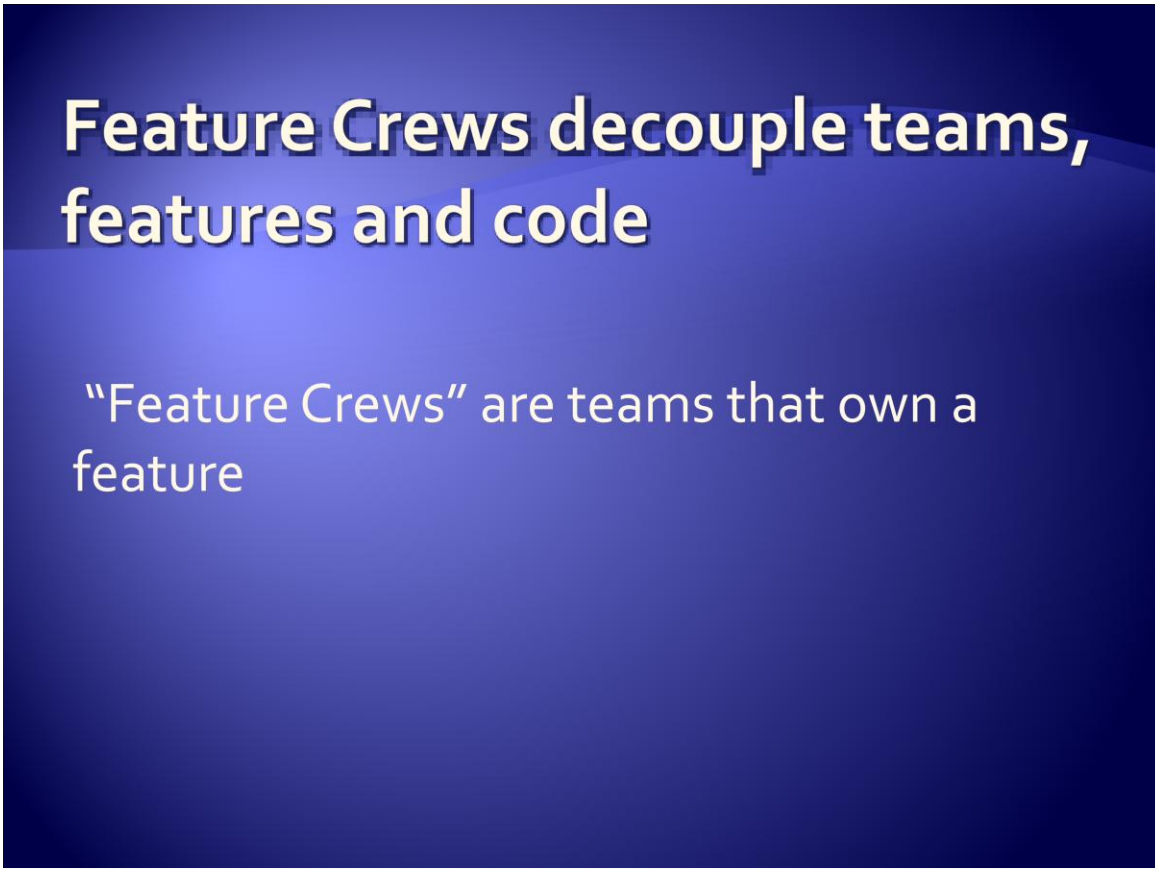
A feature crew is a **small interdisciplinary team** charged with delivering a single (small) feature as part of a larger end-to-end user experience within the product (sounds familiar right?).

Typically our **crews consisted of** a couple of developers, 2-3 testers and a Program Manager, who played the role of the customer and to some extent the team facilitator.

The Feature Crew model itself **doesn't really prescribe *how*** the team builds software – although it does make some recommendations – it focuses on the end result of the crew.

Feature Crews deliver their feature, or sub parts of it, over **one or more five week iterations and integrate it** into the main product.

Feature Crews deliver features that **meet a common set of requirements** "done", these are called Quality Gates.



Feature Crews decouple teams, features and code

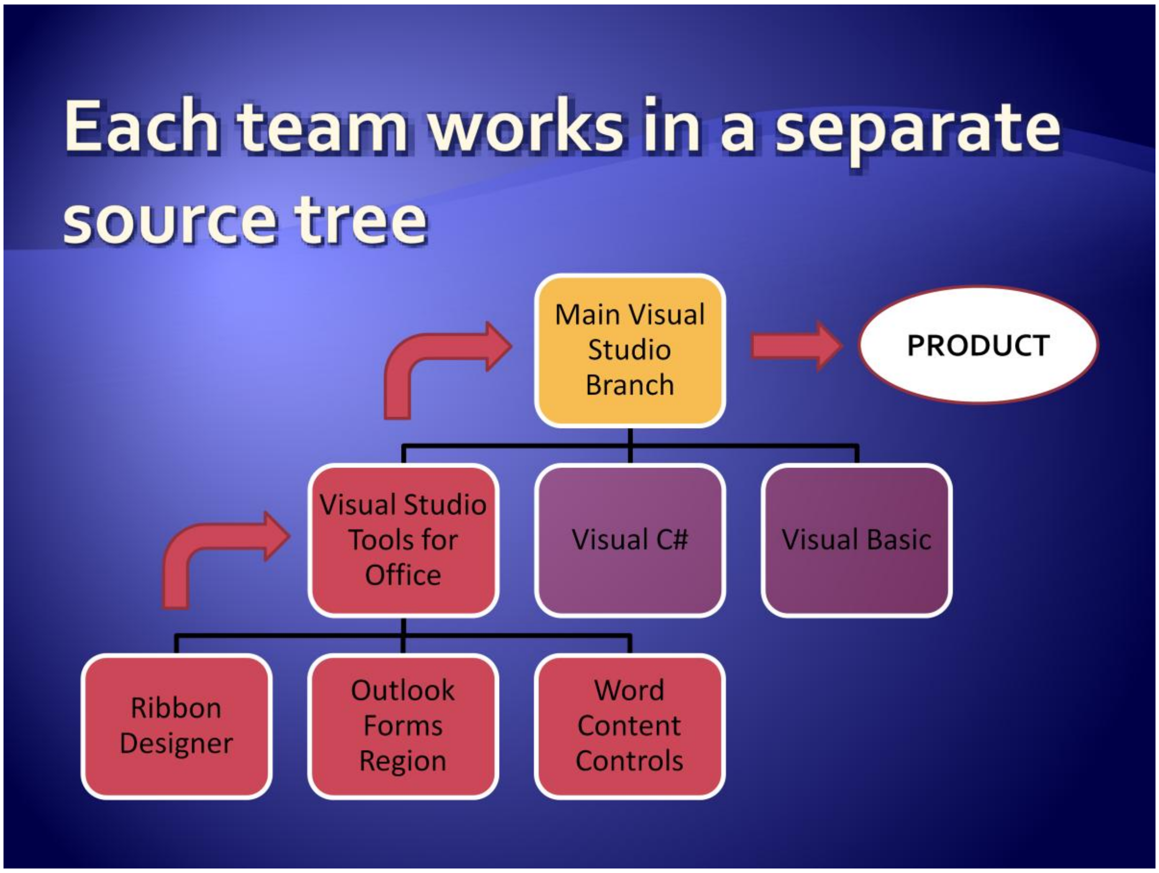
"Feature Crews" are teams that own a feature

DECOUPLE… (PARTIAL) OWNERSHIP…

In order to scale to a very large project the Feature Crews model attempts to **decouple teams, product features and code**.

Each crew works on a **single product feature** the team is **responsible** for the whole feature which is itself part of an **end-to-end user experience**. The **crew's product owner is free to make changes** to the feature provided this does not result in significant changes to the end-to-end experience.

Our PU chose to handle **experience changes by having crews surface them to the management team** at a weekly meeting.

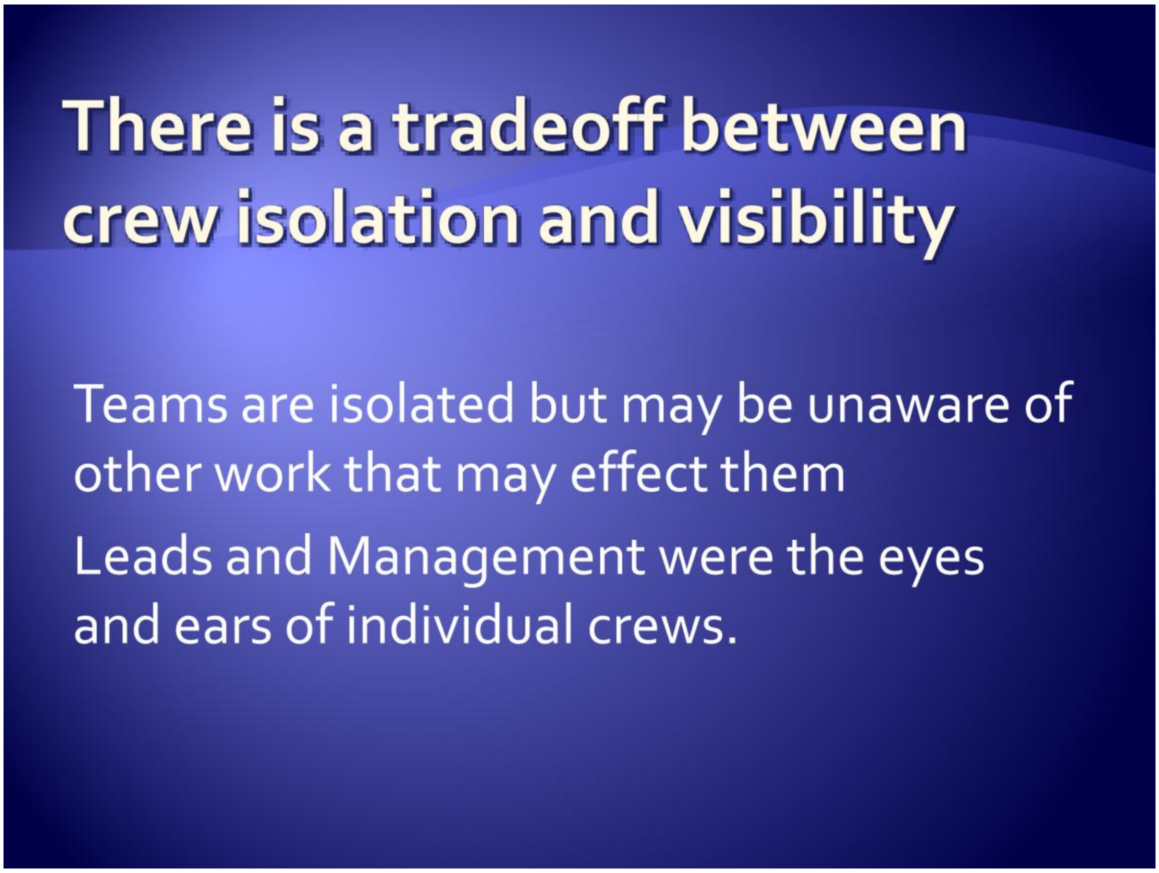


Each team works in a separate source tree

ISOLATION OF SOURCE… HOW THE RIBBON DESIGNER CREW INTEGRATES

Each crew is **further isolated** by working in its own private source code tree or "branch". When a piece of work is complete it is integrated into the parent branch and ultimately into the main product branch from which the product is build

CLICK! In order for the **Ribbon Designer crew** to get their feature into the main branch they must…



There is a tradeoff between crew isolation and visibility

Teams are isolated but may be unaware of other work that may effect them

Leads and Management were the eyes and ears of individual crews.

FLEXIBILITY… VISIBILITY… LEADS

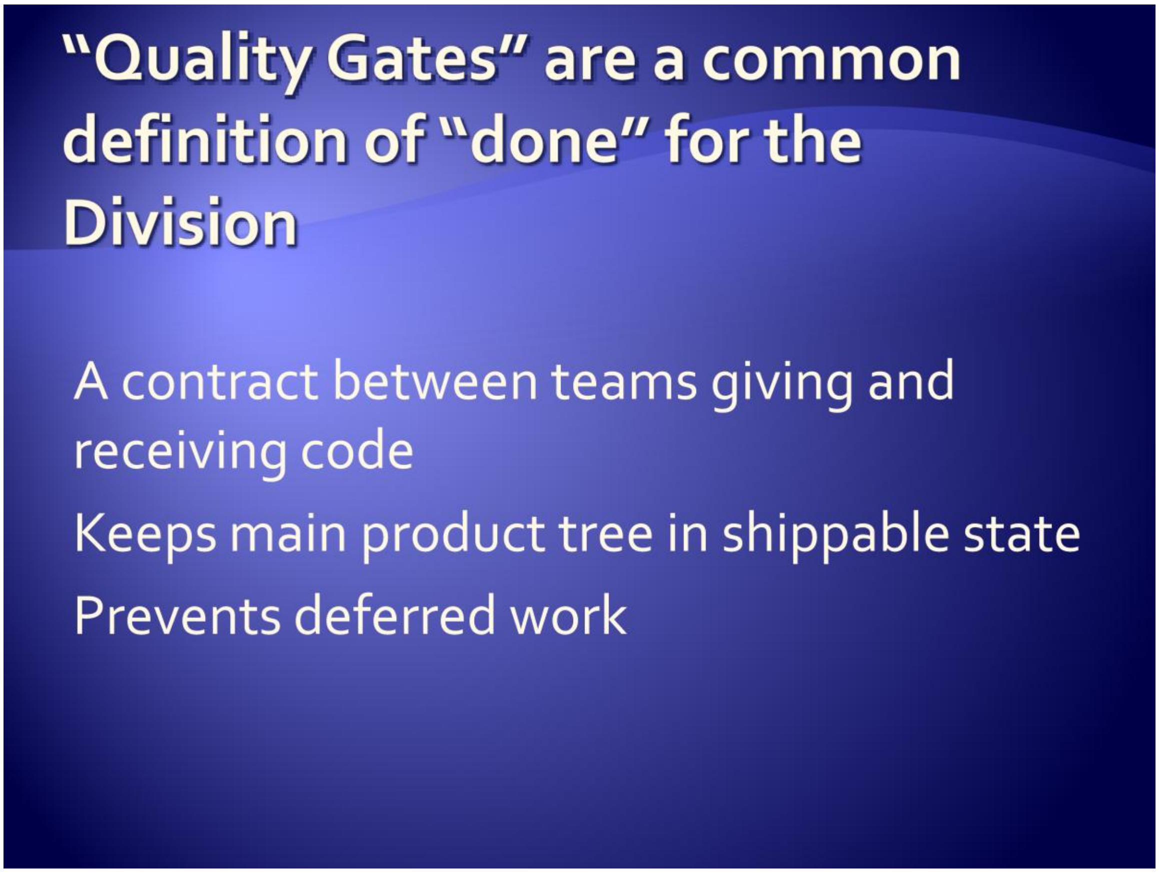
Decoupling teams, features and code gives teams the **flexibility** to create features as they see fit.

It creates **visibility issues** as teams don't really see what other teams are doing, especially for features that are dependent on other features in the same PU or even features provided by other PUs.

People: One of the **roles of management** is to **watch out** for breaking issues and give individual crews the heads up. This ties in the Diana Larson's approach to managers in an agile organization, **managers as team facilitators**.

Leads took part in the crew's daily standups and also attended a weekly management meeting to track not only progress but also **cross-features issues** that might impact individual crews.

Process: Define a **common definition of "done**" – the "Quality Gates" so that features and code meet a common standard before they move around the tree.



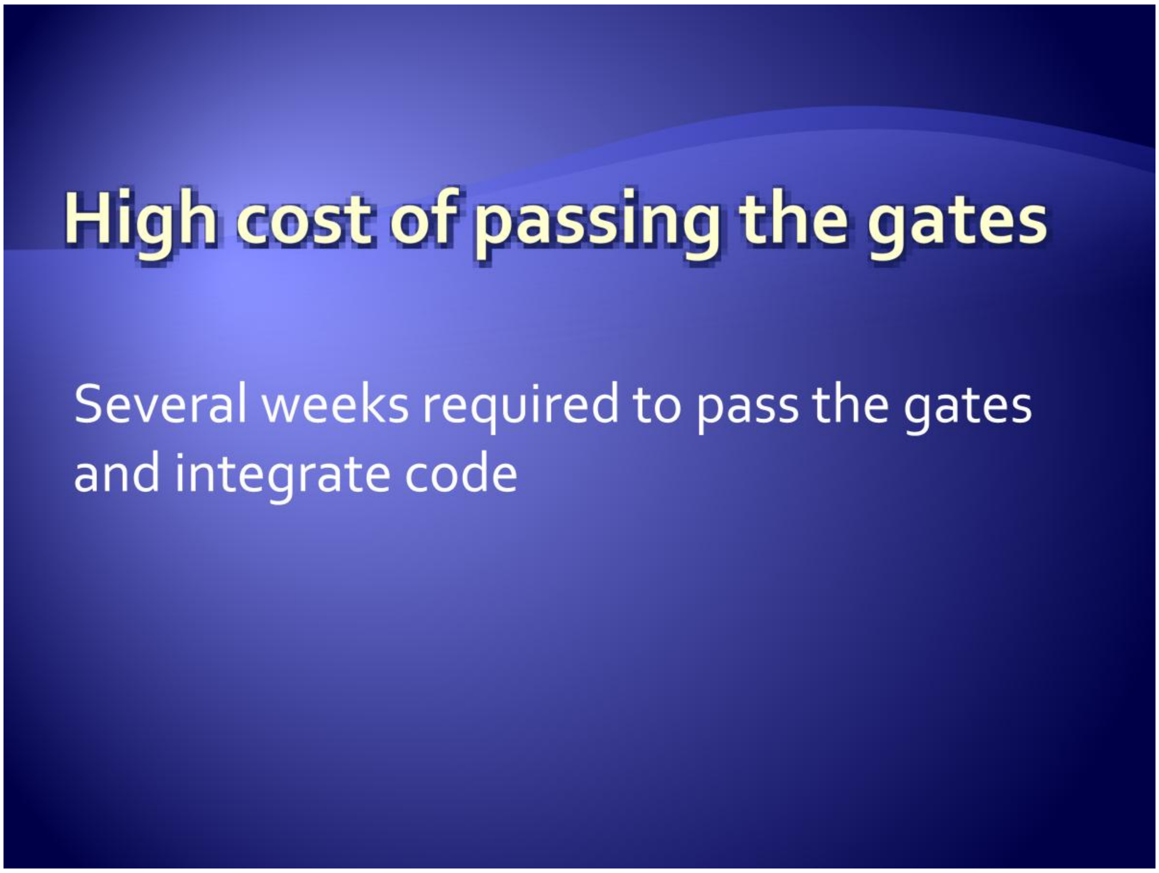
“Quality Gates” are a common definition of “done” for the Division

- A contract between teams giving and receiving code
- Keeps main product tree in shippable state
- Prevents deferred work

16 GATES… A CONTRACT… NOT DEFERRED WORK

So what do I mean by Quality Gates? A series of **16 tests** or **criteria that define if a feature meets the “done”** bar:

- Testing – unit and acceptance
- Feature Bugs Closed
- Performance
- Test Plan
- Code Review
- Functional Spec
- Documentation Plan
- Development Spec
- Security
- Samples
- Static Analysis
- BVTs Passing
- Setup Verification
- Test Matrix
- Code Coverage
- Localization



High cost of passing the gates

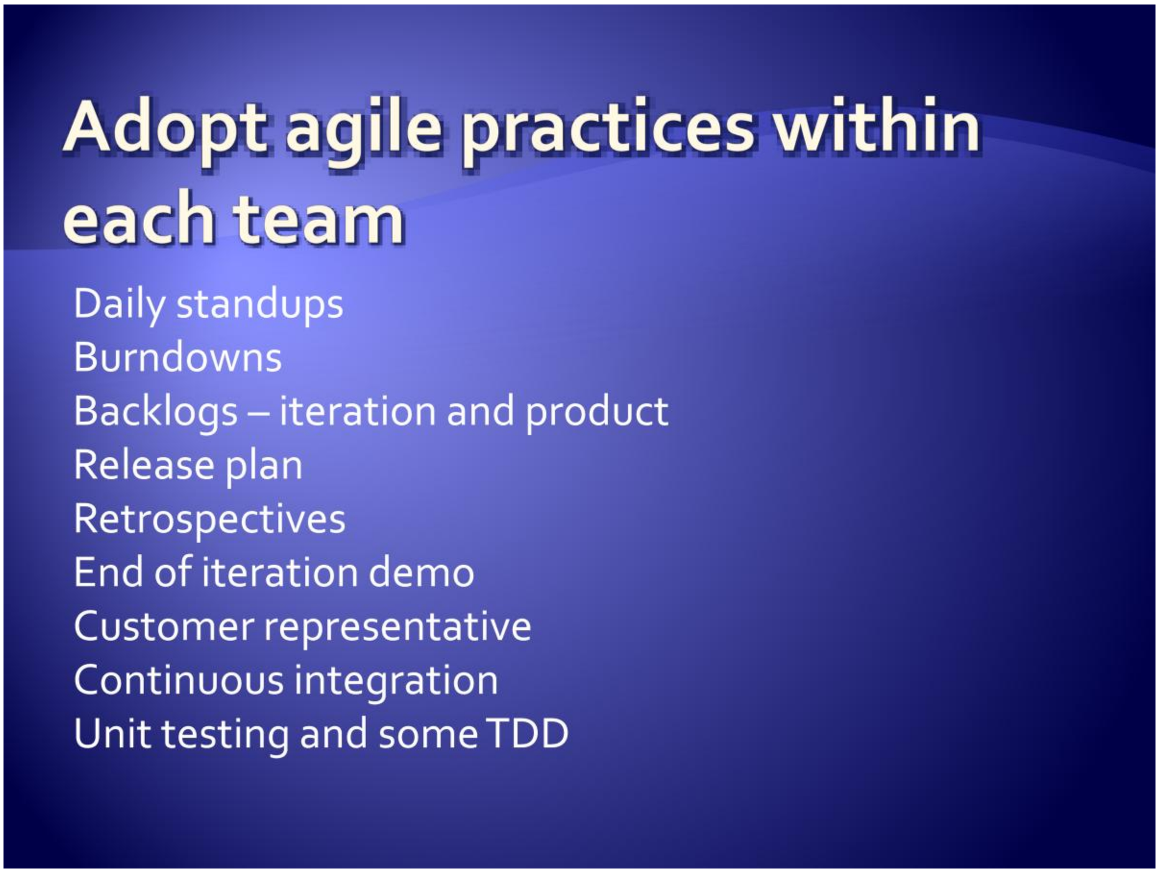
Several weeks required to pass the gates and integrate code

SEVERAL WEEKS… WHOLE DIVISION… TESTS FOR ALL PARTS OF THE PRODUCT

The Quality gates are **a comprehensive set of requirements** across the **whole division**.

The **cost of passing all 16 of these gates** was quite **significant** but the goal we to ensure that a product was **buildable from the main branch *all the time***.

Teams might take **several weeks to pass the gates** and integrate their changes into the main product branch. This meant that **crews were not able to integrate their changes at the end of each iteration**.



Adopt agile practices within each team

- Daily standups
- Burndowns
- Backlogs – iteration and product
- Release plan
- Retrospectives
- End of iteration demo
- Customer representative
- Continuous integration
- Unit testing and some TDD

SCRUM… NOT SCRUM…

Much of what we did was taken from **Scrum.** We found lots of value in these practices.

- Iteration cadence – five week iterations
- Daily standup
- Burndowns – tool for publishing burndowns and iteration backlogs to the web
- Backlogs – product and iteration
- Release plan
- Retrospectives
- End of iteration demo
- Customer representative present (the PM on the team as a proxy for the customer)

Conclusions

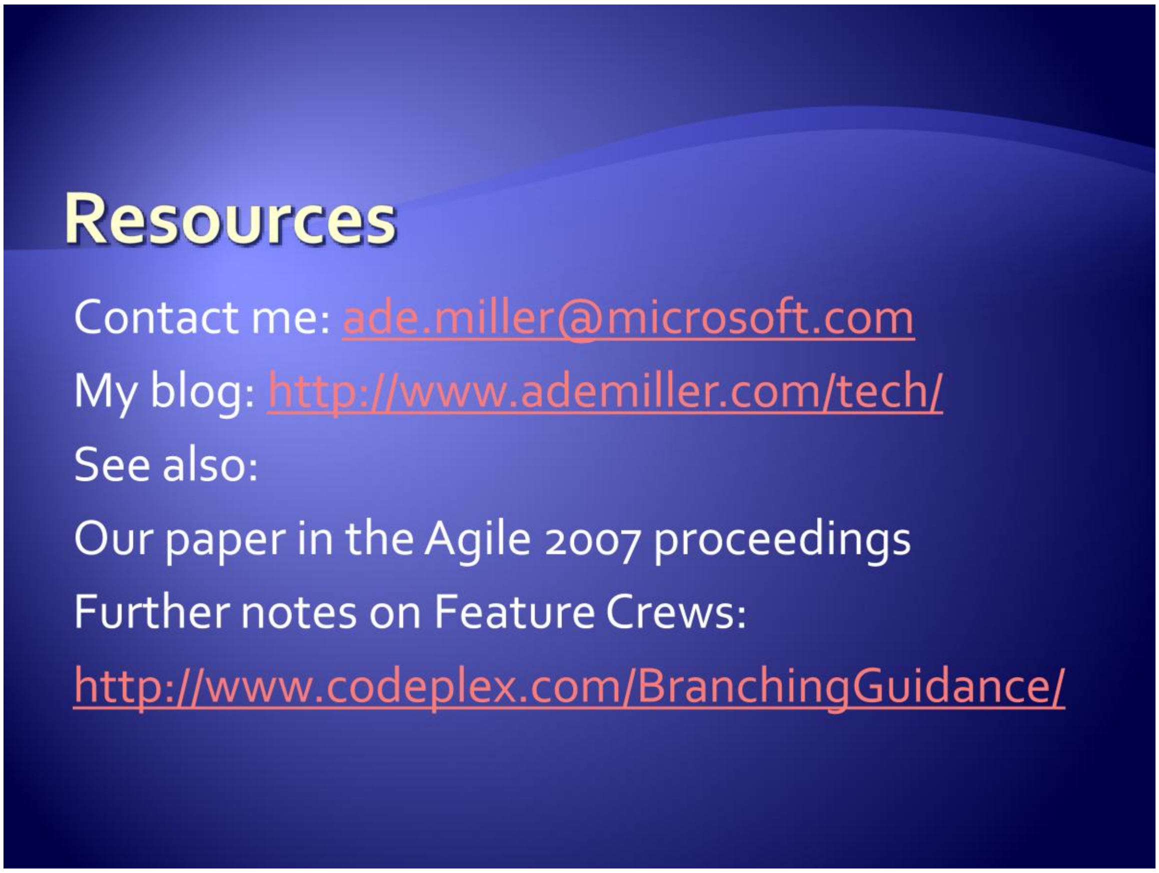
VALUE… SIZING AND BALANCING… VISIBILITY… INTEGRATION… TRANSPARENCY

Lots of **value in many agile practices**. Some barriers to some of them, for example pair programming.

Sizing and balancing crews can be hard, larger teams struggle with standups, smaller ones find quality gate work overwhelming.

The **Feature Crew model sees crews as ephemeral**, only lasting the lifetime of the feature's development which might be a single iteration. We aligned a series of features around an end-to-end user experience and kept the same crew together for the whole product cycle. This led to some people balancing issues as the loading on different features changed.

Visibility between crews was also an issue both in terms of breaking code changes in the codebase and getting interim updates to other crews.



Resources

Contact me: ade.miller@microsoft.com

My blog: http://www.ademiller.com/tech/

See also:

Our paper in the Agile 2007 proceedings

Further notes on Feature Crews:

http://www.codeplex.com/BranchingGuidance/