Build a "Development for Non-Developers" Course

Ivan Sanchez
XPConf 2018

Tribes in Software Development

Before we talk about building training courses, let's talk about "tribes" in software development. That's the main reason for this talk.

My background is coding, but in the early 2000's, eXtreme Programming changed the way I saw my role in software teams. For the first time I was:

- Working directly with the customer and the users.
- Being part of a small team, where people could not be limited by their job titles
- Experiencing a highly collaborative work environment
- Feeling as part of the customer's team as I was part of the "dev" team.

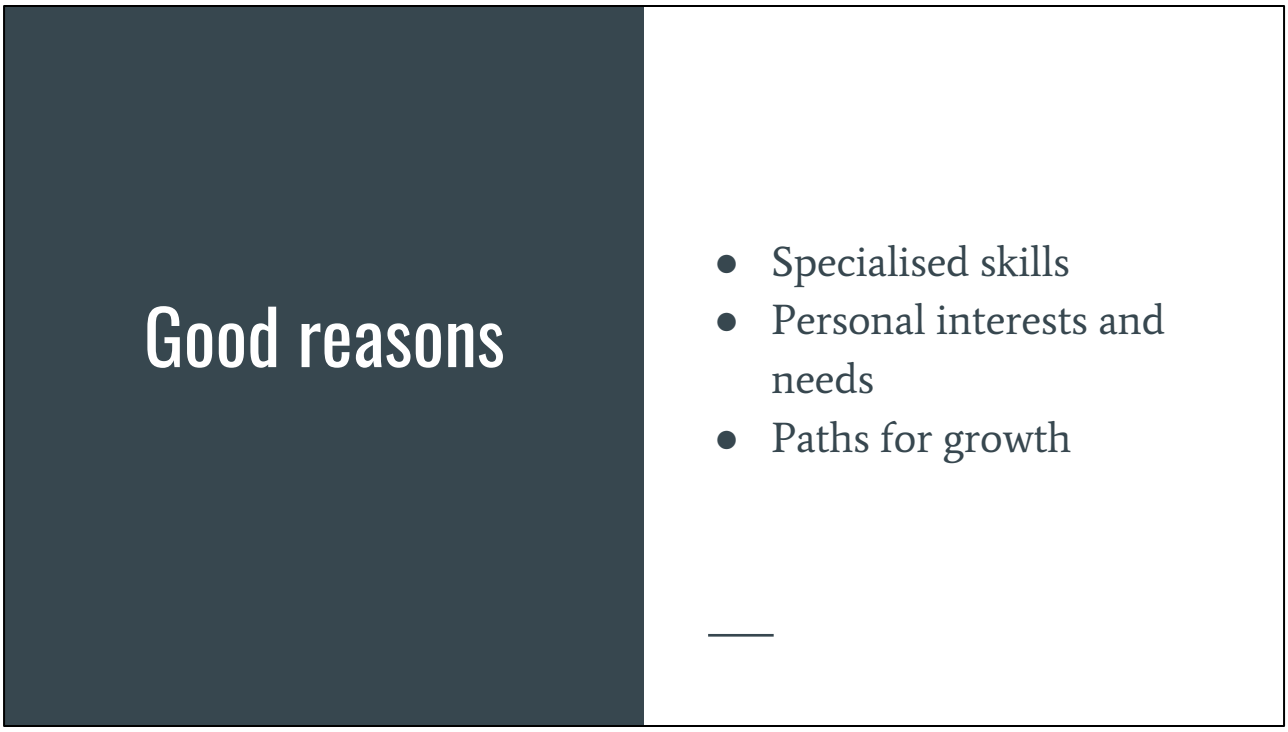
Tribes?

A small group of people, with shared interests and habits, and strong sense of separation from other groups

Here's our definition of a tribe for today.

How many people work in cross functional teams? Let's think about the different "jobs" in software projects: Programmer (frontend, backend, mobile, operations, dev in test), Business Analysts, User Experience Designers, Graphic Designers, Testers, Data analysts

Any others?



Separation of roles does happen for good reasons:

- Some skills in software development require a lifetime to become good.
- Different things motivate different people. People also have different needs and constraints.
- Both people and companies benefit from defining clear paths for career growth.

It’s important to remember, though, that this separation does not come for free.

Roles are different than tribes

Distinguishing between those two concepts is important. A role is something a person does in a particular situation. A tribe is something people feel part of.

Much of the friction inside software teams come from using particular roles as a way to separate its different tribes.

Why does that happen?

Social Identity Theory

The sense of who people are is based on the groups they're members of.

(Based on Henri Tajfel and John Turner work in the 1970's)

Some relevant characteristics

1. Groups are formed arbitrarily
2. Boundaries are enforced by inclusion/exclusion behaviours
3. Group loyalty trump overall results

The Social Identity Theory explains many of our behaviours, and some of its characteristics are especially relevant for software teams.

First, it's worth noticing that usually the way we group ourselves is primarily random. Anything can be used as a way to create a new group (or "tribe"), and once that happens, all the other characteristics will follow. The vast amount of different jobs in software teams is an excellent example of this. Most of those exist mainly by accident.

Second, once groups are determined, people will show behaviours to enforce their identification within their groups, as well as actions to exclude people. That's because our social identity (and, in consequence, our self-esteem) is bound to the group, and we will unconsciously act to protect it. The most common way that takes place in software teams is by having strict rules of who's allowed to do certain things.

Finally, maintaining the identification with the group takes precedence over external expectations of results. That explains why is common for people to prefer (even if unconsciously) their project to fail rather than cross the boundary of what's expected based on their job titles.

UML Artists

An example of “tribe” that was very strong in the 90’s and early 00’s was the System Analyst.

This role had clear boundaries: it was all about translating a specific domain problem into painfully detailed specification (mostly using UML diagrams). Those specs would then be handed over to programmers whose jobs were mainly to translate those specs into working software.

They would not code, as that was not part of their jobs. They would also discourage programmers from talking to the business people, as they were the ones who were supposed to understand and design the solution. Nowadays we accept that this is a very ineffective way to deliver software, but at the time it was considered “the right way”.

We can see the three characteristics of “tribes” here: random group, inclusion/exclusion behaviour and group loyalty trumping overall results.

If a role can only be performed by certain people, you got two tribes

That’s a rule of thumb to identify the different roles in software teams.

It’s worth highlighting that the reason why people can/can’t perform a role is irrelevant. No matter if it’s a company policy, a lack of skills, or just by accident, all the characteristics of tribes will start to appear.

Reducing negative tribe effects

- Fluid roles
- **Share**
  - Responsibility
  - **Knowledge**
  - **Skills**
- Culture of inclusion

Although tribal behaviour is always present to a certain degree, there are things we can try to reduce its negative effects.

Fluid roles are about being removing as many restrictions to who can perform specific tasks.

Another way is by sharing as much of the responsibility as possible. That can be achieved if (on top of fluid roles), people have enough shared knowledge and skills so they can work closer together. Cross-functional pairing and pair rotation also help here.

And for the cases where only certain people have the ability to perform a task, a culture of inclusion can help to reduce the barriers between the different tribes. A great example of this is when meetings are open for the whole team, as opposed to only specific people. Mobbing sessions open to the entire team are also a good way to help people feel included.

For this talk, we'll concentrate on how to share knowledge and skills across different disciplines.

Knowledge and skill sharing via cross-functional training

That should really be the title of this talk (oh well, too late).

Knowledge is about understanding concepts and principles. Skill, on the other hand, is the ability to apply the knowledge in particular situations.

In the past years, I've been trying to use teaching as a way to mitigate the adverse effects of tribal behaviour. That kind of training tends to be short, collaborative, practical sessions, focusing on real-life problems and aiming to give people actionable skills or at least knowledge that can increase collaboration on their projects (which is an actionable skill on its own).

Picking the right topics

We identify learning opportunities in two ways.

1. By paying attention to people's interactions:
    a. Terms that are unknown to other people
    b. Practices that people are skeptical about
    c. Processes that are unclear or confusing
    d. How people perceive others' routine
2. By looking at actions that can only be performed by certain people (focus on low hanging fruits):
    a. Build and publish a simple web page
    b. Run the application locally
    c. Write test scenarios
    d. Make changes to text
    e. Discuss architecture
    f. Experience technical debt

Facilitated Training

Teaching is also a skill, and probably one of those that take a lifetime to master. It can also be a tribe on its own, so it's important to be mindful of that.

There are a few practical rules that can help fitting this kind of training into a team's routine:

- Focus on giving people new skills rather than just knowledge
- Keep it short
- Make it practical rather than reading/listening/watching (design experiences)
- Respect other people's pace and needs.
- Incentivise people to talk/teach/build

A good book to get started with is Training from the BACK of the Room. Another one is The Surprising Power of Liberating Structures (about facilitation in general).

There are many opportunities to practice teaching. Apart from your own team, local meetups can also be a good starting point.

Example: Web Development 101

- Web fundamentals
- HTML and CSS basics
- Version Control
- Continuous deployment

- Example of a real course
  - 8 hours in total
  - Mix of materials (slides/links for people to read) and group/hands-on exercises with tutors.
  - Designed to use the real tools developers use:
    - Basic networking (HTTP/browsers/DNS/servers/IP/hosting)
    - Sublime Text
    - Git
    - GoCD and build automation
- Open to people of multiple teams
- Attended by people from variety of tribes, including:
  - Technical support, who benefited directly from understanding more how the systems get built.
  - Subject matter expert, who wanted a taste for a potential career change
  - Project managers, who wanted to communicate better with developers
- Great feedback
- Run multiple times. Various developers volunteering to help
- Considered to become part of HR training for the whole company

From Team to Tribe

This kind of training helped in many fronts:

- Better communication (**mutual understanding**)
- Sparked desire of further learning
- Gave people confidence to act beyond their job titles (**special powers)**

Most important: **they are baby steps towards turning the team itself into a tribe**.

Thank you - Questions?

Ivan Sanchez

@s4nchez

ivan@gourame.com

Has no question? Steal some!

- Should everyone in the team be able to code?
- How long does it take to design a course?
- How does something like DevOps fit into this picture?
- Where can this go wrong?