

Software patterns for **productive teams**

Radoslav Georgiev, @Rado_g, **EuroPython 2019**



3rd EuroPython for me ✓

Goal of this talk:

Be **practical, pragmatic & provide value.**

Goal of this talk:

“Aha! We should try this”

moment.

Context:

I'm CEO of **HackSoft** - a **software development company**.

I'll provide **cherry-picked**
examples on the topic from **our**
experience.

Agenda

1. Team leader's perspective.
2. Software development.
3. Features.
4. Explicit is better than implicit.

Team leader's **perspective.**

What I care for, when I'm a team lead.



Team leader's perspective

1. **Productivity** (we need to ship features)
2. **Confidence** (we need to keep the product stable)
3. **Independence** (make their own decisions)
4. **Well-being** / stress of team members (burnout is bad)
5. **Less context switching** for everyone (don't break the flow)
6. **Someone being blocked** by something (feeling unproductive)
7. **Morale** (overall feeling)

Constant **regressions**.

Constant regressions

Problems:

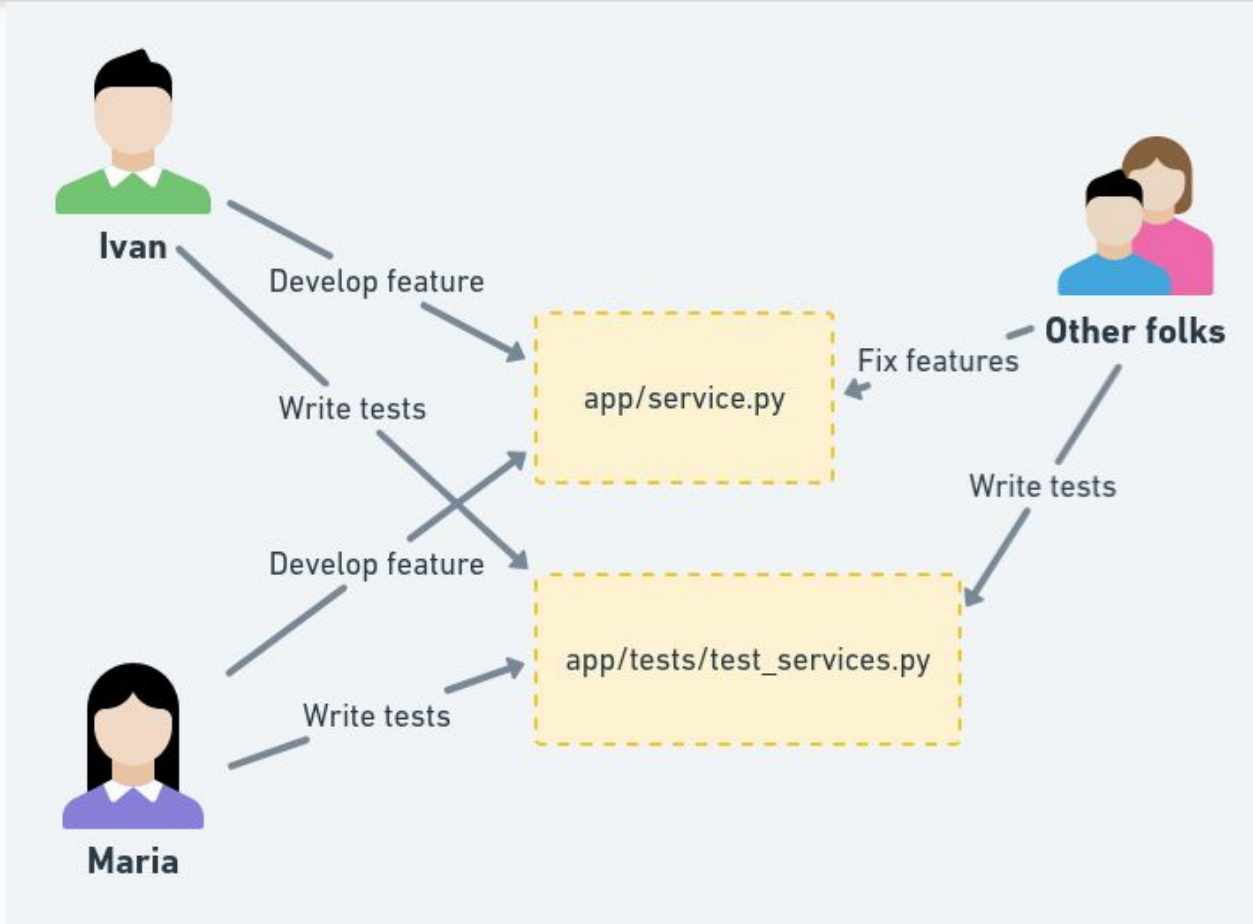
- **Production is constantly broken** / something's not working.
- **Quick "proof of concept"** is being turned into production-ready version.
- **Can decrease team morale.**

Possible solutions:

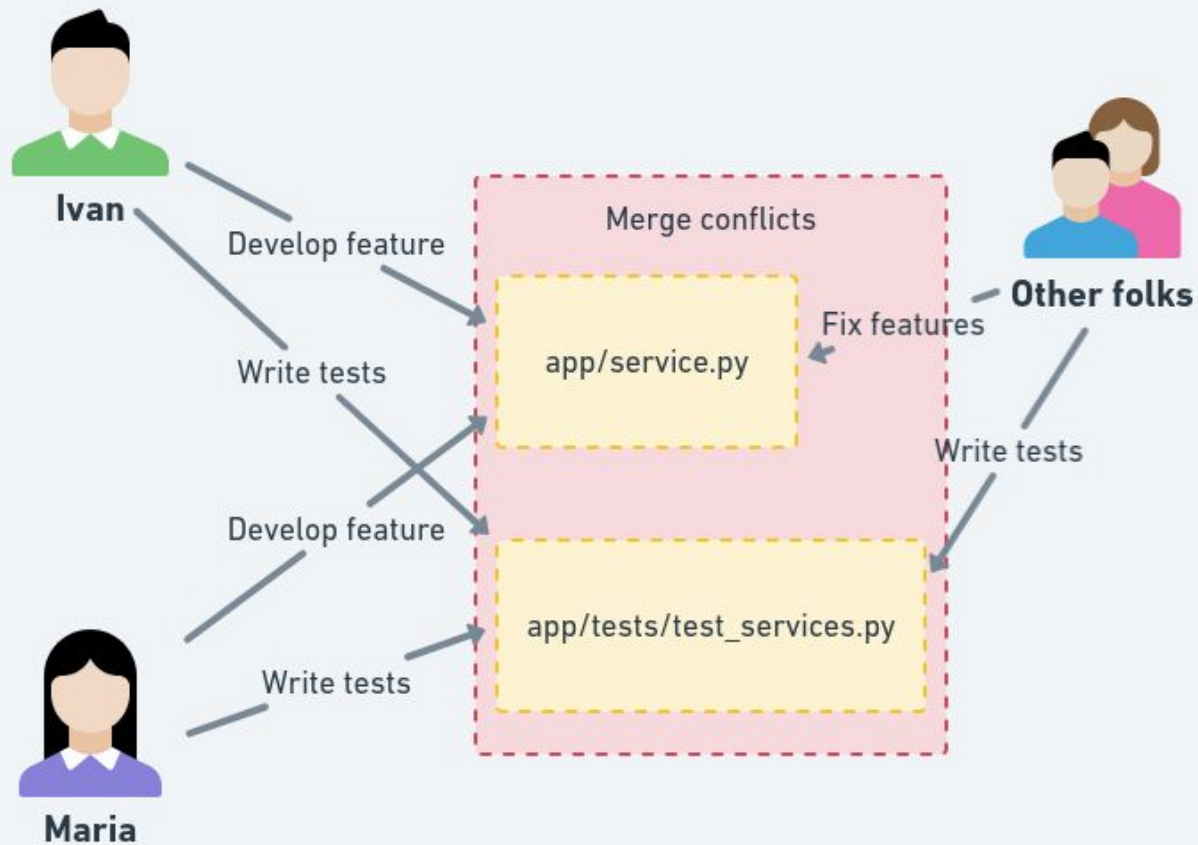
- **Stop all feature development** until software is stable again.
- **Add CI & run tests** / lints if you don't have one.
- **Add a staging environment** & don't test on production.

Constant **merge conflicts**.

Constant merge conflicts



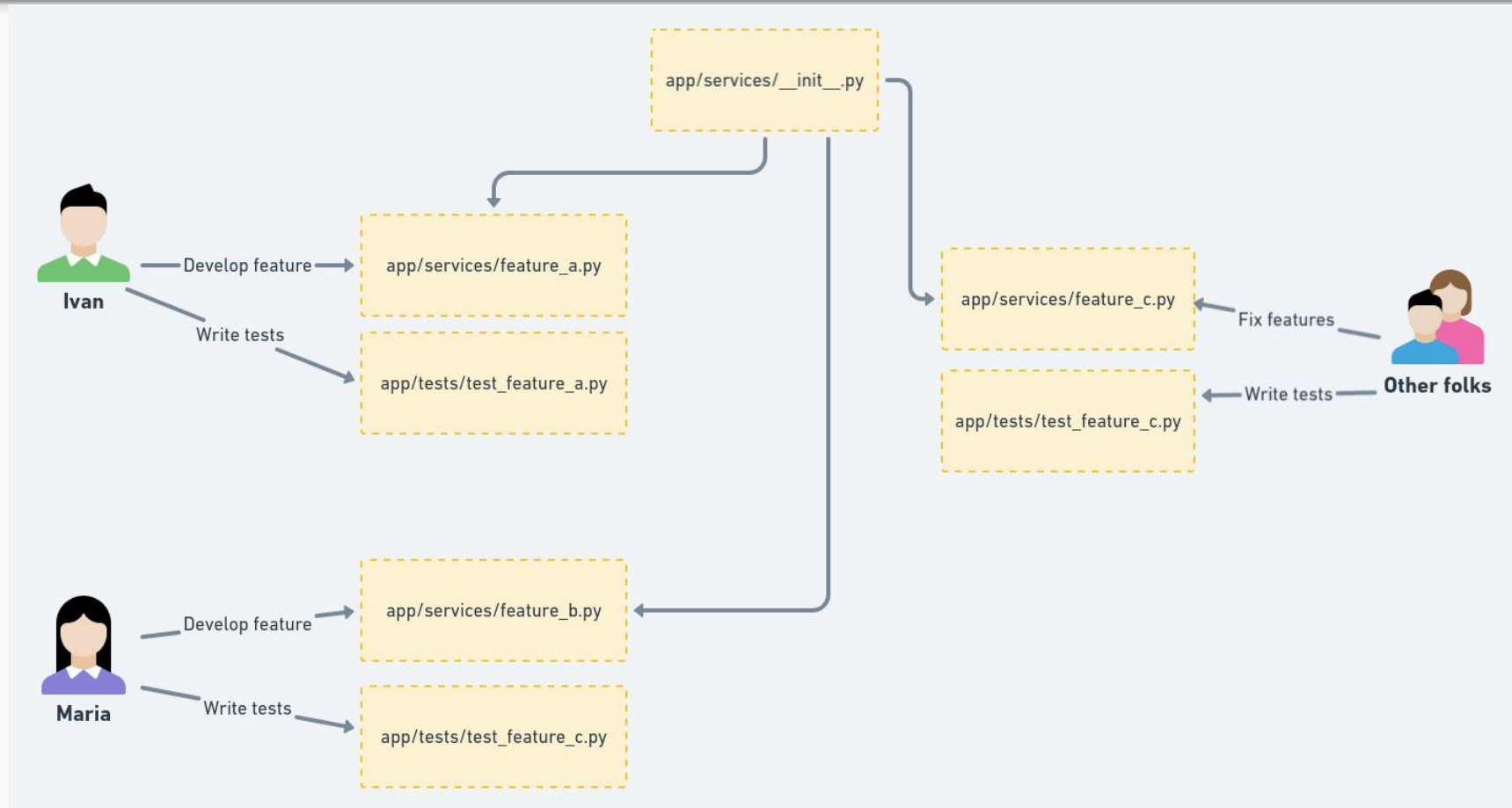
Constant merge conflicts



Constant merge conflicts

- **Split** python modules by domain.
- **Split** big test files into test file per thing that you are testing.
- **Constantly watch for merge conflicts** - this means something's not right.

Constant merge conflicts



Local setup.

A specific type of hell.



Local setup - accounts

- **Devs can't even start working** on a feature if they can't get something running locally.
- **Make sure everyone has an account** / access / keys for everything needed. Do that before they need it.
- **Keep a spreadsheet of accounts & 3rd parties.** Easier to track & manage.

Local setup - accounts

1	Dev	Service	Access type	Hac access	Comment
2	Radoslav Georgiev	Sentry	Account	Yes	@hacksoft email
3	Radoslav Georgiev	AWS S3	Keys	Yes	Keys exported upon creation & sent over Slack.
4	Radoslav Georgiev	Heroku Staging	Account	Yes	@hacksoft email
5	Radoslav Georgiev	Heroku Production	Account	Yes	@hacksoft email
6	Krasimira Badova	Sentry	Account	Yes	@hacksoft email
7	Krasimira Badova	AWS S3	Keys	Yes	Keys exported upon creation & sent over Slack.
8	Krasimira Badova	Heroku Staging	Account	Yes	@hacksoft email
9	Krasimira Badova	Heroku Production	Account	Yes	@hacksoft email
10					
11					

Local setup - documentation

- **Relentlessly document** everything related to local setup.
- **Test it** and keep it updated.
- GitHub / Confluence / **whatever is working for you.**
- **Onboarding new people is your final test.**

Local setup - setup scripts

./setup/bootstrap.sh # get a clean & ready to go local dev environment

./setup/xero.sh # Setup additional 3rd party

./setup/gocardless.sh # Setup additional 3rd party

./setup/everything.sh # Setup all 3rd parties in a clean local dev environment

Speed of tests.

Very important & often overlooked.



Speed of tests

- If you are working in an environment with small PRs, **merge often & deploy often** ...
- ... and your tests are taking **10 minutes** to run on CI ...
- **you are not going to feel very productive** & you'll often find yourself waiting on CI.

pytest-xdist / **py.test -x -n 4**

Optimize your tests. It pays off!

Features

Make sure everyone are on the same page with this.



Feature descriptions

- If the features are described poorly, **people are going to build the wrong thing.**
- **Clients often don't know the exact details of the things they want**, so ask them a lot of questions!
- **Make sure everyone on your team actually reads the feature descriptions** fully, before starting to work.

Feature blocking



Maria

Working on →

Feature A

Relies on →

Common piece of
abstraction / Feature C



Ivan

Working on →

Feature B

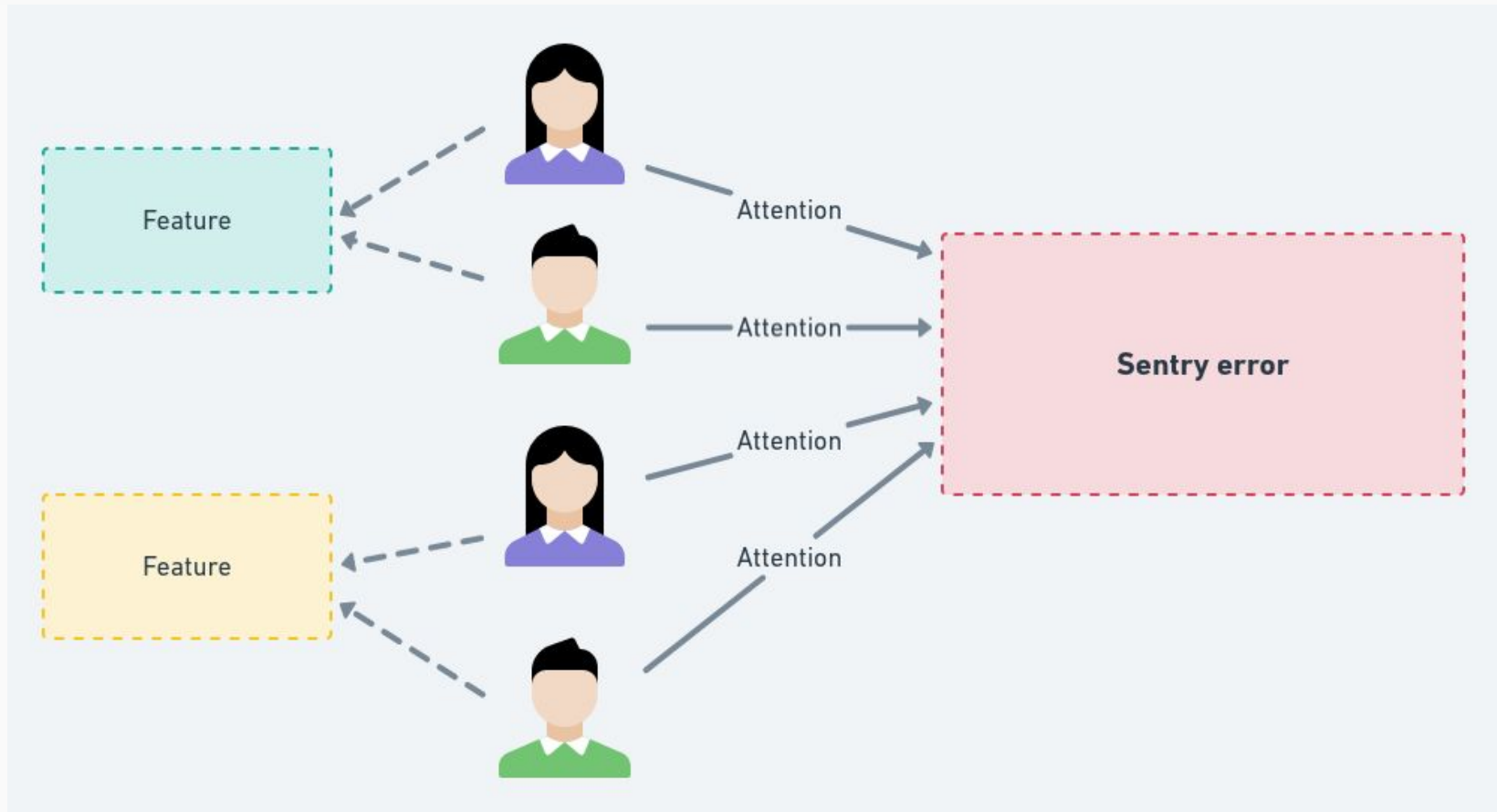
Relies on →

Feature blocking

- **Pair people around shared parts**, so they are on the same page.
- **Identify such scenarios quickly & resolve them.** Otherwise work is going to be deleted / undone.
- **Such scenarios may cause conflicts.**

Explicit is better than implicit.

There's a bug!




There's a bug!

- **Have an explicit “firefighter” for the week.**
- **Rotate everyone on that position**, each week.
- **This “firefighter” is the first responder when there's an issue.** A lot of the issues can be resolved quickly, without sacrificing all of the team's attention.

Explicit **Git** & **GitHub** workflows.

No matter what you use.



Refactoring PRs separated from feature PRs.

Easier to read, easier to catch problems.



Team rules.

Team rules

- Write down everything from **“This is how we do things here”**.
- **Better visibility** at team dynamics & **explicit expectations** from everyone.
- A great tool for **onboarding new people**.
- Revisit & update!

Have an **explicit team lead.**

Otherwise, there is going to be an implicit one.



Have an explicit team lead.

- **Know who the leader is.** That's the person making the calls when needed & the person who's responsible for the team success.
- **Team leads should focus on enabling their teams do their job well.** If this means less coding - then so be it.
- **We rotate team leads every week,** so everyone knows what it's like to be on that position. Gives perspective.

Conflicts.

You cannot avoid them, but you have to handle them.



Conflicts

- Catch early & try to **overcommunicate** with all parties involved.
- Read books on management & leadership. **Use your gut feeling.**
- Have **perspective** on what's important.
- Beware of **toxic people & malicious obedience.**
- **Fire**, if necessary (easier said than done)

Adapt.

Adapt

- If something's not currently working - **understand why & make changes.**
- Establish processes but **don't follow them blindly.**
- Teams are **different** (people are **different**).
- **Things change.**

Ask your developers for pain points. They will tell you.

And do something about them.



Thank you. **Questions?**

Radoslav Georgiev, CEO of HackSoft
@Rado_g
radorado@hacksoft.io