TOP 5 ELIXIR MISTAKES

At Erlang Solutions, we perform Elixir code reviews for a living. We look at the code of everyone from startups and Fortune 500 companies, and have noticed a few patterns.

Based on our years of experience reviewing Elixir code, here are our top 5 Elixir gotchas!

1. Too much config
2. Too many agents
3. Too many macros
4. Simplicity slipping away
5. Do I need a process?

Learn more about the work we do with Elixir and Phoenix.

Erlang SOLUTIONS

TOO MUCH CONFIG

1

Abusing config files, leading to code that's not flexible at runtime.
This is particularly relevant for library authors. It's preferable to provide public APIs which accept config options, so that we maintain flexibility.

SYMPTOMS

Your system is configured properly in development and tests, but the configuration looks odd in production, especially when using releases. This probably means that it's pulled from application config at compile time, and it can't be modified later at runtime.

WHAT TO LOOK FOR

In libraries, look for all occurrences of Application.get_env/2 and friends.
In applications, search for places where result of Application.get_env/2 and friends is assigned to a module attribute, like:
@attr Application.get_anv(:my_app, :config_var)
Remember that module attributes are resolved at compile, so it's better to move the call for configuration variable to function body.

ROOT CAUSE

It's logical that developers reach to Mix.Config when it comes to configuration - the location of these files (the config / directory) suggests that it's the way to go.

HOW TO MOVE FORWARD

In libraries, allow users to pass configuration with every function call (you can optionally use Mix.Config but at least allow the caller to override it by explicitly passed configuration).
In applications, refactor code which uses configuration pulled in at compile time.

Erlang SOLUTIONS

TOO MANY AGENTS

2

It's easy to reach out for an Agent to store some data. This comes at a cost though, which is the introduction of a bottleneck. It's always important to think about how the introduction of a new process may affect the overall system performance.

SYMPTOMS

If you notice that a part of your application is slow, even though the work is distributed among concurrent workers (like, for example, Phoenix request handlers), this means that there is a synchronisation point somewhere and the workers can't go faster than that. A good metric to check is to see whether increasing number of workers doesn't result in a speedup.

WHAT TO LOOK FOR

Look for named processes! (usually name is provided via :name option to start_link functions). If a process is named, it means that it's a singleton in the whole system and needs to be designed with care to not become a bottleneck. Also look for places in code where one process, e.g. a GenServer, spawns a dedicated Agent which only it has access to. It most likely means that the Agent's state can be held directly by the GenServer.

ROOT CAUSE

An agent promises to be a simple solution for sharing state between processes, quoting official Elixir documentation: "Agents are a simple abstraction around state." They have a clear and pleasant-to-use functional interface and they abstract away the process of handling messages.

HOW TO MOVE FORWARD

If there is really a need to share state between processes, ETS is the way to go. It was designed exactly for this use case and has unmatched performance on the BEAM.

Erlang SOLUTIONS

TOO MANY MACROS

3

While macros are powerful constructs that can help reach usable DSLs, they can often be misused. Can a macro be replaced by a function call? What's the trade-off between the two approaches?

SYMPTOMS

Long compilation times. Macro usage has a huge impact on compilation time and introduces unnecessary dependencies between the modules, slowing down Elixir's parallel compiler.

WHAT TO LOOK FOR

First, search for definitions of __using__ macro. If the only thing they do is injecting aliases or imports, it's probably not worth it because it introduces many dependencies between the modules.

ROOT CAUSE

Macros are very powerful and appealing. The Elixir team have created a macro implementation arguably better than anything seen elsewhere, which can be very tempting to use in a lot of situations.

Macros can be used to abstract away complex functionality and provide programmers with the ability to use their own domain specific language to describe generic, but repe system behavior. They can help to eliminate boilerplate code, for example when implementing crosscutting concerns such as transaction handling on functions or security checks for REST services.

HOW TO MOVE FORWARD

Functions!
Most macro usages can be replaced by function calls. Use macros sparingly, only when you absolutely need to modify the code at compile time.

Erlang SOLUTIONS

SIMPLICITY SLIPPING AWAY

4

Finding it hard to stay functional? Functional programming promises a way to tame complexity. Don't let it slip through your hands… functional programming isn't something that's enforced by a language as much as it is a mindset.

SYMPTOMS

A code base that is hard to get to grips with. What's it like when a new team member comes along to a project? How long do they have to riffle around in a code base before they get to grips with it enough to deliver value? If the answer is too long, given the project size, then this can indicate that simplicity is slipping away.

WHAT TO LOOK FOR

Pure functional code (value transformations) interleaved with side effects (including message passing). That means a clear distinction between the functional and the operational where there is a concentration on business logic through a functional approach, the substance of what it is that your systems do, with a delivery of that value through operational abstraction.

ROOT CAUSE

Developers are used to working this way. Other programming paradigms don't put such emphasis on separating pure and impure code. They can bring baggage from other paradigms or models and are too slow to adapt or never do so at all. Adopting Elixir isn't as much about adopting a language as it is about adopting a mindset: therein, in part, lies the beginning of systems that are accidentally complex.

HOW TO MOVE FORWARD

Extract pure functions and data structures to separate modules. Move side-effects to the edges of your system and let those pieces be the clients of the functional parts.

Erlang SOLUTIONS

DO I NEED A PROCESS?

5

When building an Elixir system at some point every developer will eventually face a question: "Do I need a separate process for the activity I'm implementing?" Approaching this problem carelessly may result in a system with too many or too little processes.

The thing is to have the processes in balance. The Erlang VM is known for being able to run hundreds of thousands processes concurrently, but before getting there make sure there's appropriate rationale behind that.

SYMPTOMS

A long running activity blocks another one that could be run concurrently and entirely independently of the fist one but sits in the same process and is affected by the long-runner.
A crash in a process prevents its further execution even if the code it was to execute could run regardless of the error.

WHAT TO LOOK FOR

Look at the behaviours, like GenServers, and places where we call other components / external API thay may potentially take a long time to finish or generate an error. In the first case, ask yourself if this call has to be synchronous, and in the latter, if the potential error from such a call should crash the entire GenServer process.

ROOT CAUSE

It takes time to get used to thinking in terms of concurrent activities running in isolated processes. For those less experienced it is not obvious to separate activities into processes.

HOW TO MOVE FORWARD

Given two activities: A and B, the reasoning whether to split them into separate processes may go like this:
Should A and B be isolated? If so, then split A and B into process. If atomicity is what is needed, one process should be enough.
Should the latency of A not to slow down B? If so, put them into separate process.

Erlang SOLUTIONS

ABOUT ERLANG SOLUTIONS

With more than 100 specialists working out of London, Stockholm, Krakow, Budapest, and New York, Erlang Solutions has a team of software experts passionate about BEAM technologies, including Elixir and Erlang/OTP, and with an unwavering belief in the open-source future.

Offering a wide range of services from consultancy and development to architecture reviews and training, and by building trusted, fault-tolerant systems that can scale to billions of users, Erlang Solutions' technical expertise is the backbone to numerous global business operations.

"Having Erlang Solutions assess your designs / implementation before launching any Elixir based product should be a part of every company's prelaunch checklist."

Binh Ly, Chief Technology Officer at TalkShop

LET'S TALK ABOUT:

+ Architecture, prototypes, and software development
+ Code Reviews - we'll review your code and help you make it even better
+ Adapting your team skill set to the Erlang VM environment, so they can be productive as early as possible with minimal disruption
+ Project Management - we'll provide an end-to-end solution that is professional, fault tolerant, concurrent, and reliable

Erlang Solutions builds scalable systems for over 300 clients to power their emerging sectors; Blockchain, FinTech, Media, Telecoms, Gaming, eHealth, and Internet of Things. Contact us to discuss your project info@erlang-solutions.com.

ABOUT US

www.erlang-solutions.com
info@erlang-solutions.com
+44 (0) 20 7456 1020

Erlang SOLUTIONS