The route to the successful adoption of non-mainstream programming languages

Lessons from adopting Erlang and Elixir

By Francesco Cesarini
& Mike Williams

Erlang SOLUTIONS

AUTHORS
Francesco Cesarini
Mike Williams

CONTACT
Erlang Solutions
London HQ
The Loom
14 Gower's Walk
E1 8PY London
United Kingdom

general@erlang-solutions.com
www.erlang-solutions.com

© COPYRIGHT ERLANG SOLUTIONS LTD 2017

Contents

AUTHOR
Francesco Cesarini
Mike Williams

CONTACT
Erlang Solutions
London HQ
The Loom
14 Gower's Walk
E1 8PY London
United Kingdom

general@erlang-solutions.com
www.erlang-solutions.com

© COPYRIGHT ERLANG SOLUTIONS LTD 2018

An introduction to taking the road less travelled

Erlang SOLUTIONS

An introduction to taking the road less travelled

You've looked at the alternatives and come to the conclusion that a programming language which has never made it in the top 10 of the Tiobe Index is the right tool for the job. This new language will help you solve your productivity, maintainability, reliability, scalability, [insert your favourite] problems, making your team more efficient, improving the quality of what you ship whilst saving you lots of money along the way. Whilst the reasons might be the right ones, using the right tool for the job is a big step from using the tools at hand.

Introducing a new technology in an organisation is never easy, be it a fast growing gambling start-up or a Fortune 100 company. Alongside technical challenges, you need to deal with organisational, political and human factors. And if your company is politicised enough, you will have to dodge the vultures circling the project waiting for the spills so they can revert to their old ways (all within their comfort zone) just to say "I told you so."

In this guide, two Erlang old timers share the lessons learnt from decades of helping companies successfully develop and ship Erlang based products, including Elixir. Mike is a co-inventor of Erlang and has managed teams with hundreds of developers, whilst Francesco spends a good chunk of his time setting up development centres, teaching Erlang, mentoring, architecting systems, and then writing books about it.

The first pitfall in adopting a language is not having the right people in place who can take the lead and help you build a core team.

Whilst our experiences are directly related to Erlang, we know that much of what we write is language agnostic. Be it Elixir, Go, Scala, Rust, F#, Haskell or any of our other favourite languages which, despite their merits, never made it mainstream. The route to successful adoption is the same.

All references to companies and people in this whitepaper are fictional. If you recognise yourself or a company you/we might have worked for, be reassured that all of this happened twenty years ago. And if you were around us twenty years ago and have a sense of déjà vu, then be assured that all these mistakes happened last month (and are happening now). The fact is, we learn from history that we often learn nothing from history. There are a number of the same pitfalls, which need to be avoided when a new technology is introduced in a company, irrespective of decade.

The first pitfall in adopting a language is not having the right people in place who can take the lead and help you build a core team.

Mike Williams

Mike Williams co-founded the **Ericsson Computer Science Laboratory** where, among other things, Erlang was created. He developed the first Erlang virtual machine and worked out the primitives for fault handling and dynamic code replacement. Since the 1990s, Mike has been in charge of both large and small units within Ericsson which develop software.

@FrancescoC

Francesco Cesarini

Francesco Cesarini is the founder of **Erlang Solutions**. He has used Erlang on a daily basis since 1995, starting as an intern at Ericsson's Computer Science Laboratory, the birthplace of Erlang. He moved on to Ericsson's Erlang training and consulting arm working on the first release of OTP, applying it to turnkey solutions and flagship telecom applications. In 1999, he founded Erlang Solutions.

Since then, Francesco has also co-authored the books **Erlang Programming** and **Designing for Scalability with Erlang/OTP**.

Building the core team

Erlang SOLUTIONS

Building the core team

But before you start thinking of hundreds, start small.

On one end of the scale, we often hear complaints about how hard it is to find Erlang developers. Java, C++ and Web developers are everywhere. They are easy to recruit. Not all might be top notch, but at least we can recruit them. Despite that, look at Ericsson who use Erlang in projects with over a hundred developers in each. Members join and leave the team, move to other Erlang or non-Erlang projects and can easily be swapped around.

Ericsson had to start somewhere; in the early days, everyone on the internal Erlang mailing list knew each other personally. Today, their Erlang programmer count is in the thousands and those using it daily are in the hundreds. But they are not alone. The reason you might not hear from companies such as Klarna, OpenX, AlertLogic, WhatsApp, Cisco and **bet365** (to mention but a few) is that they just get on with it. They are too busy building teams to complain how it's hard to find Erlang developers.

Members join and leave the team, move to other Erlang or non-Erlang projects and can easily be swapped around.

We have seen Erlang used by teams of a hundred developers who produced successful products, we have also seen teams of two or three developing, deploying and maintaining a huge code base. Not to mention companies who adopt non mainstream languages through acquisitions, successfully introducing the technology in the wider organisation instead of making the crazy decision to rewrite the product or service in Java.

But before you start thinking of hundreds, start small.

Start with a few experienced leaders

Like all programming technologies, you must have one or two experienced, good software developers who can lead the way.

Like everything in life, even a programming language can be used and abused. You need people who will stop you from making the wrong decision and avoid mistakes others have done.

Create a small team for the project initiation. Fill it with a few highly experienced experts who will work to shape a plan and get the change sponsors to a common level of understanding.

They have to be enthusiastic, but also make sure they understand and embrace the programming model.

Those who were around in the very early days at the time when C++ and OO was all the hype might remember that we spent a lot of our time convincing people that doing OO programming with Erlang was not a good idea, and that, if unwilling to make the move to functional and concurrency oriented programming, they were better off sticking to C++ or Java.

No technology (or varying levels of enthusiasm) can replace experienced software developers who know what they are doing.

While you're cherry-picking these leaders among men, see if you can seed one of them as a representative on the steering board. Their experience and enthusiasm will be an asset when it comes to changing hearts and minds across the organisation.

Developer vs programmer

We use the term "software developer" rather than "programmer", because software development entails much more than programming.

When building your team, you need people who understand and can support the product life cycle, combined with subject matter experts who understand the industry vertical you are active in. In small teams these may well be the same people.

Software development entails much more than programming.

You need to ensure you have developers who can define, program, build, test, deploy, document, package, write user manuals, and maintain your system.

Delegation

You do not want to rely on a lonely hero programmer who does not understand the value of delegation and knowledge sharing, believing that DevOps means being woken up in the middle of the night to address customer complaints about the system not working (and then receiving the employee of the month award for cleaning up the mess they created themselves).

Your operations team should be the ones noticing there is a problem through automated alerts, not your customers!

While you might need at least one hero programmer in your team setting the pace, they should never be left alone. Pair them up with a finisher. If you are struggling to find experienced Erlang developers, pick the Erlang champion in your organization and pair them up with a contractor or consultant.

Together with a very small team of two to five, get them started on a prototype. It will allow you (and them) to validate their ideas on a small scale and make mistakes which will not result in the project failing.

Design by prototyping

Erlang SOLUTIONS

Design by prototyping

A prototype is, by its nature, the result of a lot of "trial and error."

You are bound to make mistakes when developing your architecture, the earlier you can correct these mistakes or find better ways to do something, the easier it is to change.

Prototyping

Erlang is an ideal language for rapid prototyping. Applications build themselves, when you make a change in your business logic, all you need to do is replace the modules where the changes have been made.

You can often do this "on the fly" without stopping the application you are developing and testing. When your application crashes, and it will during the development, you get a very accurate pin-point of where the crash happened.

You might work on a functional prototype, a prototype addressing fault tolerance, scalability or performance. It should address the tricky parts of your application, parts that may be algorithmically complicated or critical for the performance.

Optimising prototypes

You will cut corners and take shortcuts to get something up and running quickly. Make sure you address areas of concern of all the stake-holders involved, technical and non-technical.

A developer who does not know Erlang might have burnt his fingers on performance when dealing with a Java monolith, whilst an Erlang expert might be more worried about the complexity of some aspect of the problem, requiring an end-to-end solution which can be used not only to show that the ideas work, but also as a basis for later estimates.

Optimising performance must be made based on metrics and benchmarks derived from the prototype. Measurements often reveal very surprising bottlenecks - sometimes not at all where you expect!

> ***Prototype measurements can reveal surprising bottlenecks - sometimes not where you expected!***

We once reviewed a system handling 30 requests per second. Our client, migrating from Ruby, seemed very satisfied with the results. A similar product we had worked on, using the exact same libraries, was managing 1,000 requests per second.

Prototyping also has the effect of increasing your team's understanding of the application and vertical you are in. Unless you are re-developing an application you have built before, there is a lot of hidden baggage that will come to the surface as the project progresses.

Save your rationalisations

Documentation might not be critical at this stage, but be sure you have described the rationale behind your choices. Documenting the rationale will ensure the reasoning behind your design choices are not forgotten, and it will avoid you having to repeat them every time someone joins the team.

Prototypes are often used to interact with the project sponsors and higher management, giving them the background information needed to make a go / no-go decision on whether to invest in a large project with a new programming language. By showing early results, you can both build confidence and show that you are developing the right thing.

Throw the prototype away

When you are satisfied that you have proven or disproven the goals set out in the proof of concept, and get the go ahead to expand the team and the budget to expand the project or build the application, write a short and succinct description of the principles and throw the prototype away.

> ***A prototype is, by its nature, the result of a lot of "trial and error". The code in it won't be good and what's more, only understood by the people that built it.***

A prototype is, by its nature, the result of a lot of "trial and error." The code in it won't be good and what's more, only understood by the people that built it. Project sponsors and stakeholders might have a different view. Make sure you are able to diplomatically push back requests to call your prototype a product and start shipping it to your customers. It happens every time.

At this stage you will hopefully find flaws in your architecture and be aware of improvements that can be made. Make small prototypes to test alternatives. Don't be afraid of re-writing parts of your application if things aren't working out. At the end of the day, patching and hacking code costs a lot more than re-writing it. When you have completed the prototype and decided to adopt a new programming language, the time has come to expand your team.

Don't forget to test

At the same time as you make your prototype, make a prototype of the automatic test suite which you will need as your system grows. Too often testing and verification is implemented as an afterthought, resulting in a buggy product which is hard to maintain.

The cost of implementing test cases and testing a system is similar to the cost of developing it. Don't underestimate it.

Build a self-sustaining team

Erlang SOLUTIONS

Build a self-sustaining team

Create an environment to grow and foster the mindset for success.

It's vital that you keep the team that built the prototype and use them to mentor the new recruits. In doing so, you need to make sure your team becomes self-sustainable.

It is best if the mentor can sit next to the new recruits, communication needs to be quick and efficient from both sides. As members move to new positions or get assigned new POCs, you need to be able to replace them, be capable of mentoring the new joiners, bringing them up to speed.

Our philosophy is to work with internal teams to grow the team's skills, knowledge and capabilities. Create an environment to grow and foster the mindset for success.

If you are a small company, a minimal team should consist of 4-5 people.

If you are a large company, 10-12 developers and up.

Not everyone needs to be using or developing in Erlang, but they should be passionate about it, be able to jump in and be productive at short notice, and be capable of rotating other team members out.

Transferable skills

It is always a mistake to advertise for programmers with experience in the language you are adopting; you want good and experienced open-minded software developers who understand different programming paradigms and approaches. They do not need to know the programming language, but they should be capable of understanding what is meant by using the right tool for the job.

> ***Erlang, you can learn easily. Good software practices, less so.***

If you acquire a company, you will get a pool of great developers who already have the domain knowledge. Back in 2004, a company in Silicon Valley was advertising for Erlang developers with at least a decade of Erlang experience. They had adopted Erlang as a result of an acquisition at a time when there were maybe 20 developers in the world who qualified (including the authors of this blog).

Ironically, that company used to employ another ten, but had just shut down their Swedish operations, making them redundant. Don't get too hung up on location. Focus on talent and productivity.

A team that grows with you

As your software project progresses, the focus of your work will change, at the start your focus will be on developers with good architectural and programming skills; later on configuration management, verification and testing; then deployment, operations, maintenance and documentation.

> ***Build a team of developers who can embrace the skills needed across the whole of the development life cycle.***

You should avoid different people for each category, rather you should have people who themselves can embrace the skills needed for the whole of the development life cycle. Handing over knowledge from person to person is error prone and costly, when a person works across several areas this can be avoided.

Cut out the bad habits

Learning Erlang/OTP is easy, as it is a very compact language and framework. The difficult part is unlearning other programming language models and getting rid of the bad habits they bring with them.

This is where mentorship and code reviews become an essential part of the process. Review all code, not only code in the application itself but test cases, simulators and tools.

Code reviews should be led by experienced developers focusing not only on quality, style, and documentation, but also on educating developers on the system as a whole.

New recruits often get up to speed fastest when making changes to existing code. Be cautious and review new code written by new recruits carefully.

Ericsson did a study in the late 90s looking at the productivity of graduates and experienced developers in large projects after they had attended Erlang and OTP training. They concluded that a graduate straight out of university became productive after about a month, as they could easily pick up new programming models and handle change.

It took an experienced software developer who attended the same courses three months. They picked up Erlang easily, but had to spend more time unlearning their previous programming paradigms and embrace the Erlang way. Code reviews and mentorship in this phase were critical. When we say productive, we obviously mean productive in their respective levels of seniority.

Invest in training

When building your team, do not underestimate the value of training and study circles. Where there is significant change being effected within an organisation, there must also be adequate training and support provided to the those impacted, with an effective feedback path to ensure that issues with change do not turn into objections and resistance.

> ***We often see companies give their developers a book hoping they will become productive. Why not? They are smart, well paid, and as a result, should be capable of picking up a new language on their own.***

We often see companies give their developers a book hoping they will become productive. Why not? They are smart, well paid, and as a result, should be capable of picking up a new language on their own.

It might work for polyglots who have used other functional and concurrent programming languages, but never, ever, take it for granted. If you want to save money and get your developers productive quickly, invest in a proper training and mentorship, especially if you are ramping up a team of ten or more developers. It will allow them to quickly understand and embrace the new programming paradigm and reduce beginner mistakes.

If you go down the book route, don't be surprised if the project runs into trouble. By being penny wise and pound foolish, you'll scare away the Erlang/OTP experts, as they'll burn out spending all their time refactoring and debugging bad code with no time left over to mentor those who really need it.

And remember, stakeholders outside of your core development team will need training to sustain the excitement and investment in the project. A tactic that has worked for us in the past is "boot camp" training; we've seen it create the right mindset and understanding of "change tools" within change leadership teams.

Industry subject matter expertise

Erlang is just a programming language. That's the easy part. Learning the application domain is probably the hardest task of any software project. This is where the vertical subject matter expert needs to come to the rescue.

> ***Learning a programming language is the easy part, never underestimate vertical subject-matter expertise.***

Accept the fact that you will have to train your new recruits in Erlang and maybe some of the other technologies you use, but also accept the fact that they might not be vertical subject matter experts. They might be shipping code after a few months, but it's going to take them much longer to learn the necessary domain expertise.

Plan and act with long term adoption in mind

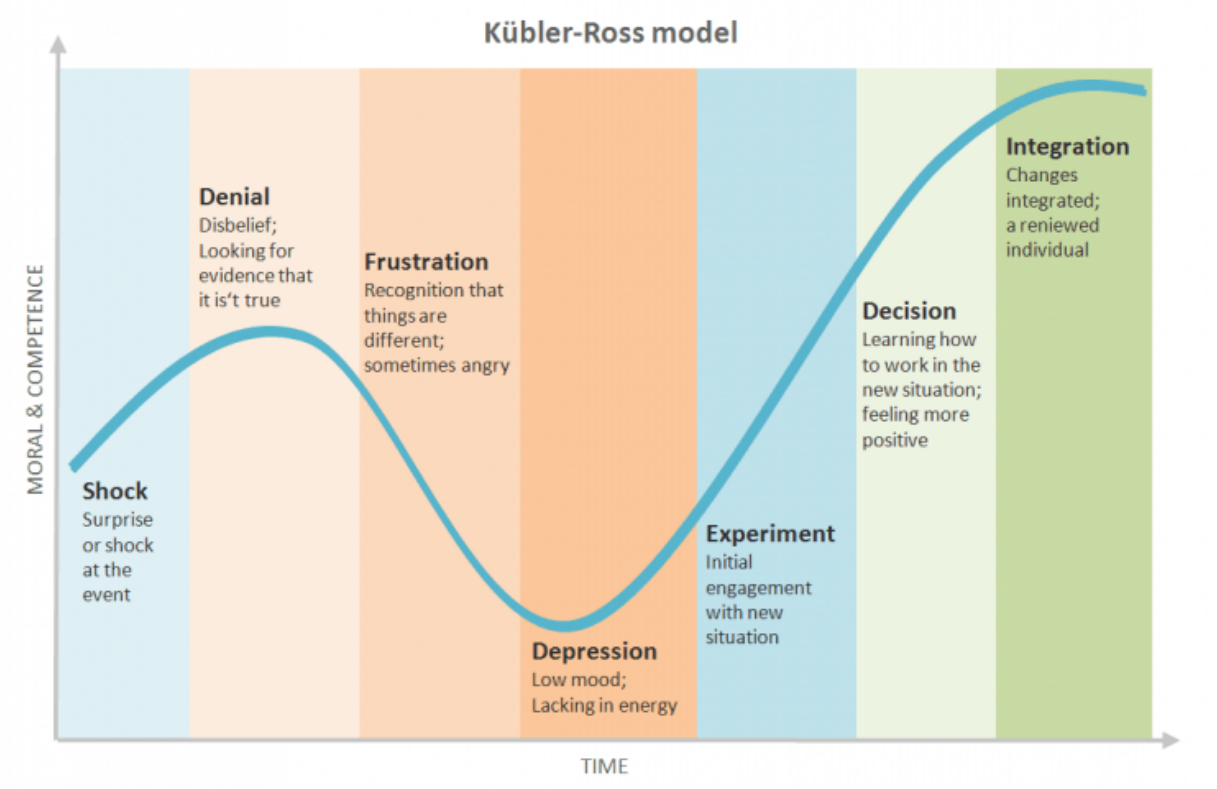
If you work for a large company and inherit a team, remember that you can't force them to use technology they don't like. If you are going to use Erlang/OTP or any other non-mainstream language, make sure the team understands the advantages and benefits, and receive the right onboarding.

> ***One of the main reasons for projects moving from non-mainstream languages is their i nability to make the team self-sustainable.***

One of the main reasons for projects moving from non-mainstream languages is their inability to make the team self-sustainable. Not having a recruitment pipeline and losing key members of the team puts pressure on the other members.

A key aspect of these kind of projects is changing individual mindsets and transitioning from established ways of doing things. This involves creating energy and enthusiasm by showing the way from small safe incremental steps to the new operating model.

Incremental is key. I have seen the pace of change play a crucial role in change adoption. If there is too much change then teams can feel overwhelmed. It can turn a "change advocate" into someone who is passive or worse, actively resisting change. This behaviour can also be triggered if change creates more work or duplication.



Source: www.cleverism.com

The Kübler-Ross Model

The **Kübler-Ross Change Curve** is a powerful tool to understand where people are on the change journey. Those driving change will be further along the change curve than those who have change forced upon them.

Prepare a toolbox of techniques to help trigger and stimulate movement through the change curve. For example, "town halls" as a vehicle for ensuring that objections to change are aired so that objectors to change do not get stuck in the "denial" region of the change curve.

Remember: projects where the right tool for the job is used will often consist of multiple programming languages. Not everyone is expected to work on Erlang, but they should know it well enough to be able to help out when necessary. Don't freeze out the naysayers, you may need them down the road.

Join the community

Take on summer interns and thesis students; this is an excellent way to evaluate developers. When they graduate, they might join you. Present at conferences and get your name out. Where possible, if there is a fit with what you are doing, join academic research projects. Your costs are often covered through grants.

If you are a large corporation, provide regular courses open to everyone in the company to recruit internally. Run hackathons and study circles and host meetups to generate interest and get a contact network. Use and contribute to open source projects, and where possible, release and maintain components others might find useful. Those who see what you are doing out of curiosity and like what they see, are likely to join the team when an opening appears.

Integrate your toolchain

Erlang SOLUTIONS

Integrate your toolchain

Don't cut corners!

While building the prototype, or as soon as you get the go ahead for the main project, you need to put your workflow in place.

It is critical this is done before onboarding a larger team. Sometimes, this toolchain and development environment gets dictated to you. Other times, you have the luxury of starting fresh and apply the best practices that prevail.

Working on the workflow will include integrating with build and test tools, code repository, continuous integration and deployment servers. You need to do this whilst ensuring that once your commit hooks are in place, you run all your test cases and get a proper OTP release on the other end, shipped in a tar file, an image, package or container.

Building a toolchain

If you are setting up a toolchain from scratch, a typical stack at Erlang Solutions might include **Jenkins** for CI, deployment and automated tasks, **Ansible** for Configuration Management and deploy scripts. We use **GitHub** and **GitLab** for revision control.

We run all of our tests with Common Test, triggered by Jenkins as a result of each pull request. Through **gadget**, we run **Elvis** to enforce stylistic guidelines, **dialyzer** to find software discrepancies such as type errors and dead code, and xref to test module dependencies. All of this, of course, complemented by **rebar3**.

Go agile

Other companies have taken the introduction of a new programming language as an opportunity to change not only the toolchain, but also the culture. Going from a strict waterfall approach to an agile one, paired up with a newly designed framework and infrastructure has allowed many to reduce the time to develop and deploy a new service from months (many, many months) to weeks, sometimes days and in some cases, even hours.

Reduce the time to develop and deploy a new service from months (many, many months) to weeks, sometimes days and in some cases, even hours.

Don't cut corners

No matter if you are integrating in your existing toolchain or rolling out shiny brand new one, invest the time to ensure you can seamlessly run tests and produce a proper OTP release. Whilst you must be aware that there will be a learning curve, do not be tempted to cut corners, as you will pay the costs in reduced productivity and quality later down the line.

Do not be tempted to cut corners, as you will pay the costs in reduced productivity and quality later down the line.

Invest in your internal libraries and interfaces

Erlang SOLUTIONS

Invest in your internal libraries and interfaces

Your first project might be painful...

Unless you are a newly founded start-up, it is likely that your system will have to interface existing services, libraries and even other programs. Are you using a payment system?

Create an OTP application that exports easy to use Erlang APIs you can reuse across projects. In it, add all of your standard metrics, alarms, notifications and logs.

Be prepared for your first project to be painful, there will be a lot of integration to put in place. But over time, it gets easier!

A messaging or notification system?
Do the same.

Authentication, authorization and policy server for your users?

Or what about an SNMP monitor or other logging and metrics tools used to interface your O&M infrastructure?

Build reusable OTP applications

Make sure you define your APIs well and isolate these components into reusable OTP applications. Your first project might be painful, as there will be a lot of integration to put in place.

But as you roll out more products and services, it gets easier. After a few years, expect a fully operational system, integrated into your infrastructure to be written just by putting together your existing libraries and a little bit of business logic.

Don’t ignore your operations team

Erlang SOLUTIONS

Don’t ignore your operations team

Hearts and Minds.

Never underestimate the power of winning the “hearts and minds” of your operations teams when migrating to a new set of tools and technologies.

Some programmers think that once they’ve written their code and deployed it, that’s where their responsibility ends, as they are not the ones who get called in the middle of the night to solve problems they have created. Anyone who has not worked in operations and is not given proper guidelines will probably fail to consider the operational aspects and DevOps.

The focus of change is creating sustainable cultural shifts.

In our experience of transformation projects, technology is the easy part. New technology may trigger disruptive behaviours but the key to the success of transformation projects is gaining the buy-in from stakeholders and staff whose way of working is being disrupted. Top of that list: operations.

Be sure that the focus of change is creating sustainable cultural shifts and embedding DevOps principles.

Never underestimate the power of winning the “hearts and minds” of your operations teams when migrating to a new set of tools and technologies.

Visibility

The Erlang VM gives you a level of visibility no other VM does. This is rarely used, and very few metrics are exported, so...

> ***...when something goes wrong, you will end up looking for a needle in a haystack, but without the needle or the hay.***

Make sure that what you develop is integrated in your infrastructure with existing tools, metrics, logging and alarm systems. The operations team want the same visibility (or better) than they have with systems written in other languages, and they will want your system to be integrated into their existing tool set. Forget them learning something new.

How many times have companies turned down Erlang because it does not run on the JVM? Not because the JVM is the best VM out there, but because operations know how it works.

Giving your DevOps team the same or better visibility as the JVM, with the same tools they are used to will reduce resistance to change.

Maintenance

Another common problem is that Erlang systems rarely fail, and there is rarely the need or opportunity to address minor issues. The let it crash approach, where failure is isolated and recovery built in the business logic, means that the operations teams do not learn the necessary maintenance skills for handling minor incidents.

> ***Make sure you plan accordingly and give your DevOps team not only all the training, support and visibility they need, but also the developers' time.***

So when things go wrong, they go wrong big time, requiring intervention from third line support or from the developers themselves.

Conclusion

In conclusion

Popularity should not come between you and the right tool for the job!

The popularity, or lack there of, of a programming language can mean that it is hard to find and keep talented software developers to build your application. It can be seen as risky by business decision makers, and rejected by DevOps who find solace and comfort in the tried and tested.

As two Erlang old timers who have spent decades helping companies successfully develop and ship Erlang based products, a language that has never made the Tiobe Index top 10, we're here to tell you that popularity should not come between you and the right tool for the job.

With due diligence and careful, reasoned planning, you can have your cake and eat it too.

When adopting a non-mainstream language, you need an internal champion around which you can build a core team. Put them to work on a prototype. It will allow you to make mistakes on a small scale, away from production.

Use prototypes to address areas of concern, and make sure you are able to solve the problem, because it is not good enough to have ideas, you must also be able to implement them and know they work! This approach will tell you if you are right on track, but most importantly, also allow you to fail fast and disprove your ideas, cutting your losses early. Failing is acceptable, if done on a small scale.

Failing is acceptable, if done on a small scale.

If the prototype process is successful and makes way for a proper project, you will need to expand the team and onboard more developers. Before doing so, make sure you have a proper workflow in place and have integrated with your toolchain.

This means integrating existing tools dictated to you or picking tools that are considered cutting edge. What is important is that your workflow gives you an artefact that consists of a proper release that can be deployed to your target environment.

When expanding your team, look not for programmers, but for talented software developers who understand what the right tool for the job means.

In conclusion

Make sure your team becomes self-sustainable by maintaining an active recruitment pipeline, visibility in the community, contributing back to open source, and taking on interns. They do not need to know the language in question, but have the aptitude to pick it up. Proper mentorship, training, and change management is key.

Finally, you need to invest in your internal libraries and interfaces, as they will provide a solid, reusable code base in projects to come. Some of the libraries will be used to interface towards your wider OAM infrastructure, giving your operations team comfort with the same level of visibility and familiarity as other systems they support through the tools they are used to working with.

References & further reading

Learn more about Erlang

Learn more about Elixir

Kübler-Ross Change Curve

[Case study]
Innovation and Scalability: bet365 relies on Erlang-based system to deliver smooth service to 11 million players

[Case study]
Bleacher Report uses Elixir to handle 8x more traffic and pushes the new content 10x faster

[Webinar]
Learning Erlang - Easier than you think

[Conference Talk]
0 – 100 MPH - Launching a New Product at Scale (with Erlang)

Erlang Solutions

Say hello!

The world's most used systems look to Erlang Solutions for technical expertise and scalability.

Erlang Solutions specialises in giving business truly scalable solutions through the creation, integration, delivery, and lifetime support of products and services based on **Erlang** and **Elixir** technologies and other technologies. Our team designs and delivers scalable solutions for end-to-end systems for clients ranging from start-ups to Fortune 500 companies.

We specialise in providing our clients with truly scalable solutions. That's why leading finance, social media, healthcare, advertising and gaming organisations look to us for technical solutions.

Request a call

To request a call or to ask a question, contact us and one of the team will respond to your inquiry as soon as possible.

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

Erlang
SOLUTIONS