Choose Your Risks, Or They'll Choose You

Your software team's playbook for mitigating unintended consequences in financial services and how feature flags can help.

By Ariel Pérez

Choose Your Risks

Developing software for the financial services industry is a unique world in which to work. There are a different set of obstacles, regulations, and no real rules to the road. This eBook discusses the challenges and opportunities for today's product development teams working at large financial institutions. It dissects some common misconceptions from trying to do "the right thing" to meeting regulatory and safety standards. It also provides guidelines for achieving your goals while securing reliable risk controls in the process.

What We'll Cover

About the author:

Ariel Perez

Ariel is currently the VP of Engineering for Measurement & Learning at Split. He is responsible for the technical direction and leadership of the engineering team. They manage Split's massive data pipelines and the statistical engine charged with enabling customers to derive insights from their features.

Ariel Pérez has had a diverse career in software engineering and product development. He has worked at large global enterprises like JP Morgan Chase, small startups like Try The World, and even ran his own product and engineering consulting practice. As he's gained experience across different domains and faced numerous technical challenges, one core belief has continually been reinforced in his mind: that putting people first is what has the biggest impact of all in any organization. He has always strived to be a leader, rather than a manager in all his roles, whether they were purely in Engineering Management, in Entrepreneurship, in Product Management, or a combination of them.

When he's not leading teams and building highly scalable distributed solutions, you can find Ariel in a dance studio teaching a latin dance class, training a dance team, or on a dance floor - dancing to live music.

Financial Services Development:

It's a Never Ending Game of Risk Mitigation

For software developers working in financial services, mitigating risk goes with the territory. It's like a never-ending game of Whac-A-Mole, where you navigate the dangers as they appear. The regulators in the industry are quick to point to watch-outs, but the truth is, no one ever tells you how to avoid them. Developers are alone, exposed to potential errors. The good news is: Responsibility and autonomy can also be rewarding if you play your cards right. Embracing a reimagined software delivery strategy that's powered by feature management can help; it might even earn you a few extra arcade tokens.

Risk mitigation is arguably the number one priority in financial services firms' software, and the controls make product development teams feel anything but agile. With constant policy changes, layered regulatory environments, and emergent security threats, there's too much at stake not to have an overabundance of caution. However, product development teams may lose sight of some very important opportunities if safety is the only focus. There's always room to further accelerate feature delivery and empower a stronger culture of experimentation. Even at the world's largest financial institutions, this can be done in a manner that doesn't undermine risk controls, but further strengthens them.

I was able to embrace Agile at a Fortune 50 bank without losing rigor in our approach to risk management, and you can, too. I want to share my approach with other companies and product development teams in the industry. The following plays are designed to help you achieve your goals in financial services software without collateral damage along the way.

Key Priorities For Financial Services Software

According to a survey conducted by EY Research, here's what respondents in financial services indicated as a top priority for their software development team:

89% Say Enhancing Cyber & Data Security Is a Priority

85% Implementing a Digital Transformation Program

82% Gaining Efficiencies Through Technology Adoption

77% Think They Need To Improve Risk Management

Safe Moves With Risky Outcomes

And How to Improve Your Play

If we speed up, we might miss something. But if we slow down, we might fall behind. If we release a feature now, it could break. If we keep it in testing, it might never see the light of day. We constantly weigh these risks and more as engineers in the financial services industry. Many work to squash out all possible sources of error. However, those who think perfection is attainable might be opening the door to a whole new world of unintended consequences.

Here's the reality: With so many risks in financial services software, there's no way to eliminate them all completely. The only way we can advance our practice is to take a more holistic view of risks, so we can better identify which ones are most worth taking. Let's discuss some common risk controls applied to software development in financial services, and how they might be more dangerous than you think. Then, let's look at some alternate plays, so you can start taking risks without really taking risks.

1. The Branch That Ends In Conflict

The Challenge:

Sheer Size

Large institutions dominate financial services with massive scales. Hundreds (sometimes thousands) of software engineers could be working on the same application simultaneously. That's a lot of people that might potentially disrupt the development process. As a result, projects get broken out into silos with numerous testing environments and multiple layers of approval. Meanwhile, you're just trying not to step on toes or break the build.

Your Move:

A Long-Lived Feature Branch and Release Branch

To mitigate risk, you're probably deploying to a long-lived feature branch, a release branch, or both. Although this strategy encourages further validation and testing in a safely staged environment, it's actually introducing a whole new set of risks. It's as if these branches become isolated, like islands. Feedback on how a new

branch will respond to being integrated isn't truly known until after it's merged with the main line. By the time that actually happens, a merge conflict is almost inevitable. Still, more importantly, unintended interactions between features from different branches will finally rear their ugly heads, exposing your applications to anything from performance issues, to inconsistent behaviors, or even crashes.

Alternate Play:

Think Bigger Than a Branch, Deploy to the Trunk

Trunk-based development is an industry best practice, and the fastest moving companies have been embracing it for years. The goal is to achieve continuous integration, pushing your software changes to the main line as often as possible. The more often, the better. So, when you merge, you can make it much more likely that the main line will be ready to deploy at a moment's notice.

But let's get real for a minute. You know that continuous integration can't be achieved overnight, especially at a large financial institution with thousands of code changes and branches already in motion. You're absolutely right. However, you can slowly transition now by merging unfinished work right to the trunk. This is where feature flags shine.

Feature flags let you deploy work-in-progress code right to the trunk, wrapping it up in a safety blanket that notifies your team right away. "Hey, this isn't done yet," says the feature flag, "But when the time comes, I'm living proof that you can merge me without breaking things." Instead of writing your code on a separate branch with plans to merge later, you're creating what's called a branch by abstraction within your existing source code. It's there behind a flag. It's passing the validation process. It just hasn't been finished yet or turned on for your customers.

To unite silos, processes, and remove potential merge conflicts at large financial institutions, give the biggest olive branch possible: the trunk.

2. The Big Bang Release That Blew Up

The Challenge:

Release Risk

To keep our banking customers engaged and constantly innovating personalized and secure banking experiences, you've got to release and deploy new features. It's part of the job we signed up for. But as you already know, there are many risks and costs involved with the process. You want to pressure test your changes, ensure everything runs smoothly, and reduce the impact of downtime that might disrupt your customers.

Your Move:

A Batched Release Strategy

This strategy is a common way to mitigate release risk in the finance industry. It makes a lot of sense. You want to decrease the impact of downtime, so you deploy on nights and weekends. You want to test and validate as much as possible through a contained release branch. You want to reduce the effort and cost involved with deployment, so you pay it all at once, and coordinate a big-bang release. This is the name of the game in finance. However, you're introducing a slew of new risks by batching your releases together.

With an everything-at-once strategy, more things can go wrong, which leads to more problems to sift through, and there's a higher chance you'll have to roll it all back. The minimal downtime you planned could be extended for a while, and it's no vacation.

Alternate Play:

Risk a Little, Not It All: Try Small, Frequent Releases and Deployment

As we previously mentioned, feature flags can help you move past long-lived feature branches so that you can strive for a trunk-based approach. In doing so, you gain the ability to empower fast and frequent release/deploy strategies. The value of this is getting your features out there often. What you get back is real-time insight into how features perform in the real world. Because your focus is more granular, you'll instantly see which features are causing problems. Turn off the bad ones; there's no need for a stressful rollback of an entire release branch.

A beautiful thing about feature flags is that they separate release from deployment. This means you put a feature out there and keep it turned off. Then, you can see how it behaves without actually showing it to your customers.

This is how you move away from the big bangs that can blow up your batched release strategy. Small and frequent releases/deployments are a smaller risk to take that can help gain efficiencies. “Aim small, miss small”: That’s the name of the game.

3. Slowing Down for Safety Reasons

The Challenge:

Code Breaks

Overregulation in financial services tends to prioritize risk mitigation over release velocity. That’s no surprise. But it’s not like you can stop releasing new software innovations to your customers. Your product management team would despise you, and your customers would leave you for a new cloud-based banking provider. We have to push on while trying to balance all of the risks. Thinking about all the errors that might happen in production will only paralyze you. We have to face the facts. Mistakes happen; it’s a matter of minimizing the consequences as much as humanly possible.

Your Move:

Slowing Down Your Releases

If you’re shipping less often, it’s because you’re making sure you’ve got it right. You’re testing new features to double-check they won’t be flukey. You’re updating to meet customer requests. You want your new feature release to be perfect, and that’s a natural way to think. But, as you’re striving for perfection, remember this: Perfect doesn’t exist in software development. In fact, perfection is the enemy of progress.

By chasing perfection, you slow things down. Here’s what typically happens. If a break occurs (and they often do), there’s no way to get your time back. Since you spent so much of it preparing for your release, you’ll have little time left to roll back, fix, and re-release. A strategy like this only guarantees one thing: It’ll take you longer to fix the problem.

Alternate Play:

Don't Slow Down for Safety, Speed Up the Fix

Many development teams are saying this, and it truly works in practice. Safety is really just a result of speed. Increased velocity can actually eliminate risks.

How do you make it happen? It requires investing in the things that allow you to go fast with confidence: TDD, CI/CD, automated testing, observability/monitoring, with feature flags and insightful data woven throughout. This helps the team exercise a critical muscle. Introducing features regularly and tracking the impact, allows you to respond the moment something breaks. A little blip is a quick fix, and that's as simple as toggling off a feature.

Shift to strategies you can actually control. If we know breaks are inevitable, be a better fixer, not a product of the underprepared.

4. The Cautious Move Not to Move

The Challenge:

Legacy Technology

Cloud adoption and microservices are the new norm, but migrating to them doesn't come without the risk of a critical data leak and a plethora of other potentially catastrophic failures. It's why so many financial institutions are hanging on to legacy technology. This spans old mainframes, exhausted database providers, even ancient versions of Java, .NET, and Python. You're probably pressured to keep up with agile transformation in your engineering department, which leaves you weighing some difficult decisions. Do you risk migrating to microservices? Or, do you risk keeping legacy running on its last leg? Talk about a lose-lose situation.

Your Move:

"If It Ain't Broke, Don't Fix It"

If you're a software engineer in the financial industry, you're constantly triaging risks as they appear. It makes sense that you'd choose to avoid a time-consuming and costly migration if you don't have to. But here's the reality: The longer you run your legacy solutions, the higher the likelihood these things will lose their resiliency. Eventually, they get harder to update, more expensive to test, and trickier to modify without breaking things. As a result, they become more vulnerable to security risks. You might think it's a safe move to keep legacy, but you're introducing a whole other list of risks that worsen over time.

Alternate Play:

Move to the Cloud and Microservices, Avoid Lifting Everything at Once

Even though it's an investment and the process takes time, migrating away from legacy to microservices is worth it. It's a long-term commitment to agile transformation and security. The longer you wait to migrate, the more your legacy technology becomes outdated and potentially irreparable. Might as well get going now to avoid these risks.

So what if you're a major bank? It's not like you can snap your fingers, and your entire global infrastructure suddenly teleports to the cloud. Seriously, there is a feasible strategy, and it's about starting small. This is where feature flags are a life saver.

Feature flags help you carve out small pieces of code little by little. Then, simply migrate to the cloud in itty-bitty chunks. Doing this allows you to control the traffic from the old system to the new one in percentages. If something is failing, you can shut it off with very little traffic disturbance.

Want to go even further by tracking the progress of your migration? Use a feature management platform that attributes data to every flag. With intuitive insights, you can run parity tests between the old and new systems. You can measure to see which one runs faster or slower or which throughput is higher or lower. Basically, you can get as granular with performance metrics as you want.

When you rely on feature flags with measurement and learning, it's less of a lift when you move. *Try it for yourself. Use this sample code for Tap Compare with feature flags while migrating from S3 to Mongo:*

```
switch (storageSourceTreatment) {
    case MONGODB:
        identityResult = _mongoIdentityStorageDAO.fetch(
        …
    case MONGO_COMPARE:
        identityResult = _mongoIdentityStorageDAO.fetch(
        …
        try {
            Optional<Identity> s3Identity = _identityStorageDAO.fetch(
            …
            IdentityComparisonResults results = compareIdentityResults(identityRe
s3Identity);
            if (results.differenceDetected()) {
                _log.warn(results.differenceReport());                    …;
            }
        } catch (
          …
      default:
        identityResult = _identityStorageDAO.fetch(
```

5. Observability Without Context

The Challenge:

The Cost of Troubleshooting

A lot is riding on the way we troubleshoot at major financial institutions. Due to the large number of environments and the sensitivity of data involved, when bugs are introduced, it becomes very difficult to locate and extract them. If you don't move fast enough, it gets serious quick. Outages create downtime, costing on average $5,600 per minute, but that number is even higher at large financial institutions. This puts enormous pressure on DevOps teams, and time to resolution has never been more critical.

Your Move:

Debugging With Feature Observability

With the superabundance of debugging tools available at your disposal, you're making observability possible more and more each day. From modern logging to APM, to observability stacks and more—whatever it takes to help you quickly identify the cause and location of a problem, you'll do it. Whatever methods you can adopt to reduce your time to resolution, it's your number one priority. This is the right strategy. You're not introducing new risks by doing this, however, there's a way to take things one step further.

Additional Play:

Limit Risk Further With Feature Aware

Imagine if you didn't just have observability. Imagine if you had granular awareness within your observability tools. With Feature Aware, you can know exactly how a particular feature impacted a certain metric. That's something only the best feature flagging tools like Split can do. While your observability stack can help you find correlations, only Split can help you determine causation.

With Feature Aware, you can enable more verbose logging for specific users. You can alert on particular metrics like latency and error states that have degraded your system. A Feature-Aware engine tells you, for a fact, that the feature you rolled out is the root of the problem. Furthermore, when integrating feature management platforms with tools like Bugsnag or Datadog, you can see feature context come together with monitoring metrics to be able to quickly identify the cause and location of the issue at hand.

Context and awareness are gamechangers. They'll make troubleshooting and triaging problem-causing features more efficient, regardless of the number of environments involved.

3 Rules Every Software Developer Should Live By

1. Ship Faster
2. Know the Impact of Everything You Release
3. Unite Teams Across Silos

Bank on Feature Management for a Culture of Change

The right feature management and experimentation platform can help you change the culture across product development teams. Eliminate uncertainty with modern trunk-based development through a single source of truth that everyone can work across. Feature flags unlock the velocity teams crave. Engineers become emboldened to test new features as an outcome. If something breaks, they can put everything back together with the push of a button.

Feature management removes the job's ancillary, administrative, and tedious aspects, so engineers can focus on the products they're building. As a result, they start feeling more purposeful, more productive, and less stressed. They receive regular feedback from the features they release, helping them appreciate the ways their work is benefitting the business. After all, engineers want to know that they're making a difference.

Thanks for reading. I hope this playbook helps you choose your risks wisely, so you can advance software delivery practices at your financial institution and beyond. If you're looking to get started with feature management and experimentation, Split is designed to scale with enterprise banks and financial institutions. Its unique architecture is one-of-a-kind for data security and PII protection. Plus, it has a patented attribution engine that automatically connects customer event data to feature flags for performance and behavioral context at the feature-level. This 100% feature observability is a game-changer for those who develop and deliver. Eliminate the guesswork with a deeper attention to data. Request a live demo with a Split sales representative to learn more.

Split as a Return on Investment

From decreasing downtime to helping teams deploy more frequently (and with less risk), the right feature management & experimentation platform pays in more ways than one.

- ✓ Gain **10.7%** Engineering Efficiency
- ✓ An Average of **$16M** Efficiency Savings Per Year
- ✓ Increase Deployment Frequency Up to **50x**
- ✓ About **1 min** MTTR Average
- ✓ Decrease the Cost Per Feature Change By **54%**
- ✓ **100%** Engineers Agree That Feature Flags Are Important to CI/CD

split®

Split is revolutionizing software delivery with its **Feature Data Platform ™**, pairing the speed and reliability of feature flags with data to measure the impact of every feature.

Schedule a demo with us or visit **split.io** to learn more.