

BUGS & AUTOMATION: THE NEXT GENERATION OF QA

A DevOps approach to error monitoring and reporting

CONTENTS

- Introduction
- Why is error tracking so important in game development?
- KPIs that make a difference
- The importance of automation
- Characteristics of an effective solution
- Vital studio-wide benefits
- Leveraging customer support
- Integrated Unity DevOps solutions

INTRODUCTION

Every game developer knows the pain of debugging. This necessary-but-tedious process ultimately improves your chances of launching a successful game, but when you're toiling away fixing your code, it's hard not to think, "There are better ways I could be using my time."

Fortunately, there are many opportunities to optimize your development pipeline, so you can spend less time worrying about catching and squashing every little bug and still go to market with an amazing game.

Upgrading your DevOps solutions and automating development processes like error monitoring and reporting can give you a big chunk of your precious time back.

Bug and error tracking will always be a part of making games, but automating the process can help your team focus their attention elsewhere. With the right set of tools, bugs turn into signposts that point you toward optimized code and a better product. The earlier you implement error-tracking tools, and the better integrated they are with your existing workflows, the greater their impact will be.

This e-book takes a look at error monitoring and reporting from a DevOps standpoint. Read on to learn how to use smart DevOps solutions to speed up development, minimize costs, and deliver a better user experience to your customers.



→ POINT #1

WHY IS ERROR TRACKING SO IMPORTANT IN GAME DEVELOPMENT?

REVIEWS CAN MAKE OR BREAK A GAME'S SUCCESS.

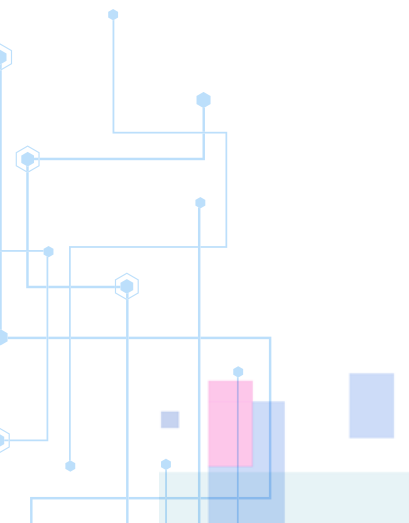
Game development is extraordinarily competitive, and first impressions count for a lot. If your game is rife with bugs upon release – especially if it's a hotly anticipated title – online conversation will not be kind, and in the worst cases it can leave your studio's reputation in tatters. A buggy release can take down a studio of any size.

Error monitoring and reporting has to be an integral part of every stage of development, but it's mostly used in the playtest, beta, and release phases of a game. Internal testing captures only a fraction of errors, and some bugs will get past the most careful developers and QA teams. That's where error tracking makes the difference.

When something does go wrong with code, it's crucial that a reliable, automated tool immediately alerts developers with a meaningful report that includes the entire context of an error. This account should include data such as the device, OS, version, quantity of available memory, and where in the code the error occurred.

It's easy for a smaller indie to put off implementing a sophisticated error monitoring and reporting system. When you're crunching to get a game out the door, it's hard to think about improving your DevOps processes.

But sound DevOps processes will ultimately drive your studio's success. It's essential to have a system to know exactly how your code is performing. And effective tracking doesn't have to break anyone's budget.





→ POINT #2

KPIS THAT MAKE A DIFFERENCE

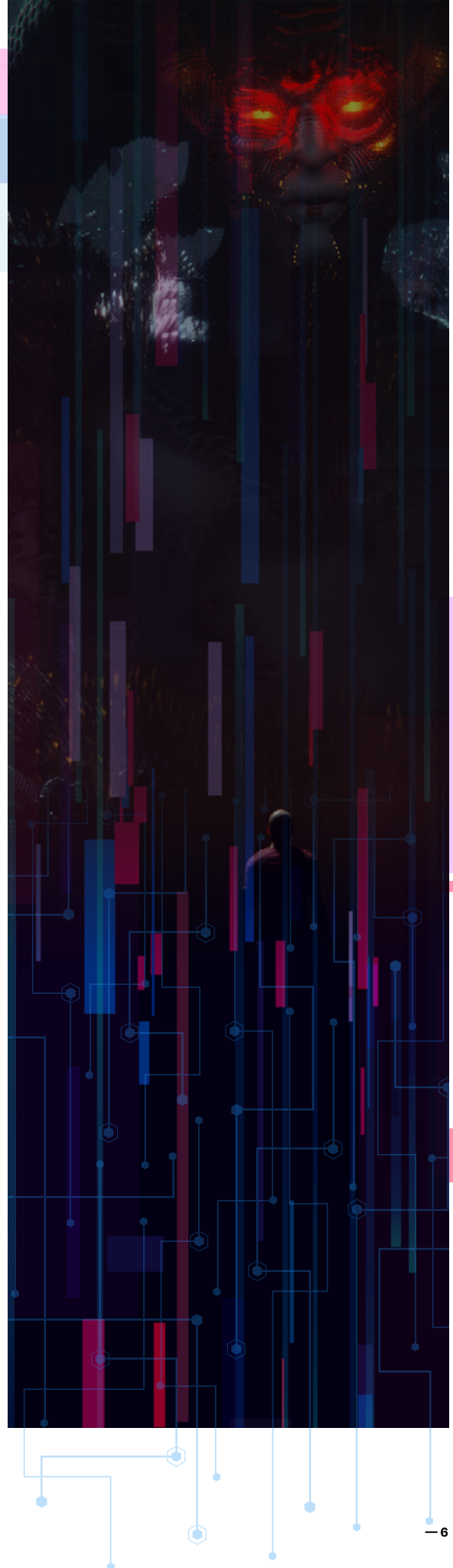
DON'T LET SUPERFLUOUS DATA GET IN THE WAY OF QUICK ACTION.

Trying to sift through too many key performance indicators (KPIs) can lead to information overload and data that lacks context. As with any communication, useful error reporting has to be concise and focused on what's relevant and actionable. Superfluous data from tracking too many metrics just gets in the way.

For game development, the most important KPIs are:

- **Number of errors:** This leading indicator immediately points to the health of your development processes. It lets you isolate faulty workflows or programmers who may need training or greater supervision.
- **Number of users impacted by an error:** Prioritize fixes through careful analysis of this essential metric. An error may only be affecting a few power users, but if they happen to be your game's biggest in-app purchase (IAP) consumers, your priorities are clear.
- **Errors introduced with a release update:** Deltas that occur soon after a patch or update to your game help focus investigations on a typically limited amount of newer code.
- **Errors per application module:** This measurement can yield a double benefit, offering more focused research as well as identifying individuals or teams whose performance is better or less advanced than their peers.
- **Errors occurring within five minutes of gameplay:** Glitches that get in the way of new players getting hooked on your game are going to be much more costly than errors that occur further down the road.
- **Crash- and error-free statistics:** Monitoring the percentage of sessions without crashes allows you to measure your game's aggregate session stability, and you can track this metric over time to see how stability is evolving. Keeping tabs on how many users didn't experience crashes is useful for tracking your players' experience.

For your error monitoring processes themselves, the most important KPI is mean time to detect (MTTD). How long does it take before you're aware of bugs that may be eating away at your success?



→ POINT #3

THE IMPORTANCE OF AUTOMATION

MINIMIZE MANUAL PROCESSES THAT INTRODUCE UNCERTAINTY.

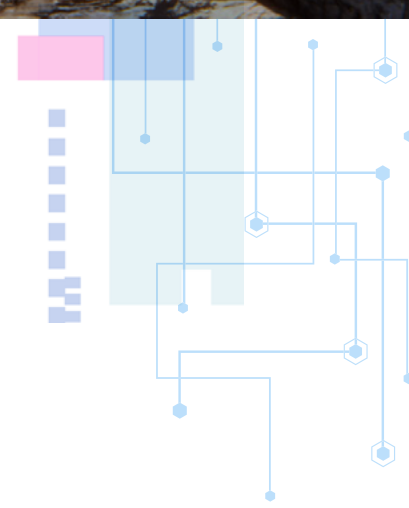
Automation is the backbone of error monitoring and reporting, and it's key to virtually all DevOps practices.

Game code errors are cold, hard, objective events. They occur on specific devices running specific apps on a specific OS under specific conditions. Collecting and presenting that data in a useful way requires minimizing manual processes that introduce uncertainty.

For example, do you want a customer support tech collecting and categorizing bugs at the end of the day, or do you want bugs prioritized automatically the moment they happen? Do you want someone researching the context of each error, or do you want as many relevant data points as possible, automatically collected and immediately available?

Automated error tracking can collect data from development, QA, and production environments, capturing crashes and exceptions from every client, console, engine, and server platform that your game or game component runs on. This approach typically creates structured, searchable error reports that you can configure for your particular needs, giving you better insights into where to look for issues in your code.

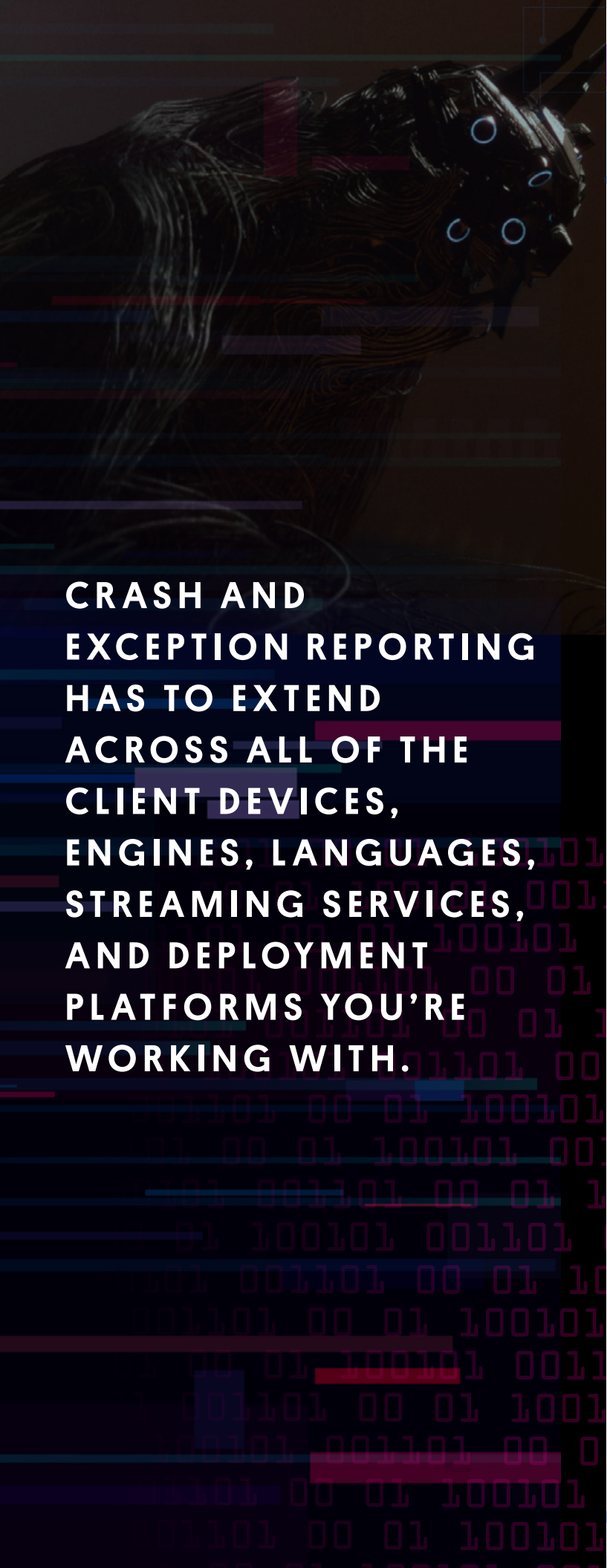
Importantly, this level of reporting lets you catch issues that users haven't noticed yet that are still impacting performance. Adding these kinds of automated error tracking capabilities is rarely expensive, and even a small studio can benefit tremendously – especially if your QA team is a part-time intern, or simply you, working in the kitchen late at night.





→ POINT #4

CHARACTERISTICS OF AN EFFECTIVE SOLUTION



CRASH AND EXCEPTION REPORTING HAS TO EXTEND ACROSS ALL OF THE CLIENT DEVICES, ENGINES, LANGUAGES, STREAMING SERVICES, AND DEPLOYMENT PLATFORMS YOU'RE WORKING WITH.

LEARN THE MUST-HAVES FOR ANY ERROR MONITORING AND REPORTING APP.

As with any DevOps tool, effectiveness is often a function of ease-of-use. A complex UI may not affect a developer, but if the developer asks an entry-level support person to generate a report and then also has to spend time explaining how to do it, the solution's effectiveness drops considerably.

A number of features should be considered as must-haves for any error monitoring and reporting app. Crash and exception reporting has to extend across all of the client devices, engines, languages, streaming services, and deployment platforms you're working with. And of course, it has to be compatible with your programming languages and tools, with a plug-in for your development platform. It ought to offer flexible deployment options such as hosting in a multitenant or dedicated instance, or deploying on-premises.

In addition, you'll need to know:

- **The exact cause of an error:** As much as possible, reports should include the methods and functions that generated an error along with a detailed stack.
- **Which errors are related:** Similar bugs should automatically be grouped and categorized based on their similarities, as well as around user-defined criteria. For example, you'll probably prioritize crashing bugs, then assigning different types of errors to different developers. This capability should include some degree of deduplication to handle multiple reports and notifications. At times, too much data can be as bad as too little.
- **How to triage effectively:** Can you configure alerts so that the right people know what's going on as soon as possible?
- **How to zero in on the error:** Are the search and query capabilities flexible enough that you can find errors using a wide range of attributes?

→ POINT #5

VITAL STUDIO-WIDE BENEFITS

NO GAME CAN SURVIVE WITHOUT EFFECTIVE ERROR MONITORING.

Long ago, after cranking out a batch of code, a programmer would usually have to go back and comb through their work to find where they misspelled a function or miscounted delimiters. This was a huge time sink, but soon enough, coding platforms evolved, and now it's possible to catch and fix these issues almost immediately.

Game code is infinitely more complex, but the imperative to find and eliminate errors in early development stages remains strong. Every studio needs to do this in order to:

- Accelerate development cycles
- Minimize patches and emergency fixes
- Spend more time creating new features and product improvements

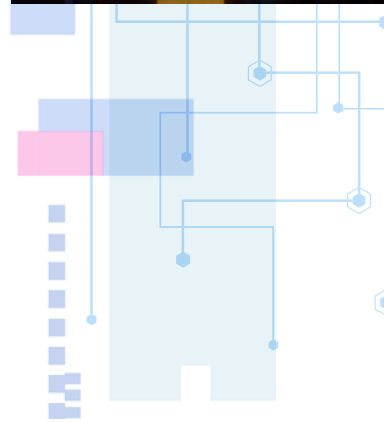
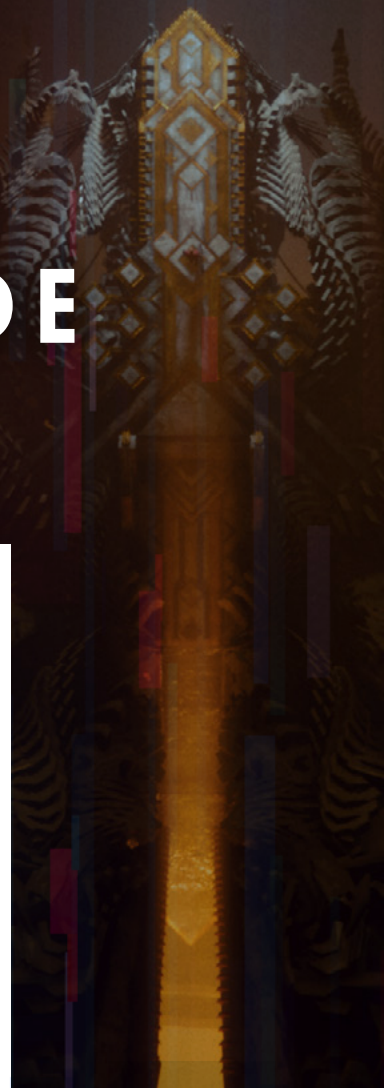
Often players haven't progressed to a level where a bug lurks in wait shortly after a release. The error is there, but it hasn't been reported yet.

Increases in monthly active users (MAUs) can also lead unexpected bugs to surface, and it's difficult to anticipate errors related to high player counts when you're testing with a smaller cohort in-house. More players mean more error logs, so you'll have to work harder to stay on top of everything.

When automated error tracking meticulously tracks code exceptions, QA and developers can spot invisible issues and fix them before they adversely impact the player experience. Whether you're building games or creating AR filters, user experience is what drives customer retention, and, in turn, your success.

In game development, eliminating crashes and performance-robbing exceptions is *everything*. The competition to retain players with a great user experience is stiff – no game can survive without effective error monitoring.

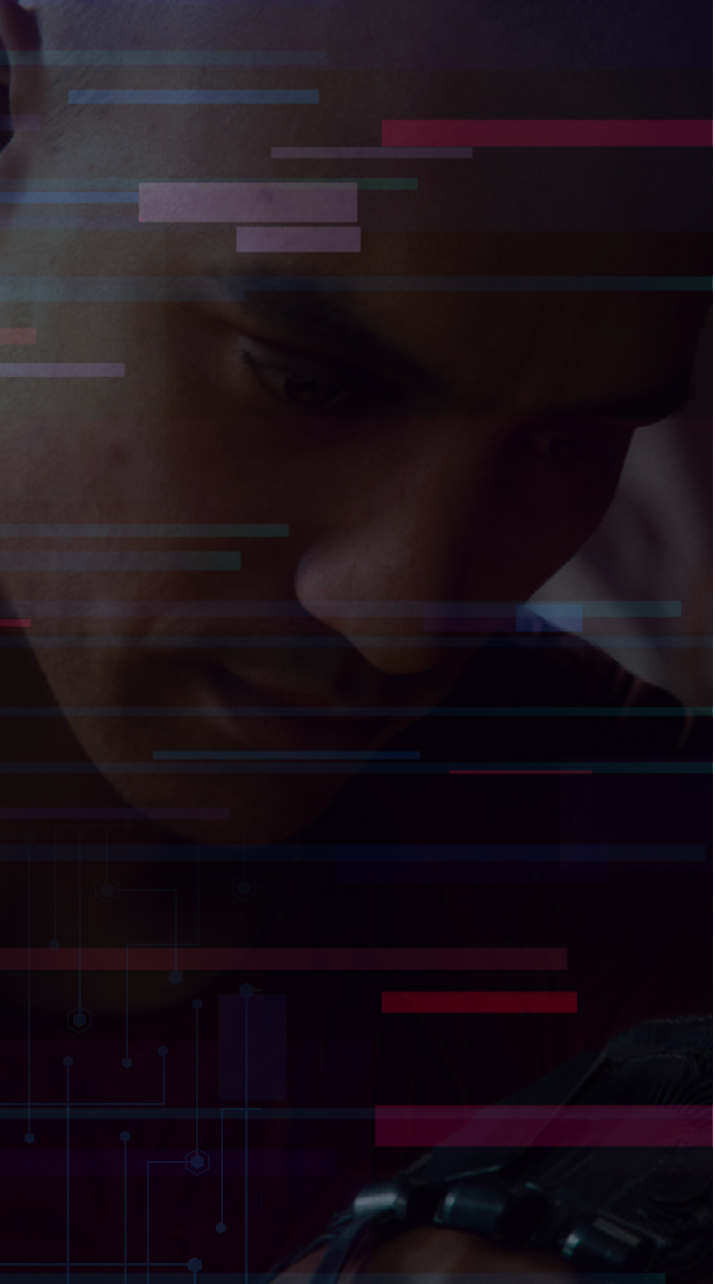
Of course, with timely, automatic error monitoring and reporting that groups errors and shows the detailed context in which they occurred, mean time to repair (MTTR) lessens considerably. Developers spend more time on new content, and overall game quality – the user experience – surges.





→ POINT #6

LEVERAGING CUSTOMER SUPPORT



IN-GAME BOTS LINK ERROR-MONITORING, PLAYERS, AND DEVELOPERS.

Despite the most rigorous QA and testing, errors will persist, and help from players can be a huge help in finding fast fixes.

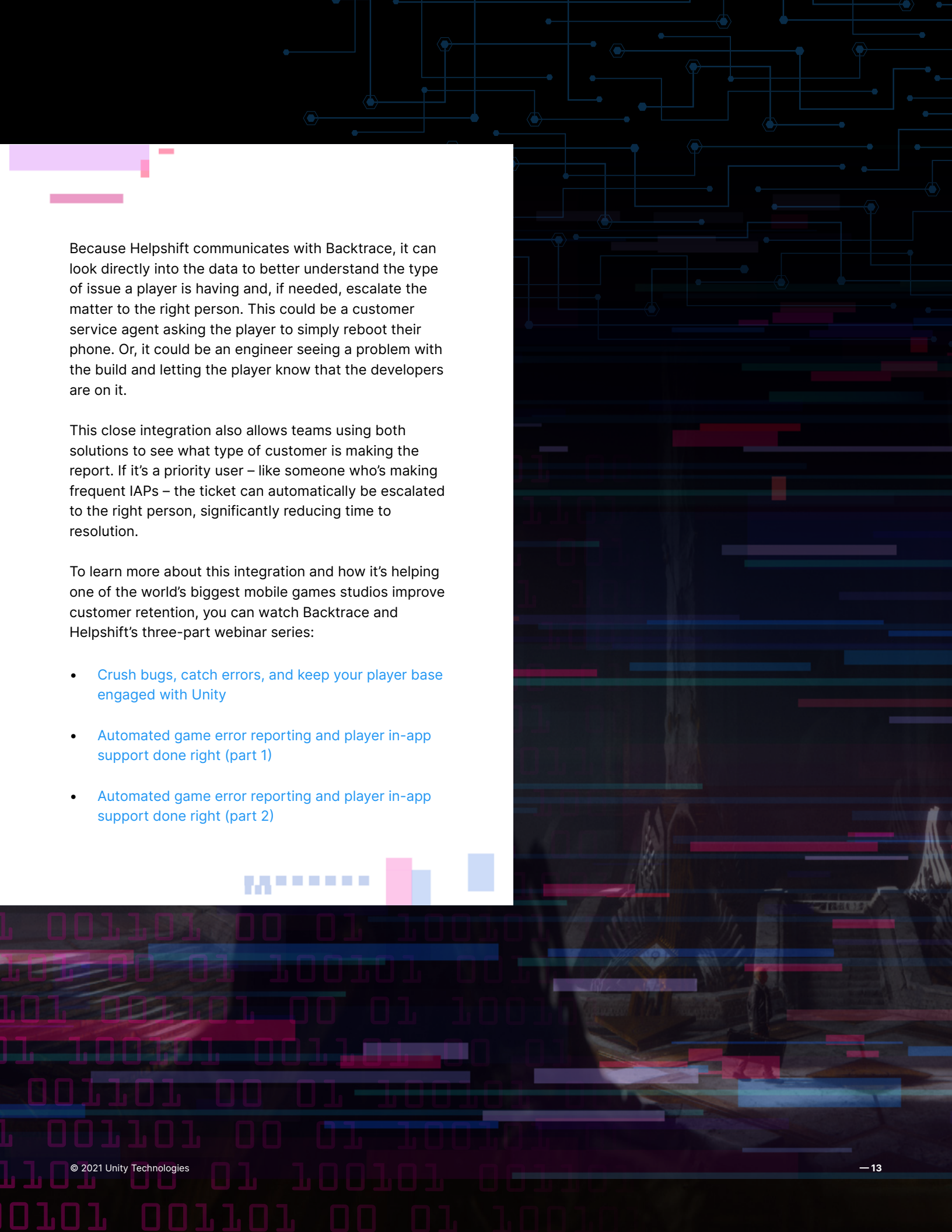
When your error reporting pipeline feeds directly into your customer success pipeline, both teams benefit. And when you add automations into the mix, the process only becomes more efficient. Players can be part of the bug-fixing solution without leaving the game – significantly reducing churn.

One example of a highly efficient automated error reporting integration is the relationship between [Backtrace](#) and [Helpshift](#), two of Unity's Verified Solutions Partners. Backtrace automates crash and error reporting, while Helpshift is an automated customer support app that's fully integrated into a game.

Over the past year, Backtrace and Helpshift have been working on a partnership to provide the next generation of help for engineers, support teams, and gamers. They've built an integration between their two products that provides customer support with full visibility when a user reports an error.

If there's a serious error, a dialog box can pop up and ask the player to confirm the issue and provide additional context. If it's a known bug in the error monitoring system, the dialog can inform the player and tell them how soon to expect a fix. An automatic follow-up communicates when things have changed.

WHEN YOUR ERROR REPORTING PIPELINE FEEDS DIRECTLY INTO YOUR CUSTOMER SUCCESS PIPELINE, BOTH TEAMS BENEFIT. AND WHEN YOU ADD AUTOMATIONS INTO THE MIX, THE PROCESS ONLY BECOMES MORE EFFICIENT.



Because Helpshift communicates with Backtrace, it can look directly into the data to better understand the type of issue a player is having and, if needed, escalate the matter to the right person. This could be a customer service agent asking the player to simply reboot their phone. Or, it could be an engineer seeing a problem with the build and letting the player know that the developers are on it.

This close integration also allows teams using both solutions to see what type of customer is making the report. If it's a priority user – like someone who's making frequent IAPs – the ticket can automatically be escalated to the right person, significantly reducing time to resolution.

To learn more about this integration and how it's helping one of the world's biggest mobile games studios improve customer retention, you can watch Backtrace and Helpshift's three-part webinar series:

- [Crush bugs, catch errors, and keep your player base engaged with Unity](#)
- [Automated game error reporting and player in-app support done right \(part 1\)](#)
- [Automated game error reporting and player in-app support done right \(part 2\)](#)

