2021 Guide to

# Faster Continuous Integration Builds

By: Jim Holmes

INCREDIBUILD

# Table of Content

# We Hate Slow Builds Already—But Here's Another Reason To

Slow builds are a headache for every project team that has worked on any moderately complex software project. Delivery teams struggle to get in a stable cadence of work: specify requirements, discuss the work, do the work, build and check the software, push out to the appropriate environment. Getting to a point where your team is comfortable with that cadence is hard! Anything that slows that cadence directly impacts a product's overall quality and value.

Harder yet is to maintain that cadence as the project continues to grow and mature. More features are added, dependencies and complexity increase, and any number of other issues hit home. Too often, the build process becomes the major impediment to a project's success. Have you been a part of a project where the all-inclusive build is only done once a night or, worse yet, left until the weekend?

Living in this sort of environment turns in to a serious risk for the project's quality: delays in builds cause teams to cut corners in critical areas. Tests don't get written—it takes too long for the red, green, refactor flow of good test-first development, much less test-whenever-development. Tests don't get executed as part of every local build—who can wait the extra time on top of an already slow build? Tests get left out of the scope of execution—I only have enough time to run these few tests; forget checking integration tests for side effects!

In this paper, we will look at a number of quality-related concerns around slow builds, and we will investigate several approaches to addressing these concerns.

# The Timebox Rules Everything

24 hours per day, seven days a week, four weeks per month. You can't get around those time constraints. You can add more workers, but still, there's only so much you can do in a fixed amount of time. As build cycles lengthen, teams have to consider what they can tack on to the process. Often tradeoffs are made which put a project's overall quality at risk, in favor of focusing on adding in more functionality, linked resources, additional projects, etc.

In the following section, we'll take a deeper look at the ramifications of those decisions.

# Improving Your Delivery

It doesn't matter if you are doing agile, waterfall, scrummerfall, chaos, or some other methodology of software development: Your team and customers benefit from you working to improve how you deliver software. Smoothing your delivery process means less drama at release time. Improving your delivery process means less angst and fear when adding features or modifying your codebase. Improving your delivery process means your customers get better value faster—it's not a myth!

Continuous improvement [1] is a concept focused on helping teams and organizations get better at how they deliver software. You don't have to buy off on everything described in the referenced article; simply focusing on a few things may be enough for you—and likely your build cycle should be near the top for consideration!

Feedback cycles are one of the crucial themes running through continuous improvement, Lean software, agile software, and other philosophies/methodologies. A feedback cycle (or feedback loop) is an expression of the time it takes for a change in the system to be reflected. Feedback cycles exist at the long/slow scale (did a customer's bug get fixed in the latest release?) too much smaller/faster (did the unit test I just wrote pass when I ran it?).

Much has been written on feedback cycles; the three posts listed below would be a great starting point for learning more:

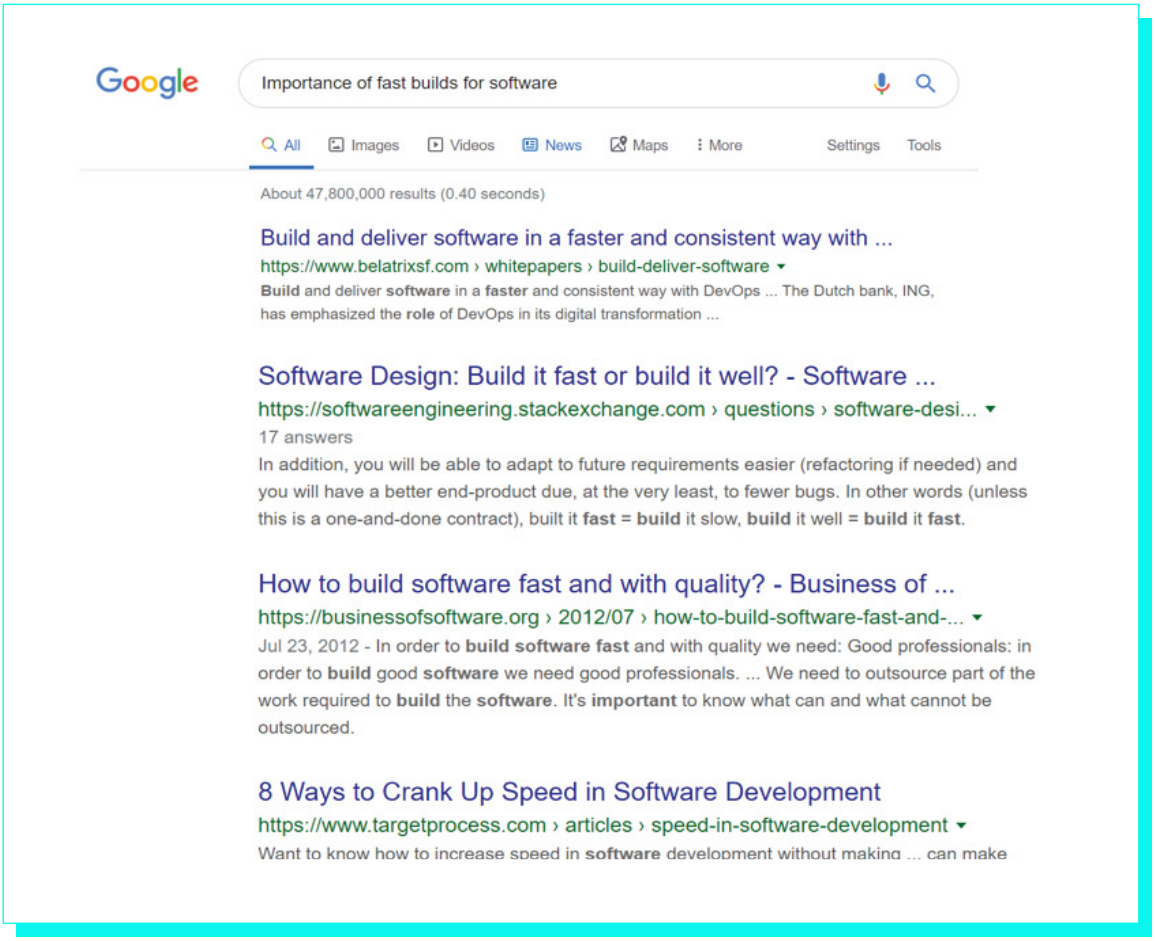
- Scott Ambler's [2] article on how faster feedback cycles are central to Agile
- Ron Jeffries [3] calls out the importance of testing to good feedback
- Gary Bernhardt's post [4] on blisteringly fast feedback

1 See a great introductory article at http://leankit.com/kanban/continuous-improvement/
2 www.ambysoft.com/essays/whyAgileWorksFeedback.html
3 ronjeffries.com/xprog/what-is-extreme-programming/
4 https://www.destroyallsoftware.com/blog/2014/tdd-straw-men-and-rhetoric

# Industry Leaders on the Power of Fast Builds

Thankfully, the software industry has thought leaders who have been emphasizing the importance of a fast build cycle for a long time. There is a solid body of writing and speaking in this area. Michael Feathers' wonderful standard Working Effectively with Legacy Code repeatedly talks to cutting build time as part of reining in a codebase. James Shore's and Shane Warden's seminal work The Art of Agile Development (and other books) speak of a ten-minute build cycle.

A simple Google search shows us how thinking around this has changed over the last decade or so. You can see back as early as 2001 where Joel Spolsky and others were preaching the importance of a daily integration build.




The industry has continued to evolve since then, with the focus now on Continuous Integration. The Continuous Integration [5] philosophy speaks to building and integrating on a daily basis—but encourages teams to look to much more frequent build/integrations. Entire tool platforms have evolved to help teams embrace more frequent builds/integrations/deployments on a nearly

5 https://www.thoughtworks.com/continuous-integration

continuous basis, often every five to ten minutes. This dramatically speeds up the feedback cycle. Teams get near-instantaneous pass/fail status when integrating instead of waiting days!

Our industry is continuing to evolve and we have matured to understand even daily feedback on integrating, building, and testing the entire system may be too long. Jez Humble[6] and other industry thought leaders are preaching the value of continuous delivery, not just integration!

# Impacts of Slow Builds

Slower builds have many impacts across a team's efforts. Those impacts hit morale, costs, and the ability to create high-quality software.

## Slower Delivery of Value to Customer

At the end of the day, nothing matters if software teams aren't shipping value to their customers. Everything that's a roadblock to progressing code to production and handing it off to end users is an impediment. In Lean Software Tom and Mary Poppendieck, famously ask "How long does it take to deploy one line of code to production?"

What happens if you have a serious defect in your production environment, say one that puts your revenue at risk—a crucial security flaw, perhaps? Can you get your issue fixed, promoted, built, tested, and deployed in a reasonable amount of time? Or are you so constrained by your build process that it might take hours or -- worse yet -- days to roll that change out?

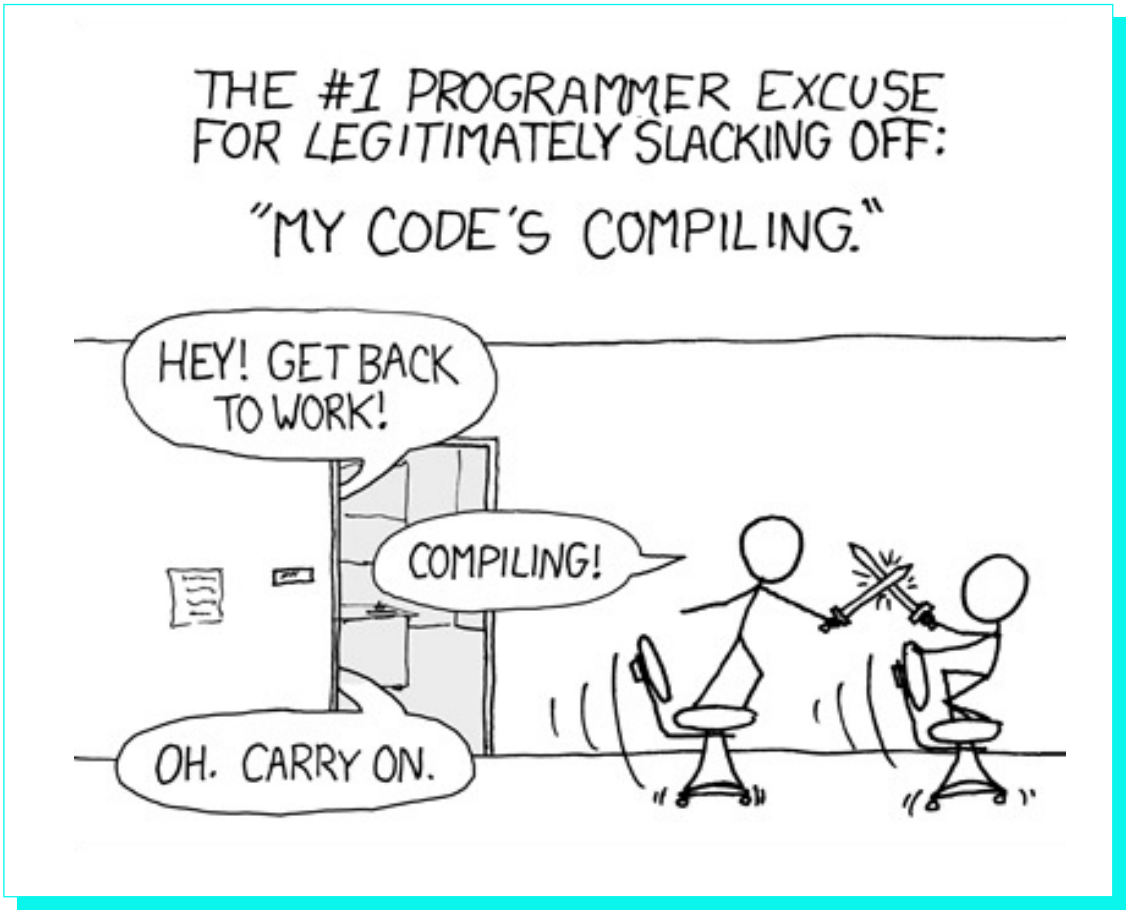
## Dropping Out of "The Zone"

Software testers and especially developers love to debate about getting "in the zone." The Zone is a highly focused state of concentration where a person turns in to a "super producer." Programmers cherish getting in the zone because they're able to rapidly solve complex problems, knock out difficult tasks, and deliver value in extremely short periods.

Not everyone's happy when the build speeds up. XKCD's infamous comic[7] is a humorous take on a team that fills the time with slow builds in "interesting" ways. That said, perhaps you can find another way to meet your teams' needs for creative anachronism!

6 http://continuousdelivery.com/

7 https://xkcd.com/303/ (Creative Commons BY-NC 2.5)




Source: xkcd

## Worse Maintainability

Humans avoid painful situations. Draw what metaphors you will (physical injury, emotional stress, SharePoint development projects), but brutally long build cycles certainly fit in this category. Desire to avoid dealing with painfully long integration and test builds can actually have a negative impact on the maintainability of projects. Teams will put off the pain of trying to run an integration build which incorporates multiple components, branches, projects, etc. if that build takes too long to run. The integration will be put off to a nightly build, then to a weekly build, then to once an iteration, and so on.

Teams in this state have surrendered to the complexity, rather than working to resolve it. Martin Fowler has a terrific article on doing integrations as frequently as possible.[8] His theme is "if it hurts, do it more often," meaning work through the painful parts of your delivery pipeline to focus on delivering value as quickly as possible.

8 http://martinfowler.com/bliki/FrequencyReducesDifficulty.html

## Automated Tests Ignored

Sadly, slow builds can also cause teams to rely less and less on automated test suites. Think about it: if you're already suffering with long build times, a natural (but WRONG!) reaction is to avoid doing anything to add to that build time. This has been proven out in the software industry as teams write fewer tests and run those tests less frequently. This leads to increased regressions, especially in more complex systems, as teams lose the benefits of their automated test suites.

One practical example of this is ModuleWorks, a provider of CAD/CAM components. ModuleWorks' complex distribution architecture requires building 32 different sets of releasable binaries to support customer configurations. Their build times were so long that developers were only integrating once a day—and their GTest suite for their C++ code was rarely being run.[9] Only after implementing Incredibuild, discussed below, were they able to implement continuous integration and automated testing.

## Devs Narrow Scope of Executed Tests

Another way teams will reduce build times is to cut the scope of tests they're executing. Instead of running all tests in unit or integration suites, developers might limit the test execution scope to the specific component they're working on at the moment.

This bad practice results in a loss of exposure to potential side effects as teams write code. Think about it: one of the main reasons for a suite of automated tests is to provide a safety net for ongoing development! A well-written, thoughtfully constructed suite of tests gives us confidence to alter parts of the system knowing with utter certainty that tests lock down behavior in other areas of the system.

Retalix, a division of NCR Retail, offers a streamlined suite of systems for online, mobile, and in-store retail systems. Retalix had a suite of 15,000 unit tests that took 12 minutes to execute—after a complete optimization of every developer's system. 12 minutes to run unit tests is far too long, and Retalix's development teams weren't utilizing the tests as much as they should have. (Read the referenced case study to see how Retalix brought their test runs down to 1 minute 20 seconds![10])

9 ModuleWorks Accelerates Testing & Continuous Integration Enabling Advanced Manufacturing, Incredibuild Case Study at https://www.incredibuild.com/case-studies/moduleworks

10 See "Retalix Case Study" https://www.incredibuild.com/case-studies/retalix

### Static Analysis and Other Quality Approaches Ignored

Tools like Visual Studio's Code Analysis Tools, NDepend, SonarQube, or similar items, can be an extraordinary help in ensuring a codebase's quality and maintainability. These tools flag quality issues like complexity, dependencies, code metrics, etc. and can be used as a gate for moving your code through your delivery pipeline.

Unfortunately, these tools take time to run, which means they're certainly not going to be utilized in environments where builds are already the bottleneck to productivity and delivery.

## Speeding Up Builds Manually

Teams learn to overcome slow builds in many ways, other than just continuing to suffer with them. Lots of effort, sometimes spent over a year or more, is distributed across reconfiguring complex build cycles, standing up new hardware, and reworking entire test suites.

### More Hardware Fixes Everything!

Naturally, many organizations first look to hardware for speeding up slow builds. In many cases, this can make sense; it's easy to swap out a build server or two. What happens when that doesn't improve the problem much, though? Besides, hardware takes time to procure, set up, configure, and maintain.

If you're looking to create a new pool of remote agents, then you'll need to handle the procurement, setup, configuration, and maintenance for those as well! Moreover, integrating those remote agents into your build system can be extremely difficult. You'll need to make sure all your tool chain handles parallelization: build server, build system, test frameworks, etc. Teams who are looking for a balance of payoff / effort may have a hard time justifying the effort involved. Regardless, you are still constrained by the speed of your strongest build hardware—which may not be enough. Buying additional hardware that may only be needed for peak times may be an expensive proposition, rather than optimizing your already-existing hardware and existing processing power.

### Manually Split Into Units / Tasks

Builds sometimes can be sped up by manually splitting the build job in to multiple pieces and executing those pieces manually.

This solution is fraught with peril, though. Teams, already burdened with enough work delivering software, now have to manage the build process manually through dependencies, updates, and new tasks. It's a costly effort to handle this sort of thing manually, and forces the teams to build deep expertise in the build pipeline.

Dependency management is a painful but crucial part of any software system. Platform dependency management tools like Bundler for Ruby, Maven for Java, NuGet for .NET, and others, are complex tools that are meant to offload the pain of dependency management. Teams splitting up builds now own much of this dependency management, since the build pipeline won't know how to deal with dependencies between separate build jobs.

### Cost of Effort

It's critical that teams keep in mind the costs of efforts around speeding up the builds. Many teams spend months, if not years, tuning their build processes. Emanuil Slavov's "Need for Speed" talk at ISTA Conference 2015 [11] laid out a story of an 18 month journey to cut test execution time from three hours to three minutes. That's a terrific improvement; however, an incredible amount of effort was expended.

Was that effort worth the expense? For Emanuil's team it certainly was; however, one can only wonder what other options they might have explored which could have given them back hundreds or thousands of hours to spend focusing on delivery instead of infrastructure.

## Speeding Builds and Tests With Incredibuild

Organizations looking to speed up their build times have a fine line to walk between accomplishing the speed increases and throwing too many people and resources at the effort. Emanuil Slavov's effort, mentioned earlier in this paper, took 18 months to achieve their goals. They realized tremendous results after a long period of hard work updating infrastructure, build configurations, data management, and test automation strategy. While the results were terrific, perhaps that effort could have been better spent directly delivering value if other options for build optimization were available.

### Overview of Incredibuild's Architecture and Solutions

Incredibuild offers teams and organizations just that capability: rapidly scale out build infrastructure to dramatically improve build times—all with a minimal investment in setup, configuration, and management of build resources. Incredibuild enables organizations to quickly stand up pools of build agents with a central build coordinator. Developers, testers, or anyone with Incredibuild installed to initiate a build which is distributed across the pool of agents. Splitting a long-running build in to tasks and executing those in parallel reaps huge rewards with a minimal investment in time and resources.

Additionally, other long-running tasks can be handled by Incredibuild. Game development organizations are using Incredibuild to speed up their graphics and audio rendering jobs, tasks

11 http://www.agiletestingdays.com/session/need-for-speed/

that are well-known to take inordinate amounts of time. Other organizations leverage Incredibuild to dramatically cut automated test suite execution times and speed up code analysis.

It is important to understand that Incredibuild's agents are not the same as those for Jenkins, Team Foundation Server, or TeamCity. Those tools' agents are complementary to Incredibuild's agents. For example, Incredibuild frees Jenkins build agents from the performance limitations imposed on it by its own hardware. The two sets of agents can be used in conjunction with each other to truly transform a build workstation into a system using potentially hundreds of processor cores and gigs of memory.

Incredibuild's capabilities don't require organizations to invest in multiple new systems to scale out their long-running tasks. Instead, organizations can use Incredibuild's agents on existing systems. Incredibuild will only distribute tasks out to systems which have an appropriate amount of free CPU cycles and memory. This means you can have agents running on systems idle in a corner, on systems lightly used by administrators or program managers, even on systems actively used by developers. Later below, you'll also see how those agent systems don't even need source files, libraries, or build tools installed.

Where needed, you can easily scale out Incredibuild's agent pool to cloud resources to get hundreds, or even thousands of cores handling your build chores!

## Scaling Out to Cloud

Cloud platforms like Amazon AWS, Pivotal's Cloud Foundry, or Microsoft's Azure lets organizations rapidly scale out their system infrastructure. Why shouldn't those same platforms support an organizations build and delivery pipelines?

Frankly, Incredibuild doesn't care where you have your build agents defined, it only needs to be able to communicate with them. Incredibuild can use any cloud-based infrastructure as long as it supports the provisioning of Virtual Machines. Generally that means you'll need some form of VPN established with those cloud-based systems; however, if you're using those same platforms for your system development it's likely you'll already have something in place.

Things are even easier if you are using Microsoft's Azure: Incredibuild has out-of-the box support for working with Azure. Set up your VPN, push Incredibuild agent software to your Azure instances and Poof! you're able to pull those instances into your agent pool.  You can even use Incredibuild to automatically provision new instances if you need them.

This ease of use runs throughout Incredibuild's entire feature set. The goal of Incredibuild is to ensure organizations spend less time setting up and configuring build infrastructure, and more time focusing on delivering value to their customers.

## Low-Impact Solution

Incredibuild focuses on ease-of-use so teams can quickly get back to work. Installation and configuration of Incredibuild is very simple, even in large-scale environments. Default settings are carefully chosen so teams can see great benefits with an out-of-the-box installation. For example, sensible minimums are pre-defined for CPU utilization boundaries, available RAM, disk caching, etc.

Incredibuild even takes it a step further by providing tools to create an agent install package that holds project-specific configurations. Running that installer will set up the agent software, set project-specific customizations, and establish a connection to the Incredibuild controller—all in hands-off mode.

Finally, pool agents don't need any additional project-specific software installed. You don't need to install Visual Studio, Xbox development environments, gcc, dependent libraries, templates, or anything similar. Incredibuild's process virtualization will bundle those resources and push them to the agent. The agent receives that bundle and is able to perform all the assigned tasks without ever having to rely on local resources. This is a huge time-saver for organizations since teams never have to worry about updating libraries, patching tools, or applying service packs to complex systems. (Note: System-level resources such as anti-virus and operating system patches still must be managed!)

This approach has enabled Incredibuild's customers to rapidly and dramatically improve their build and packaging times, as can be seen in several real-world examples.

## Some Real-World Time Improvements

Organizations around the world have reached out to Incredibuild for help in improving long build cycles in a wide range of domains.

### Maturing from Long Builds to Continuous Integration

Algotec builds the Carestream Vue product suite, in use by thousands of medical facilities across the globe. The Vue suite helps optimize medical imaging and is built on a complex codebase of over 400 C++ projects, many with over 30+ files in each project. Source files make extensive use of processing-intensive features like macros, TLB imports, and custom build steps.

As a result, the suite could only be built once per night. Developer integrations of changes were slow and painful, and resulted in very long feedback cycles to the developers. Long feedback cycles often lead to a downward spiral of quality, and can be a show-stopper in a team maturing to processes like continuous integration or delivery.

Algotec took the easiest possible step to implementing an Incredibuild solution: simply installing an agent on each developer's system to utilize existing systems on Algotec's network. This one step garnered improvements of 90% in smaller projects. Larger projects saw a smaller improvement at first: 140 minutes down to 40 minutes. Algotec and Incredibuild teamed up to optimize dependencies and other project settings, finally paring builds down to 34 minutes, an 80% improvement.

Realizing these time savings allowed Algotec to move into a continuous integration flow, running their builds multiple times per day. Algotec was also able to begin implementing automated testing, something that was infeasible without the extraordinary timesavings from implementing Incredibuild.

### Time Savings Equate to Better Testing

ModuleWorks builds extremely complex, highly customized solutions for Computer Aided Manufacturing industries across the globe. Many of their customers require very specialized configurations, resulting in ModuleWorks having to build at least 32 different binary sets for each major release.

Because of their complex build requirements, ModuleWorks developers were constrained to integrate sequentially. Worse yet, because of the length of the build (several days!), developers were only able to integrate a few times per week. Additionally, tests written using Google's GTest framework were suffering due to length of execution. New tests weren't being written, and existing tests weren't run frequently enough to head off increasing quality issues.

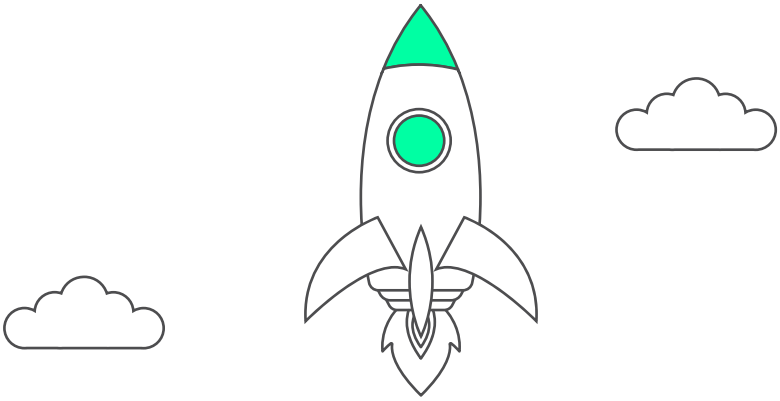
Simply by adopting Incredibuild, ModuleWorks was able to cut their build times from 30 minutes to three. Test execution times for their GTest suites dropped by 86%. Before Incredibuild's implementation, a full build and test cycle would take over three hours; after bringing in Incredibuild a full build cycle is less than 30 minutes.

Saving over 2.5 hours per build cycle has energized ModuleWorks' developers to do more automated testing. Better yet, ModuleWorks has seen significant drops in regressions since the test suites are growing and are executed more often. Overall quality has dramatically improved for ModuleWorks' shipped products.

# Don't Tolerate Painfully Long Builds; Deliver Better Products!

Long build cycles (regardless whether it's build, build and test, or build and test and other tasks) are a huge hit on a team's productivity. As you have seen in this paper, long build times also impact an organization's level of quality they are delivering.

Any organization can choose to invest large amounts of time and resources into cutting their build cycle. Alternatives exist to dramatically cut build times without such dramatic investments of time and capital. Any organization looking for a fast win on improving build times ought to consider Incredibuild. Better yet, go get a free trial license of Incredibuild and install it. The minimal effort in setting up and configuring Incredibuild makes it easy to try it for yourself!



## Install Incredibuild for Windows or Linux at

## https://www.incredibuild.com/free-trial

### or from within Visual Studio navigate to File > New Project > Build Accelerator