From Chaos, through Fear, to Confidence.

It's important for a business of any size to have confidence in the software it produces; from a start-up focused on rapidly delivering new features, to a large enterprise that prioritises performance and stability. At each end of the spectrum future growth (or defending your current position) depends on releasing working software at a cadence which meets the demands of the marketplace.

JetBRAINS

endjin

work smarter

Many organisations don't have rigorous Application Lifecycle Management processes in place – they have evolved to their status quo based on heroic effort, a modicum of arrogance and a touch of luck. The problem with this approach is that it works...until it doesn't; everything will be just fine until things start going inexplicably wrong. Suddenly, bugs in the production environment can't be replicated in the development environment; or deployments that used to take just a few minutes of manually copying binaries now seem to take hours because new features require a series of environment tweaks to run properly.

Then, one day, code that ran in the development environment, once deployed, doesn't run in the production environment, and the site goes down. This deployment can't be rolled back because the developer responsible for the deployment forgot to back up the original binaries. After some frantic debugging, with a manager pacing to-and-fro in the background, it is discovered that a new feature requires an environmental tweak that hasn't been documented.

Once the site is back up and a calculation has been made of the amount of money lost by the outage, an all hands development meeting is called to figure out what went wrong and how to stop it from happening again. The outcome of the meeting is that one of the developers is blamed for not communicating the new environmental requirement and placed on performance review. A mandate is made that all developers must communicate such requirements into a run-book used by the individual doing the deployment.

The Business also decides that to mitigate the risk to income due to an outage caused by a deployment, all future deployments must be made at 2am on a Sunday morning and a new Ops team is formed from the existing group of developers who are now responsible for the nightshift at the weekend to do the deployments. The morale of the development team has taken a nose dive and individuals are worried about being identified and punished for causing issues in the production environment in the future. The new Ops team are worried that they will be blamed for failures of omission by the development team.

A month or so later, the next planned release is postponed because the scheduled work has not been completed. Two weeks later, after much pressure is applied from The Business, the development team grudgingly agree that they are feature complete and ready to do a deployment at the weekend.

The deployment goes without a hitch – new environmental tweaks required are documented in the run-book; but over the next couple of weeks the number of calls to customer support have increased. The large number of support requests start to absorb a high percentage of the available development resource in order to investigate and close the issues. Half way through the month the next deployment is postponed by two weeks because none of the features in development are near completion and another two weeks later the team still believes there are two more weeks of work to be done to get to feature complete.

At this point there hasn't been a new deployment in two months, questions are being asked higher up in The Business about the development team's capability and voices of concern are being raised about how "fit for purpose" the existing platform is. Something has to change, but no-one is quite sure what is needed.

The above is a fictional, generalised account of several real world projects we have been engaged to help rectify. It is a common situation that many, many companies, both large and small, sleepwalk into.

What seem like common sense decisions, such as creating Development and Operational teams end up creating walled gardens where information flow is stunted and a culture of blame and blame mitigation is fostered, under a general cloud of fear and opaque processes. This is precisely the type of situation the DevOps movement is trying to address and rectify.

Within its first 4 years, RecruitmentGenius.com has filled over 60,000 jobs for 9,000 companies. To deliver this volume of growth they had to aggressively develop their online platform. Like any good start-up they rapidly needed to adapt their software architecture and product feature-set to take advantage of commercial opportunities as they arose. This strategy proved to be very successful and helped them grow the business to become the UK's No. 1 online recruitment agency in 2013.

Planning the next phase in Recruitment Genius' life, from start-up to SME, The Board of Directors wanted to invest in a new platform, designed to harness all the competitive advantages cloud computing has to offer, to enable them to offer new products and services to their customers in a timely fashion, while keeping a lean and agile mentality. They had noticed that not only did it seem that the last few feature releases had been harder to deliver, as if their process had developed a level of friction that was affecting their ability to ship new features, but there had also be an increased occurrence of issues in the production environment that had raised concerns about the way they managed the quality of the output from development.

Following a recommendation from one of endjin's other customers, Recruitment Genius approached endjin to assist them with their platform migration to Windows Azure and to help improve their end-to-end software development processes.

More fascinating than the technical details of the migration and the use of cloud technologies to enable a business to elastically scale to meet peak demands, is the story of the 18 month journey endjin embarked on to help Recruitment Genius move from chaos to confidence;

Before a line of code was changed, endjin ran a workshop to perform an end to end, drains up review of the existing platform with Recruitment Genius' development team. Several interesting issues were revealed. Among them was a realisation that architectural drift has occurred; there was a disparity between what the team believed were the processes implemented by the system, versus what the code was actually doing.

Next endjin embedded with the team to observe how "work" flowed through their development process, from initial requirement inception, through development and into production. This revealed that they were working in a highly chaotic fashion; requirements were often partially formed ideas that didn't clarify the end goal state or acceptance criteria.

The specifications of feature requests were often framed in terms of the underlying database constraints and how this affected the code in layers above, rather than considering the desired end user experience and working down through the layers to achieve it.

Because the requirements weren't fleshed out it was difficult to estimate the work involved in order to deliver the feature request. The development team always delivered, but they were expending heroic levels of effort to do so.

The team was using Team Foundation Server Version Control, but found its implementation of branching unwieldy, so had decided to carry out all development on the main code branch. The Team Foundation Server was also hosted on a fragile virtual server which had developed a habit of crashing, which left the developers in a disconnected state, unable to commit their code and collaborate; they had developed a sceptical and untrusting relationship with their tools.

For ease of development, every project was included in a single solution file, which stressed even the most high specification development rig. A clean build took around 12 minutes to run and the solution was lacking test coverage. Manual tests were carried out against the development environment.

Because there were no defined acceptance criteria, development carried on until it was declared "good enough" and then it was released into production.

Deployments were carried out by performing a build on a local development machine. The build artefacts were then transferred over a remote desktop connection to the production server. Deployments were undertaken with a certain amount of trepidation as there had been a history of deployment issues.

Having reviewed the production process, endjin spent some time with the management team and sought information on two fronts: how new work is conceived and prioritised; and how development progress is reported to The Board.

After some time to digest and reflect upon all the information gathered, endjin proposed two concurrent work-streams.

The first was a re-engineering effort to migrate the platform to Windows Azure. The second was an Application Lifecycle Management programme to overhaul the delivery practices to focus on delivering business value through code quality, along with a new RAG format for board reporting of development activity.

The journey can be broken down into the following steps:

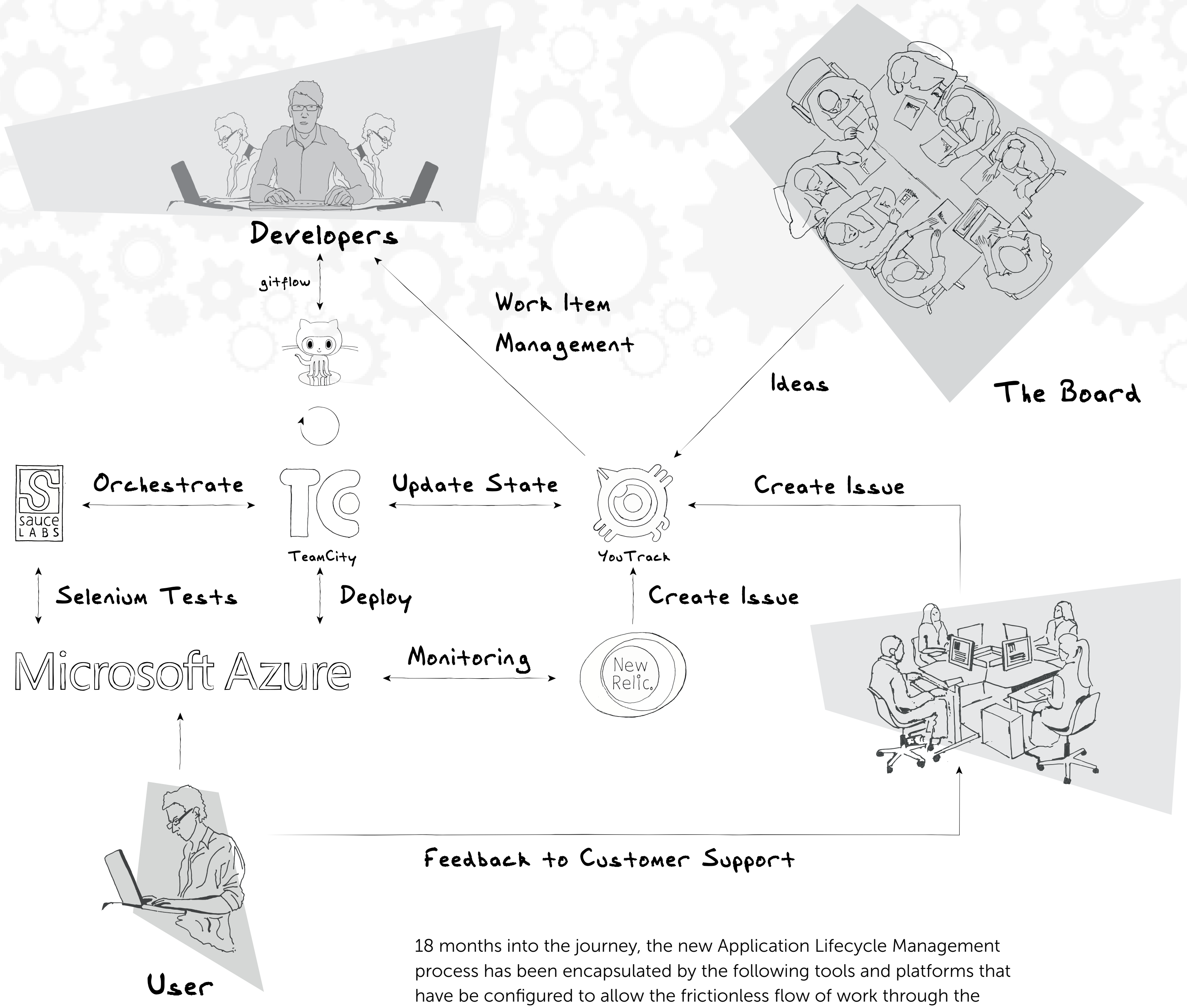
Step 1: Restructure the source code tree

The first step was to create some order and reasoning behind the layout of the source code tree, so that members of the development team know where code, build artefacts and binary dependencies should be stored. This should prevent the source tree from becoming more chaotic over time.

Step 2: Split up the solution

Over 4 years, dozens of projects had been created inside a single solution; the sheer volume of code put a huge amount of strain on the IDE and generally slowed the developers down. One of the tacit aims was to make adding any new feature, or making a change, as fast as when the organisation was a young start up.

The solution was split up into five smaller logically separate solutions, this significantly reduced the time to open & compile the solutions. Once in the smaller solutions, aggressive refactoring was carried out (with the aid of ReSharper) to enforce standard architectural principals (layering, single responsibility, separation of concerns). Redundant code was extracted from the solution and archived.



18 months into the journey, the new Application Lifecycle Management process has been encapsulated by the following tools and platforms that have be configured to allow the frictionless flow of work through the system. This quickly provides feedback about the various quality gates and enables rapid deployment to any environment.

Step 3: Migrate solution to GitHub

Once the solution has been refactored, the next step was to pick a reliable version control system that the development team could depend upon. GitHub was chosen, not only for its great branching support and useful features such as the ability to code review and comment on commits, and a wiki for storing information about the system, but also because it was an externally hosted solution, which meant there was one less internal system to support.

Step 4: Introduce MSBuild based automated build framework

We replaced the error prone, manual build steps with BuildFx, endjin's MSBuild-based automated build framework process. Executing any unit or integration tests as part of the build also made a huge difference to the developers' confidence levels. They know that by running a single command file, within a few minutes they can do a full clean build and test run, and all the build artefacts are packaged into a single location, ready for deployment.

Step 5: Install TeamCity

The next logical step from having an automated build process that developers can run locally, is to ensure that this build process is executed every time a developer commits a code change. If a code change causes a build failure, everyone should be notified and a fix should be applied. The most common problems it helps to detect are when developers fail to commit all files in a change set, or when a seemingly small change affects the behaviour in another part of the system and causes a unit test to fail.

One of the other major behavioural attitudes it helps to eliminate is the "it works on my machine" mentality – because a TeamCity builds starts in a pristine state it will expose any missing files, or old versions of binary dependencies that contaminate a developer's machine.

TeamCity was installed and configured to automatically build each of the solutions, every time a change was made. Additional build steps for finding code duplicatesand running code coverage reports on unit test run and build features, such as AssemblyInfo Patcher and Build Files Cleaner were configured, which help improve the overall quality of the solution.

With TeamCity 8.1 pre-tested commit functionality has been added for Git via the new Automatic Merge build feature. This is more commonly known as a gated check-in and means that changes are built on a private branch, and those changes are merged into the main branch only if the build finishes successfully. This ensures that other members of the team don't have to deal with broken builds as the main branch should always be in an unbroken state.

Step 6: Introduce GitFlow

Now the foundations required for providing a feedback loop to developers about the quality of the code they are committing has been established, the next step was to reduce the friction and complexity of collaboration on a single codebase. GitFlow sets out to answer the following questions that plague every software development team:

How do you:

- Keep your repository tidy?
- Have consistency between projects?
- Get your developers up to speed quickly with our procedures?
- Enable the team develop a feature in isolation from the main branch, but also allow them to collaborate?
- Stabilize before a production release?
- Manage production releases?
- Manage production hotfixes?

The solution GitFlow proposes is a prescriptive branching model, born of experience, which answers the questions above, and a set of Git Extensions that allow you to manage and automate work through that model.

TeamCity enables GitFlow support via its branch specification feature; TeamCity will automatically monitor each of the dynamically created GitFlow branches and build any changes committed to that branch.

For a more detailed description of Gitflow and configuring your TeamCity environment see the following blog posts: http://blogs.endjin.com/tag/gitflow/

Step 7: Install YouTrack

Once the main engineering practices had been established, it was time to focus on the overall development process & methodology for creating, elaborating and delivering work. The existing process relied heavily on email. There was no overall list of work, and no way to capture estimates or track progress.

YouTrack was selected because of its low friction approach to creating and tracking work items, and also for its integration with TeamCity. When a developer committed a code change, they could enter the ID of the item they were working on; this allowed anyone looking at TeamCity to click a link and go directly to the item in YouTrack.

Endjin ran a backlog creation session where the whole team collaborated on gathering existing feature ideas and entering them into YouTrack as a single backlog. The Product Owner transcribed product & feature ideas that were stored in multiple different locations (note books, draft emails, spreadsheets, and documents) in to YouTrack so that there was a single product backlog that could be easily groomed and prioritised.

Step 8: Introduce User Stories and Behaviour Driven Development

Once there was a unified backlog of work, the next step was to convert them from generalised feature requests to user stories that explain the actors involved, the context and reasoning behind the feature and a set of acceptance criteria.

This change helped on a number of fronts. It helped the developers understand the reasoning behind some of the feature requests; it started to instil the notion of the personas using the platform, and make developers think about implementing features from a user experience perspective: from the user interface down to the database, rather than the traditional approach which limits the experience based on the constraints made by underlying technology choices.

Once the team understood how to articulate requirements in terms of user stories, the next skill to adopt was the translation of the user stories into a series of executable specifications that the developers could create to help them elaborate what behaviours were required to deliver a user story. These executable specifications were written in a mixture of MSpec and SpecFlow, both of which were run by TeamCity as part of the continuous integration process. Code coverage reports were used to identify areas of code that were not exercised by the existing specifications, indicating that some scenarios and edge cases were missing and would need to be dealt with.

Step 9: Introduce Iterative & Incremental development

Now that they had a backlog of well understood work, and a process for turning those ideas into provable code, the next thing to tackle was to evolve the mind-set from developing a big stack of features and releasing them in one go every month or so, to building iterative and incremental changes and releasing them as quickly as they pass through defined quality gates. This is both a behavioural and mind set change. The process was instigated by using the Agile Board features of YouTrack to plan what work was going to be delivered on a weekly basis. It allows easy visualisation of what the team is working on, how close work themes are to being complete and whether they are in a position to be deployed (changes moving from a feature branch to the main release branch in GitFlow).

Step 10: Introduce automated deployments via TeamCity

Work was starting to flow rapidly through the system, so the next bottleneck to tackle was the ability to rapidly deploy to any given environment. Endjin integrated their DeployFX for Windows Azure framework, which is responsible for both the configuration management and deployment process. It was originally integrated as a build step, but since TeamCity 8.x it has been packaged up as a Meta Runner.

To enable automated deployments, the development team made changes to the platforms' configuration files, so they could be automatically updated by the deployment process depending on which environment had been selected as the deployment target. The TeamCity continuous integration process was also modified to publish deployable build artefacts that DeployFx for Windows Azure could consume and deploy.

Deployments were now so straight forward that any member of the team (with the correct security permissions) could instigate a deployment; developer, tester or product owner. Because deployments now took minutes, dozens were performed a day, which rapidly reduced the fear that had been associated with manual deployments and increased confidence that each production deployment would work perfectly.

Step 11: Introduce Web UI Automation with Selenium and Sauce Labs

Now that changes could automatically be deployed into any environment, the team decided they wanted to create an extra quality gate: a series of end-to-end UI automation tests that would run on demand. By this point the team had adopted SpecFlow for all integration based specifications and endjin build a custom SpecFlow plugin which allowed the developers to tag different scenarios with a list of browsers and platforms that wanted to run the tests against. The SpecFlow plugin is available at: https://github.com/endjin/Endjin.Selenium.SpecFlowPlugin

Rather than manage their own browser test environment, the development team decided to use Sauce Labs a cloud based Selenium test infrastructure SaaS product. Sauce Labs will automatically provision your browser / OS combination, run your test suite against your web application and report the results. Videos of each test run are recorded to allow you to diagnose any test failures. TeamCity was used to orchestrate the running of these tests; the results are surfaced as standard unit test results showing success and failure.

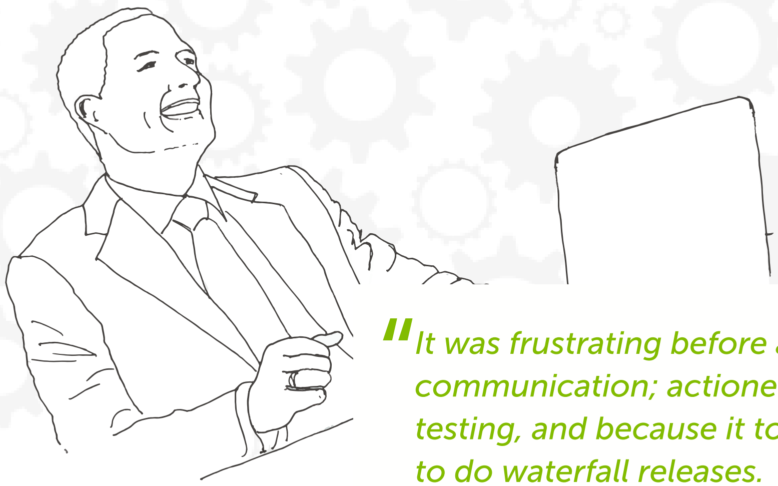
Sauce Labs have released a TeamCity plugin that allows orchestration of Selenium Tests and will also gather test result videos and screenshots and enables TeamCity to publish them as build artefacts: https://github.com/rossrowe/sauce-teamcity-plugin

Step 12: Install NewRelic for application monitoring

The final feedback loop created was extensive application monitoring of the platform. New Relic was chosen because of its tight integration with the Windows Azure Platform. The development team introduced semantic logging into the platform, which allowed them to reconstruct the context of any issue in the test environment in order to diagnose, debug and fix the issues. An additional build step was added to the deployment process to notify New Relic that a new deployment had occurred – this enabled the development team to observe longitudinal data to see if any specific deployment caused performance issues.

Errors raised by New Relic were turned into backlog items and entered into YouTrack; these could be prioritised at the start of each iteration depending on their level of urgency. Problems that customers reported, were also captured by the customer support staff and were also captured and prioritised in the same manner.

These two feedback loops ensured that problems were captured, prioritised, fixed and deployed rapidly, ensuring the platform continually improved, became faster and more stable as time goes on, rather than the more common occurrence of long term performance degradation.



"It was frustrating before as ideas would get lost in an ether of communication; actioned in no particular order; lacked robust testing, and because it took so long to deploy we were forced to do waterfall releases.

But since Endjin automated everything, all our ideas are logged and prioritised using YouTrack and we avoid a mess of email communication as the team collaborate together in one central place.

GitFlow ensures our developers don't tread on each other's toes and we are in control of which features make it into any given release. TeamCity takes care of automated testing and deployment to Azure in minutes rather than days.

Automating these tasks ensure we rapidly deliver product enhancements to our customers. The outcome is our developers are happier, our customers are happier and that makes my life as a product owner much easier."

Geoff Newman Managing Director,
Recruitment Genius

Who are Recruitment Genius?

Recruitment Genius has rapidly grown to become the UK's largest online flat-fee recruitment company. Their business model has helped fill over 59,704 jobs for over 8,800 UK companies for an astonishing flat-fee of only £199. Many well-known brands such as Tesco, Shell, Sony Playstation and Ford use Recruitment Genius, as well as many other small but growing companies, to place everyone from board level to shop floor. recruitmentgenius.com

Who are endjin?

Established in 2010, endjin has become the number one Microsoft Windows Azure Partner in EMEA, delivering projects using cloud, mobile, desktop and embedded technologies in retail, financial services, and consumer applications. endjin.com

Technologies used

TeamCity YouTRACK

New Relic. Microsoft Azure

sauce LABS

endjin
work smarter

Strategy Experience Development Cloud

2 Leathermarket Street, London, SE1 3HN
t 020 8720 7287 e hello@endjin.com
endjin.com