Crazy Fast Build Times

Daniel Worthington-Bodart (http://dan.bodar.com/) BCS SPA 2012 Conference, 3 July 2012

Introduction

Talk is the result of an article published earlier this year, which caused a high level of hits on Dan's web site.

System builds that take 10 seconds used to be thought of as very acceptable. Not any more. You can make improvements.

Measure everything!

Reason for the talk

We got here without realizing why. Picture a frog in water that gradually heats up – it never feels uncomfortable. Build times gradually increase – who cares? At what point should you do something about it?

History

In 1999, Kent Beck famously said that a ten-minute build was OK – anything shorter would not give you time to get a coffee!

Today we are still in the same situation despite vastly faster machines. What gives?

Examples

A five-year old app with 150K lines of code took roughly 45 minutes to build: now 3 minutes or less.

A recent 20K app builds in 15 seconds, including unit tests, acceptance tests and deployment.

These techniques are not only applicable to green field developments.

Divide and Conquer

A very simple approach. Split the build into fast and slow portions. This doesn't really solve the problem – what tends to happen is that the broken long build never gets fixed.

Really, the application is telling you by the long build times that it needs to be split into components that can be built separately. This will also allow you to test in a much more focused way.

Testing

Reduce duplication. Don't keep testing the same workflow.

Production quality dummies – in-memory versions of collaborating components, such as databases. Run unit tests on these and include just a small number of unit tests on the actual component, so that its equivalence to the dummy can be confirmed.

Contract tests – police the APIs between components.

Libraries

Just reading in a single XML file to configure Spring for a given unit test took 14 seconds. It turned out that it could be reduced to under a second by using lazy instantiation, which was amortised over thousands of tests to drastically reduce the build time.

Replacing GWT with a simple REST framework reduced the amount of code by 86% and allowed use of HTMLUnit for testing instead of Selenium. The pyramid of technology choices clearly has a huge impact on build and test times.

Mixed-mode is possible – only use JavaScript for those tests that require it, for example.

Containers

Starting and stopping an application server per test case is a huge overhead.

Using a dynamic port per test case avoids wait times. An in-memory HTTP interface tests the units in isolation from the server, which is even better.

Hardware

The same build on the same hardware runs 15% faster on Macs than on Windows, and another 15% faster under Linux.

Future

Dan has looked for speed improvements elsewhere. The Java compiler can be invoked with switches to avoid output of intermediate files to disk.

Question

Build tools used (Maven, etc) are not as fast as Ant – should you not use Ant instead? Probably not very significant. Scala compilers need to be improved in efficiency. The main thing is to measure and to be aware of tradeoffs available.

Why not just advise pushing as much as possible down to unit tests? Definitely the way to go; but you do need end-to-end testing. Try to pick the right API level to run acceptance tests – e.g. drive programmatic interfaces rather than HTTP interfaces. Dan's organization no longer distinguishes between different

levels of tests. Key question about each test is “does it give us confidence”. Other important questions are “is it reliable” and “does it give rapid feedback”?

What do you do with the time you’ve saved? Well, there’s never enough time, but an example story: an Oracle database field was misused (null and empty string are not the same, but were treated as such by the software). Caught in the final integration test. Shouldn’t the real Oracle database have been used in testing all the time? Answer: over three months of development, the dummy database probably saved £25000 of developer effort. The cost of fixing this late-detected defect was at most one developer day (£1200). Definitely worthwhile.

Don’t rely on automated testing too much. Developers still have to try out the app from the UI to give a level of confidence that the change has worked. And rapid deployment is a very important factor in enabling them to do this.

How can you avoid build times drifting upwards again? Giving a shit. Someone on the team should be the champion for this. Be passionate about keeping builds streamlined and lean. In Dan’s projects, the build time is a very important statistic, which is tracked along with memory usage, story points delivered, defects etc.