Continuous Delivery with a legacy VCL application

Who am I?

Who is albelli?

What is the application

Albelli Windows Editor, used by albelli, bonusprint, önskefoto and fotoknudsen in EU, Vistaprint in EU, NZ, AUS, SG, JP and IN

Legacy ≠ obsolete or maintenance, instead it is an older application that was not designed according to modern software engineering practices.

Actually, current use (for albelli brands) exceeds 20k users per day, more than 800 k orders per year.

How do we update our customers?

Why is CD hard for older applications?

* Unit testing might be unavailable / impossible to write
* Build pipeline might be non-existent
* Manual regression testing takes a lot of time before release, increasing time before feedback

Consumer software

* No fixed hardware profile, due to amount of use and location of customers, direct contact hardly possible

State of affairs 2 years ago

* Automatic build of main executable on commit, but nothing more, no linked projects
* All 3rd party code (including pre-compiled packages) recompiled on every commit (sources were included in project search path)
* Almost 300 warnings, and 450+ hints in codebase.
* Building a release meant triggering all translations separately
* Gathering installers
* Change XML files to announce updates (tool to do in bulk, still a manual step)
* Upload installers to S3
* Upload XMLs

Next to manual regression testing, this process could take up to four weeks, also due to lack of faith in the quality of the product

Remove warnings

* + Hints and warnings can be bugs!! (E.g. unitialized variables)
  + New warnings unspottable
  + Theory of broken windows
  + Boyscout rule
  + Coding standards / formatting guidelines

Introduce unittesting

* Quick test = quick feedback

Creating the build pipeline

CI Server  
Jenkins, Finalbuilder, TeamCity, ContinuaCI.  
Finalbuilder in my experience has better Delphi integration, TeamCity was already available.

For Release Candidates and Master branch builds:

Commit -> build .exe -> translate .exe (x8) -> Pack installers   
- Change from dcc32 to msbuild for version number integration  
- every build is unique  
- All code resides in 1 repo  
 - based on *What* is the artefact we’re deploying. All dependencies for the same artefact reside in the same repository.

* To keep branch information intact across branches in TC
* Because even though the build is meant for building a single artefact, there are in between steps and artifacts possible (in our case, exe, translated exe, installer, set of all installers)

Automating the deployment

* Automate the modification of xml files
* Create NuGet Packages for both installers and xmls

Introducing UI testing

Especially with older applications, Unit Testing might not always be possible rwithout heavy refactoring. With visual oriented applications or applications relying heavily on databases, unit testing might even be impossible.

Also, refactoring just for the sake of introducing unit testing might not be a desirable way to go. This code is battle tested and hardened over the years of use by hundreds of thousands of customers. (Fun fact, albelli reached 1.5 million active users (order at least 1 product last year, order volume is xx % from the windows editor).

What is it that you are testing? With UI testing we are testing the result of the underlying code, but is the UI the end result? For us, it is the actual book being produced. As we have had probems with the actual rendering of the PDF, while everything in the model and UI seemed OK, testing the PDF became vital to our confidence in the editor. After all, when 1) each time the action is performed in the same manner 2) the UI reacts in the same way and 3) the PDF is produced indentical to the previous time (an known good original), then we know that the feature under test (FUT) is behaving as expected.

Make installers as stand alone as possible. Trade off size <> dependencies on first start = less behaviour to test when running tests.

Introduce Ranorex and Eggplant

First attempts with Eggplant

* Unstable, finicky, depending on sreen resolution

Ranorex scans UI parts, and can address UI components directly. Through IAccessible interface

Not depending on resoution, works out of the box for almost all VCL components

Tests written in C# .NET, Visual Studio Community compatible

Some component not accessible? Or not deep enough to adress different parts of the component? Implement IAccessible yourself!

Introduce BDD

Introduce SpecFlow

BDD - > Generate test framework -> Build the test