Incremental Development and Regression Testing

INCREMENTAL DEVELOPMENT REQUIRES AUTOMATED TESTING. OTHERWISE, IN TIME, THE BURDEN OF REGRESSION TESTING WILL BRING DEVELOPMENT TO A CRAWL

Why we must automate testing

By Hans-Eric Grönlund

Incremental development has dominated the field of software development since the 1990's[1]. For good reasons; the idea to evolve a system through small incremental releases has many advantages, including:

- It reduces risk
- It allows feedback to guide the development process which helps in building the right system (as opposed to the system as it's designed up front)
- It opens up the development process and encourages customer involvement
- It can bring real business value earlier

Few people today question the value of the incremental development approach. Still, developing software incrementally comes with a consequence that may seriously affect a team's ability to deliver high quality software rapidly.

A Simplified Model

Let's look at an example to see where the problem is.
Suppose we're building a system using a traditional method. We do an up-front design of the system, we build it, and at the end we test it for acceptance.
For the sake of simplicity assume that the effort of acceptance testing the system is illustrated by the box in Image 1.

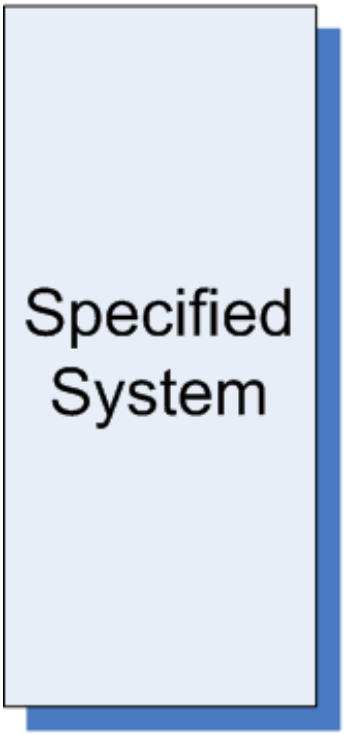

Image 1: The amount of acceptance testing using a traditional approach.

2

Now, suppose we're creating the same system using an incremental approach; Instead of building it all in one piece we develop the system incrementally, for instance in three iterations.

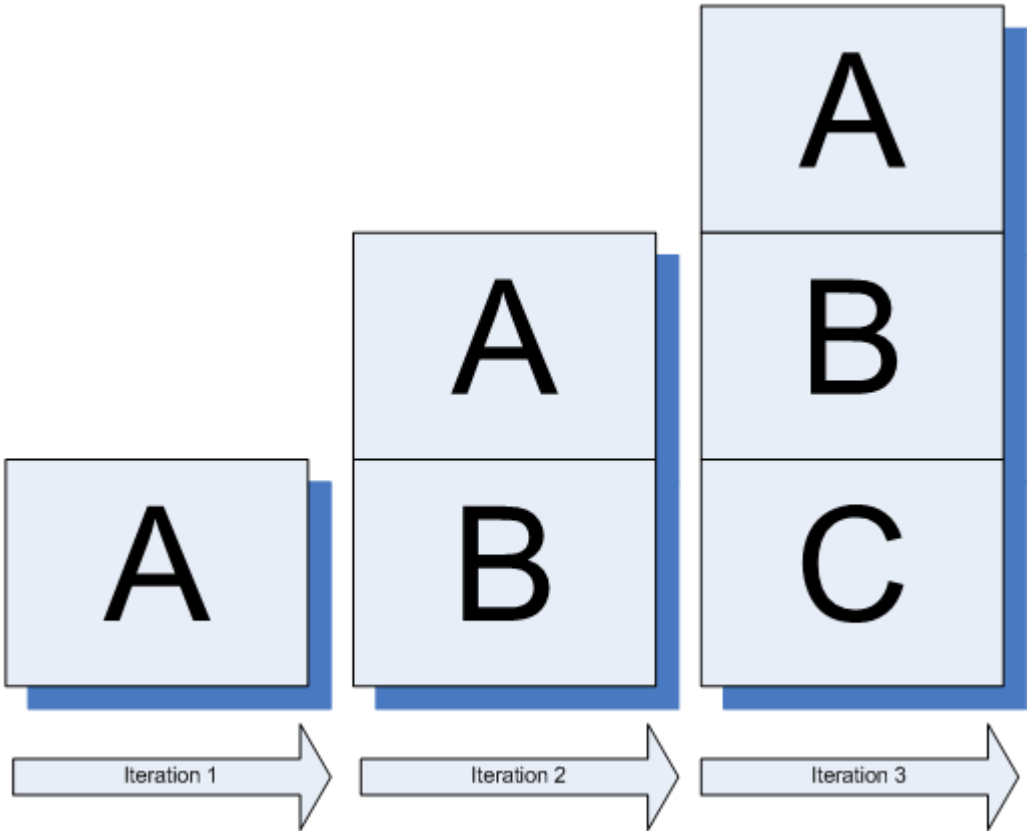

Image 2: An incremental approach: plan to release the system in three steps.

The system is developed in three iterations, each adding a new chunk of functionality. For reference, let's call them A, B and C respectively.

The Problem

Incremental development dictates that each increment be released—or at least potentially releasable. For a system increment to be releasable we need to do a complete test of it; not only the new functionality, but all previously built features for regression[2]. So building the system incrementally requires more testing. The question is how much more?

Let's assume that acceptance testing a piece of functionality and regression testing it requires the same amount of work. Let's also assume that the three chunks of functionality are of equal sizes. We could then compare testing effort between the two approaches.

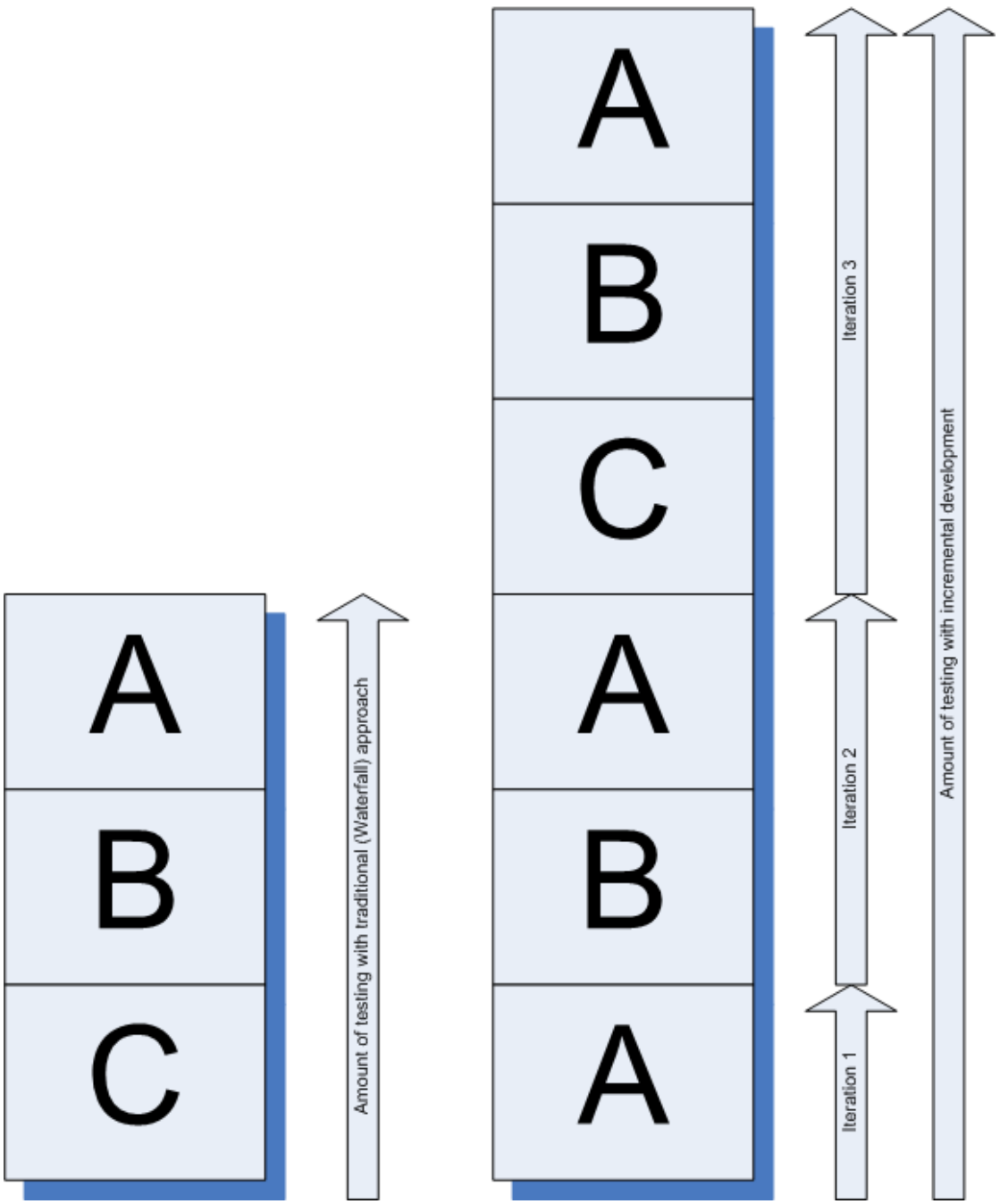

Image 3: Comparing testing effort between traditional and incremental approaches.

Given the assumptions, testing effort with an incremental approach and three iterations is twice that of the traditional approach. At first sight the factor two may seem like a reasonable price to pay for increased feedback, but the effect is non-linear and increases rapidly with the number of increments. With five iterations comes three times the testing effort, at seven the amount is quadrupled, etc. Table 1 shows how the testing effort in an incremental development grows with the number of increments.

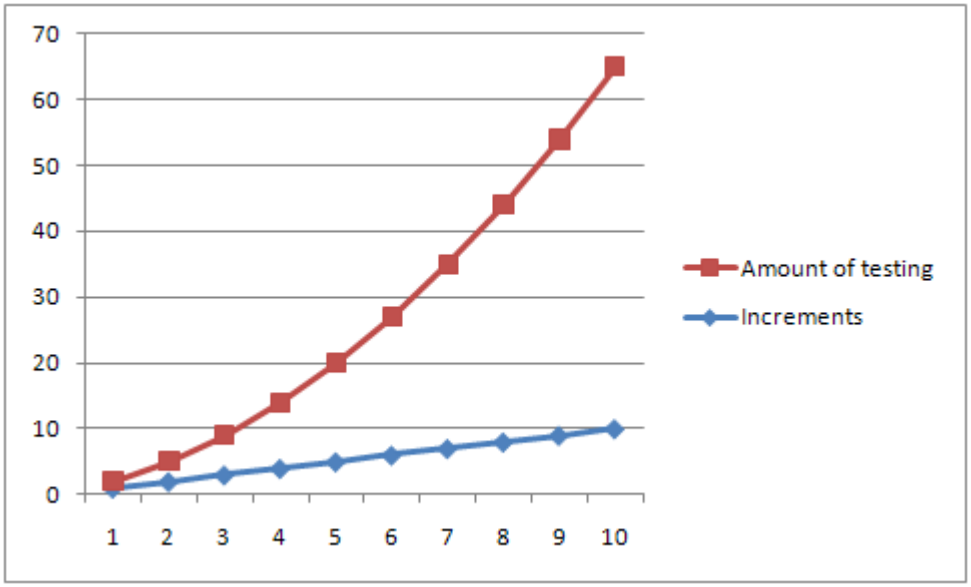

Table 1: Testing effort grows rapidly with the number of increments.

Of course, the assumptions made in the example aren't realistic. For instance, testing new functionality isn't exactly the same as checking for regression in old features. Also, the traditional approach of delaying acceptance tests until the end of the project will most likely result in a bigger effort than the simple model give the appearance of.

Still, the quadratic growth of regression testing is a very real problem with incremental development. A sneaky problem too; the effect is barely noticeable at the beginning, but the increasing load of testing that comes with each increment soon affects the team's velocity.

Possible Solutions

At some point the burden of acceptance testing gets so severe that the team is forced to take action. The simple solution is to ask for more resources, but that is neither an effective nor a long term solution. The amount of work is accelerating and keeping up is practically impossible.

Instead, what usually happens is that the team starts compromising with full regression testing. That too is a very unsatisfying solution. By not testing all previously deployed functionality, the team opens a way for subtle bugs to reach production. That will send the system into a spiral of ever decreasing quality, and poor quality is the main reason for low productivity in teams.

Another common way to deal with the problem is to reduce the number of increments by making fewer but bigger releases. This is giving up on incremental development and moving towards a waterfall method, nullifying most of the benefits; not the direction we'd want to go, as decades of experience have shown us.

So we need to release often, and we need to test all features when we do. That leaves us only one choice: automation, the only long term solution to the regression testing problem. A regression testing suite that takes days for a tester to run manually can be executed in a matter of minutes by a computer. This releases a lot of testing resources, which can be used to create more automated tests and trigger a positive spiral instead of a negative one.

How to Get There

The first thing we need to realize is that automating acceptance testing requires a lot of effort. It cannot be done half-heartedly. Many teams fail because they don't focus on the mission. They let one or two individuals work in semi-isolation to try and make it happen. The rest of the team carries on as before in an attempt to keep production of new features going.

This is wrong. Instead, the best way to ensure success is to “stop the line” and involve the whole team in the endeavor.

Next we need to define an implementation strategy and set goals. Should we strive for complete automation or are there parts of our system where the cost-benefit ratio is more favorable? Implementing automatic acceptance testing is always a lot easier in the beginning, while legacy[3] systems will be more challenging. In that case, use a divide and conquer strategy; Start with low hanging fruits and continue from there.

Depending on the chosen strategy you’ll need some kind of supporting infrastructure for your automated tests. There are plenty of alternatives of acceptance testing frameworks.
You can use scripts for batch-like systems, recording tools that invoke your user interfaces, tools like Fitnesse, xUnit, rSpec, etc to write acceptance tests in the same language that you write your code.
A good thing with the last category of frameworks is that they may be used to write acceptance tests before the features to be tested are even implemented. This is a huge advantage because the tests provide the best type of requirement specifications. They are therefore useful during development and not only for regression testing.

Do whatever works for you but be determined and be patient. Eventually, you’ll be glad you made the investment.

About the author

Hans-Eric Grönlund has been a passionate software developer since 1990. Today he spends most of his time as an IT consultant, leading agile teams using Scrum and Kanban.

Join him at his website www.hans-eric.com.

Notes

1. Actually, incremental development is as old as the field of software creation itself. It's been utilized in various forms since the 1940's, but the emerging of the so called waterfall models in the 1980's made such an impact that they are now considered "traditional". [CLVB03]

2. As anyone involved in software development can testify, accidentally breaking of existing features can be quite common. Even small changes can have unforeseen effects on a seemingly unrelated part of the system under development. What's worse is that because the cause-effect relationship is diffuse for bugs introduced this way, they can be quite troublesome to fix once discovered. For that reason regression testing is of utmost importance to ensure quality.

3. According to Michael Feathers' definition [MF05], code without test is legacy code. I use the same definition applied to systems: a system without covering regression tests is a legacy system.

References

- **[CLVB03]**: Craig Larman & Victor R. Basili, Iterative and Incremental Development: A Brief History, IEEE Computer Society, 2003 http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf
- **[MF05]**: Michael C. Feathers, Working Effectively With Legacy Code, Prentice Hall PTR, 2005
- Steve Freeman & Nat Pryce, Growing Object-Oriented Software, Guided By Tests, Addison-Wesley 2010
- Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code, Addison-Wesley, 2007
- Fitnesse Acceptance Testing Framework, http://fitnesse.org/
- Creating Automated Tests in Visual Studio 2010, Microsoft Software Development Network http://msdn.microsoft.com/en-us/library/dd380755.aspx
- The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber and Friends http://pragprog.com/titles/achbd/the-rspec-book