# C Sharp Enterprise Zone

# Chapter 2: Enterprise Application development concepts

An Enterprise Application in its broadest sense is an application that manages information in an enterprise. Typically they are the systems used in the daily operations of a business and are required to be robust, secure and scalable because a loss of uptime in the system results in a decrease in productivity of its users.

This chapter will first cover the development philosophies and practices that we value in developing enterprise applications, and then outline a set of infrastructure challenges that just about every enterprise application developer encounters. These challenges will be expanded on in the rest of the book.

## Development philosphies and practices to value

Over the years of delivering custom developed enterprise applications we have come to value certain development methodologies and philosophies very highly. These are important to cover as they have deeply affected the way Habanero was put together, and if you are to use Habanero in any capacity they are philosophies we hope you will value too.

### Source Control

For every kind of development using some form of source control is an absolute must. Even if you’re developing software as a one-person team a source repository is essential as it brings with it many advantages, such as:

1. It allows you to undo program-breaking changes easily. Because of this it allows you to experiment with an idea knowing that you can always revert if things get out of hand
2. It records changes made to the source code over time, allowing you to retrieve the source of a project as at a particular date, or to see exactly when a particular change was made (and by whom if necessary)
3. It allows multiple people to work on the same set of code at the same time without overwriting each others’ changes
4. It allows you to release a bug fix for the current live version of your software even while you’re developing new functionality
5. It becomes the de-facto master-copy of your code, which along a solid backup process means you always know that your source code is safe and sound (not on someone’s computer somewhere…)

Of course there are many ways that source control can be used and an astounding selection of tools that can be utilized. The important thing is that source control is used, not the particular tool to accomplish it.

At Chillisoft we use Subversion for all our source control needs and we’ve found it to be a top class solution as well as being free and open source which means it has the advantage of being supported by many other open source tools. To serve the repository we use VisualSVN Server and as our standard front end we use TortoiseSVN (no Visual Studio integration though, we have never found that to be a particularly helpful thing).

Some of the big distributed open source projects like Mozilla and Linux Kernel are using distributed version control systems. If your team is geographically distributed it might make sense to go with something that supports this kind of structure better, like Git.

#### Further reading

For the what, why and how of Subversion read *Pragmatic Version Control Using Subversion (2nd Edition)* by Mike Mason [Mason].

For the distributed world the guys at the Pragmatic Bookshelf have put together a similar book for Git called *Pragmatic Version Control Using Git* by Travis Swicegood [Swicegood].

### Test driven development

If there is a single practice that has dramatically changed the face of software development over the past decade it is Test Driven Development. It is a simple-to-explain concept but it leads to a paradigm shift in the way one develops and thinks that we believe will become the baseline standard for development in the years to come.

It is often noted that the cycle of testing, debugging and correcting takes a large majority of a traditional project’s development time. This is largely because a lot of testing is done manually by a team of testers. These testers in turn provide feedback to the developers, who correct the flaws and pass the system back to the testers. Once again the testers must test the system in its entirety to ensure that no new bugs have been introduced. And so, slowly, the bug count drops to the goal of zero.

It’s immediately obvious that some sort of automated testing process would substantially speed up this cycle. The question has always been how to do this effectively without extending project *development time* writing code that doesn’t add functionality. Tools, such as the industry standard xUnit (jUnit in Java, NUnit in .NET) have helped with this because they remove most of the overhead involved in setting up test suites. But more importantly the practice of test-first development, or test-driven development, affects the way systems are designed (for the better) and results in a significant decrease in the length of that testing, debugging and correcting cycle.

TODO: basic TDD structure, Red Green Refactor.

Habanero and all production projects at Chillisoft are developed using a test-driven development philosophy. At the time of writing this there are over 6000 automated tests for Habanero itself, which means that we can be confident that in the process of creating new functionality we are not breaking other features.

#### Further reading

The original, seminal book on TDD is *Test-Driven Development: By Example*, by Kent Beck [Beck]. It is a basic introduction to the concepts and contains a really good extended example of developing using TDD. We recommend being extremely rigorous with the principles laid down by Beck in this book.

Once you’ve mastered the basics, Gerard Meszaros’ *xUnit Test Patterns: Refactoring Test Code* is an excellent book on real-world testing philosophies and patterns.

### Continuous Refactoring

Refactoring is another practice that has entered the developer’s vernacular in the past decade and become standard. Martin Fowler coined the term and defined a refactoring as “a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” [Fowler, 53]. Refactoring is changing code structure, but it is not altering behaviour.

A constant vigilance for the smells of bad code is required if we are to stay ahead of code entropy. Because of this the steps of test-driven development include a refactoring step after each passing test. Its one thing to say “this code needs refactoring, I’ll add that to the to-do list,” and another to continuously refactor things little by little so that those big refactoring jobs never make it onto the to-do list. We have realized that with each line of code written our programs move closer to chaos unless this practice of continuous refactoring is applied. Areas of our programs become no-go zones that no one can maintain or improve upon unless we’re constantly applying the rule of not allowing software entropy to gain a foothold.

Refactoring without automated tests is a dangerous activity so the practices of test-driven development and continuous refactoring go hand-in-glove; one cannot be practiced without the other.

Many refactorings can be performed better by automated tools (or at least the substeps can), and we highly recommend using a plugin for Visual Studio like Resharper to make continuous refactoring that much easier.

#### Further reading

The original book on Refactoring is Martin Fowler’s *Refactoring: Improving the Design of Existing Code*. In it Fowler has made a catalog of standard refactorings that outline step by step how to perform things like extracting a method or moving a method without breaking other code. He also covers a variety of “code smells” and possible refactorings to apply in these cases to alleviate the smell.

*Refactoring to Patterns* by Joshua Kerievsky builds upon Fowler’s book by showing how you can refactor existing code into well structured patterns. Test-driven development advocates a practice of less design up front (countering the ills of applying patterns before they’re need), while refactoring to patterns shows us how we can still end up with a pattern based design via refactoring.

### Continuous integration

### Agile methodologies and Principles

### Domain Driven Design

## Infrastructure Challenges

There are a set of challenges that just about every enterprise application is required to overcome. In this section we will go over these one by one, discussing what the challenge is, when and why it needs to be overcome and briefly how this can be achieved by using Habanero. The infrastructure challenges discussed here are particular to the design philosophies outlined above, but many will apply given a different set of design philosophies too.

### Data storage and retrieval

### Domain modelling and object-relational mapping

### Concurrency control

### Atomic transactions

### Application Layering

### Change

## References

[Mason]

Mason, Mike. *Pragmatic Version Control Using Subversion (2nd Edition)*. Raleigh, North Carolina: The Pragmatic Programmers LLC, 2006. ISBN: 0-9776166-5-7 ([http://pragprog.com/titles/svn2](http://pragmaticprogrammer.com/titles/svn2))

[Swicegood]

Swicegood, Travis. Pragmatic Version Control Using Git. Raleigh, North Carolina: The Pragmatic Programmers LLC, 2008. ISBN: 9781934356159 (<http://pragprog.com/titles/tsgit>)

[Beck]

Beck, Kent. *Test-Driven Development: By Example*. Boston, MA: Addison-Wesley, 2003.

**[Meszaros]**

Meszaros, Gerard. *xUnit Test Patterns: Refactoring Test Code*. Boston MA: Addison-Wesley

**[Fowler]**

Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Boston, MA: Addison-Wesley, 2000. CBS 87.

**[Kerievsky]**

Kerievsky, Joshua. *Refactoring to Patterns*. Boston, MA: Addison-Wesley, 2004.