SIX DEVELOPMENT TOOLS ALL GRADUATE DEVELOPERS SHOULD KNOW... BUT USUALLY DON'T.

Introduction:

As a senior developer I am actively involved in the interviewing and integrating of both graduate developers as well as junior experienced hires into our software development teams here at BSG.

Having myself been a product of the varsity environment, I am fairly familiar with the level of theoretical and practical knowledge that can be gleamed from a mainstream Information Systems course, both from an analytical as well as a technical point of view.

Based on my own experience, as well as my own observations, the various IS classes from around the country tend to produce great analytical thinking developers with a strong understanding of commerce (accounting, finance, economics, statistics, etc.) which is to be expected, as most of these departments fall under the Commerce faculty banner and are more geared towards producing analysts rather than hardcore developers. Conversely these same institutions produce highly focused Computer Science graduates who can easily program a microprocessor, write a custom driver for a piece of hardware, or optimise a search algorithm, all in no less than four distinct programming languages. Outside of the varsity landscape, private FET (Further Education Training) institutions produce graduates with a broad range of basic programming language skills, just about enough to get started as a junior coder.

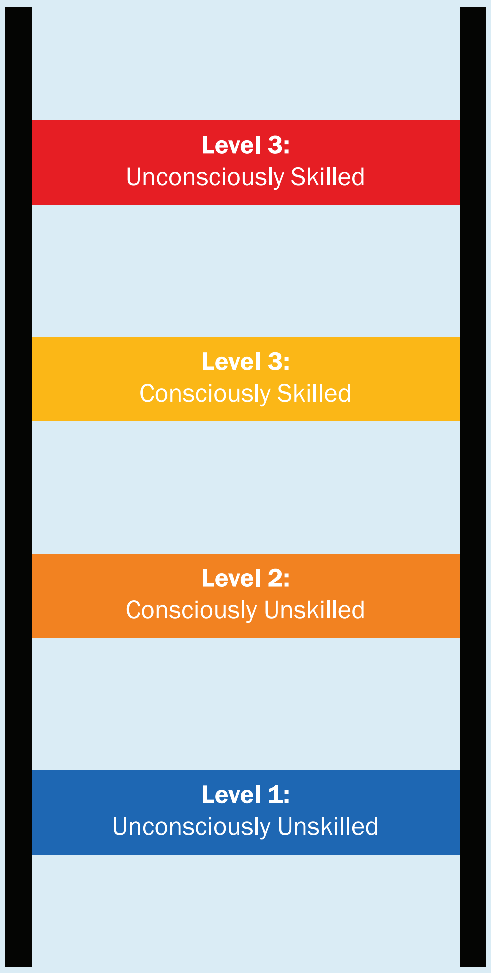
What all of these groups of developers tend to have in common, is a distinct lack of practical (and most of the time, any) exposure to the development tools and techniques which have been a staple part of many software development houses for many years. The kinds of tools that allow developers to collaborate and work together as teams of highly skilled professionals.

The aim of this article is to introduce a tertiary level student, intending to enter the real world of development in the next few years, to 6 of these tools. Thinking back to when I was at varsity, I knew I would have greatly benefitted from this kind of technical perspective with which I could have familiarised myself prior to landing my first programming job.

Although this article is written from a Microsoft .Net perspective, all 6 types of tools are equally valid and widely used across all mainstream programming languages. Once you understand the concept it's simply a matter of a Stack Overflow query, to find the most relevant tool for your preferred language.

The tools chosen all have more than 4 of the following characteristics:

- Reduce effort
- Reduce errors
- Increase collaboration
- Widely used across the industry
- Not specific to a specific architecture layer
- Tend to be unknown to graduate developers



Using the competency model above, the aim of this article is to move a graduate developer from a level 1 to level 2 i.e. being consciously aware that these tools exist and that they are something that should be investigated further. Come work at BSG and we can potentially help you move to level 3 or 4 over time.

1. Continuous Integration - Ci (Aka Build Server)

CI is a piece of software that is configured to build software projects automatically without direct user interaction. A typical software project would usually have multiple build configurations targeting different project stakeholders and triggered by different events.

Although the types of functionality can vary greatly depending on the characteristics of the project, some of

the more common build
steps usually include:

- Compiling of the source code
- Versioning and packaging a solution
- Deploying and configuring the application
- Running regression tests and calculating the code coverage
- Storing of build artefacts
- Communicating results to stakeholders

This has the direct benefit of creating an automated, repeatable and faster deployment process.

Once the build server has been configured correctly, it can substantially reduce errors as well as reduce the dependency on a developer's time to deploy a solution and ultimately allow non-technical individuals (such as analysts, product owners, testers etc.) to trigger builds on their own.

Part of the benefit includes real time and visual feedback. This usually takes the form of email notifications and various graphical tools via a web portal. Typical information would include:

- Build status
- Error messages
- Version numbers
- Test coverage
- Specific failing tests
- Deployment packages ready for release into production

Within the .Net team at BSG, we tend to configure our projects with 4 distinct builds, as specified in *Table 1*.

	Continuous	Nightly	Test	Package
Purpose	Ensure integrity of source code	Early access to new functionality	Stable playground to test new functionality	Full package ready for formal release
Audience	Developers	Users	Testers / Analysts	Maintenance Staff
Trigger	On code check-in, with 3 min quiet period	Once every evening	On demand	On demand
Automated Tests	Yes	No	No	No
Archive Artefacts	No	No	No	Yes

Table 1 .Net Build Specification at BSG

Based on the *Table 1*, for an agile development team, a continuous build is vital for real time feedback about broken builds, failing tests and code coverage. And this needs to occur as close to the original check-in as possible. Business users require a daily view of where the development is headed as well as a playground to visualise new functionality as soon as possible. Testers tend to require new builds in an ad hoc fashion. And finally, when the functionality has been coded by the developers, approved by the business users and tested by the testers it is ready for shipping to a more formal environment and thus needs to be packaged and made easily accessible to the relevant IT support personnel.

In order to get started with CI, you require 2 pieces:

1. You would need to select a lightweight CI tool such as TeamCity, Hudson, Jenkins etc., of these the .Net team at BSG makes exclusive use of JetBrains TeamCity as it:
    - Is relatively simple to setup as well as to configure new projects
    - Is relatively light-weight in terms of system resources
    - Supports multiple programming languages
    - Includes a good code coverage tool (DotCover)
    - Is free
2. You would probably require a crash course on a build scripting language. From a .Net perspective, an understanding of MSBuild would be an advantage but there are others, some which are .Net specific and some are geared towards other languages. Although not strictly required, more advanced build processes can greatly benefit from custom-built MSBuild (or equivalent) Tasks and Targets.

2. Version Control System - VCS (Aka Source Control)

VCS is software used to manage the concurrent editing of a group of files, at the same time, by a group of people.

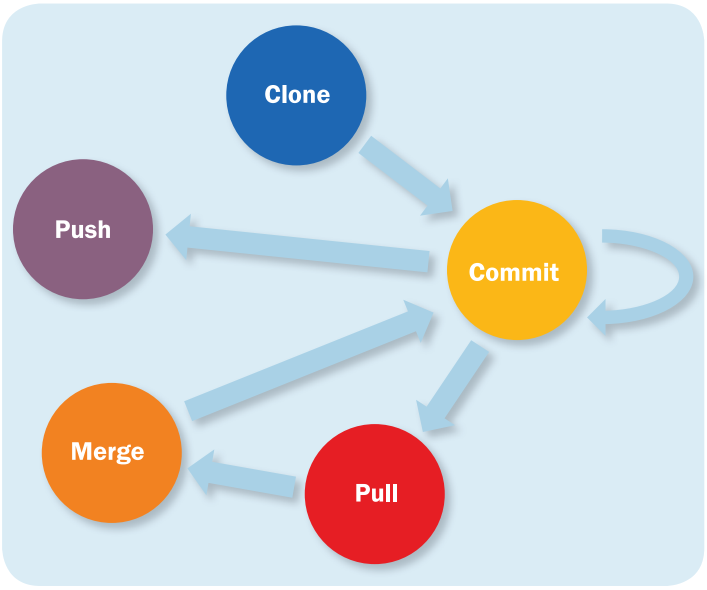
Developers can thus work on the same source code files at the same time and in doing so the VCS tool should be able to:

- Resolve merge conflicts
- Revert to previously saved state
- Keep a history of changes and individuals responsible for those changes

Teams which do not make the use of a VCS, tend to spend a great deal of effort merging changes from multiple sources, which is both tedious and error prone.

Exclusive locking of files should be discouraged within development teams (except for binary files, aka non-text files, which do not merge well) as it tends to create an unnecessary developer friction which is counterproductive for an agile team. In addition, the merge tools of today are sophisticated enough to be able to handle even the most advanced merge conflict. The key is update your source (aka pull) as often as possible to ensure you are up-to-date with the rest of the team.

At BSG we make use of the newer generation VCSs, which are known as Distributed VCS or DVCS. The major difference is that each developer has local access to the entire repository (i.e. every single revision) which is usually cloned from central location. The developer would commit locally many times whilst working on a piece of unfinished functionality and then once finished "commit" or "upload" (aka push) back to the shared central location. This 2 stage commit process provides the developer with a safety net of saved state without impacting the rest of the team with broken or unfinished code.



In order to get started with VCS, one requires up to 3 separate pieces (the last 2 of which are optional).

1. Get hold of a file-based DVCS such as Mercurial or Git. The former has tended to provide a better developer experience in a windows environment than the latter, but this is due to change in the future. File-based VCSs tend to be more portable and require far less server infrastructure.

2. In order to interact with the VCS, various client tools will be required. Both come with their own command line tools but if you prefer a more graphical experience, then *TortoiseHg* for *Mercurial* and *SmartGit/Hg* *for Git* are great tools to get you started. A merge tool (which is usually packaged with the GUI tools above) is also required (when not using the provided console application), *KDiff* is a solid performer.

3. When working within a team, sharing of code becomes important, and when you don't have the infrastructure to host your own repository, one can always turn to various cloud services to fill the need. *GitHub* and *BitBucket* are two such services, *BitBucket* in particular offers free private repos for teams of up to 5, which should be perfect for small projects such as the ones usually found within the varsity environment.

3. Design Patterns

Design Patterns are reusable software solutions to solve commonly occurring problems.

Essentially Design Patterns could be described as a form of industry best practice to solving specific, recurring problems. Thus one does not need to "reinvent the wheel" as the solution to the problem has already been developed, tested and widely accepted.

Design Patterns can also provide a form of "short-hand technical speak" between developers. For example, one developer could ask another to create a specific component as a Singleton. Provided both developers have some experience with the Singleton pattern, no further instructions should be required.

Another advantage to understanding certain patterns is that a developer should be more capable of grasping a larger framework which is based on that particular pattern. For example, those developers that previously had exposure to the Model View Controller pattern should have been readily able to understand the mechanics behind ASP.Net's MVC web framework and thus become more productive, in a shorter space of time, when compared with those developers to whom the MVC pattern was completely foreign.

The same way Patterns are common ways of solving problems in a correct manner, Anti Patterns are common ways of doing things wrong. Related to Anti Patterns is the concept of Code Smells. Code Smells are used to describe bad coding practices that just feel wrong or go against good programming practices, e.g. when a developer creates a large class with a few hundred lines of code.

A word of warning however, inexperienced developers, specifically those who have just been introduced to the concept of Design Patterns can fall into the trap of the Shiny Hammer Syndrome, whereby every problem is a nail which your newly acquired pattern tool can solve, i.e. Patterns are used to solve problems which are not fit for purpose.

Getting started with Design Patterns is a simple as reading various books or blogs on the subject.
This could include:
- The Gang of Four (GoF)
- Martin Fowler's Enterprise Patterns
- O`Reilly's book - Head First Design Patterns
- PluralSight's Design Pattern video course

4. Test Driven Development - TDD (Aka Test First Approach)

TDD is a development technique whereby developers begin writing unit tests before writing the actual implementation of the code. The tests describe the intended functionality by asserting the result of an action under test. These tests will initially fail (represented by most GUI test runner tools as red) as the functionality does not yet exist to change the state. The tests should begin passing (represented by the change from red to green) as the code to change the state is added. It is this red to green transition (i.e. making the tests pass), that allows the code to be driven by tests and not the other way around (i.e. logic followed by tests).

At a recent varsity presentation, when asked if any of the teams had been using a TDD approach, no hands went up. When asked further if anyone had ever heard of TDD, still no hands went up. Then going back to basics, when asked if anyone knew what unit tests were, still no hands went up. Then remembering the testing theory I had learnt back at varsity, the students were asked if anyone of them knew about white box and black box testing, then all hands went up.

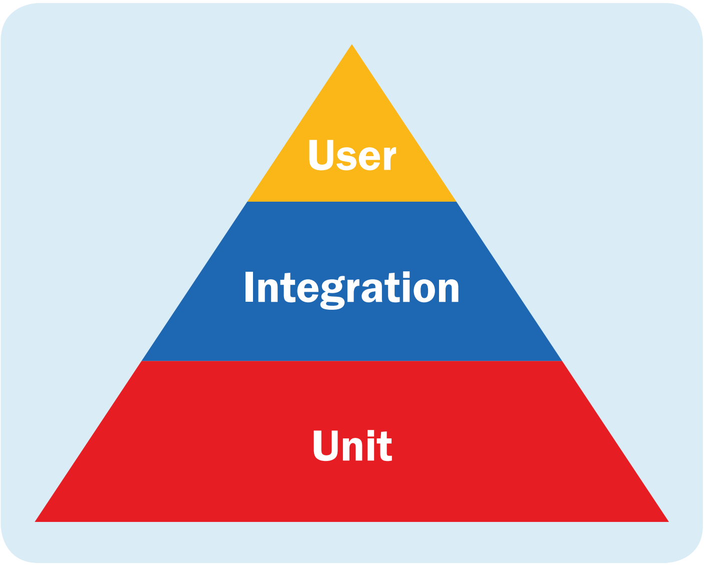
So what I remember of testing theory at varsity was something along the lines of a paragraph or two about white box testing, black box testing, load testing, stress testing etc. It seems as if this abstract theory is still being taught 8 years on. It was surprising to find that TDD theory (of which there is quite a bit) had not yet been incorporated into mainstream IS courses.

The advantages of a TDD approach are numerous; and the reasons for both developer and business stakeholder resistance to TDD are equally high (and beyond the scope of this article). In a nutshell, TDD can direct developers to make better use of coding principles (some of which include SOLID, KISS, YAGNI, DRY, etc.). Similarly as tests are written before code, a closer to 100% code coverage (% of code executed by a test suite) will be a natural by-product. This in itself facilitates regression testing and provides a safety net for large scale code refactoring.

One of the most important aspects of TDD is that it encourages developers to code smaller reusable components and only the code that is actually needed. The more complex the code, the more complex the test. And complex code is harder to maintain.

However just as one can write bad code, one can also write bad tests. Below, we'll discuss the types of tests and the unit test structure used by the BSG .Net team to overcome this issue.

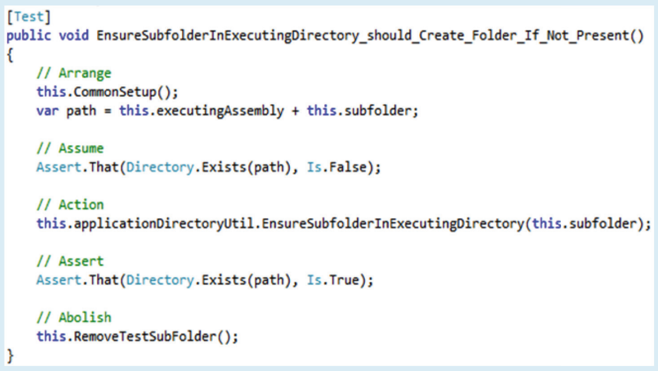
Within the BSG .Net team, we define 3 main categories of tests. All of the tests mentioned below will run on top of an xUnit type framework of which the team has traditionally used nUnit.



Unit Tests are designed to test one and only one class, all other classes are mocked out using special frameworks (such as Moq). Thus they are independent of any integration points, such as a database, file systems, etc. There tend to be many hundreds of these tests on any given project and thus they are required to run extremely fast.

Integration Tests vary, from testing across multiple classes, to testing external integration points (such as database calls, file systems, web services). They tend to be fewer in number when compared to Unit Tests. The key is that they tend to cross between layers of classes. They are harder to set up, as mocking is generally not used, therefore a clean, known state is harder to achieve.

User Tests are driven from the UI layer right down to the data access logic. In effect they are a sanity check and are therefore few in number. They tend to be more brittle, especially when UI changes occur. Depending on the UI technology, various UI runners can be used, Selenium is a good example of a web runner that can be used across multiple browsers.



```
[Test]
public void EnsureSubfolderInExecutingDirectory_should_Create_Folder_If_Not_Present()
{
    // Arrange
    this.CommonSetup();
    var path = this.executingAssembly + this.subfolder;

    // Assume
    Assert.That(Directory.Exists(path), Is.False);

    // Action
    this.applicationDirectoryUtil.EnsureSubfolderInExecutingDirectory(this.subfolder);

    // Assert
    Assert.That(Directory.Exists(path), Is.True);

    // Abolish
    this.RemoveTestSubFolder();
}
```

Within the .Net team we subscribe to the principle of 5 A's, which is an extension to the 3 A's (commonly found around the web) when structuring our Unit Tests.

Based on the above example, we divide a typical test into the following 5 sections:

- Arrange – Set up variables and supporting code (optional)
- Assume – Assert known state before action (optional)
- Action – Method under test (preferably a single line)
- Assert – Asserts to ensure action performed work correctly
- Abolish – Clean up after test (optional)

In order to get started with automated testing, you would need a testing framework, usually written in the same programming language as your source code. A good start would be nUnit which is the granddaddy of unit testing in the .Net space, although any xUnit framework should suffice, including Microsoft's own MSTest.

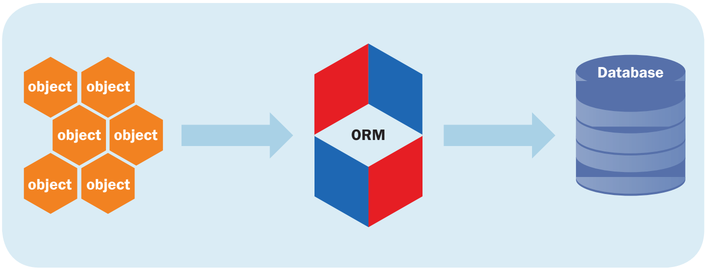
You also require a test runner, a piece of software to execute the tests. Once again nUnit comes with its own built-in runner application. There are also a host of other 3rd party runners (both free and paid for) which are extensions to Visual Studio, in order to run your tests in the same IDE environment, where you write your actual tests and your code.

Once you are comfortable with the concept of writing tests, you can extend your knowledge by looking into mocking frameworks such as Moq, as well as looking into UI testing tools such as Selenium (for html interfaces) and if you use a lot of JavaScript, the same testing conventions can be applied to your scripts by using tools such as Jasmine, PhantomJS and Sinon.JS.

And finally a whole different technique of development and testing can be achieved via BDD (behaviour driven development) and to support this approach, one could look into tools such as SpecFlow and the Cucumber DSL (domain specific language). Discussing BDD is beyond the scope of this article.

5. Object Relational Mapping – ORM

A framework which automatically maps an Object Orientated model to a relational database schema, i.e. (in the .Net world) two-way mapping between C# domain classes and SQL tables.



On a side note, one might ask if an ORM actually fits the mandate of this article, to provide developers with a toolset that is independent of the type of project and thus is generic as possible. The answer is both no and yes.

No, because ORMs are usually only found in the classic "Data Access Layer" and thus have a very specific purpose, we have purposely left out frameworks specific to particular layers (such as reporting frameworks, UI frameworks etc.) and rather focused on tools and frameworks which are either applicable to multiple application layers or fall outside the source code completely.

And yes, because the vast majority of software projects require state to be persisted and this typically occurs within relational database management systems (RDBM). Also the types of projects that students and graduates generally get exposed to early on tend to be of the more classic Line of Business (LOB) variety. Likewise, multiple different UI layers generally communicate to the same service and / or data layer. As such we see an ORM as a tool which should benefit the vast majority of graduate developers both at varsity and in the real world.

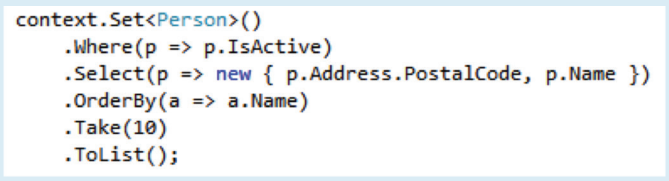
The key benefit of an ORM is to reduce the amount of boiler plate code required to map data between SQL queries and domain objects. Specifically it reduces the amount of TSQL required for the following actions to almost zero:

- Table Definitions (columns, keys, indexes, data types)
- Basic CRUD operations
- Complex queries

A good ORM will also provide features such as Change Tracking (makes note of specific changes to specific properties), Lazy Loading (automatically queries and loads child entities when accessing these child properties off a parent entity) and Eager Loading (automatically extends the initial SQL query to join child tables and populate the child properties).

One of the positive by-products of using an ORM is that during development, the database schema can be easily regenerated and thus supports concurrent updates. Maintaining a central copy of the database or a common script file is both tedious and error prone.

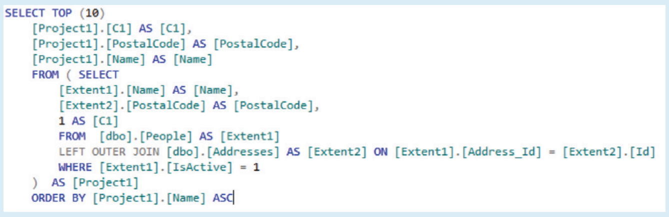
In Fig. 1 and Fig. 2 you will see the difference between using an ORM (Fig. 1) versus hand crafting the SQL code (Fig. 2) by hand:



```
context.Set<Person>()
    .Where(p => p.IsActive)
    .Select(p => new { p.Address.PostalCode, p.Name })
    .OrderBy(a => a.Name)
    .Take(10)
    .ToList();
```

Fig. 1 Code written using ORM

On the previous page (Fig. 1) is a fairly contrived query, written in C# code, notice that it includes a filter (active Persons only), a projection (selecting only a postal code and the Person's name), a join (Address that is linked to the Person), some ordering (on name), only take the first 10 records and then finally mapped into a C#.



```
SELECT TOP (10)
    [Project1].[C1] AS [C1],
    [Project1].[PostalCode] AS [PostalCode],
    [Project1].[Name] AS [Name]
    FROM ( SELECT
        [Extent1].[Name] AS [Name],
        [Extent2].[PostalCode] AS [PostalCode],
        1 AS [C1]
        FROM  [dbo].[People] AS [Extent1]
        LEFT OUTER JOIN [dbo].[Addresses] AS [Extent2] ON [Extent1].[Address_Id] = [Extent2].[Id]
        WHERE [Extent1].[IsActive] = 1
    )  AS [Project1]
    ORDER BY [Project1].[Name] ASC
```

Fig. 2 Handcrafted SQL Code

Contrasting to this is the TSQL, above (Fig. 2), which also happens to be the actual TSQL generated by the ORM framework. You'll notice that it's more verbose and harder to follow. It also doesn't include any mapping logic to actually transform the result into C# classes.

Using an ORM can be a double edged sword at times. In that it is true that an ORM can create efficient queries quicker, but at the same time it is just as easy to create highly inefficient queries, specifically when the developer is new to the framework. An ORM is not an excuse to never learn SQL, understanding SQL is vitally important to getting the best out of the ORM. The take away here is: with great power comes great responsibility.

Using the example above, if the developer does not take deferred execution into account, and places the call to ToList() on the second line instead of the last, they would transform the previous TSQL query, pulling down all rows and all columns of the Person table, across the wire into memory, and then proceeding to query the database for each Person record to obtain the corresponding Address record, in effect creating a n+1 query. This scenario is more common than one might think.

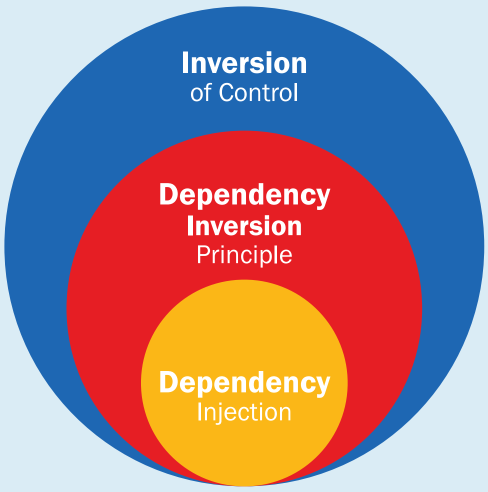
At BSG, the .Net team uses Entity Framework Code First. Written by Microsoft, it is rapidly establishing itself as a mature ORM in the .Net space. The Code First approach, together with the Linq providers, the fluent API, the built-in migration and seeding frameworks makes EF a solid choice for .Net ORM based development.

As long as you are prepared to live with, or work around, its various limitations (which are beyond the scope of this article), EF is a good starting point for your own projects. Similarly, other languages such as Java, Ruby, Python, PHP and even JavaScript have their own dedicated ORMs.

6. Inversion Of Control Containers – IOC (Aka Dependency Injection Container)

A software framework used to automate the building of complex objects as well as to manage their application lifecycle.

An IOC container fulfils two primary functions. Firstly it is responsible for building (instantiating) classes, usually ones that have multiple dependencies and sub dependencies etc. This occurs through the use of some form of Dependency Injection pattern, usually Constructor Injection (although other methods do exist).



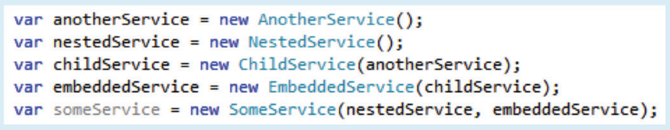
Secondly, it can be used to manage the lifecycle of the object that it has just instantiated. This is easier to explain by describing the 4 main lifecycle mangers used in many of IOC containers available today:

- Singleton – Only one instance will be created for the life of the container, the same instance is returned whenever it is requested.
- Transient – A new instance is always created whenever requested from the container.
- Per Scope – In many ways this is a hybrid between Singleton and Transient, in that for a given scope (how this occurs depends on the framework), only one instance is ever returned. For another scope, a different instance will be returned, e.g. Http Request would be a type of scope.
- External – The instance is provided to the container that is created outside of the container and will not be disposed when the container is disposed.

In effect, this means that any lifecycle code that used to be included inside the class can be removed and controlled by the container itself, e.g. a class following a Singleton pattern can now be created as a standard class and requested with a Singleton lifecycle manager. Or even better, can be updated to a different type if required, with zero code changes to the actual class.

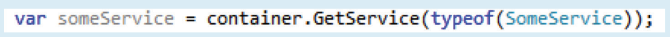
By using an IOC, one gains from having to write little to no code to build up an object and / or manage its lifespan. One implementation can be swapped out for another (e.g. swap an actual implementation for a stubbed version, e.g. used in unit tests) and changes to the classes hierarchy will require far fewer code changes across the application.

In another contrived example below one can see the difference between building an instance of a class manually (Fig. 3) versus registering all the components on start-up (not shown) and requesting a new instance from the container (Fig. 4):



```
var anotherService = new AnotherService();
var nestedService = new NestedService();
var childService = new ChildService(anotherService);
var embeddedService = new EmbeddedService(childService);
var someService = new SomeService(nestedService, embeddedService);
```

Fig. 3



```
var someService = container.GetService(typeof(SomeService));
```

Fig. 4

Imagine if the SomeService was required in 50 different locations. And now imagine you need to add a new component into the hierarchy or want to change it to a Singleton. Using a container would require only a few lines of configuration code and zero changes elsewhere. Hand crafting the code by hand would require changes in multiple places across the system.

Just to note – in the above example we are demonstrating an example of resolving a service directly from the container. This can be classified as a bit of an Anti-Pattern and is used for illustration purposes only. In a real system you would normally register the container with a global resolver of some kind and register your Composition Roots (Composition Roots are usually the point where the composition of your object graphs or hierarchy takes place) into the container. Examples of Composition Roots could include MVC Controllers, WCF Services, Win Forms, etc. depending on the technologies used.

Conclusion

The 6 tools described in this article are actively used in the real world, and there is no valid reason not to begin investigating and even integrating some (or all) of them into your projects at varsity.

Similarly, if you are a graduate, or inexperienced developer, who has not been exposed to these tools in your place of work now is as good a time as any to introduce both yourself and potentially your teams, to these tools.

About the Author

The author is a Senior Developer who has been working at BSG, Cape Town since graduating from the University of Cape Town with a B.Bus Sci (Hons) majoring in Information Systems in 2005. He has a passion for all things related to .Net development, agile and specifically enjoys engaging and sharing knowledge, obtained from working at BSG, with students, graduates and experienced developers alike.

About BSG

BSG is a consulting and technology company with almost two decades experience across a variety of industries. We have been an integral partner in the delivery of over 500 projects to businesses across a wide variety of industries including, retail and investment banking, insurance, mobile money, telecommunications, and oil and gas. At BSG we have a proven track record of guiding our clients to success by developing authentic relationships based on tested results and trust. BSG has delivered successful interventions across a variety of environments and levels of complexity despite various challenges.

At BSG we have the knowledge and experience to help you successfully innovate like a start-up, from whiteboard to benefit.

At BSG we believe that in order to ensure success, the development team must be as robust and encompassing as possible. As such we have dedicated project teams, often working within the client environment to ensure the most transparent and collaborative approach is achieved. Through this approach we are able to ensure constant, consistent and open communication with all stakeholders, guaranteeing that each iteration builds on the learnings of its predecessors, taking on board all feedback received from stakeholders and users. Systems are designed with the user in mind. This user-centric design approach, coupled with Scrum and Agile Development methodologies and the latest in development technologies ensures that the systems we deliver meet, and wherever possible exceed, expectations and requirements. BSG's team of highly skilled developers, fluent in .Net and Java programming languages are experienced with the latest development technologies because we believe in being a proactive force for positive change, making a difference in everything we do.

Unlocking potential ➤ Accelerating performance

BSG

22nd Floor, Triangle House
22 Riebeek Road,
Cape Town
8000

Tel 021 418 0888
Fax 021 418 0999
Email info@bsg.co.za

Oxford Terrace,
No. 3 Ninth Street,
Houghton Estate,
2198

Tel 011 215 6666
Fax 011 215 6600
Web ***www.bsg.co.za***