Returadresse: ProgramUtvikling AS. Postboks 1, 1330 Fornebu

DISTRIBUERT AV POSTEN NORGE

NORGE P.P. PORTO BETALT

No 2/2012

The DEVELOPER

NDC SPECIAL EDITION FOR DEVELOPERS AND LEADERS

NDC WORKSHOPS ON TOP OF OSLO PLAZA

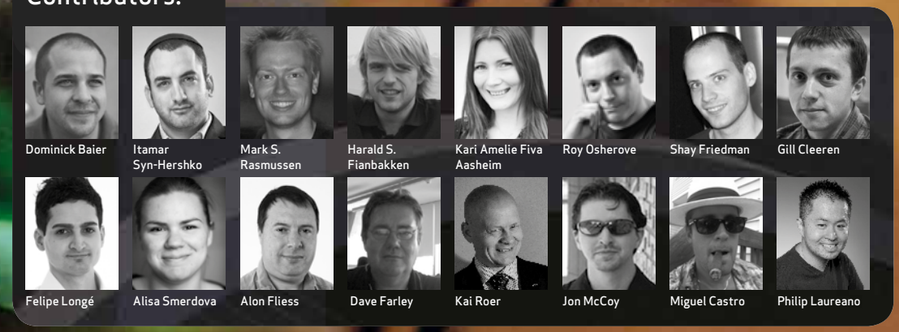
With Lea Verou among others
June 4th & 5th

READ ALL ABOUT NDC 2012
Pages 57 - 70

THE ART OF MISDIRECTION

By Dan North - Page 4

Contributors:



NORWEGIAN DEVELOPERS CONFERENCE 2012

Oslo Spektrum, Oslo
June 6-8
Pre-workshops, 4-5 June

NDC 2012 NORWEGIAN DEVELOPERS CONFERENCE

Nº 2 2012 NDC Edition

NDC 2012 – Bigger than ever!

On the 6th to 8th of June, we will again be welcoming people from all over the world to the Norwegian Developers Conference (NDC) in Oslo, the beautiful capital of Norway. This year's conference will truly be one of the world's most exciting and important conferences for developers and project managers.

NDC has become a highly international, world-renowned conference, with attendance numbers increasing steadily throughout its five year history. 45 days away from the conference, I am delighted to say this year will probably see a record attendance.

This allows us to expand the conference from three to five days. The first two are pre conference workshops will take place at the top floors of the Oslo Plaza hotel, adjacent to the Oslo Spektrum venue which will host the last three days. Given the extensive program, participants will be able to pick sessions suited to their particular interests as a sort of "tapas course of disciplines". Some companies allow their employees to visit NDC as a conclusion of the first half of the year, knowing that both the program and the big party on Thursday night will be among the highlights of the year. Other participants come on their own initiative. Whatever your reasons, I can guarantee that you will find NDC worth your time and energy.

There is a lot going on within the world of software development at the moment. Developers need to stay on their toes. At NDC, we bring the best minds together – both on stage and in the audience.

Kjersti Sandberg

Contents

Publisher: Programutvikling AS
Organisation no.: 963 201 141
Address: Martin Linges vei 17-25, 1367 Snarøya, Norway
Phone: +47 67 10 65 65
E-mail: info@programutvikling.no
Uncredited photos are from Shutterstock, except portraits.
Print run: 15,000
Publication schedule 2012:
No. 3: 22 August, No. 4: 21 November
Print: Merkur Trykk AS

Watch the magician. Watch how he drops a coin into his hand, closes his hand, shows you the closed hand, opens it with a flourish, and the coin is gone! He smiles. You look to his other hand. He turns it over and opens it with the same flourish. Not there either! Then he takes your hand, closes it into a fist, opens it and there is the coin!

The Art of MISDIRECTION

By Dan North

Now watch again. This time watch the other hand. As he turns the upper hand over and makes a fist you see the coin slip into the lower hand in a much smaller movement. He opens the empty hand and moves the lower hand to the top. Ignore the obvious movement.

Notice instead how he guides the coin between his middle fingers. Ignore how he opens the other hand and instead see him discreetly drop the coin into the first hand again. Finally feel how he slides the coin into your palm as he closes your fingers. Magic? Perhaps not.
But a classic illusion.

Magic works by misdirection. The magician exploits your tendency to look at the obvious while the real action is taking place elsewhere. When the obvious is made compelling enough it's hard to even imagine looking elsewhere. Economists have a name for the impact of only looking in one place: they call it Opportunity Cost.

Whatever you are doing right now comes at the cost of everything else you might be doing instead, and there are always other things you could be doing. But you aren't considering the other things because you're busy concentrating on the obvious.

THE HIDDEN COST OF SOFTWARE DEVELOPMENT

How does that apply to software development practices? A new practice usually starts out as well-intentioned advice: *"We tried this thing [stand-ups, pair programming, TDD, burn down charts, iterations, build automation] and it worked out well for us. Maybe you should give it a try too!"*

All too soon there's an agenda attached: *We tried this thing and we think we can make money showing people how to do it. You should use it because it works! Here's a white paper to prove it."*

(By the way, if you read through that list of practices and thought *"but those are all good things to do,"* I'm talking to you.) No-one tells you the opportunity cost of using a technique. Where doesn't it work? What else could you be doing instead? What are the trade-offs in choosing this over one of the alternatives?

Try this two-part exercise: Think of a practice or technique you use when you develop software, maybe your favourite practice. Got it? Ok, part 1: why do you do it? What benefits does it give you? You can probably think of a few. Write them down. Now, part 2: where wouldn't you use it, and what are some alternatives? Write down some pros and cons of each one. I'll wait.

Chances are you found that second part much harder. If you mostly came up with negative or ironic reasons for the alternatives, you are just listing more reasons to use the original practice. You can't see past it to the alternatives. *You've been marketed to!*

TDD UNDER THE SPOTLIGHT

As an example I'm going to pick on TDD since it's among the most

Continues on next page

dogmatically advocated practices I've encountered, but you can apply this approach to anything. We're going to look at TDD — *really* look at it — and see if we can discover where it shines and where it isn't so useful.

If you are a TDDer, take a moment to think about where you *wouldn't* use it, and what you might do instead. Recently I've seen a couple of people asking this as a rhetorical or even ironic challenge, as though you'd be mad to ever consider not using TDD.

TDD advocates say things like *"TDD lets you make steady, confident change." (It often does.)* "TDD allows for emergent design" (Maybe.) *"Automated tests act as a regression suite and stop you reintroducing bugs."* (They often do.) *"Tests act as living documentation."* (They can.) *"Test-driven software is cleaner and easier to change than non-test driven software."* (Actually I'll take issue with this one. I've seen shocking pure-TDD codebases, and clean and habitable codebases with no automated tests at all.)


© Shutterstock

Let's take a look at the opportunity cost — the trade-offs — in each of these assertions. Each one could be an article in itself. I just want to give you a feeling for finding the trade-offs inherent in soundbites like these.

THE OPPORTUNITY COST OF STEADY, CONFIDENT CHANGE

What's not to like about steady, confident change? Well sometimes you can describe the problem but you can't see an obvious answer. Financial trading applications are a lot like this: you might try several approaches and see how they perform, like a series of experiments. You want each experiment to be inexpensive so you can try several. TDDing each option will work, sure, but it is going to be more expensive than just sketching something out to see if it feels right.

What is the opportunity cost of all that extra time spent TDDing the sketches? You can sketch half-a-dozen ideas in the time it would take to TDD any one of them. And it doesn't end there. The result of one attempt can change your understanding of the problem as much as offering a potential solution, which sends you in a new, unexpected direction. Some TDD advocates will say this is a series of spikes and you don't need to TDD those. However I'm talking about putting software fully into production, and keeping the successful experiments as production software, so this doesn't really hold.

TDD locks in your assumption about the desired end goal. It assumes you know where you are heading, or at least where you are starting from. If you don't know what the solution even looks like this could be an undesirable strategy. Perhaps you should defer investing in a solution until you know more about the problem.

THE OPPORTUNITY COST OF EMERGENT DESIGN

Sometimes the right design isn't staring you in the face. It might take a shift in perspective, a reappraisal of the whole premise, to see simplicity through apparent complexity. TDD is about incremental change and improvement. It is great for finding local maxima but the best solution might require a radical rethink. The opportunity cost in this case is that we get trapped in this local maximum and miss a bigger win. In Lean terms this is the difference between *kaizen*, continuous improvement, and *kaikaku*, abrupt transformation. Neither can succeed without the other but we have tended to focus on the former.

TDD is the epitome of kaizen programming. To shine it needs an environment where you can step back, go for a mental walk around the block, come back and maybe change everything. You need an overarching vision, a "big picture" design or architecture. TDD won't give you that. However, a robust implementation of complex behaviour is going to benefit from the incremental approach TDD provides. You will notice I'm not suggesting not to use TDD but qualifying where and how it is effective.

You don't write software because you want software, you write software to provide a capability. On several recent applications we got to a point where we *rewrote* the system to add a new capability. It sounds rash, never mind wasteful, but what we had learned getting to where we were told us it would be more effective to rewrite — often in a different technology — the subset of the application that contained the real value. One system started in Scala and moved, in chunks, to Java (the wrong direction, surely!). Another started in Python and moved to JavaScript and node.js. Yet another started in Python and was rewritten into another Python app! Each rewrite was the result of getting the software in front of people, operations folks as well as users, and responding to their feedback. In that context it didn't matter how crafted or test-driven the original code was because we threw most of it away.

We hadn't over-invested in the software by surrounding it with comprehensive automated tests, otherwise it would have been an expensive exercise. But what of the software that survived? We found it was possible to produce TDD levels of quality after the fact. We know what well-factored software looks like, and by this stage we knew we were prepared to invest in this software: it had proved its value. So we started introducing TDD-style tests for the more complex or critical parts of the code, which led us to refactor it to make it more testable, which created natural seams and subsystems in the code, leading to bigger refactorings, and so on. I've been describing this technique as **Spike and Stabilize**: get something, anything, into production to solicit rapid feedback, and invest in whatever survives.


© Shutterstock

THE OPPORTUNITY COST OF AUTOMATED TESTS

Automated tests provide assurance that code is doing what it should. This begs two questions: Are there any other ways of gaining that assurance? Is the assurance valuable?

Automated tests are good at exercising specific pieces of code. Would you spot bugs in that code *without* the tests. You can't help noticing a missing or unconnected submit button. If it fails halfway through retrieving data what should it do? Once you've coded it to do that is it ever likely to stop doing that? Will an automated test give you any more assurance? What else could you do?

The value of an automated test is a function of a number of things: the criticality of the code, the impact of a Bad Thing happening, the cost of correcting that Bad Thing, which may be reputational as well as operational, the likelihood of you not spotting it with regular usage in development, your familiarity with the domain. Code at the edges of a system is often harder to test than internal code. UI widgets, external integration points, third party services, can all be expensive to automate. Does the cost justify the value? Again I'm not saying not to do this, but to question the trade-offs and decide where it makes sense. I've seen teams burn weeks of development effort creating beautiful automated test suites that provided almost no additional assurance or feedback over using the application. The reputational cost there is high: they lose credibility with their stakeholders, who would prefer to see features being delivered. Again you need to strike a balance, understanding where automation is valuable and where it is habit.

THE OPPORTUNITY COST OF TESTS AS LIVING DOCUMENTATION

There are many kinds of documentation. Valuable documentation educates you. It tells you things you wouldn't otherwise know because they aren't obvious. You want to know what makes this system different from all the other similar systems you've seen. (You may also question why you are documenting these quirks, rather than eliminating them from the system and reducing the level of surprise for a newcomer, but that's another story.)

Some automated tests read more like historical documents, giving an insight into times past. Perhaps at one point there was a silly bug where refreshing the current status fired off an email to the back office. They spotted it straight away, of course, so the developers wrote a test. There is now a test called *"should not send email to the back office when refreshing current status"*. What? Of course it shouldn't. But there are many of these, increasing the noise among the signal, so it becomes harder to find the really useful living documentation. *Living* is not a synonym for *useful*. Curating living documentation — automated tests — is as important an activity as curating any other kind of documentation. All too often it gets overlooked.

CONCLUSION

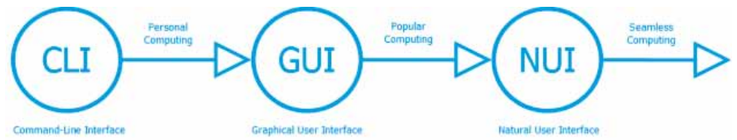
My goal is not to bash TDD or any other development technique. I see TDD as a valuable and important development technique, but there are contexts in which it shines and others in which it is a hindrance. So take nothing at face value, and instead look for the trade-offs in every decision you make, because those trade-offs are there whether or not you see them. And if you can learn to spot them you can do magic.

NUI: MAKING SCIENCE FICTION REAL

"The question persists and indeed grows whether the computer will make it easier or harder for human beings to know who they really are, to identify their real problems, to respond more fully to beauty, to place adequate value on life, and to make their world safer than it now is."

Norman Cousins – The Poet and the Computer, 1966

By Alisa Smerdova and Felipe Longé



The importance of Human-Computer Interaction (HCI) is emphasized by rapidly growing general use of computers. Even everyday life gets more and more computerized. When IBM made computers personal (**personal computing**), the Command Line Interface (CLI) appeared to be a kind of HCI. The user involved the computer in an interaction through a series of codified inputs and predefined commands. After the computational power increased, the Graphical User Interface (GUI) became possible and the first pointer device was invented. As a result, computers became fit to be used by ordinary people, which gave the start of **popular computing**. The users learned certain mouse movements and actions that allowed exploring the interface with less effort.

"That's the old way, that's the old mantra: one machine, one human, one mouse, one screen. Well, that doesn't really cut it anymore."
John Underkoffler – TED talk
"John Underkoffler points to the future of UI", 2010

Most software today is interactive and user-centered by design. However, the interaction is limited by the use of input devices and monitors. To expand it beyond those barriers, computers need to understand the space and be able to collaborate with humans. Multi-touch technology and gesture-recognition devices can be the way to the Natural User Interface (NUI). The direct manipulation of objects and spatial gestures suitable for interaction can start **seamless computing**.

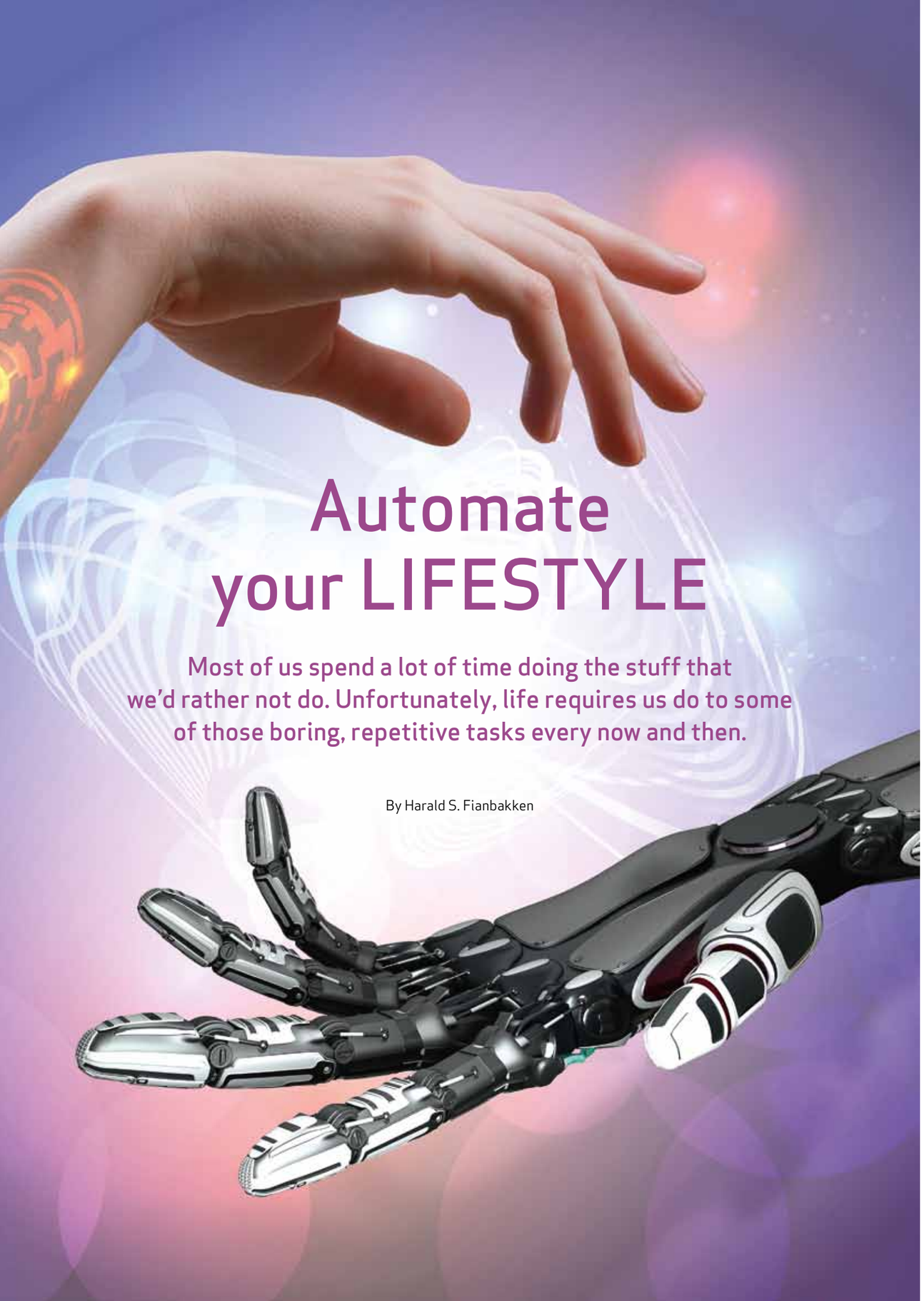
With true HCI, the user can experience the effortless transition from being a novice to becoming an expert. Potentially, with the skills that the user has already had, no additional learning can be required.

There are two basic design strategies in achieving Natural User Experience.

1) By limiting functionality and customization to get easy learning and intuitive design. An obvious example of such functionality is the iPhone (Apple's iOS). The iPhone design is oriented towards its natural user's experience on the basis of the ideas and skills acquired before (background knowledge), such as using switches or sliding book pages.

2) By merging digital objects with real ones by means of the Reality-based User Interface (RUI). The user succeeds in learning new interfaces after he starts interacting with them. For example, Microsoft Surface follows the RUI design strategy on the basis of its ability to respond to the touches of the finger tips and read tags and objects on its surface. Moreover, the RUI can be applied to the Microsoft Kinect device that has a high potential of being a NUI input device. The Kinect software can be designed to recognize a spatial gesture matching it with its graphical representation and performing a corresponding action.

Certainly, human-computer interaction is still in process of gradual development to be as transparent as possible. There is every reason to believe that modern technology will make science fiction real, and modern software is, therefore, limited only by developers' imagination.



Automate your LIFESTYLE

Most of us spend a lot of time doing the stuff that we'd rather not do. Unfortunately, life requires us do to some of those boring, repetitive tasks every now and then.

By Harald S. Fianbakken

Maybe management just enforced you to report your working hours into yet another system, or you as a developer are facing manual build & deployment schemes, or it could be as simple as sorting out the colors of your socks in your sock drawer? All these things can drain a lot of your time every week.

While others might be satisfied with their manual routines as long as they "get things done" we developers should think differently. We should ask ourselves how can we optimize our time and make life just a bit easier.

The simple answer is automation, optimizing and the DRY principle! We've all heard it and when you are writing code, it's obvious.

But when it comes to our routines (especially the manual ones), some of us tends to skip this principle or be sloppy.

We all have some sort of rituals during the day; we get up, get dressed, have coffee, leave for work, etc.

If you care about your time, you should be conscious on how you are spending it. Have you ever done the following exercise?

Make a list of the most time consuming rituals (process) you do every day (break it down as much as you like). For each process, answer the following:

- Do you enjoy it or would you like to use less time doing it?
- Is it repetitive?
- Can it be generalized?
- Can it be automated in any way?
- Can you improve the process?

You are allowed to be creative in how things can be automated and optimized! If you make coffee every morning, get a timer and automate it to make you the coffee in parallel while you do other stuff! Automation can be as fun as you want it to be.

Maybe you'll save some hours during the week? Or maybe you are perfectly happy with spending some quality time with your analog processes every day.

Either way, our digital processes should be optimized as it is natural for us developers!

If you haven't already, I suggest you to start enjoying life by learning Powershell and making it become your favorite tool on the windows platform. Automate installs, excel operations, Powershell 3 (WMF3) is just around the corner, and the beta has already been released.

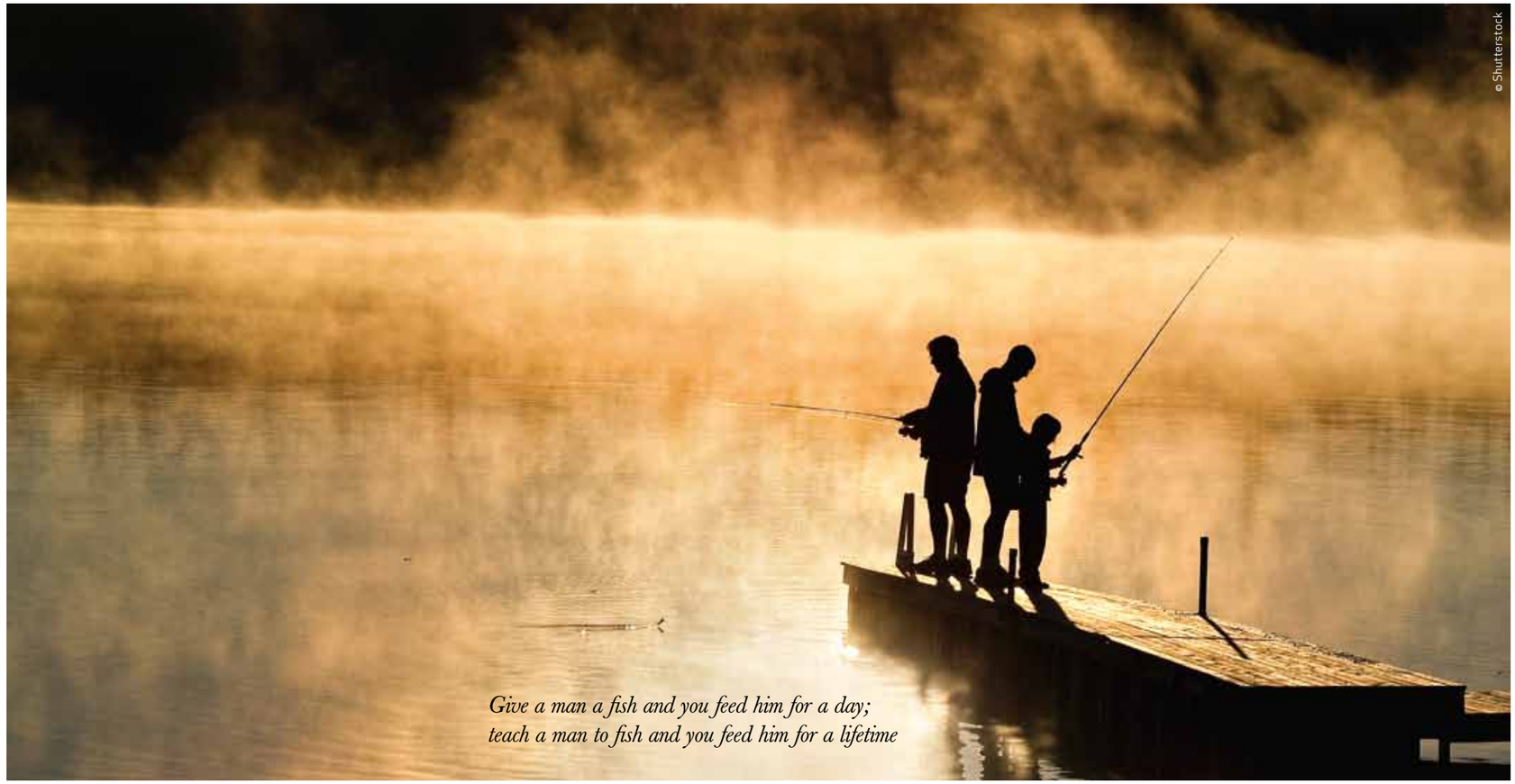
I encourage you all to take a look at the new exciting features that might help you in automating your digital processes. Features such as workflows and task-scheduling (with event triggering) will be very useful!

Be conscious about your time and spend it on what you enjoy doing! Happy coding and have fun at NDC 2012!

At NDC2012 I'll be giving my Revealing the SQL Server Magic presentation, detailing the internal storage format of SQL Server data files – all as a result of creating an open source parser for said files. Usually, the first question I get after doing the presentation is *why*?

Back to Basics

By Mark S. Rasmussen



Though seemingly a very simple question, it's an interesting one, for I have no simple answer. I'm not a database administrator with a need for the utmost inner details, I don't work on the SQL Server engine team and above all, I have no obvious use case for my project. As the tech lead of a small company I spend most of my time wrangling emails, making sure servers and all things technical continues to work while at the same time exercising my C# skills in Visual Studio.

As a developer slowly moving further and further away from "real" development, I felt like I was missing something. I wasn't just doing less development, the development I was doing was slowly being abstracted more and more away. We didn't just use "a" framework any longer; we used frameworks built on other frameworks, all supported by DALs, ORMs and a multitude of other TLAs beneath. We'd be stressed hard to get any further away from the actual bytes in which an integer is stored. Ask most successful and productive developers today what endianness a C# int would be stored in on their machine, and their answer would most likely be "endianness?"

The fact is; we rarely need to know anything about things like endianness, byte structures, binary serialization and the likes thereof. Does your application use 20 or 40 megabytes of memory? Who cares – we've got gigabytes. Will your <insert-popular-nosql-solution-of-the-day> scale to X amount of entries/reads per sec? Who cares – just add another instance. While this may work out just fine, I couldn't help but feel that I was missing out on something.

FINDING MY ROOTS USING C# AND SQL SERVER

As an experiment, I thought I'd write a simple parser for one of the simplest SQL Server data structures – a record. The parser was made and I presented it during a session on SQL Server internals – using my parser purely as an illustration of how the internals worked. What I didn't expect was the amount of positive feedback I got, as well as being urged to continue what I was doing.

Fast forward about a year later; I've now been working on my OrcaMDF project for just about a year – purely as a late-night hobby. At this point OrcaMDF will read just about 95% of the AdventureWorks sample database that ship with SQL Server 2008 R2 – not just user data, but metadata, file headers, allocation and boot structures as well.

I've had lots of fun and challenges in doing the project. While SQL Server is well documented, the official documentation stops a long way from the actual byte level. There are excellent books written by outstanding SQL Server community members, but those as well stop some way from the level I needed to actual perform the parsing. Over the last year I've read numerous SQL Server books for hints on how data was stored, I've read Books Online documentation and I've reverse engineered parts of the data file format by luring SQL Server into giving me the necessary details – all without neither looking at nor touching the binary files as those are out of bounds.

WHY DID YOU DO IT?

While I didn't have a clear answer when I started, I've slowly come to realize what I've gained by doing this project, and why I'd recommend anyone to do something similar.
I've gained an immense respect for the power there lies in knowing the implementation details of a system like an RDBMS. Don't get me wrong, I have immense respect for the inner workings of the various NoSQL solutions out there as well; I just happened to choose SQL Server for my experiment as that's what I was working with.

At the beginning I had the mindset of most "web scale" developers today – once the database can't handle it any longer, ditch the RDBMS/ACID mindset and shard it out, possibly by using a NoSQL solution. Instead of concentrating on the details, just add enough boxes for the system to reach a state where it performs satisfactorily.

Today I take a different approach. I know exactly how my schema choices affect what's stored on disk. Knowing what's stored on disk, I know how SQL Server is able to traverse my data, whether by seeking or scanning. By knowing just these two details, I'm able to use my intuition to predict a large set of performance characteristics without having to read up on them. I can skip most articles on indexing strategies as I can figure it out myself, knowing what lies beneath, knowing how SQL Server interacts with the disk and what's put in memory. Having reached this point, I realize how far you can scale single-box systems (which does not exclude sharding – this is just about scaling those individual shards).

As a developer, what I find even more valuable is the immense experience I've gotten working with the lowest levels of data structures. I read in data files as humongous byte arrays, starting from fileData[0] and continuing till the end of the file, shifting bits left and right, reversing endianness where appropriate.

Just as I had an epiphany when I completed my first machine architecture course at university, I can feel how my newly gained knowledge affects me, even when working on the 10th tier in a web solution.

Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime.
Now get out your fishing pole and start learning how to fish!

The RavenDB .NET client API is very simple to use, making it quite easy to miss more advanced stuff that can actually make a difference in your application.

By Itamar Syn-Hershko

RavenDB[1] is a transactional, open-source document database written in .NET, offering a flexible data model designed to address requirements coming from real-world systems. RavenDB allows you to build high performance, low-latency applications quickly and efficiently. Built-in with RavenDB is a fluent and easy to use .NET client API, replication and sharding support, caching, a management studio, full-text search support and many other powerful features which are sometimes left forgotten.

In this article we are going to explore two of these lesser-known features, and show how they can greatly improve your application and your business.

1: SUGGESTIONS

Imagine the following scenario: You go on a website, and look for some info on an acquaintance. You search once, twice, but can't find it, so you give up. A few days later you discover Danny is actually spelled Danni, but it is already too late. The website you gave up on lost you - a potential user, or worse: a customer.

Sounds familiar? This happens time and again with too many websites and applications. Trying to guess what the user was actually looking for and trying to provide him with meaningful alternatives is being considered by many developers overkill. By doing so, they don't realize the full potential of their application and lose both customer and business.

RavenDB provides a very easy and intuitive way of providing alternative results for queries that returned no results, just like Google's "Did You Mean?". Applications backed by a RavenDB store can, with a very few lines of code, provide their users with suggestions of alternative queries they can try if their original query did not yield good results.

Here's everything you need to do to add such a feature to your e-store:

```
public ActionResult ProductByTitle(string searchTerm)
{
    // The query - get a product by it's title via the Products/ByName index
    var query = from product in RavenSession.Query<Product>("Products/ByName")
                where product.Title == searchTerm
                select product;

    var result = query.FirstOrDefault(); // Issuing the query

    if (result != null) // We found an exact match
        return View("ProductPage", result);

    // Query had no results
    var sgst = query.Suggest(); // Look for suggestions based on the query

    // One suggestion found - the search term was probably misspelled
    if (sgst.Suggestions.Length == 1)
        // If you care about SEO, do a 301 redirect instead
        return ProductByTitle(sgst.Suggestions[0]);

    // Show suggestions to the user
    // Don't forget the condescending "Did you mean?" :)
    return View("Suggestions", sgst.Suggestions);
}
```

2: MORELIKETHIS

Let's continue with the e-store model. Many big online stores - Amazon for instance - maximize their profits by showing users "related products" in product pages and during check-out. Similarly, websites like CNN can get users to spend more time at their website by showing links to related content at the bottom of an article. Unlike what you might think, this doesn't require a full editorial staff to through all your content. It can be done fairly easily by comparing data in one content entity to the rest of the content, and showing the highest ranking ones to the user. So the question remains – how can you do this easily and efficiently? With RavenDB adding this feature is just a matter of dropping a bundle on the server (that is, copying a DLL to the /Plugins folder on the server[2]), and adding a reference to that bundle's complementary client DLL. Once that is done, all that is left for you to do is approach the RavenDB server with a document ID, and tell it which index to use for the comparison:

```
var list = session.Advanced.MoreLikeThis<Product>("Products/ByName",
                                                  "products/123");
```

To get the most out of this feature, you want the lookup to be performed on text properties like title and description only, and make sure they were indexed as Analyzed. Doing so will utilize RavenDB's full-text search capability behind the scenes, and will maximize relevance of the products considered relevant.
It is also possible to perform fine-tuning and adjustments, for example to hand-pick what properties to use for the actual comparison, and what is the minimum or maximum word length[3]:

```
using (var session = documentStore.OpenSession())
{
    var list = session.Advanced.MoreLikeThis<Product>("Products/ByName",
            new MoreLikeThisQueryParameters
    {
            DocumentId = key,
            Fields = new[] {"Title"},
            MinimumWordLength = 3,
    });
}
```

THERE'S A LOT MORE

RavenDB has many other cool features you can use very easily to enhance the user experience of your application, and probably your profits too. Checkout the official docs at http://ravendb.net/docs for more.

REFERENCES

[1] http://ravendb.net/
[2] http://ravendb.net/docs/server/bundles
[3] http://ravendb.net/docs/server/bundles/morelikethis

© Shutterstock

PRO GAMERS can create better AGILE TEAMS

By Kari Amelie Fiva Aasheim

Research shows you'll need about 10 000 hours to reach an elite level at something. Research also shows that many young adults have spent about 10 000 hours playing games since their early childhood. They've played board games, video games, pc games, console games, arcade games and online games. These gamers have unlocked a secret valuable for IT companies on the lookout for new employees.

Every Easter a "geek conference" is held in Vikingskipet, Hamar, where kids can bring their computers and set up their coolest game-, play, music and digital chill out area. A lot of companies are there as well trying to reach these aspiring new IT geniuses. Norway's minister of Government Administration, Reform and Church Affairs, Rigmor Aaserud, also paid them a visit to inspire the youth to continue their interest in IT. However, the Minister did not seem to find anyone she wanted to talk to, saying "I have to find someone who is not just playing games" (Aftenposten nettavis 05.04.2012).

Dear Minster of GARC, there in the conference hall you have the second generation of gamers. Fresh out of the oven, with more than 10 000 hours spent on gameplay, and these kids are – our future. Through game playing they have all developed a skill set which everyone should admire. Given that a young adult in his/her early 20's have spent 10 000 hours in front of a screen, it would be a huge waste to not try our very best to learn from them. But what is it they are so darn good at? And how can we use that to create new value and inspire new jobs or just improve our existing work environments?

Kids are not the number one consumer of video games, adults in the age 30+ are. Trained gamers are excellent at quickly getting an accurate overview of a chaotic scene. They will immediately start organizing the playing field, getting a more structured view of the challenges set before them. When they know what the goal is, they will automatically start thinking tactics. They will automatically start moving things around, like pieces on a chessboard, playing cards in solitaire, or troops in an RTS (Real-Time Strategy) game, so that they can start strategically moving in on their target.

And as important, gamers are used to fast iterative learning. They might fail, but they will immediately revise their game plan and get right back in the game, more experienced and with more determination.

If our minister of GARC and other companies could learn to appreciate and value this mindset, they could then mold these gamers into a new kind of project members and team players. Then we might see a shift in how teams work together. Gamers playing Halo, Battle Field or even World Of Warcraft will create guilds, or ideal teams, which are based on performance rather than employer, age, sex, senior/junior level, or cultural background.

Teams who play games together develop a collective mind set which again leads to improved team loyalty, motivation and effectivity in their grand mission to win, score goals or reach the next level. Transfer that to an iterative development scene, and you will have a team performing brilliantly.

Companies need to understand how to better motivate their teams (1up!), and teams need to learn from gamer teams on how to communicate effectively, even though they are completely virtual.

C#ing, The Dynamic Way

By Shay Friedman

C# is a static language. It has been that way since the very beginning and we all got to love it the way it is, one way or another. However, since .NET 4.0, C# supports the *dynamic keyword* which gives it dynamic capabilities. This doesn't make C# a full blown dynamic language, but it definitely opens up a lot of interesting scenarios which have been available only for dynamic language developers.

WHAT IS THE DYNAMIC KEYWORD?

The dynamic keyword enables a feature called "dynamic member lookup". The dynamic type itself is treated as a regular *System.Object* except one very important difference - any member access - let it be a method, a property, an indexer or any other member - is not verified during compilation time. The resolution of this member is done during runtime, instead.

For example, in the next sample, the call to the *DoCrazyThings* will be resolved only during runtime:

```
dynamic crazy = GetCrazyInstance();
crazy.DoCrazyThings();
```

That's all very cool, but how is this going to help me in my everyday tasks? Read on.

USE CASE 1 - GENERICS

Dynamic language developers, like Rubyists or Pythonists, know this in their hearts - once you write a method, you make it as generic as it can be. The advantages of that are enormous - after a while you find yourself just reusing code because the generic methods you've written fit your new code like a glove.

C# has Generics, which is a wonderful thing. However, Generics in C# is limited by constraints which makes it much less generic than it could have been. For instance, can you write a single method that adds two objects using the plus operator? The answer is no. In such cases, we can use the *dynamic* keyword to get rid of the limitations of C#'s Generics.

In the sample below I use the *dynamic* keyword to create a generic method that uses the plus operator (+) to add two dynamic parameters:

```
public T Add<T>(dynamic a, dynamic b)
{
        return a + b;
}
```

This method accepts **any** two objects that support the plus operator between them. For example, two integers, two strings, a string and an integer and even custom classes that overload the plus operator. It removes the need to create an overload for each of these cases - write once, use everywhere:

```
int         res1 = Add<int>(1, 2); // = 3
double      res2 = Add<double>(1.5, 2.75); // = 4.25
string      res3 = Add<string>("Hello", " World"); // = "Hello World"
CustomClass res4 = Add<CustomClass>(
        new CustomClass(), new CustomClass());
```

USE CASE 2 - METAPROGRAMMING

A language that supports metaprogramming usually means it supports two features: a way to investigate objects during runtime (reflection), and a way to create or change objects during runtime - "programs can write programs".

In C#, reflection has been around for a while and is an elgant(ish) way to support the first feature of the two. However, whoever wanted to create or alter objects during runtime had to dive into IL tricks, *Reflection.Emit* and other only-super-smart-people-understand stuff. The dynamic keyword made this a bit better.

A part of the *System.Dynamic* namespace is the *DynamicObject* class. This class, when overridden, enables you to create entirely dynamic classes. These classes get to control every aspect of interaction with them - method calls, property sets, etc. For example, the next class will write out any call to methods it receives. Notice that the actual methods do not really exist:

```
public class WriteOutMethodCalls : DynamicObject
{
 public override bool TryInvokeMember(
   InvokeMemberBinder binder, object[] args, out object result)
 {
  Console.WriteLine("You tried to invoke {0}", binder.Name);
  result = null;
  return true;
 }
}
```

When using this class we need to initialize it and save the result in a *dynamic* type variable:

```
dynamic w = new WriteOutMethodCalls();
w.Hey();
w.Woohoo();
w.YouCanCallAnyMethodYouWant();
```

This is just a sample, of course. At the moment numerous folks have already taken advantage of this type of classes to create very interesting frameworks. Take a look at Massive, Simple.Data, EasyHttp and ElastiCObject - they all use *DynamicObject* based classes in order to do their magic. And the first time you see what they're capable of, it actually looks like magic.

CONCLUSION

In this article I went through two possible reasons to use the dynamic capabilities of C# - generics without limitations and metaprogramming. In addition to these you can easily interop with DLR-based languages like IronRuby and IronPython, use it for Reflection scenarios and more. In conclusion, try it out and use it when you think it fits. Once you really need it, you won't believe how you managed without it so far.

This article is about how to choose the right cloud services, services providers and what to pay attention to. From a management side, not the technical side of things.

THE CLOUD SECURITY RULES

- choosing right from an organizational point of view

By Kai Roer

FIVE AREAS TO CONSIDER

When an organization considers to use cloud services, and also when your organization is already using them, there are quite a few pitfalls worthy of investigation. The five most important aspects are:

- Business model
- Risk assessment
- Compliance
- People
- Technology

Any organization considering cloud services, must address all the topics. Let us take a look at each one, and what makes it important.

BUSINESS MODEL

The cloud services and their providers comes in all shapes and sizes - from free Gmail, calendar and simple CRM tools, to full-fledged, specially designed enterprise software and services. The huge diversity allows for an equally diverse number of business models for cloud service providers. From free, via freemium to tailored, highly priced solutions - the ways of making money by offering cloud services are many.

Things you may want to consider when looking at the service provider's business model, include:

- How do they (the service provider) make money?
- What is the actual price we are paying?
- How important is customization vs. price for us?
- Will they have the financial strength to stay in business?
- What kind of support will we receive? Is it adequate for our needs?

Another important aspect to consider is how can using (or not using) cloud services impact your own business model(s)? This question may not seem so obvious, so let me use the music industry as an example. Over the past 60 years, they have defined and fine-tuned their business model of distributing media (LP, Cassettes, CD, DVD), as well as harvesting licensing fees from music played in radio and TV stations. Then came Internet. And cloud services like Spotify quickly became a threat to their current business model, forcing a change. Today, we have not seen the end of it, yet already the music industry is adapting and changing their business models to allow for services like this to distribute their music.

Things you may want to ask about your own business model, include:

- How can (and will) the use of this particular service impact our business model?
- What can we do to leverage the changes in how we implement and use (cloud) services?
- How is cloud services impacting our industry today? And tomorrow (in the future)?
- What else can we do to create even more impact?
- Will our business die if we choose not to implement the cloud service?

I believe the largest impact of the cloud is on the business model - how we think and generate business. If you want success by using (or not) cloud services, you really need to consider how the cloud is impacting your business and your business model today and tomorrow.

RISK ASSESSMENT

Risk is everywhere. The secret to mitigate them, is to know them. The more you know, the easier they are to control. Using a good framework based on best practices is key to any security, and should also be applied when you consider the cloud.

Even in a perfect world, things go wrong. In the cloud, you may have less control to fix, investigate and rescue information. You may also experience that your data and services are un-available at a time when you need it. Accidents happen all the time - choosing the right provider, with the right level of service for your needs, will prove invaluable in case of disaster.

Some questions to ask, might be:

- To what level is your cloud service provider going to be responsive if symptoms of a problem occurs? What if it is a non-technical problem?
- What are the risks for us by using this particular service provider? Can we accept those risks? Why?
- Where is our disaster recovery plan? Who will update it to cover the cloud services?
- Who are in charge of risk assessment and mitigation in our organization? How is the cloud service provider supporting us at this stage?

There is nothing to beat a good old-fashioned risk assessment exercise. If you do not have the expertise in-house, there are quite a few specialists available to you. Use them.

COMPLIANCE

In today's business environment, many organizations struggle with staying up-to-date on new regulations and legislations. The commercialization of cloud services hardened this challenge further. Where-as most business' would be living in happy ignorance of legal issues outside their own borders in the past, today they have to comply with laws in countries where their cloud providers operate. Not only are you legally obliged to know those laws, you are responsible for making your organization follow them.

Add to the complexity - your cloud provider may use data-centers in different locations - in different countries and on different continents. The complexity does not relieve you from your responsibility, it just adds to the challenge.
Some questions to ask if you want to stay out of jail, and keep your stakeholders happy, include:

- Where is my cloud provider located, and where will my data be located?
- What laws will govern my data? Where is the jurisdiction?
- Can I afford a law suit?
- What regulations (in addition to laws) must we comply with?

One example that comes to mind is the huge differences in privacy laws in the USA and EU. Even within EU, you will discover differences between the different countries. Make a point out of knowing which laws you must comply with before you choose cloud - doing it afterwords may be very costly.

PEOPLE

In all the buzzing about cloud services and cloud security, it is very easy to forget the users, the people, the human factor, as I like to coin it. In a machine, you can press the button and things are moving as expected (at least after some tests and adjustments). With people, as we know, things are usually different. They may be forgetting their phone on the train, they may share their computer with a friend or colleague, they may happily use that USB-stick they got for free at that last event.

Contrary to popular belief, there is no way to control people 100%. You may have your policies, you can add technology to monitor and surveil your employees, yet no matter what path you choose, people are bound to make mistakes. The more complex a tool is to use, the harder it is to avoid errors. The more flexibility a solution offers, the larger the risk of doing things wrong.

Yet, if you remove all the options, and treat your employees as machines, they will leave you faster than you can replace them.
Some questions that may help you help your people to succeed with the cloud, may include:

Continues on next page

- What training will be required to help our staff to use the new tools efficiently? How do we make that available?
- What possible errors and mistakes do we think people may do with this tool? What can be done to mitigate it?
- What changes do we need to make to our policies and user contracts?
- What do we think about mobile devices? What about users that bring their own device?
- Is our organization ready to make the migration to cloud? What do we need to change to prepare it?

People tend to be flexible, as long as your give them time enough to change. Expect some to complain about the changes, and some to embrace the cloud.

TECHNOLOGY

When it comes to the actual technology of cloud services, there is not much you need to know about. The cloud technology is not really something new, it is merely another iteration of data-warehousing, software distribution, and scalability. What is important is that there are plenty of hard-working, highly skilled people around to take care of the technology, so you can focus on the organizational and business aspects of the cloud.

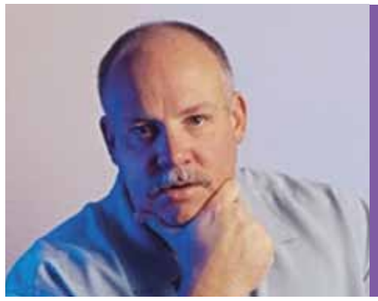
Some questions you may want to ask your technical staff, include:

- What can I do to help you guys do your job better?
- What other services providers would you like to consider?
- What alternative solutions exists if we choose not to move to the cloud?
- What other questions should I be asking you guys?

If your organization decide to use cloud services, you will gain better results faster if you trust your technical staff to do the implementation and roll-out while you focus on the business impact. As with all organizational change, nothing is going to happen if the top-tier management is involved and actively supporting the change.

ABOUT THE AUTHOR

Kai Roer is a European management consultant who focus on information security. He is the editor and co-author of The Cloud Security Rules, where 16 of the worlds leading authorities on cloud security came together to create a resource for leaders, managers and decision makers on how to choose right in the cloud jungle. Roer is also the author of The Leader's Workbook, and the founder of The Roer Group. He has been speaking and training in more than 20 countries around the world. You can read more about him on his personal website: http://kairoer.com



Courses with MIKE COHN

CERTIFIED SCRUMMASTER - CSM – MIKE COHN

This two day course—taught by author and popular Scrum and agile trainer Mike Cohn—not only provides the fundamental principles of Scrum, it also gives participants hands-on experience using Scrum, and closes with Certification as a recognized ScrumMaster.

DESCRIPTION

During the ScrumMaster class, attendees will learn why such a seemingly simple process as Scrum can have such profound effects on an organization. Participants gain practical experience working with Scrum tools and activities such as the product backlog, sprint backlog, daily Scrum meetings, sprint planning meeting, and burndown charts. Participants leave knowing how to apply Scrum to all sizes of projects, from a single collocated team to a large, highly distributed team.

YOU WILL LEARN

- Practical, project-proven practices
- The essentials of getting a project off on the right foot
- How to write user stories for the product backlog
- Why there is more to leading a self-organizing team than buying pizza and getting out of the way
- How to help both new and experienced teams be more successful
- How to successfully scale Scrum to large, multi-continent projects with team sizes in the hundreds
- Tips and tricks from the instructors ten-plus years of using Scrum in a wide variety of environments

COURSE DATE

London: 30. April
Oslo: 4. June
Oslo: 3. September

CERTIFIED SCRUM PRODUCT OWNER - CSPO – MIKE COHN

Certified Scrum Product Owner Training teaches you, the product owner, how to use the product backlog as a tool for success.

As you watch the product take shape, iteration after iteration, you can restructure the Product Backlog to incorporate your insights or respond to changes in business conditions. You can also identify and cancel unsuccessful projects early, often within the first several months. The Certified Scrum Product Owner; course equips you with what you need to achieve success with Scrum.

Intuitive and lightweight, the Scrum process delivers completed increments of the product at rapid, regular intervals, usually from every two weeks to a month. Rather than the traditional system of turning a project over to a project manager while you then wait and hope for the best, Scrum offers an effective alternative, made even more attractive when considering the statistics of traditional product approaches in which over 50% of all projects fail and those that succeed deliver products in which 64% of the functionality is rarely or never used. Let us help you avoid becoming one of these statistics.

YOU WILL LEARN

- Practical, project-proven practices
- How to write user stories for the product backlog
- Proven techniques for prioritizing the product backlog
- How to predict the delivery date of a project (or the features that will be complete by a given date) using velocity
- Tips for managing the key variables influencing project success
- Tips and tricks from the instructors fifteen years of using Scrum in a wide variety of environments

COURSE DATE

London: 2. May
Oslo: 5. September

2012 NDC

NORWEGIAN DEVELOPERS CONFERENCE

june 6-8 Oslo, Norway

We are proud to present the fifth NDC
Be sure to check out our amazing speaker line-up

Hot topics for devoted project managers and .net developers
• 2-day workshop • 3-day conference with 7 parallel tracks

SPEAKER AT NDC

MIKE COHN

SPEAKER AT NDC

ARAL BALKAN

Pre conference
Workshops
June 4-5th

WORKSHOP AT NDC

LEA VEROU

SPEAKER AT NDC

DAN NORTH

SPEAKER AT NDC

PHIL HAACK

Some of our signed speakers:

Jon McCoy	Sahil Malik	Paolo Salvatori
Laurent Bugnion	Jonas Follesø	Richard Campbell
Gojko Adzic	Miguel Castro	Carl Franklin
Igor Tkachev	Geoff Watts	Robert C. Martin
Esther Derby	Greg Young	Bruce Lawson
Udi Dahan	Cory Haines	Christian Johansen
Steve Faulkner	Chris Hardy	Venkat Subramaniam
Billy Hollis	Tom Ball	Itamar Syn-Hershko
Roy Osherove	Michael Friis	
Brad Wilson	Chris Mills	

WORKSHOP AT NDC

CORY FOY

Visit ndcoslo.com for updates on speakers and workhops

Get your All Access Pass now – See back cover for details

organizers ProgramUtvikling Microsoft

ndcoslo.com

NDC 2012 NORWEGIAN DEVELOPERS CONFERENCE

FUNDAMENTAL SECURITY ARCHITECTURE

"Features" in .NET Applications

By Jon McCoy

The underlying architecture of a computer relies on the streamlined pattern of interpretation, computation, and process tracking. Nowhere in this architecture is there an impenetrable trust algorithm or interpreter that asks the question of its instructions—"should I do this?"

A number of attempts at protections exist but each has an exploitable vulnerability. This allows full implicit trust and control over computers that makes all applications susceptible to modification and override of their normal functionality. In general, a programmer writes a program with a logical program flow, compiles that program, and tests it to ensure proper functionality. The CPU does not question the instruction; rather the CPU interprets it and computes the result.

The programmer of the .NET framework also functions within this implicit trust. They code in the .NET language(s) of their choice and then rely on the Common Language Runtime (CLR) to ensure that all of the application security, memory allocation, memory collection, and errors are handled properly. The assumption is made that the CLR and CPU are capable of interpreting what the programmer wanted, keeping the programmer's intellectual property safe and ensuring that his application runs smoothly and efficiently on a wide variety of architectures without the programmer needing to be concerned about the underlying hardware. The CLR removes the need for low-level programming through a .NET application's compilation into an Intermediate Language (IL). In most cases, this is efficient and secure; in the cases where it was not, tools were developed to exploit these inefficiencies and insecurities.

Tools such as Obfuscation and Encrypted Packers (some of the same tools used to protect malware) are used to protect the applications and make it more difficult for people who would extract logic/source-code from the released binary. Corporations have capitalized on marketable versions of code-obfuscation/protection software that prevent decompilation. In most cases, such software is insufficient to protect the application.

Software companies and their engineers have the fundamental belief that their millions of lines of code will be difficult and expensive to reverse engineer and that their intellectual property is protected. With advanced tools and minimal training, the effort to enter and affect other applications is diminished to the extent that entry-level individuals are able to view an application's source code or change its logic.

This ease of observation/alteration shows and reinforces the need for advanced protection systems, a solid software development life cycle (SDLC), and external expert reviews. Furthermore, new breed of protection systems and developer thinking are required.

The .NET framework is powerful and ubiquitous; however, without changes to the manner in which corporations and developers write applications, the framework will continue to be exploited. Core architecture needs to be centered on a layered and defensible position, which can subsequently enable highly secured applications. This core enablement is the area of focus for DigitalBodyGuard.

The Team Leader
MANIFESTO

This article is the first chapter of the book "Notes to a software team leader". You can find out more on twitter (@royosherove) and on the blog 5whys.com. You can purchase the (in progress) book at leanpub.com/teamleader

By Roy Osherove

The way we work with our teams must be better – better adjusted to the current reality and needs of the team, business and ourselves.

The values I will introduce here will be covered in depth throughout this book, and I hope that they can help spark a change in you, the reader, to become better.

WHY SHOULD YOU CARE?

You might feel helpless in leading your team to do the things you believe to be 'right'. This book can help.

You might feel like maybe you just want to keep your head above water, and this book will help accomplish much more than that.
You might feel clueless as to what it is you're actually supposed to be doing with your (future) team.

You might just be broken and scared because you have no idea how you are going to get out of a bad situation at work.

I hope I can help. Because I've been there, clueless and scared myself.
I was lucky enough to have some good mentors along the way who challenged me to do the things I was scared to do, to get out of my comfort zone, and to learn things I didn't even know that I didn't know them.

As team leaders, across the world, I think we all suffer from some shared basic bad experiences. Most of us weren't taught how to do this. And most of us are never going to get mentors to help us through this.

Maybe what we're all missing are some good old-fashioned people skills. Maybe we're missing some direction, some overall goal in why we're taking on this role of leading others. I feel that if we go in, unarmed, without a sense of purpose and strategy into a leadership role in software, we've already lost the battle to create real teams, yes, our head might be just above water, but are we really there just to slog through another day? Are we supposed to be as helpless as we are?

We want to be those who create real value, but also happy teams, fulfilled and loyal.

THE MANIFESTO

WE BELIEVE THE ROLE OF A TEAM LEADER IS TO GROW THE PEOPLE IN THEIR TEAM

- We believe in adjusting the leadership style to the current needs of the team, over a single style of leadership
- We believe in challenging ourselves and our teams to always get better, so:
  - ☐ We embrace taking risks for our team over staying safe
  - ☐ We embrace fear and discomfort as learning new skills
  - ☐ We embrace experimentation as a daily practice
    - – With people
    - – With tools
    - – With processes
    - – With the environment
- We believe team leaders lead people, not machines, so:
  - ☐ We embrace spending more time with our team than in meetings
  - ☐ We embrace learning people skills and techniques

DON'T BE AFRAID TO BECOME MANAGEMENT

A lot of developers who just got promoted to team leaders, or just got offered to become one, seem to be against going management. I can understand some of the reasoning, but I don't accept it. You might be afraid that your time will be sucked up by meetings, that you won't have time to do the things you love the most (like coding), that you might lose friendships with people you currently work with

And I agree that there is a foundation for those fears. We've all seen (or been) that person who doesn't have time to do what they love, or fumbles a friendship because of being a management a-hole at work, etc.

- Management, done wrong, can bring about these things. But management, done *right* negates them.

- Management, done right, is a very tough job. That's why you get paid more (unless you're in some sort of a military organization).

YOU CAN MAKE TIME FOR THE THINGS YOU CARE ABOUT

A good leader will challenge the team and people around them to solve their own problems, instead of solving everyone's problem for them. As people slowly learn to solve their own problems, your time frees up more and more to do the things you care more about, and the things that matter more (sitting down with people, actually coding to keep in track with what's going on, and more)

Doing your job as a team leader and asking people to accomplish tasks or to challenge them could indeed feel weird, but doing it will garner *more* respect for you not less respect. Yes, some things will change, but change is inevitable, you might as well own and control how it changes.

AN OPPORTUNITY TO LEARN NEW, EXCITING THINGS EVERY DAY

Nothing beats getting new skills. You, and your team, should always be getting better, and getting out of your comfort zone to learn new things.

This is essential to what a team leader does, I believe. To grow your team, you need to experience personal growth and challenges, so that you can then use those experiences and know what to expect from your team.

EXPERIMENT WITH HUMAN BEINGS

Yes. I said it. You have a team and you can experiment with goals, constraints, and leadership types. It is one of the most enjoyable and interesting things I love about being a leader.

BE MORE THAN ONE THING

You're not a developer, you're also a leader and can change things that bother you, and do things that you think are right.How many times have you said to yourself "I wish I could change X?" - as a leader you can do something about it.If you don't choose to become management - you have much less influence.

A friend once said "There are no experts, there is only us". I think that sentence is spot on - sometimes you're the person who actually has to get up and do something. You'd be surprised how many people will follow. Remember, 90% of success is stepping up to the plate.

CHALLENGE YOURSELF

Several memes I've seen talk about this basic idea:

- Do one thing every day, that scares you
- Get rejected at least once a day
- Hear the word "no" once a day

These ideas are powerful and a good way to measure ourselves if we are actually learning something. I believe that learning something truly new, is not easy or simple. In fact many times it's scary, annoying or makes you give up mid way.

Leadership, done right, can be very difficult to learn, but when you do it, you have grown so much you will be amazed looking back at the person you were before starting this journey.

Introduction to
Thread Parallel Library

DATAFLOW NETWORKS

By Alon Fliess

As we all know and as Herb Sutter predicted back at the beginning of 2005 - "The Free Lunch Is Over". The name of the game is more cores, many more cores on a single CPU chip and many CPUs in one server. The only way to get our free lunch back is to develop parallel code.

As we all know and as Herb Sutter predicted back at the beginning of 2005 - "The Free Lunch Is Over". The name of the game is more cores, many more cores on a single CPU chip and many CPUs in one server. The only way to get our free lunch back is to develop parallel code. Veteran developers, learned to develop parallel code using abstractions such as Processes and Threads. In the past, developing a parallel program implied creating threads; synchronizing the access to shared data and sending signals or messages between threads. There are several problems with this approach. First, it is hard to build, debug and maintain a hand-crafted multi-threading program. Second, using synchronization mechanisms causes performance penalty, sometimes making our parallel code run slower than a single threaded version of the same code. Think about a five lanes highway reduced to a one lane highway; the effect is similar to using a synchronization mechanism such as a lock in your code. Third, creating more threads than CPU cores, leads to a phenomenon called *over-subscription* where the operating system forces a thread to let-go its CPU and does a context switch to another thread, a process known as preemption multitasking. To get better performance, a thread-pool mechanism was invented. Thread pool breaks the one-to-one relation between a CPU core and a task. The pool has many threads, waiting to execute tasks, each time we need to execute parallel code; we dispatch a task, a delegate, to the pool. If we dispatch many tasks, they get stored in the pool, and whenever a thread finishes executing a task, it takes a new one, from the stored tasks and starts executing it. Another performance gain is that we keep the thread-pool threads alive; we don't create and destroy threads many times.

.NET 4.0 Task Parallel Library has improved the thread-pool mechanism in several ways. First, it provides the Task abstraction, we don't need to communicate with a thread-pool and instead, we use tasks as a higher level mechanism. Second, it adds a scheduler class that handles tasks execution. For example, a scheduler can decide to execute the last queued task instead of an old task to achieve a better throughput as opposed to fairness. Another mechanism that TPL uses is a Work-Stealing-Queue. In this mechanism, each thread has its own tasks queue. When a code, which runs in the context of the thread, dispatches a new task, the task goes to the running threads' queue. This means, that no synchronization is needed, since the thread handles its own task queue. It also implies better performance, since there is a good chance that the task information already exists within the executing CPU core cache. Only when a local task queue of a thread becomes empty, that thread will take (steal) a task from another thread's queue.

`Task` and `Scheduler` are both, low level classes. On top of them, Microsoft provides high level abstractions. The first is the `Parallel` class which is good for fork-join cases, for example, the `Parallel.For` that breaks the execution of a for-loop to many parallel loop chunks; each chunk is dispatched as a task to TPL execution. The same goes with `Parallel.Foreach` and `Parallel.Invoke`. The second .NET 4.0 TPL Parallel abstraction, is Parallel LINQ. The beauty of this mechanism is that you add the `AsParallel` extension method to the LINQ query, and as long as the executed code is a thread safe code, you're good to go. C# 5.0 has added another abstraction on top of TPL with the new *`async/await`* keywords. This abstraction allows the developer to write asynchronous code as if it were synchronous, making the compiler do all the hard work. A common scenario is to use this mechanism to run business logic and I/O in the background, while the User Interface remains responsive.

By now, you understand that all higher .NET parallel abstraction layers need to do is one main job - dispatch tasks to be parallel executed by the TPL engine. `Parallel.For` does it by breaking the loop range to small loop chunks, C# *`async`* tells the compiler to create a task with the payload of the rest of the function code and Parallel LINQ has a complex heuristics that breaks the query to many parallel tasks. In .NET 4.5 Microsoft has added a new abstraction on top of the TPL engine, The TPL Dataflow Network, or in short TDF. TDF does the same, it provides an abstraction that leads to dispatching many tasks to be parallel executed, but what is that abstraction?

TDF

TDF is a flavor of the actor model. In Visual Studio 2010, Microsoft introduced a native library, called Asynchronous Agents Library that implemented the actor model. TDF is the .NET flavor of that library. According to Wikipedia:

"The Actor model adopts the philosophy that everything is an actor. This is similar to the everything is an object philosophy used by some object-oriented programming languages, but differs in that object-oriented software is typically executed sequentially, while the Actor model is inherently concurrent."

TDF is used to create and execute a Dataflow network. Maybe, the most common example of a Dataflow network is spreadsheet software such as the Office Excel. In Excel we can treat each cell as an actor, when its value is changed; it sends a message to all these cells, that their values are derived from its value. Excel is maybe a good example, but any multi-stage execution system can be developed using TDF, start with a simple pipeline to a very complex graph. To sum up, TDF is a technology for message passing and parallel execution of CPU and I/O intensive applications, TDF promises high performance, high throughput with low latency.

TDF BASICS

To use TDF, you need to get familiar with the TDF building blocks and some new concepts. Each TDF network is built from message passing dataflow blocks. There are three types of blocks: Source blocks, Propagator Blocks and Target Blocks. When you build the network, you instantiate these blocks and connect them. Each block can: send or receive (or both), messages, buffer the messages in a queue, manipulate the information, has its own behavior of when, to whom and how to dispatch a message and even filter incoming messages.

Let's look at one of the predefined blocks, the `ActionBlock<T>`. This is an Execution Block. You provide the `Action<T>` in its constructor. This action gets called for each message the block receives. When the application starts, the block has no queued messages hence it has no tasks. When you (or other source block) post a message to the block, a new task is created. This task handles the message by calling the action for that message. After it finishes the action it goes back to check if there are more messages. If there are no more messages, the task exits. When a new message arrives, a new task is created to handle it. Using this algorithm reduces the likelihood to get threads oversubscription. Figure 1 & Figure 2 demonstrate this behavior:

```
var ab = new ActionBlock<int>(i =>
            Console.WriteLine("{0}) Task Id:{1}", i,
                                                Task.CurrentId));
for (int i = 1; i < 10; i++)
{
            ab.Post(i);
}
Thread.Sleep(1000);
ab.Post(10);
ab.Complete();
ab.Completion.Wait();
```

Figure 1: Task Lifetime Demo Code

Continues on next page

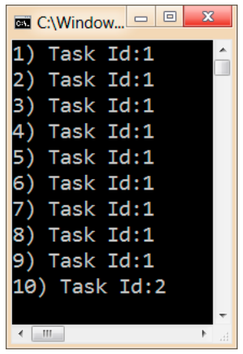


Figure 2: Task Lifetime Demo Result

Nine times Task Id 1 handles the incoming message, after that there is a gap of one second until the next message is posted. Task Id 1 is exited, Task Id 2 is created to handle the last message.

To have a parallel execution of incoming messages the `ActionBlock<T>` can be constructed with `ExecutionDataflowBlockOption.MaxDegreeOfParallelism` value that is bigger than 1.

Figure 3 & Figure 4 demonstrate this behavior:

```
var ab = new ActionBlock<int>(i =>
        {
                Console.Write("{0}) Task Id:{1}\t", i, Task.CurrentId);
                Thread.Sleep(10);
        },
        new ExecutionDataflowBlockOptions
                { MaxDegreeOfParallelism = 4 });

  for (int i = 1; i <= 10; i++)
  {
     ab.Post(i);
  }
  ab.Complete();
  ab.Completion.Wait();
}
```

Figure 3: Parallel Task Execution Code

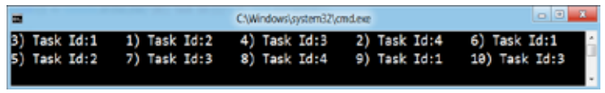


Figure 4: Parallel Task Execution Result

All TDF blocks implements the `IDataflowBlock` interface. This is a simple interface that has one property and two methods. The `Completion` property can be used to investigate the `Task`, which represents the asynchronous operation of the block. If this task is in its completed state, it means that the block finishes its execution and that it will not process anymore messages. You can check the `TaskStatus` property of the `Completion` property to understand if it succeeded (`RanToCompletion`), or failed (`Faulted`) or canceled (`Canceled`). You can also get the `Task AggregateException` if the `TaskStatus` is `Faulted` using the `Exception` property.

Member	Description
Completion	A property that provides the underlying Task object that represents the asynchronous operation and completion of the dataflow block.
Complete	Inform the block to finish all execution and no longer send or receive new messages.
Fault	Inform the block to finish all execution and move to the Faulted state

Table 1 `IDataflowBlock` interface

The `Complete` and `Fault` methods are used to inform the block to finish the execution; `Complete` on success, and `Fault` on failure. `Fault` gets one parameter, a .NET Exception object.

There are three important interfaces that extend the `IDataflowBlock` interface. These are the `ISourceBlock<out Toutput>`, the `ITargetBlock<in TInput>` and the `IPropagatorBlock<in TInput, out TOutput>` which extends the formers. Blocks that provide messages implement the `ISourceBlock`, while blocks that receive messages implement the `ITargetBlock` interface. Intermediate blocks implement the `IPropagatorBlock` interface that extends both `ISourceBlock` and `ITargetBlock`. The `TInput` & `TOutput` generic types, represent that types of the data that each block handles.

In addition to the methods that come from `IDataflowBlock`, the `ISourceBlock` interface defines four more methods. The `LinkTo` method is used to connect a target block. The `ConsumeMessage` is called by a target block to get a message. The other two methods serve for transaction-like operations. These are the `ReserveMessage` and `ReleaseMessage` that can be used to prevent deadlocks. Most of the blocks that implement `ISourceBlock`, implement also the `IReceivableSourceBlock`. This interface provides two methods, the `TryReceive` and `TryReceiveAll` that can be used to synchronously get messages from a block without the need to be linked to it.

In addition to the methods that come from `IDataflowBlock`, the `ITargetBlock` implements the `OfferMessage` method. A source block calls this method to offer a new message. `OfferMessage` & `ISourceBlock`'s `ConsumeMessage` have a well-defined handshaking protocol for negotiating passing messages ownership.

Only block implementation code uses the `OfferMessage` and `ConsumeMessage` directly. Your code should use the extension methods, which come from the static `DataFlowBlock` class such as `Post` or `PostAsync` to post messages to a target block; `Receive`, `TryReceive` or `ReceiveAsync` to get a message from a source block. The `DataflowBlock` class offers other useful extension methods such as `OutputAvailableAsync` that enables asynchronous wait until a message can be received, which works very well with C# 5.0 *`await`* keyword, the `LinkTo` group of methods that ease connecting the blocks to and form the network, `Encapsulate` that provides a way to build a new block from other TDF blocks, `AsObservable` and `AsObserver` that play as a glue between Rx and TDF, and some other utility functions.

PREDEFINED BLOCKS

The predefined blocks fall into three main categories:

1. Pure Buffering Blocks – deal with various ways to buffer and distribute messages
2. Execution Blocks – manipulate the incoming messages
3. Grouping Blocks – provide various methods to group or batch messages from single or multiple sources

When you instantiate a block, you may shape the block behavior by providing a block option object. All block option types are derived from the `DataflowBlockOptions` class. This class groups a set of properties such as the maximum number of messages that can be buffered, the block task cancelation token, the maximum messages a task can handle and the TPL task scheduler that will execute the task of the block.

Execution blocks can be manipulated by using the `ExecutionDataflowBlockOptions` which in addition to the properties of the base class `DataflowBlockOptions` provides the `MaxDegreeOfParallelism` for concurrent block execution as we've seen in Figure 3.

Grouping blocks can be manipulated by using the `GroupingDataflowBlockOptions` that adds to the properties that come from its base class, `DataflowBlockOptions`, the `MaxNumberOfGroups` that should be generated by the block and the `Greedy` property that determines whether the block should greedily consume offered messages, a behavior that may situate a deadlock.
Table 2 summarizes the predefined blocks:

Block	Type	Category	Description
ActionBlock	Target	Execution	Invokes an action for every message
BatchBlock	Propagator	Grouping	Batches input messages into an array
BatchedJoinBlock	Propagator	Grouping	Joins a specified number of inputs of potentially differing types to a batch of tuples
BroadcastBlock	Propagator	Buffering	Stores and distributes a copy of the last message received to all of its targets
BufferBlock	Propagator	Buffering	Simple Buffer
JoinBlock	Propagator	Grouping	Joins a number of inputs of potentially differing types to form a tuple
TransformBlock	Propagator	Execution	Transforms an input message to a different output message (similar to LINQ Select)
WriteOnceBlock	Propagator	Buffering	Handles receiving and storing at most one element

Table 2: TDF Predefined Blocks

BUILDING THE NETWORK

You create the network by instantiating predefined, custom and encapsulated blocks and by linking them together. You can provide properties to each link by using a `DataflowLinkOptions` instance. The `Append` property defines whether the link should be appended to the source's list of links, or whether it should be prepended. The `MaxMessages` sets the number of messages that can be consumed across the link, which provide a way to unlink a block from the network after it consumes this number of messages. The last option that can be set is `PropagateCompletion`. When set to true, this option, instructs the block to propagate completion and faulting notification to its target blocks.

SENDING AND RECEIVING MESSAGES

You can send and receive messages from TDF network in synchronously or asynchronously manner.
To send messages synchronously, use the `Post` method: `block.Post(message)`. There are two ways to read messages synchronously; the first way is to use the `Receive` method. This is a blocking method; you will wait until there is at least one message in the output buffer of the block. Another way is to use the `TryReceive` method. This method returns false if there are no messages in the output buffer: `while(block.TryRecieve(out message))`
To send messages asynchronously, you can use the new *await* keyword with the `SendAsync`:

```
for (int i = 0; i < 5; ++i)
{
        await block.SendAsync(i);
}
```

To receive message asynchronously use `await` keyword with the `ReceiveAsync`:

```
for (int i = 0; i < 5; ++i)
{
        Console.WriteLine(await block.ReceiveAsync());
}
```

USING TDF TO SOLVE A PROBLEM

To demonstrate the power of TDF, we are going to simulate an electrical logic circuit known as a Full Adder. It simulates adding two binary digits with carry and output the result and the carry:

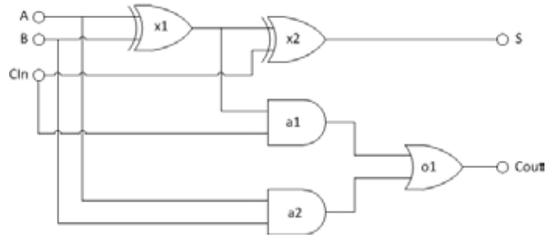


Figure 5: Full Adder Circuit

Continues on next page

To simplify the building process, we have created a class called `Gate` which gets the logic function as an `Action` (e.g. using a lambda expression). A Gate is built from three blocks, a join block that joins two inputs and forms a tuple, a transform block that executes the logic action and a broadcast block that pushes (clones) the output result to all other connected gates. To handle the logic circuit inputs (A, B, Cin), we have three broadcast blocks. To handle the output we have a join block that joins the two results (S, Cout). Figure 6 shows the TDF network for the Full Adder. Figure 7 shows the source code and Figure 8 is the result.

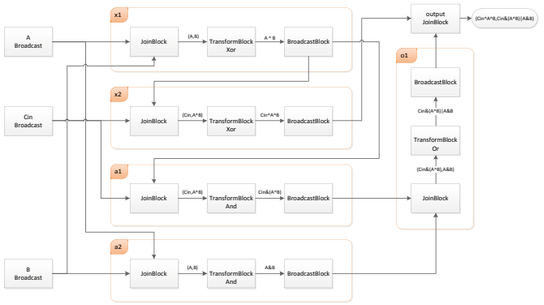


Figure 6: Full Adder Implemented as TDF Network

```
using System;
using System.Threading.Tasks.Dataflow;

namespace DataflowLogicCircuit
{
  sealed class Gate
  {
    private readonly JoinBlock<bool, bool> _joinBlock =
                        new JoinBlock<bool, bool>();
    private readonly TransformBlock<Tuple<bool, bool>, bool>
                        _transformBlock;
    private readonly BroadcastBlock<bool> _broadcastBlock =
                        new roadcastBlock<bool>(x => x);

    public Gate(Func<bool, bool, bool> logicFunction)
    {
          _transformBlock =
                              new TransformBlock<Tuple<bool, bool>, bool>(
                        inputs => logicFunction(inputs.Item1, inputs.Item2));
          _joinBlock.LinkTo(_transformBlock);
          _transformBlock.LinkTo(_broadcastBlock);
    }

    public ITargetBlock<bool> Input1
    {
      get { return _joinBlock.Target1; }
    }

    public ITargetBlock<bool> Input2
    {
      get { return _joinBlock.Target2;}
    }

    public ISourceBlock<bool> Output
    {
      get
      {
        return _broadcastBlock;
      }
    }
  }
```

```
class Program
{
  static void Main(string[] args)
  {
    Func<bool, bool, bool> and =
                              (input1, input2) => input1 && input2;
    Func<bool, bool, bool> or =
                              (input1, input2) => input1 || input2;
    Func<bool, bool, bool> xor =
                              (input1, input2) => input1 ^ input2;

    var A = new BroadcastBlock<bool>(x => x);
    var B = new BroadcastBlock<bool>(x => x);
    var CIn = new BroadcastBlock<bool>(x => x);

    var x1 = new Gate(xor);
    var x2 = new Gate(xor);
    var a1 = new Gate(and);
    var a2 = new Gate(and);
    var o1 = new Gate(or);

    A.LinkTo(x1.Input1);
    A.LinkTo(a2.Input1);

    B.LinkTo(x1.Input2);
    B.LinkTo(a2.Input2);

    CIn.LinkTo(x2.Input2);
    CIn.LinkTo(a1.Input2);

    x1.Output.LinkTo(x2.Input1);
    x1.Output.LinkTo(a1.Input1);
    a1.Output.LinkTo(o1.Input1);
    a2.Output.LinkTo(o1.Input2);

    var output = new JoinBlock<bool, bool>();
    x2.Output.LinkTo(output.Target1);
    o1.Output.LinkTo(output.Target2);

    Console.WriteLine("A + B + Ci= S (Co)");
    Console.WriteLine("------------------");
    for (int a = 0; a <= 1; ++a)
    {
      for (int b = 0; b <= 1; ++b)
      {
        for (int cin = 0; cin <=1; ++cin)
        {
          A.Post(a != 0);
          B.Post(b != 0);
          CIn.Post(cin != 0);
          var result = output.Receive();
          Console.WriteLine(
                                        "{0} + {1} + {2} = {3} ({4})",
                                        a, b, cin,
                                        result.Item1 ? 1 : 0,
                                        result.Item2 ? 1 : 0);
        }
      }
    }
  }
}
```

Figure 7:
Full Adder Implemented as TDF Network Demo Code

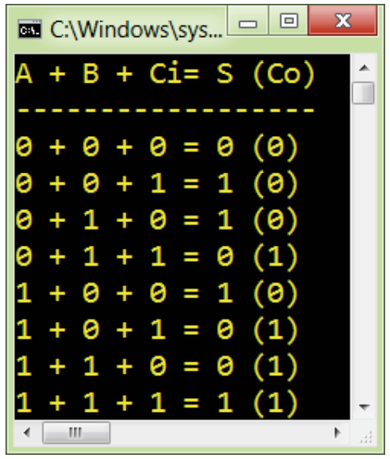


Figure 8:
Full Adder Implemented as TDF Network Demo Result

SUMMARY

Task Dataflow provides a new way to architect and design high performance, high throughput, low latency parallel solutions. There are many areas that this approach can fit easily, from simulating hardware to handling multistage pipeline of execution. You can design the network using the predefined blocks, or you may extend these blocks by implementing your own custom blocks. You link all the blocks to form a network and you post messages to the network. TDF plays nice with the new `async/await` C# 5.0 keywords. Since it uses the same TPL low level mechanism as the `Parallel` class and PLINQ, you can use all of them together without having a performance penalty from over subscription of threads.

Another .NET technology that has some overlapping with TDF is Reactive Extension (Rx). If you are not familiar with Rx, I encourage you to learn about it. TDF and Reactive Extensions are better together. You will find that sometimes using LINQ in Rx is easier than forming a TDF network, and sometimes the mission is too complicated for Rx, while a TDF network provides the needed control. TDF blocks can easily communicate with Rx and vice versa.

Securing (ASP.NET) Web APIs

These days all the kool kids (TM) talk about lightweight and REST web services and how they are far superior than the dinosaur SOAP/XML combo we got used to over the years (mild sarcasm). But they indeed have a point that HTTP/JSON based services allow for more efficient modeling of (public facing) web APIs especially in the face of web based and mobile architectures. Microsoft accounts for this by releasing a brand new service framework called "ASP.NET Web API" as part of the MVC 4 wave. But is this approach maybe a little bit too lightweight - especially when it comes to security? We'll find out.

By Dominick Baier, Thinktecture

ASP.NET Web API is a framework for building HTTP-based services. Though it mimics the way MVC works (routes, controllers, actions), it is hosting independent and can be hosted inside IIS or some other arbitrary server (a self-host based on WCF is included in the bits as well).

By default (though there are also other options) you implement your service by deriving from a class called *ApiController* and start implementing action methods that correspond to the various HTTP verbs, like Get, Put, Post and Delete. The controller infrastructure then uses the model binding approach to surface input data into the action methods, and a content-type based serializer (or formatter) to send the responses back. In addition to this abstraction model, you have full access to the HTTP protocol to tweak the various headers, encodings and status codes. See figure 1 for the birds-eye view architecture diagram.

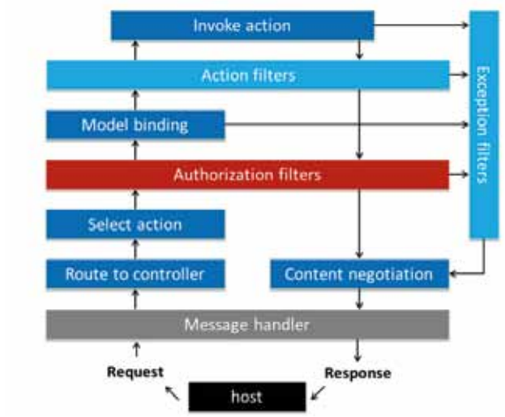


Figure 1: Web API Architecture

AUTHENTICATION FOR HTTP-BASED SERVICES

When looking at the (humongous) WS-Security spec for SOAP based services, it appears that in comparison HTTP does not have that much to offer. But that's not true at all.

In contrast to the SOAP world, the HTTP community made the decision that they will always use SSL to protect the transport channel, which removes a lot of the complexities you have to deal with unsecured communication protocols. By removing that complexity (and taking the SSL requirement for granted), we can focus again on the actual payload, and not how to protect it.

The moral equivalent of the security headers in SOAP are HTTP headers in REST. And even more specifically, a special header called *Authorization* is typically used to transmit credentials (sometimes other means like query strings or cookies are used, but that is just an implementation detail). The authorization header can transmit arbitrary credentials as long as they can be encoded as a string. Examples would be simple username/password pairs, API keys or full-fledged security tokens like SAML, SWT or JWT. To give the recipient a hint which type of credential is submitted, the authorization header consists of two parts: the scheme (credential type) and the value (credential). After parsing that header, the service can then decide if he wants to accept that credential, and if not send back a *Www-Authenticate* header that specifies the preferred scheme along with a status code of 401 (unauthorized). If he wants to accept the credential, he verifies the header value and if successful, establishes a principal for that user.

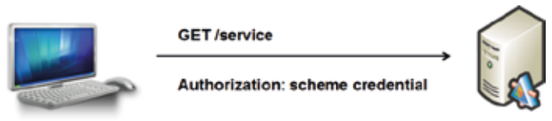


Figure 2: Authentication in HTTP-based Services

SECURITY IN ASP.NET WEB API

When looking at the architecture diagram, there are several building blocks where security features can be implemented. Let's start at the bottom of the stack.

THE HOST

ASP.NET Web API can inherit the client security context from the host - in fact this is the default behavior. That means that when you host the API in the context of a web application, the surrounding security setup (e.g. Forms or Windows Authentication) is just re-used. You would "see" the same client as in a web form or an MVC controller. This is e.g. very convenient for Ajax-style callback scenarios. But this also means that you tie yourself to the hosting environment (e.g. ASP.NET) which is fine in many situations, but Web API has more to offer.

MESSAGE HANDLER

Message handlers are already part of the Web API specific plumbing and thus hosting independent. They are the most low-level extensibility points that Web API has to offer and ideal for implementing behavior that should apply to all or a group of related services in your application. The heart of a message handler is a method that gets to see all incoming requests and outgoing responses. In here is a perfect place for inspecting incoming HTTP messages and looking for credentials, e.g. the on authorization header. From there the message handler can validate the credential, establish the principal as well as orchestrate the various error status codes and response headers. At [1] you can find an extensible authentication framework based on message handler that supports Basic Authentication, SAML and SWT tokens out of the box. In addition this framework also implements claims-based identity and the WIF pipeline for Web API. This is definitely worth a look.

AUTHORIZATION FILTERS

Authorization filters can control authorization on a controller or action method basis. As you can see they are still quite early in the pipeline to avoid using too many resources or CPU cycle before we actually know if the user is allowed to access the resource he tries to. Technically speaking an authorization filter is a special type of .NET custom attribute. The runtime inspects the controller, and if it finds such an attribute, it gets invoked. Inside the attribute you can check the identity of the client as well as details about the current request. Based on that information you can either grant access, or send back some alternative response, e.g. an unauthorized status code (401). By deriving from *AuthorizeAttribute*, you can write you own custom authorization filters to enforce your authorization policy.

MODEL BINDING

Model binding is the process of converting HTTP input data (form fields, query strings etc.) to a .NET type, often called the input model. You can place several attributes on the model to control how that conversion should take place. You can define validation rules on the model, e.g. required fields or that a certain field should be an email address, URL etc. You can use the .NET *DataAnnotations* namespace for that. In addition you can also control which fields should be bound at all, and which should never be bound (both white- and black-list approaches are supported). And as always keep in mind: all input is evil, until proven otherwise (see [3]).

EXCEPTION FILTER

Exception filters are essentially error handlers. They can be either controller local, or global. In there you get to see error details and get to decide which parts of those details are for internal logging purposes and which are for the end user. Be sure to not leak internal information to your users (or potential attackers).

SUMMARY

As you can see, you can implement just every security scenario using HTTP-based services. The main difference compared to SOAP is really just the SSL requirement. Once you decide going down that route, you have many, many framework choices for implementing your services. ASP.NET Web API is the new framework from Microsoft which feels very familiar if you have done MVC or WCF work before. I also briefly described the several architectural pieces that can be used to implement security features in such services. Web API is still work in progress, but it has all the relevant knobs and switches to build robust and secure applications that can be used in a variety of security-relevant scenarios.

[0] http://aspnetwebstack.codeplex.com/
[1] https://github.com/thinktecture/Thinktecture.IdentityModel.Http
[3] http://odetocode.com/Blogs/scott/archive/2012/03/11/complete-guide-to-mass-assignment-in-asp-net-mvc.aspx

DISCLAIMER: This article is based on Beta 1 of MVC4 and ASP.NET Web API. You can also download the open source version at [0]. Be aware that this is work in progress, and some code snippets may not work with the latest build and/or previous or future versions.

WRITING EFFECTIVE ACCEPTANCE TESTS

In Continuous Delivery acceptance tests act as an executable specification of the behaviour of a system. This is, intentionally, a very behaviour-focussed view of the tests and without this focus there are some bear-traps in the process awaiting the unwary.

Evidently if these tests are to act as our executable specifications they must executable - that is to say they are computer programs and suffer the same strengths and weaknesses as any other computer program. The key weakness that I have in mind is that they are easy to design poorly.

By Dave Farley
Author of Continuous Delivery Reliable Software Releases Through Build, Test and Deployment Automation
www.davefarley.net

THE TOUGH PART OF AUTOMATED TESTING

The commonest question that I get about CD is how can you possibly keep all of those acceptance tests running? This is born out of the all too common experience of automated functional tests, namely that they are fragile and very prone to the application changing beneath them and so invalidating the tests. This is a common problem, but it is amenable to solution with good design. The key here is a clear separation of concerns and perhaps more important terminology.

The root of this problem of fragility is that the tests know too much about the way in which the problem is solved and they express too little about what the test is really trying to assert. This is in part a symptom of us programmers being lazy and not properly understanding the problem, and rather focussing only on the solution.

Such tests are tightly-coupled to a specific implementation and as a result are very fragile in the face of change. One reaction to this is to say "automated testing is too hard so we won't do it". The trouble is that if it works automated functional testing is enormously valuable. If only we could find a way to make it work and stand-up in the face of change...

WHAT IS THE RIGHT LANGUAGE FOR ACCEPTANCE TESTING?

My bet is that half my readers, are now thinking of languages like "Java", "C#", "Ruby", "HTML" - wrong! The only sensible language for testing is the language of the problem domain. Successful tests must express a business idea. They must assert an outcome relevant to the problem domain not some technical translation of it.

My work is currently in the field of finance, I write trading systems. For my business the core behaviours of our system are primarily about to placing orders in a market. I will use order placement as an example, but what I say for order placement is true for every interaction with our system. We have created a Domain Specific Language (DSL) for writing our acceptance tests. The DSL has several important properties, but most important of all is the level of abstraction and the importance of capturing the intent of a test in the language of the business.

A typical instruction in a test to place an order in a market looks like this:

```
placeOrder("market: myTestMarket", "10@50");
```

I can specify more detail if I need to describe the precise nature of an order in more detail, but if all I want to do is place an order for 10 units at a price of 50 in "myTestMarket" this will suffice. We have tens of thousands of test cases that place lots of orders and the vast majority specify no more than these few details. There is nothing in here that gives any clue about the technicalities of how the order was placed, and that is the point.

For my current system you can place orders via it's user interface, Java client, C# client, HTTP protocol or FIX (Financial Information eXchange) protocol. Each of these channels of communication with our system have automated tests, implemented in our DSL that we have implemented for the purpose.

Our technical compromise that allows us to test these different channels is that you must specify the channel of communication that is being used for any given test case. We have two ways of doing that:

```
ui.placeOrder("market: myTestMarket", "10@50");
fix.placeOrder("market: myTestMarket", "10@50");
api.placeOrder("market: myTestMarket", "10@50");
...

 or

  @Channels(UI, API, FIX)
public void someTestCase()
{
       placeOrder("market: myTestMarket", "10@50");
}
```

Our DSL runs test cases as a JUnit test. In the second example we have implemented a simple custom JUnit test runner that, given an @Channels annotation, will re-run the test case on each channel.

Beneath the surface of our DSL there are layers of code, some shared between channels some specific to them. Fundamentally the DSL is divided into two layers, a layer that presents the language that tests can be written in, and a layer of protocol drivers that describe how that the interactions implied by the DSL instructions are expressed for a given underlying channel. In once case this is the simple formulation of a message and it's dispatch. In another there is code that knows how to populate each field in the UI to specify the order and which buttons to press to send it..

We have had many cases where radical changes in the implementation, or even changes in the underlying technology, have left the tests passing. We had to re-work the relevant protocol driver, but the test cases, where the real value and the majority of the work resides, remain valid though any change except a change in the requirement itself.

At its root this approach is only about good software design, separating the concerns of the test case from the concerns of the underlying technology. By simply layering the test infrastructure and being cautious about the design of the DSL to avoid inadvertent coupling, we side-step the common problem of fragile functional tests.

When Windows Phone 7 was first introduced, some people were a bit disappointed by the lack of not being able to run code in the background. Through its application lifecycle management, Windows Phone effectively only allows one single application to run (apart from some first-party apps written by Microsoft). That app is the foreground application. When another application becomes the foreground application, the other application basically stops running (in other words, it doesn't get any CPU cycles anymore).

Working in the background in WINDOWS PHONE 7.5

By Gill Cleeren

In Windows Phone 7, this application would become tombstoned, meaning that it's removed from memory but state is saved so that the developer can act on the fact that the user might return to the application. Since Windows Phone 7.5 (the Mango release), the app can also become dormant. This new state keeps the memory assigned so that when resuming the app, the user is returned to the same state. While the application wasn't in the foreground anymore, it wasn't able to do any processing though.

Both tombstoned and dormant states don't allow the application to do any background processing. Still, Microsoft did add quite a few controlled options to do background processing in the Mango release. Through the use of some OS-managed services, developers get the ability for running code, downloads, audio and more in the background while at the same time, the OS is in control (and therefore, the developer can't drain the battery or extensively use the network). For the end-user, this of course results in a better experience. Let's take a look at some of the options Microsoft added for background processing.

BACKGROUND AGENTS

When we talk about background processing in Windows Phone, it's a different kind of processing than we are used to think of. When thinking of Windows itself, an app can be running in the background. In most cases, this is the same application instance, which can still use CPU cycles and still has memory assigned to it. In Windows Phone however, when the application isn't running in the foreground, it doesn't get access to the CPU. The way Microsoft implemented background processing is therefore through a separate part, namely an agent. An agent is a separate assembly, included in the XAP file of your application. Its execution is managed by a system service, not by your application. You can think of an agent as a block of code that is called on certain intervals by the OS. It runs in a different process than your application, meaning that it's not dependent on your application to run.

Adding an agent is easy in Visual Studio. When looking at the New Project Template dialog, you'll notice there's a template called the "Windows Phone Scheduled Task Agent". This template creates a separate project in your phone solution. Its most important part is a class that inherits from ScheduledTaskAgent. In its OnInvoke method, we can write code that will execute when the agent is ran.

What can we ask an agent to do? Within the agent, we have access to most parts of the phone API, although some are not allowed. Obviously, it's not possible to access anything in the UI (remember, this agent is being executed in the background while the main application is not required to run). For other obvious reasons, the agent's code can't interact with the camera or the microphone (I assume most people wouldn't like an application that installs a background agent which takes pictures behind their backs…). Apart from some other exceptions, a background agent can do quite a lot: it can do tile updates (in fact, it's one of the most interesting ways to update a live tile!), it can send toast notifications, perform location lookups, interact with isolated storage and even work with sockets. It's easy to see that there are numerous great scenarios to build with the background agents, including location trackers, tile updates and polling a service.

An agent is however limited in the amount of time it can run. In fact, there's two ways of running an agent: through a PeriodicTask or through a ResourceIntersiveTask. The first one runs every 30 minutes

(depending on several factors including load on the system) whereas the latter runs when specific conditions are met (such as being connected over WiFi and to AC power). Depending on what you need your agent for, you must select either of these two (or both). If you require a long-running synchronization process, a ResourceIntensiveTask might be best. If you need to perform tile updates, a PeriodicTask will be your best bet.

BACKGROUND TRANSFERS

Downloading files from within a Windows Phone application requires the user to have your application as the foreground application as long as the download takes. This might not be the best experience for your user.

Windows Phone has another service on board known to mankind as the BackgroundTransferService which as its name implies, allows us to perform transfers of files in the background. From within an application, we can create a BackgroundTransfer instance, pointing to the file we want to upload or download (using GET or POST as the method). We can then register that BackgroundTransfer with the service, which is managed by the OS itself. Therefore, the download is running outside of the application and is therefore not interrupted when the user leaves the application or even reboots the device. The transfers run in the background until complete a completion, the file is physically placed in the isolated storage of the application that initiated the transfer. When running, the application can then access the file regularly.

There are some limitations, mainly to keep things running smoothly. For example, there's a limit on transfers that an application can initiate as well as a limit on file size. Also, device-wide, a limitation is set by the OS on the number of queued transfers.

ALARMS AND REMINDERS

For some application scenarios, developers may want to notify their users about a certain time-related condition being hit. The background agents are most of the time not a good fit for this, since in most circumstances, they only run about every 30 minutes. If we want to notify the user about a reminder or wake him using an alarm, chances are that having to wait (in the worse case) 30 minutes might not be a good idea.

For this specific reason, Windows Phone contains an Alarms and Reminders API. They're both similar in the way they work but there are some specifics to each of them. An alarm is something we set to appear on the screen at a specific time, allowing the user to snooze or dismiss. The Reminder is similar, but here, the developer can specify more context (such as title of the reminder). Both allow to start the application that created them by tapping the alarm/reminder surface.

BACKGROUND AUDIO

Playing audio in the background was originally only possible for the built-in Zune media player. Since Windows Phone 7.5, Microsoft opened this up for application developers, so that they can build their applications that integrate with Zune. This way, it's possible to create a playlist from within your application and keep playing audio even when your application isn't the foreground application anymore.

SUMMARY

Adding background capabilities in Windows Phone opened up a lot of new scenarios in Windows Phone. A lot of applications today in the marketplace already make use of this added functionality. Maybe now is the time to extend your application as well to make the most of what the OS has to offer!

2012 NDC

NORWEGIAN DEVELOPERS CONFERENCE

june 6-8 Oslo, Norway

We are proud to present the fifth NDC
Thanks to our partners who make it happen!

For information about partnership deals, please contact:

henriette.holmen@programutvikling.no

tel.: +47 976 07 456

Become a partner today!

Partners

99X
Agile Alliance
amende
BEKK
BookWorld
bouvet
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING
creuna
ciber
computas
CONCEPTOS CONSULTING
DIPS
EASYQUEST
ELAN A Manpower Professional Company
epinova content management experts
EVRY
GEODATA
glasspaper
Vinmeta
itera CONSULTING
JetBRAINS Develop with pleasure!
knowit
MAKING WAVES
mesan
Microsoft
miles
NNUG
nova net
oktaset by KodeCompagniet
pluralsight hardcore developer training
ProgramUtvikling
Schlumberger
SPT GROUP
SYBASE An SAP Company
Systek
steria
telerik deliver more than expected
ViaNett SMS GATEWAY SPECIALIST - SMS BILLING
VISMA
WEBSTEP

organizers ProgramUtvikling Microsoft

ndcoslo.com

NDC 2012 NORWEGIAN DEVELOPERS CONFERENCE

Understanding Dependency Injection

Dependency Injection is an approach to providing one type with instances of other types on which it may depend. Though defined in Wikipedia as "an approach to testing computer programs", it's more than just about testing. That being said, testing remains an extremely good reason to use DI and one of the more common.

By Miguel Castro

Dependency Injection continues to be an integral part of good software testing. Dependency Injection is a form of Inversion of Control and is sometimes confused as being the exact same thing. Inversion of Control is a software principle whereas control of a class' initialization and execution is delegated to another class, sometimes a container or a manager. That being said, Dependency Injection is a slight subset of Inversion of Control.

SOFTWARE COUPLING

When writing software, it's always a good idea to separate the concerns of the types involved. In fact there is a buzz phrase that describes this process, "separation of concerns". The concept is that each type having either only one, or a very fine-grained set of responsibilities. Of course if we translate our software to the line of business for which it is built, it immediately becomes apparent that a business process involves many steps and thus many types. Most of the time there is a need for one type to work hand-in-hand with another and even become a sub-part of another. In other words, one class may "depend" on another. In fact, the parent type would very likely contain an instance of the type on which it depends, usually passed in through a constructor or a property. The problem is that this creates a coupling between the two types. Not only can the parent type not exist (or compile) without the dependent type, it can be stuck with the way that type wants to do things.

I'll describe this using what I think is one of the best examples to demonstrate a business process that can easily be subdivided, e-commerce. Among other things, three of the functions that are involved in a check-out process are payment processing, customer update, and notification. After an order is placed, a customer's credit card must be charged, after which the customer record needs to be updated with the order he/she placed and the product purchased. Lastly, the customer should probably receive an email with their receipt and order summary.

Let's start with a type that will contain the data involved in my example. This type is called **OrderInfo** and looks like this:

```
public class OrderInfo
{
    public string CustomerName { get; set; }
    public string Email { get; set; }
    public string Product { get; set; }
    public double Price { get; set; }
    public string CreditCard { get; set; }
}
```

To act upon this data, I have a type called **Commerce** with a method called **ProcessOrder**, which performs all the aforementioned sub-processes. In the interest of separating the concerns, I have types called **BillingProcessor**, **Customer**, and **Notifier**. The idea is that each of them might be used from another class in another situation, but it's also a good idea not to put everything in the **Commerce** type to encapsulate the check-out process.

Here are the three classes. Notice that in the interest of simplicity, their methods have no real functionality. If they did it would be complex and involve several resources, such as a database. Also note that each of their methods act upon different pieces of the **OrderInfo** type.

```
public class BillingProcessor
{
    public void ProcessPayment(string customer, string creditCard, double price)
    {
    }
}
public class Customer
{
    public void UpdateCustomerOrder(string customer, string product)
    {
    }
}
public class Notifier
{
    public void SendReceipt(OrderInfo orderInfo)
    {
    }
}
```

The **Commerce** type incorporates each of the other three types by receiving an instance of each in its constructor.

```
public class Commerce
{
    public Commerce(BillingProcessor billingProcessor,
        Customer customer,
        Notifier notifier)
    {
        _BillingProcessor = billingProcessor;
        _Customer = customer;
        _Notifier = notifier;
    }
    BillingProcessor _BillingProcessor;
    Customer _Customer;
    Notifier _Notifier;

    public void ProcessOrder(OrderInfo orderInfo)
    {
        _BillingProcessor.ProcessPayment(orderInfo.CustomerName,
                                         orderInfo.CreditCard,
                                         orderInfo.Price);
        _Customer.UpdateCustomerOrder(orderInfo.CustomerName,
                                      orderInfo.Product);
        _Notifier.SendReceipt(orderInfo);
    }
}
```

Assuming that I have an instance of **OrderInfo** with the necessary data, using the **Commerce** type can involve something like this:

```
Commerce commerce = new Commerce(new BillingProcessor(),
                                 new Customer(),
                                 new Notifier(),
                                 new Logger());
commerce.ProcessOrder(orderInfo);
```

The problem here is two-fold. First, the **Commerce** type is completely and utterly coupled to the other four types. Not only can it not work without these, but it is totally locked into the implementation each provides. What if the system we're writing changes payment gateways in the future? This is very plausible; in fact it's happened to me. I even changed back later when the rate structures changed again. If my application was written like the example above, I would have to rewrite the entire **BillingProcessor** class, only to rewrite it again later. Of course I can have two different classes, but that will then mean changing the code in the **Commerce** class in order to alter which other type gets received.

The second problem is that the **Commerce** class is difficult to test. Eventually I will have to test the production functionality of the four classes that house my sub-processes, but what if I want to test the **ProcessOrder** method of the **Commerce** class to see how it handles the combination of the four processes without having to deal with the resource access that those processes might undertake. Writing a unit test for this method means having to bring in all four of the other types and whatever resource access was written into them.

This is "coupled" behavior, and does not account for good software practices. Now I'm going to make adjustments to the code above in order to decouple my components and make everything more extensible and testable.

The first step to decoupling my application components is to abstract out the implementation from the definition in the three process classes. This way I can provide a production implementation and test implementation later. So the first thing I'll do is refactor the definition of the classes out to three interfaces.

```
public interface IBillingProcessor
{
    void ProcessPayment(string customer, string creditCard, double price);
}
public interface ICustomer
{
    void UpdateCustomerOrder(string customer, string product);
}
public interface INotifier
{
    void SendReceipt(OrderInfo orderInfo);
}
```

Continues on next page

Then I'll modify the four process classes to implement their appropriate interface. In the interest of space, I'll include only the **BillingProcessor** class here.

I can now write as many different billing processors as I want, each providing a different implementation of the interface. The key now is to modify the **Commerce** class so that it receives injections by way of the interfaces and not the concrete types as before.

```
public class BillingProcessor : IBillingProcessor
{
    void IBillingProcessor.ProcessPayment(string customer, string creditCard,
                                          double price)
    {
    }
}
```

Notice that the constructor arguments and the class members are now interface types. The method calls in the **ProcessOrder** method are exactly the same. The **Commerce** class just cares that it can make these method calls and leaves what they actually do to whatever the implementation classes that were sent through the constructor. The **Commerce** type's dependencies were injected through the constructor without being coupled to the **Commerce** type. The only coupling that took place is through the interfaces and they act as plumbing so this is good coupling.

```
public class Commerce
{
    public Commerce(IBillingProcessor billingProcessor, ICustomer customer,
                    INotifier notifie)
    {
        _BillingProcessor = billingProcessor;
        _Customer = customer;
        _Notifier = notifier;
    }
    IBillingProcessor _BillingProcessor;
    ICustomer _Customer;
    INotifier _Notifier;
    public void ProcessOrder(OrderInfo orderInfo)
    {
        _BillingProcessor.ProcessPayment(orderInfo.CustomerName,
                                         orderInfo.CreditCard, orderInfo.Price);
        _Customer.UpdateCustomerOrder(orderInfo.CustomerName, orderInfo.Product);
        _Notifier.SendReceipt(orderInfo);
    }
}
```

This is Dependency Injection in its simplest form. Since dependencies were "injected" using the constructor, this is referred to as constructor injection. Dependencies can be set through properties, using property injection, or through methods. Since my components have been decoupled from one another through the use of interfaces, I have allowed my application to grow and change in the future; since components can be swapped if and when necessary. In fact, in the context of Visual Studio projects, the **Commerce** class can sit in one assembly (possibly the main application), the process classes in a another (or even four different ones), and both sides share a third assembly containing the interfaces.

When I proceed to write a unit test against the **Commerce** class' **ProcessOrder** method, I would no longer need the production instances of the four process classes, and certainly not even a reference to assembly in which they live. I can write test versions that implement the interfaces in a totally different way, perhaps doing nothing at all except providing a dummy return value when one is expected. In fact, ideally I wouldn't need to write test implementations at all but can use a mocking framework instead to create them on-the-fly within the unit test itself.

The one thing that has not changed with this refactoring is the fact that whatever called the **Commerce** class still needs to instantiate the three process classes in order to inject them into the instance of **Commerce**. This is where a Dependency Injection container will come in real handy.

DEPENDENCY INJECTION CONTAINERS

If I break it down in the simplest and rawest way I could, I will tell you that a Dependency Injection containers are about two things, R&R. No, not rest & relaxation; we're software developers; we'll get none of that. I'm talking about registration and resolving. This is the acts of storing a list of types and later retrieving instances of them at will. The DI container is the tool that turns DI into architectural patterns that lets us easily and automatically, satisfy a type's dependencies. It is a repository that typically associates interfaces with concrete types.

HOW A DI CONTAINER WORKS

Dependency Injection containers all work in a very similar fashion. At the beginning of an application's execution-cycle, you need to register associations of concrete types to interfaces which they implement. The way registration functionality is exposed is one of the things that make containers different from one another. The idea of "resolving" is that later when an instance of an interface implementation is requested, the container can offer an instance of the appropriate concrete type. What makes this process special is that it happens recursively, as many times as necessary. Let me explain what I mean.

At the start of an app, you register 20 types with 20 interfaces. When you request a type from the container by specifying an interface, not only do you get an instance of the associated type but the container offers a bit more. It looks at either the constructor arguments or the public properties and determines whether they are interface types. If they are, it attempts to resolve them as well and set the argument or property value to the instance it resolved. The container then repeats the process for each of those resolved types as well. By the time it returns to you the type you requested, it has with it all its dependencies, and their dependencies, and so on down the line.

There are many options for Dependency Injection containers for .NET developers, some even from Microsoft. The Unity Container is Microsoft's answer to the call for Dependency Injection tooling and will be the one on which my example will be based. Microsoft also offers the Managed Extensibility Framework, or MEF. While many will argue that MEF is not a DI container, it offers many of the functionalities expected of one and is actually one of my preferred technologies for implementing Dependency Injection.

In order to integrate the **Unity** container into my previous example, I simply need to register all of my interface-to-class associations. In a real application, this is a task that is typically executed when an application starts up and the container is stored statically so it can be accessed throughout the entire application.

Here's the modified code for using Microsoft's **Unity** container in order to resolve the dependencies for the **Commerce** class:

```
UnityContainer container = new UnityContainer();
container.RegisterType<IBillingProcessor, BillingProcessor>();
container.RegisterType<ICustomer, Customer>();
container.RegisterType<INotifier, Notifier>();
container.RegisterType<ILogger, Logger>();
OrderInfo orderInfo = new OrderInfo()
{
    CustomerName = "Miguel Castro",
    Email = "miguel@dotnetdude.com",
    Product = "Laptop",
    Price = 1200,
    CreditCard = "1234567890"
};
Commerce commerce = container.Resolve<Commerce>();
commerce.ProcessOrder(orderInfo);
```

Instantiating the **Commerce** class by "resolving" it through the Dependency Injection container will force the container to go through its constructor arguments, checking for interface-types and attempting to resolve each of them as well. Since I've registered those types earlier, instances of the associated concrete types will be instantiated and passed into the **Commerce** class before it is returned to me for further usage.

Other popular containers out there include **Castle Windsor**, **NInject**, **StructureMap**, & **Spring.NET**. Each offers slightly different features than others, but in their core they all offer a way to register types and to resolve them and their dependencies.

CONCLUSION

The concepts of Dependency Injection and the implementation using a container can be employed in any application at any level. WPF, Silverlight, and Metro applications typically use DI to satisfy dependencies for ViewModels used in bindings. ASP.NET MVC applications use DI to serve controllers and thus automatically satisfy controller dependencies. Even ASP.NET Web Forms can benefit from DI by sub-classing the Page Handler Factory that serves up Web Forms and satisfy its dependencies, though this must be done through property injection and not constructor injection.

Dependency injection solves several problem areas in development. It allows us to decouple code components from one another and it lets us not worry about instantiating classes to send into other classes. Solving these two problems also sets up our software for easy testability later. There are many elements of dependency injection that I could not cover in this article. Among them is the ability to resolve multiple implementations of an interface as well as defining which of several registered implementations is to get resolved, possible through configuration files. Most containers provide both of these features, each in its own variety.

The examples in this article and the accompanying code, downloadable at http://www.dotnetdude.com/downloads, should give you a great head-start on using dependency injection. The downloadable code provides full examples of several DI containers including MEF. It also includes a make-shift container written from the ground up so you can see how the internals work. And lastly, I provide you will complete implementation examples for WPF, MVC, and WebForms using both Unity and MEF. The three scenarios I provide you should also serve as a good kick-off point for just about any application you are writing. Whether you're using a Microsoft container or a third-party one, using DI in your applications will ensure you're writing decoupled, manageable, and testable components, and also very importantly, it's really, really cool.

Not Your Father's C# v.Next++

A brief insight into one of the most powerful languages in the .NET platform, how you can use it today, and why it might just make Rosyln obsolete, even before its release.

By Philip Laureano

THERE'S SOMETHING NOT QUITE RIGHT HERE.

It looks like C#, and it even can compile code written in C# 4.0.
It has Design by Contract.
Non-nullable types.
It also has macros that allow you to add your own keywords to the language,
manipulate its syntax tree and let you do your own metaprogramming at will.
No, these are not the "Roslyns" that you're looking for. It's a .NET language called Nemerle, and the best part is that it's available for production right now, and you don't even have to wait for Microsoft to finish C# vNext in order to use it.

I CAN'T BELIEVE IT'S NOT C#

For the most part, Nemerle's curly-braced syntax is nearly identical to the C# language, and this traditional "Hello World" program in Nemerle illustrates just how similar the two languages can be:

```
public class Program
{
  public static Main() : void
  {
            Console.WriteLine("Hello, World!");
  }
}
```

In this case, the only difference seems to be the syntax for declaring the void return type; other than the return type, it's practically identical to its C# counterpart:

```
using System;
public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Of course, the example above is quite trivial, and in the real world, the applications that you will create and maintain will no doubt be infinitely more complex than just greeting the rest of the world. In fact, if you're dealing with at least a medium-sized application, there's a good chance that it will be throwing a few exceptions of its own, as well as catching exceptions and errors that are caused from calls to third-party libraries. In such cases, having unit tests can ensure that you can verify that the application behaves correctly from an outside perspective, but such black-box testing can't always help you diagnose the problem if you need to "crack open" the black-box when the application fails. There has to be some sort of sanity check to ensure that your application is always in a valid state. There also has to be sanity checks that ensure that the third-party libraries that your application calls behave as they are supposed to behave so that it will make it easier to figure out what went wrong if the application fails.

In C#, such diagnostic tasks have typically been relegated to simple assertions made from classes in the System.Diagnostics namespace. For example, the following assertion code prevents the application from throwing a DivideByZeroException when the caller tries to divide a number by zero:

```
using System;
public class Math
{
    public int Divide(int a, int b)
    {
        if (b == 0)
            throw ArgumentException("b cannot be zero");

        return a / b;
    }
}
```

DESIGN BY CONTRACT

In contrast, Nemerle has language extensions that allow you to declare the same assertions from the above example without polluting your method bodies with numerous assertions. Here is the same code example written in Nemerle:

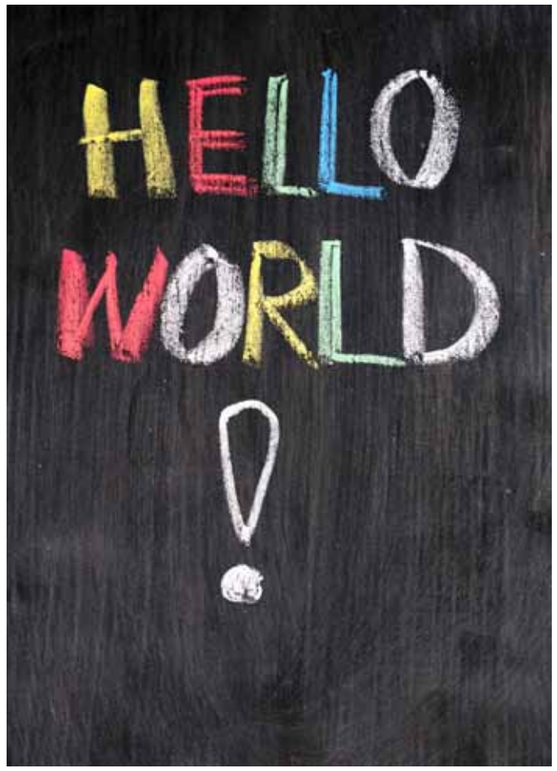
```
using Nemerle;
using Nemerle.Assertions;
using System;
public class Math
{
    public Divide(a : int, b: int) : int
        requires b != 0 otherwise throw ArgumentException("b cannot be zero")
    {
        a / b;
    }
}
```

The example above is mostly self-explanatory, and as you can see, Nemerle makes it trivial to add preconditions to any given method. It even allows you to add postconditions to your methods so that you can ensure that your code remains in a valid state after a given method executes, as this example shows:

NON-NULLABLE TYPES

The postcondition above is fairly straightforward, and you might have noticed that there's nothing preventing the caller from passing at least one null string as part of the method call. If you were to call the Concat method with one or more null strings, the program will inevitably terminate with a NullReferenceException. At first glance, it's tempting to put a preconditions that verify that both parameters have non-null strings, but Nemerle has a macro called [NotNull] that ensures that those two strings can never be null so that you won't have to repeatedly write the null reference preconditions yourself:

```
using Nemerle;
using Nemerle.Assertions;
using System;
public class Concatenator
{
    public Concat(a : string, b: string) : string
        ensures value != null // Make sure that it never returns null
    {
        // Equivalent to
        // return a + b; in C#
        a + b;
    }
}
```




```
using Nemerle;
using Nemerle.Assertions;
using System;
public class Concatenator
{
    public Concat([NotNull] a : string, [NotNull] b: string) : string
        ensures value != null // Make sure that it never returns null
    {
        // Equivalent to
        // return a + b; in C#
        a + b;
    }
}
```

Conceptually, this is no different than writing your code as:

```
using Nemerle;
using Nemerle.Assertions;
using System;
public class Concatenator
{
    public Concat(a : string, b: string) : string
        requires a != null
        requires b != null
        ensures value != null // Make sure that it never returns null
    {
        // Equivalent to
        // return a + b; in C#
        a + b;
    }
}
```

Continues on next page

Now, earlier in this article, I mentioned that **most of the above code** is self-explanatory, but what if I told you that all the language features that I mentioned above are not actually a part of the Nemerle language?

The truth is that Nemerle is so powerful that all of the features that I have mentioned so far are actually macros, and the Nemerle compiler itself is completely unaware of what non-nullable types are, much less aware of how to implement Design by Contract on its own.

EVERYTHING IS A MACRO

In essence, *there is no proverbial spoon* here in the compiler implementation; it's macros being used all the way up and down the Nemerle language, and its macro usage is so pervasive that even common language constructs such 'if' statements and for (and even foreach) loops are also macros. What this means for the average user is that you can still use Nemerle in the same way that you would use C#, but the Nemerle compiler is so flexible that you can extend the language to meet your business requirements in significantly less time than it takes for Microsoft's C# language team to develop a working CTP that you might not even be able to use in production code. In short, Nemerle provides programming language enhancements at the speed of 'right now'.

THE OPEN-CLOSED (COMPILER) PRINCIPLE

Unlike Roslyn (aka C# 5)--which makes you 'fall into the pit of success' by restricting you from doing anything harmful with the compiler—Nemerle assumes that if you can write your own language macros, then you are a *qualified base jumper* into the pit of success itself. (Needless to say, it's up to you to pack your own parachute, but I'm sure you are already aware of that fact).

That being said, Nemerle allows its users to extend the compiler implementation through macros without touching the Nemerle compiler's original source code. For example, take a look at the following simple Nemerle macro definition:

```
macro m ()
{
    Console.WriteLine ("compile-time");
    <[ Console.WriteLine ("run-time") ]>;
}
```

METAPROGRAMMING OUT OF THE BOX

Now, the first thing you might have noticed is the strange <[and]> characters, which is analogous to the <% and %> characters in ASP.NET. The code outside the <[and]> characters executes when the compiler builds your code, and everything inside those brackets will be translated into IL by the Nemerle compiler and that code will run wherever the m() macro is used in your code.

The most interesting part about that macro snippet is that the call to Console.WriteLine("run-time") within the brackets is actually the Nemerle compiler allowing you to construct an entire syntax tree using just Nemerle code itself.

Yes, repeat after me: Nemerle lets you build syntax trees, using Nemerle's syntax itself.

Unlike Roslyn, Nemerle doesn't force you to manually construct the syntax tree yourself using the its syntax tree object model. In most cases, all you need to do to create a syntax tree is to embed a Nemerle code fragment inside an <[and]> block. For example, the following macro adds the 'repeat' keyword to the Nemerle language a certain number of *times* and injects the *body* into the compiled code:

To use the new syntax, all you need to do, in turn, is to call it from your code:

```
macro repeatmacro (times, body)
syntax ("repeat", "(", times, ")", body)
{
    <[ for (mutable t = $times; t > 0; --t) $body ]>
}
```

To use the new syntax, all you need to do, in turn, is to call it from your code:

```
repeat(6)
{
    Console.WriteLine("Hello, World!");
}
```

Predictably, the output will be:

"Hello, World!"
"Hello, World!"
"Hello, World!"
"Hello, World!"
"Hello, World!"
"Hello, World!"

To recap: we just added a language syntax extension in less than six lines of code. Nemerle allows you to implement in six lines what would take a six month release cycle for a closed language team to implement and release. Needless to say, it's a very powerful tool to have, and it will make it easier for you to work with a language that is closer to the business language of your customers.

"You"-driven compiler development, done your way

The possible uses of Nemerle's macros are only limited by your own imagination. Having the power of the Nemerle compiler in your hands means that you and you alone are responsible for the extensions that you create. "With great power," as the saying goes, "comes great responsibility".

I hope you brought a big-enough parachute.

COURSE OVERVIEW OSLO

ProgramUtvikling

COURSE TITLE								
AGILE	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Advanced Project Management - Tom & Kai Gilb	3		11				IT Fornebu	17 900
Agile Estimating and Planning - Mike Cohn	1				21		London	7 000
Coaching in softwarem development teams	1	2					IT Fornebu	3 900
Effective User Stories for Agile Requirements - Mike Cohn	1				18		London	7 000
*From User Stories to Acceptance Tests - Gojko Adzic	3	14			26		IT Fornebu	17 900
*Intercultural strategy for distributed and virtual teams - Kari Amelie Fiva Aasheim	2	10					IT Fornebu	6 900
*Kanban Applied - Mattias Skarin	2					24	IT Fornebu	13 900
*Lean QA: Smart Kvalitetssikring - Tom & Kai Gilb	2		14				IT Fornebu	13 900
*Mastering Agile Practice - Kevlin Henney & Jon Jagger	3					3	IT Fornebu	17 900
SCRUM	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Certified Scrum Product Owner - CSPO - Geoff Watts	2		28			3	IT Fornebu	13 900
Certified Scrum Product Owner - CSPO - Mike Cohn	2				5		Felix, Oslo	13 900
*Certified ScrumMaster - CSM - Geoff Watts	2					1	IT Fornebu	13 900
Certified ScrumMaster - CSM - Mike Cohn	2		4		3		Felix, Oslo	13 900
*Effective Agile Development with .NET (including CSD certification) - Gáspár Nagy	3	7					IT Fornebu	17 900
*Improve your scrum team - Advanced Scrum Master course - Geoff Watts	2		26				IT Fornebu	13 900
*Smidig utvikling med Scrum med Arne Laugstøl	1		24					5 900
TESTDRIVEN DEVELOPMENT - TESTING	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Clean Code: Agile Software Craftsmanship - Robert C. Martin	2					15	IT Fornebu	13 900
*From User Stories to Acceptance Tests - Gojko Adzic	3	14			26		IT Fornebu	17 900
*Legacy Code -Test Writing Techniques - Michael Feathers	2							13 900
*Rapid Software Testing - Michael Bolton	3				24		IT Fornebu	17 900
*Test-Driven Development & Refactoring Techniques - Robert C. Martin	3	9				17	IT Fornebu	17 900
*Test-Driven Development in .NET Master Class (TDD in .NET Course) - Roy Osherove	5						IT Fornebu	22 900
*Test-Driven JavaScript med Christian Johansen	3			29		24	IT Fornebu	17 900
*Whole Team Approach to Agile Testing - Janet Gregory	3				11		IT Fornebu	17 900
MANAGEMENT	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Agile Management - Jurgen Appelo	2		26				IT Fornebu	13 900
*Lead Better - Essential Skills for Software Team Leadership - Roy Osherove	2						IT Fornebu	13 900
DESIGN - ANALYSIS - ARCHITECTURES	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Advanced Distributed Systems Design using SOA & DDD - Udi Dahan	5		11				IT Fornebu	22 900
*Agile Architecture and Design - Robert C. Martin	4			27			IT Fornebu	22 900
*Agile Design and Modeling for Advanced Object Design with Patterns - Craig Larman	4	7				15	IT Fornebu	20 900
*Analyse og design med Scrum	4	8					IT Fornebu	19 900
*Architecture Skills - Kevlin Henney	3				26		IT Fornebu	17 900
*Arkitekturer med C#.NET	4		12				IT Fornebu	19 900
*Domain-Driven Design (DDD) - Course design Eric Evans - Instructor Kristian Nordal	4				25		IT Fornebu	20 900
*The Architect´s Master Class - Juval Lowy	5							23 900
MOBILE APPLIKASJONER	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Advanced Android Application Development - Gonçalo Silva/Luigi Agosti	3		20				IT Fornebu	17 900
*Android Application Development (Intro + Advanced) - Gonçalo Silva/Luigi Agosti	5		18				IT Fornebu	22 900
*Aral Balkan's iOS SDK Dojo	3	30					IT Fornebu	17 900
*Architecting Mobile Solutions - Dino Esposito	5				10	29	IT Fornebu	22 900
*Designing the mobile user experience - Aral Balkan	2	7					IT Fornebu	13 900
*Intro to Android Application Development - Gonçalo Silva/Luigi Agosti							IT Fornebu	13 900

*Courses in the list that are asterisked are included in the framework agreement programme.

MICROSOFT	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
* Web Development - Scott Allen	5			20		22	IT Fornebu	22 900
*70-513 - WCF 4 with Visual Studio 2010 - Sahil Malik	5	21		20		8	IT Fornebu	22 900
*70-513 - WCF 4 with Visual Studio 2010 - Sahil Malik - Kongsberg	3			13			IT Fornebu	17 900
*70-516 Real World LINQ - Data Access, Entity Framework and Beyond - Scott Allen	4	7				15	IT Fornebu	19 900
*70-583 - Microsoft Azure - Sahil Malik	3				6			17 900
*ASP.NET MVC - Arjan Einbu	4		18				IT Fornebu	19 900
*C#.NET: Utvikling av applikasjoner i .NET med C# - Arjan Einbu	5	7	25	27		8	IT Fornebu	22 900
*Design Concepts, Architecture, and Advanced Capabilities - Billy Hollis	4				3			19 900
*Federation & Claims-based Identity with .NET and the Windows Identity Foundation - Dominick Baier	2	10			27		IT Fornebu	13 900
*MS 70-515 Web-utvikling med ASP.NET - Arjan Einbu	5	21		20			IT Fornebu	22 900
*Silverlight 5 Workshop - Gill Cleeren	4	7					IT Fornebu	19 900
Windows 8 fagdag	1	3					MS Norge	0
*WPF - 70-511 / 10262A Windows Presentation Foundation (WPF) - Arne Laugstøl	4		18		10	29	IT Fornebu	17 900
SHAREPOINT	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*SharePoint 2010 and Office 365: End to End for Developers and Designers - Sahil Malik	5		25		10		IT Fornebu	22 900
JAVA	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Core Spring - Kaare Nilsen/Mårten Haglind	4	21		20		8	IT Fornebu	20 900
*Effective JPA - Industrial strength Java Persistence with JPA 2.0 and Hibernate	2	10			27		IT Fornebu	13 900
*Effective Scala med Jon–Anders Teigen	3		13		10		IT Fornebu	17 900
*Maven 2 - Trygve Laugstøl	2				6			13 900
*Programming Java Standard Edition	5	21		13			IT Fornebu	22 900
*Spring and Hibernate Development - Kaare Nilsen/Mårten Haglind	5	21		20		8	IT Fornebu	22 900
HTML5 - JAVASCRIPT	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Introductory JavaScript for programmers - Christian Johansen	2			27		22	IT Fornebu	13 900
*Test-Driven JavaScript med Christian Johansen	3			29		24	IT Fornebu	17 900
*Using HTML5 and JavaScript to build Web Apps - Remy Sharp	3	30			9		IT Fornebu	17 900
C++	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Advanced C++ programming - Hubert Matthews	4	21			3		G.H.Kongsberg	19 900
*C++-501: C++ for Embedded Developers - Mike Tarlton	5				17		IT Fornebu	22 900
*Programming in C++ with Mike Tarlton	4		25		17		IT Fornebu	19 900
XML	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Exchanging and Managing Data using XML and XSLT - Espen Evje	3	14				17	IT Fornebu	17900
DATABASE	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Course 10776A: Developing Microsoft SQL Server 2012 Databases - Espen Evje	5		18	20			IT Fornebu	22 900
*Databasedesign, -implementering og SQL-programmering med Dag Hoftun Knutsen	4	29					IT Fornebu	19 900
*Oracle PL/SQL- programmering	3							17 900
*Oracle SQL- programmering med Dag Hoftun Knutsen	3			25		17		17 900
EFFECTIVE CONCURRENCY	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
Effective Concurrency - Herb Sutter	3				24		London	17 900
PROGRAMMING	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
*Introduction to Python Programming - Peet Denny	3				13			17 900
*Objektorientert utvikling	4		18					19 900
PRE-CONFERENCE WORKSHOPS AT NDC	**Days**	**May**	**Jun**	**Aug**	**Sept**	**Oct**	**Location**	**Price**
A day of WCF - Miguel Castro	1		5				Radiss. Plaza	4 900
Clojure: Up Close and Personal - Robert C. Martin	1		4				Radiss. Plaza	4 900
CSS3: Not just for Designers - Lea Verou	2		4				Radiss.Plaza	7 900
Develop Mobile Applications with C# and .Net - Chris Hardy	1		5				Radiss.Plaza	4 900
Hunt The Wumpus: Acceptance tests to code - Robert C. Martin	1		5				Radiss.Plaza	4 900
Node for a real-time web - Remy Sharp	1		5				Radiss.Plaza	4 900
Rails for the .NET Developer - Cory Foy	2		4				Radiss.Plaza	7 900

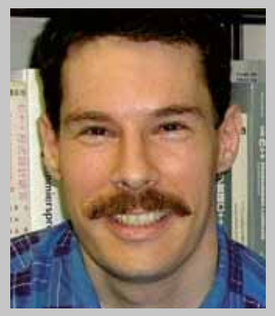
COURSE OVERVIEW LONDON

DeveloperFocus

DeveloperFocus LTD, London by ProgramUtvikling AS
www.developerfocus.com

COURSE TITLE								
AGILE & SCRUM	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Acceptance Test-Driven Development with David Evans	2					19	London	£1395
Advanced Scrum Master Course - Geoff Watts & Paul Goddard	2			11			Holborn	£995
Agile Estimating and Planning Training - Mike Cohn	1					21	London	£700
Architecture with Agility - Kevlin Henney	3						London	£1195
Certified Scrum Master - CSM - Geoff Watts	2			4		4	Holborn	£995
Certified Scrum Master - CSM - Mike Cohn	2					19	London	£1400
Certified Scrum Product Owner - CSPO - Geoff Watts	2					6	London	£995
Certified Scrum Product Owner - CSPO - Mike Cohn	2	2					London	£1400
Effective User Stories for Agile Requirements Training - Mike Cohn	1					18	London	£700
Game-Orientated Agile Learning (GOAL) with Geoff Watts & Paul Goddard	2						Holborn	£995
Retrospective Techniques - Geoff Watts - WAITLIST ONLY	2							£995
Succeeding with Agile - Mike Cohn	2						London	£1400
ARCHITECTURE - DESIGN	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
The Architect˙s Master Class - Juval Lowy	5							£2500
EFFECTIVE CONCURRENCY	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Effective Concurrency - Herb Sutter	3					24	London	£1995
MICROSOFT .NET	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Advanced C++ Programming - Hubert Matthews	4		19				London	£1750
ASP.NET MVC 3 - Arjan Einbu	4							£1750
Building Applications with ASP.NET MVC - Scott Allen	3					10	London	£1750
Building Applications with ASP.NET MVC/ HTML5 Course - Scott Allen	5					10	London	£2250
C #.NET: Developing applications in .NET with C # - Arjan Einbu	5		25			24	London	£1950
Design Concepts, Architecture, and Advanced Capabilities for WPF - Billy Hollis	4	28					London	£2500
WCF - Windows Communication Foundation (WCF) 4 and AppFabric with Visual Studio 2010	5	14					London	£1950
Working with HTML 5, CSS 3, and JavaScript - Scott Allen	2					13	London	£1250
WPF-70-511 Windows Presentation Foundation (WPF) - Arne Laugstøl	5						London	£1950
MOBILE APPLICATIONS	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Mono for Android Course with John Sonmez	2					13	London	£1250
MonoTouch and Mac Development Course - John Sonmez	3					10	London	£1750
MonoTouch and Mono for Android Course - John Sonmez	5					10	London	£2500
SHAREPOINT	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
SharePoint 2010: For the .NET Developer and Architect - Sahil Malik	5	28		2			London	£1750
SharePoint Geek N8 - Sahil Malik	1	11					London	£0
JAVA	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Core Spring 3.0 - Kaare Nilsen	4							£1750
Spring and Hibernate Development - Kaare Nilsen	5							£1950
JAVASCRIPT - HTML	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
Mobile Web Apps - Remy Sharp	1			5			London	£495
Node & HTML5 for a real-time web - Remy Sharp	2		28				London	£895
Test-Driven JavaScript - Christian Johansen	3							£1500
Using HTML5 and JavaScript to build Web App - Remy Sharp	1			6			London	£495
USER EXPERIENCE & DESIGN	**Days**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Location**	**Price**
CSS3: Not just for Designers with Lea Verou	2					6	London	£825
01_Web Usability Fundamentals with Rolf Molich - Workshop 1	1						London	£299
02_Usability Testing Practicum with Rolf Molich - Workshop 2	1						London	£299
03_Advanced User Testing with Rolf Molich - Workshop 3	1						London	£299
04_Measuring Usability with Rolf Molich - Workshop 4	1						London	£299
Design Reviews with Rolf Molich - WAITLIST ONLY	1						London	£299

EFFECTIVE CONCURRENCY with Herb Sutter



24.-26. September on Grand Connaught Rooms, London

This course covers the fundamental tools that software developers need to write effective concurrent software for both single–core and multi–core/many–core machines. To use concurrency effectively, we must identify and solve four key challenges:

- Leverage the ability to perform and manage work asynchronously
- Build applications that naturally run faster on new hardware having more and more cores
- Manage shared objects in memory effectively
- Engineer specifically for high performance

This seminar will equip attendees to reason correctly about concurrency requirements and tradeoffs, to migrate existing code bases to be concurrency–enabled, and to achieve key success factors for a concurrent programming project. Most code examples in the course can be directly translated to popular platforms and concurrency libraries, including Linux, Windows, Java, .NET, pthreads, and the forthcoming ISO C++0x standard.

THE SEMINAR INCLUDES THE FOLLOWING TOPICS:

- FFundamentals
- Isolation: Keep Work Separate
- Scalability: Re–enable the Free Lunch
- Consistency: Don't Corrupt Shared State
- Migrating Existing Code Bases to Use Concurrency
- Near–Future Tools and Features
- High Performance Concurrency

ABOUT THE INSTRUCTOR

Herb Sutter is the best–selling author of four books and hundreds of technical papers and articles, including the widely–cited essay "The Free Lunch Is Over" on the software sea change now in progress to exploit increasingly parallel hardware. He chairs the ISO C++ standards committee and is partly responsible for the design of several C++0x standard features, including async, futures, atomics, lambda functions, and the memory model. He is also a principal architect at Microsoft, where he was the lead architect of C++/CLI and other technologies, and was responsible for the design of lambda functions in Visual C++ 2010 and their integration in the Parallel Patterns Library.

Price 1,995 GBP

Contact us: 0843 523 5765

Register on www.developerfocus.com

Sign up now!

ProgramUtvikling

DeveloperFocus

Training for developers and leaders

in Oslo, London or wherever you like

ProgramUtvikling offers the best courses and most flexible training to the developer community wherever you want us to. Our philosophy is to provide "practical, applicable knowledge" by providing the highest quality training with the best instructors.

In addition to our permanent staff we have a number of exciting and well-known instructors such as Herb Sutter, Craig Larman, Billy Hollis, Mike Cohn, Geoff Watts, Gill Cleeren, Sahil Malik and a great staff of excellent Scandinavian instructors.

ProgramUtvikling
OSLO - www.programutvikling.no

DeveloperFocus
LONDON - www.developerfocus.com

2012 NDC NORWEGIAN DEVELOPERS CONFERENCE June 6-8 Oslo, Norway

NDC has become one of the largest conferences dedicated to .NET and Agile development and is hosted by ProgramUtvikling AS. Meet speakers like Brad Wilson, Dan North, Mike Cohn and Lea Verou. **www.ndcoslo.com**

HTML5 • Javascript • Mobile • Agile • Scrum • Design .NET • Java • Architecture • TDD • SharePoint • C++

2012 NDC
NORWEGIAN DEVELOPERS CONFERENCE
june 6-8 Oslo, Norway

Agenda and Practical Information

All you need to know about NDC 2012 >>>

- 3-day conference with 7 parallel tracks
- 2 days of Pre-conference Workshops

ProgramUtvikling Microsoft ndcoslo.com NDC 2012 NORWEGIAN DEVELOPERS CONFERENCE

NDC 2012 OSLO SPEKTRUM,
6 – 8 JUNE
PRE-CONFERENCE WORKSHOPS
4 – 5 JUN

The Conference reaches new heights

Norwegian Developers Conference is arranged in Oslo Spektrum for the fifth year in a row.

By Linn Therese Skulstad

Energetic, inspiring and instructive. The success is repeated and this year's Conference reaches new heights. Extended to last a whole week, it now consists of two course days, three conference days and five knowledge days – you compile the days as a professional tapas. The Committee for this year's Conference has selected the best lecturers that can be found. With good words of praise from all the committee members, this is a Conference you cannot miss.

"First and foremost, you must participate in NDC because of the professional content and not least the possibilities of building a network. I think that we have developed the Conference to be one of the world's most important and impressive conferences in which to participate, in that we have extended the conference to five days. We are concerned with improving ourselves, something which we must also do in an industry which is constantly under development", says Henriette Holmen.

HENRIETTE HOLMEN, PROJECT MANAGER FOR NDC

Henriette Holmen is the Project Manager for the Conference, and is responsible for booking lecturers, employment contracts, travel booking, partners, and being in contact with external suppliers. This is a job that Henriette is very satisfied with since it offers many different challenges and pleasures, as well as giving her the opportunity of building a network in an industry she thinks is very exciting. Ms Holmen has been a part of the NDC team for three years, and together with the rest of the colleagues, she is very satisfied with having extended the Conference to a whole week. The breadth and quality of the proposed lectures were so good this year that the Committee felt it necessary to expand to eight tracks.

"We have received an overwhelming number of abstracts this year, and it is clear that the visibility of the Conference increases each year. We have received abstracts from the whole of the world and this is something we on the Committee are very proud of", says Ms Holmen.

JONAS FOLLESØ, MANAGER AND SCIENTIST IN BEKK

Jonas Follesø spends part of his time as a scientist where he is busy in the Norwegian professional environment, either as a lecturer/article author or similar activities. In addition, he works as a consultant and developer, where mobile is one of the areas he is particularly interested in.

This is the second year that Mr Follesø is a member of the NDC programme committee, and thinks that it is an incredibly exciting, stressful and challenging task, in addition to having had an extra focus on mobile as a theme.

"NDC has grown every year and when we have such an extreme number of qualified lecturers and course holders in Oslo, it would be silly not to use the opportunity to really go into depth with some themes using the pre-conference workshops", says Mr Follesø.

"NDC has developed not only into being an important arena in Norway, but in time also an internationally known conference with the focus on smooth development and alternative thinking on the Microsoft and.NET platform. NDC has also developed into being a conference which is not only relevant for .NET developers, but rather everyone who works with development in one form or another", says Mr Follesø.

KJERSTI SANDBERG, GENERAL MANAGER OF PROGRAMUTVIKLING

Kjersti Sandberg is the founder of Programutvikling AS and has organized the NDC Conference from the beginning. Her daily job is with the professional areas of sale, marketing and business development. Her role on this year's Committee has been administrative, have communication with speakers and present good ideas. "The fact that NDC has expanded into a whole week allows you to form your own professional tapas. This must be the peak for those who will get the most out of five knowledge days in an exciting, enjoyable and social way" she adds.

TORBJØRN MARØ, SENIOR DEVELOPMENT AND TEAM LEADER IN PSWINCOM

Torbjørn Marø works with .NET development connected to mobile communication in PSWinCom. PSWinCom communicates SMS messages between a few thousand companies and their customers, which includes most Norwegians who have a mobile phone. Torbjørn leads a small team of skilful developers, enjoys his job very much, and has many demanding challenges to resolve.

" I really hope that people make use of the extra days. Here they will have an opportunity to go in-depth with some very exciting speakers and developers, at a low cost compared to the price of a normal course", says Mr Marø. His role on this year's Committee has been to represent developers who are not dyed-in-the-wool Microsoft fans, but who like to keep abreast of what is going on in many arenas. Mr Marø also contributed to obtain some lecturers that have not been seen in NDC previously.

"This also includes John McCoy, who is an expert in .NET hacking – how one breaks into a .NET framework, and bypasses security measures. Another thing I have been interested in is seeing that we have a good spread among what we can call introductory presentations on the one hand, and heavier, more demanding deepening on the other. I want NDC to be a conference where all developers can find lectures they will benefit from, and I am quite sure that we have managed it".

PETRI WILHELMSEN, DEVELOPER MARKETING MANAGER IN MICROSOFT

Petri Wilhelmsen's daily work is on the Evangelist team at Microsoft Norge. There he is responsible for reaching out to developers in Norway with Microsoft technology for developers, as well as working closely with universities and schools to provide them with knowledge about Microsoft technology. In addition, he is a very active coder and has a passion for graphics programming, algorithms and game development.
"My role has been to help NDC find lecturers, themes and sessions for the agenda. This is a big job where we on the Committee must sit down together, evaluate lecturers and which sessions we wish to select from Call for papers, distribute them during the day and make tracks. It has been incredibly rewarding working with such a competent gang as this year's Committee", says Wilhelmsen. He, also, illustrates how tough it has been to choose between the various abstracts.

"A lot of good abstracts have been received this year, and it has been incredibly difficult choosing among the best. We would like to have had more days and more tracks to get in everything we wanted. Unfortunately, this is not possible and we have had long evenings discussing who we wished to include", he says.

SVEIN ARNE ACKENHAUSEN, INDEPENDENT CONSULTANT AND LECTURER

Svein Arne Ackenhausen works in Contango Consulting AS, a small consultancy company that delivers software services and training. Here he works mainly with .NET development assignments and courses. In addition to his daily consultancy work he works with the products ContinuouTests (Mighty Moose), AutoTest.Net and OpenIDE.

"NDC has grown into being a recognised conference and I have to say that the standards are set higher than in previous years. It is incredibly enjoyable to be on the Committee. It is not every day that one can select the best technical lecturers in the world. I felt a little like a three-year-old in a candy store", says Ackenhausen.

"One should register for the Conference because it is unique opportunity to learn a lot of the most exciting things that are taking place in development today from several of the best in the industry. From experience, and as mentioned by several of the speakers, it is a conference with a very good atmosphere. There is a lot to learn both in the sessions and outside through exciting discussions during breaks and in the evenings".

JAKOB BRADFORD, GENERAL MANAGER OF PROGRAMUTVIKLING AND NDC

As General Manager for NDC, Jakob Bradford has had the main responsibility for seeing that they reach their goal with the agenda and completion of the Conference.

"Both the number and quality of the abstracts have been very high this year. We have received more abstracts from more speakers than ever. The high quality meant that we increased with an extra track, so that this year there will be eight parallel tracks. The number of speakers and abstracts shows the position NDC has as we enter our fifth year. Now I am very much looking forward to being present at the pre-Conference days.

Fifth time around

For the fifth consecutive year, NDC is once again just around the corner: Get ready for three days of tasty food, great entertainment and enlightening sessions.

By Inger Renate Moldskred

Photo: Vebjørn R. S. Olsen

The band Loveshack, which is well known to many previous NDC participants will play for us. They've joined in on all of our former conferences, and is a must see at this years NDC as well. Loveshack will bring you many 80s classics, and set the mood for a grand summer party.

This year, we introduce a new way of building up the expo area and a new way to serve you food. Rather than meals at certain times, food will be served all day and in a more open arena. The new plan give more free space, with greater chance to mingle and meet new people. Expo will be the place to be, especially during session breaks. And the best part: You get to eat whatever, whenever you feel like eating.

As a tribute to the five year anniversary we will have different kinds of restaurants, each with their unique cuisine; serving Italian, Indian, Asian or American food. Placed strategically in the expo area; whichever way you turn, you'll find food up for grabs. Sit down in the restaurant areas, or take the food with you to the overflow or relax areas.

In good tradition, the .net Rocks guys Richard Campbell and Carl Franklin will join us this year as well. They'll keep you posted through these three days filled with tasty food, smart mingling and enlightening sessions. In addition to being a radio station, the .net Rocks stand will also serve as a restaurant; a barbecue corner.

The food is, as in the previous years, delivered by the Flying Culinary Circus guys. The chef quartet consists of Trond Svendgård, Fish & Seafood Expert; Hans Kristiand Larsen, Meat Expert; Mathias Spieler Bugge, Sauce & Soup Expert and Tor Jørgen Kramprud Arnesen, Herb & Vegetable Expert. In addition to having names like Elton John, Sara Ferguson, Perez Hilton and Ke$ha in their repertoire, serving food from these four talented chefs has been a great success on previous conferences. We look forward to serve you culinary food once again.

Mathias Spieler Bugge,
Sauce & Soup Expert

Trond Svendgård,
Fish & Seafood Expert

Hans Kristiand Larsen,
Meat Expert

Tor Jørgen K. Arnesen,
Herb & Vegetable Expert

Read more about the program and sessions of NDC 2012 at **ndcoslo.com**

.net Rocks

This years, the .net Rocks stand with Richard Campbell and Carl Franklin will serve as both radio station and restaurant; stop by their barbecue corner.

New for this year's conference is the band **Donkeyboy**, who will perform for us live from the stage in Oslo Spektrum; with the vast possibilities this arena gives for great sound and light shows.

Donkeyboy released their debut single *Ambitions* in March 2009 and their debut album *Caught In A Life* in October 2009, which sold 115.000 copies. This spring, **Donkeyboy** released their much anticipated second album, *Silver Moon*. As on *Caught In A Life*.

The success of the five friends, fronted by brothers Cato and Kent Sundberg, is the result of a hard working and talented band. The band has played more than 100 live shows. And this NDC they will play live for you in Oslo Spektrum.

Join us at NDC 2012 for a great food and entertainment experience, in addition to great sessions from leading experts in their genres.
And remember the two pre-conference days as well!

Oslo

The capital of Norway

Oslo is called “The blue, the green and the city in between”, and is a great destination for outdoor activities. Even in the city center, the nearest park is never more than a few blocks away. A ten-minute boat ride from the center takes you to lovely beaches on the close islands. In winter you find hundreds of kilometers of cross-country trails within the city boundaries, in addition to eight ski centers.

View from the Norwegian Opera house © Shutterstock

The city center © Shutterstock

The city offers an abundance of attractions, shopping possibilities and a flourishing cultural life. The choice of restaurants is almost unparalleled in Scandinavia. These are some of the choices you have when spending time in Oslo:

The Munch Museum
Edvard Munch’s art is the most significant Norwegian contribution to the history of art. The museum’s program comprises film screenings, audio tours, concerts, guided tours and lectures. The museum has a shop, a café and a library.

The Norwegian Opera & Ballet
After the opening in 2008, Oslo Opera House soon became a landmark and a tourist attraction. It is the first opera house in the world to let visitors walk on the roof. With three stages and state-of-the-art equipment, the opera offer a variety of performances and productions.

The Vigeland Park
The Park is a must-see in Oslo. The famous park is filled with 192 bronze and granite sculptures, far stretching lawns and long straight avenues bordered with maple trees.

Holmenkollen National Ski Arena
The Arena is the most visited tourist attraction in Norway. It includes the Holmenkollen Ski Museum & Jump Tower, shops, a café and a ski simulator.

Nordmarka wilderness area
The forest region in northern Oslo, Nordmarka, covers 430 km^2 and is a place for activities and recreation all year round.

The Vigeland sculpture park © Shutterstock


Aker Brygge, shopping and restaurant area © Shutterstock

Aker Brygge
Aker Brygge has about 70 shops, 40 restaurants and bars, a cinema and a guest harbour. This unique sea-front boardwalk is one of Oslo's primary attractions, and more than 7,000 people live and work in the area. This is where people meet and mingle.

Lofoten Fish restaurant
At Lofoten Fish Restaurant it is quality, service and expertise which matter. The freshest, top quality ingredients make them able to present a maritime menu of a high international standard.

Delicatessen Tapas Bar
This is one of Oslos most famous Tapas Restaurant and is situated both at Grünerløkka and Majorstuen. Here people can eat quality food at reasonable prices in a casual setting.

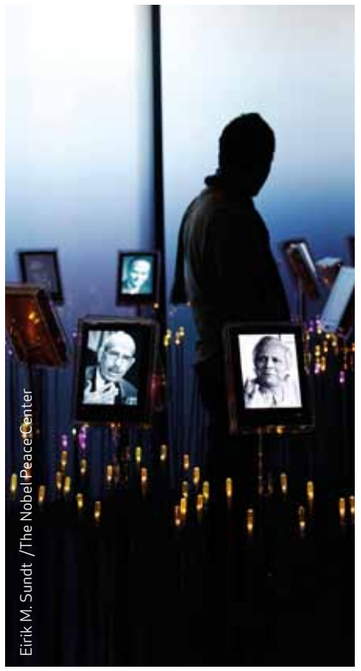
Maaemo
Maaemo has two Michelin stars and all the food is made from only organic and natural ingredients. Here you will get to taste the very best meats, seafood, poultry, vegetables, berries and herbs from Norwegian producers. Combined with an innovative kitchen and employees that are allowed to make their own mark on the food, drinks and experience, Maaemo is all that you expect of a restaurant at the top international level.

How to get around in Oslo
Oslo offers convenient public transportation, and short distances make it easy to get around by walking or biking. We recommend you to buy an all day Oslo Pass, which is the easiest and most inexpensive way to experience Oslo! The Oslo Pass provides free travel on all public transport, free admission to museums and sights, free parking in all Oslo municipal car parks, discounts on car hire, Tusenfryd Amusement Park etc. The Oslo Pass is your ticket to the city.


The Harbour and the City Hall © Shutterstock


The Royal Castle © Shutterstock

Sara Johannessen/Nobel Peace Center


Eirik M. Sundt /The Nobel Peace Center

Visit the Nobel Peace Center

Meet the Nobel Peace Prize laureates at Nobel Peace Center and learn more about them. Technology, digital and interactive installations allow you to experience their stories in new ways. Check out the magic book about Alfred Nobel and the electronic wallpapers, or find peace in the Nobel Field.

CURRENT EXHIBITIONS:
In Afghanistan: This exhibition shows the life of tough, but vulnerable American soldiers seen through the lens of Tim Hetherington/Panos Pictures, and a current portrait of Afghan women through the lens of Lynsay Addario/VII.

SHEROES: Portrays the three Peace Prize laureates 2011, Ellen Johnsen Sirleaf, Leymah Gbowee and Tawakkol Karman.

The exhibitions are displayed through documentary photography and films.

OPENING HOURS DURING THE CONFERENCE:
Every Day 10 am–6 pm
Entrance Fee: NOK 80,-,
Oslo Pass: Free
Location:
Rådhusplassen
(the City Hall Square)
How to get there:
Tram #12 from Jernbanetorget to Aker Brygge or any bus, underground or tram to the National Theatre and walk five towards the harbour.

Nobel Peace Center
Nobels Fredssenter

Oslo- in numbers

- Total area: 454 square kilometres
- Population (2011): 600,000
- Forest area: 242 km^2
- Park and sports arena area: 8 square kilometres
- Lakes within the city limits: 343
- Islands within the city in the Oslofjord: 40
- Length of the Oslofjord: 100 kilometres

Pre conference Workshops on the top of Radisson Blu Plaza Panorama

June 4th

Using HTML5 and JavaScript to build Web Apps : Remy Sharp

This one-day workshop will introduce you to HTML5 with a brief backstory, before diving into the APIs one by one. As well as going through code and showing practical demonstrations, where possible, we'll also talk about the alternatives and polyfills for old browsers that don't support "awesome" out of the box.

You're not expected to have played with HTML5 just yet, but you will need to have a reasonable understanding of HTML & JavaScript.

June 4th

Clojure: Up Close and Personal : Uncle Bob

In this all-day session you will learn Clojure by following along with Uncle Bob, step bgy step, and test-case by test-case, as you build a simple Swing/Clojure orbital simulator app. Along the way you'll learn the value of Functional Programming, hiw (and why!) to design applications that use it, and why Clojure is the next important language that you should master. This session will be extremely challenging!

June 4-5th

CSS3: Not just for designers : Lea Verou

Starting with the basics, we'll dive deep uncovering the nitty-gritty details of the CSS3 features that help you achieve those goals. You will get the chance to test everything yourself in the process with small, easy to understand examples designed to keep the learning process fun and enjoyable.
This is a two-day course.

June 4-5th

REST workshop : Erlend Hamnaberg & Trygve Laugstøl

The workshop will be an introduction to REST. We will take a practical approach and implement an actual application. We will walk you through how HTTP works in a RESTful context. The participants will learn how to use it effectively. REST is the first architectural style that elevates data to an architectural element. We will supply the participants with a toolbox for identifying, and use hypertext elements in existing and new data formats. We will also show areas where REST as an architecture and HTTP as a protocol works poorly.

June 4-5th

Rails for the .NET Developer : Cory Foy

In this two-day workshop, you'll learn the basics of Ruby on Rails and build and deploy your own website. Along the way you'll learn Test-First programming with RSpec and Cucumber, committing your app to Git (and to GitHub) and deploying your app using Heroku.

June 4-5th

Choose your own adventure : Greg Young

We will be coding 80% of the time. Of course you will not be coding my random next little start up (unless of course we all decide that two days is enough!). We will instead focus on writing code that has lessons built into it. Katas, Check. Build your own testing framework and use it. Check. Things you can't do at work but may learn something from. Check. If you want lectures there are many other workshops, if you want relevant code and less lectures; bring a laptop the more environments the marier we love polyglotism.

June 4-5th

The RavenDB Workshop Part 1 & 2 : Itamar Syn-Hershko

In this fast-paced and hands-on 2-day RavenDB workshop, you will learn how to use this quickly evolving Document Database efficiently in your applications to save time and effort on communicating with database storage.

June 5th

A day of WCF : Miguel Castro

In this one-day workshop I'll take you through the design and development of WCF services from the ground up using a best-practice approach from the beginning. We'll cover service contracts, data contracts, services, proxies, hosting, and consuming. Then we'll dive into many of the additional features provided by WCF including but not limited to transactions, fault handling, and instancing, and even security.
And time-permitting we may get into REST services.

June 5th

Hunt the Wumpus: Acceptance tests to code : Uncle Bob

In this one day-course we will begin with acceptance tests in FitNesse that describe the behaviour of the old computer game "Hunt The Wumpus". And then, step by step, test case by test case we'll use TDD, Refactoring, and Clean Code practices to get this application working. Students will learn how to use FitNesse, and how well written acceptance tests are an executable specification of an application.

June 5th

Node for a real-time web : Remy Sharp

Node allows for applications to be developed in JavaScript and is running on the incredibly fast V8 engine (the same JavaScript engine used to run Google Chrome). It can handle thousands of concurrent connections due to it's event model, and because of this event model, developing for Node is very familiar if you've already written JavaScript for the browser – as this the same event model. Topics that will be covered; the basics, debugging techniques and tools, building web servers, WebSockets in minutes, and Repo Control.

June 5th

Develop Mobile Applications with C# and .Net : Chris Hardy & Jonas Follesø

Join Chris Hardy and Jonas Follesø as they will transform you from a .NET developer into a iOS and Android mobile developer in this one day workshop on using .NET and C# to develop native iOS and Android mobile applications using MonoTouch and Mono for Android. This is your chance to find out how easy mobile development with C# and .NET really is and how you can produce stunning native applications.

organizers

ProgramUtvikling Microsoft

ndcoslo.com

NDC 2012 NORWEGIAN DEVELOPERS CONFERENCE

PROGRAM – Wednesday

Agile · Cloud · Craftsmanship · Design & Architecture · Devops · Interesting & Strange · Microsoft
Mobile · Programming Languages · Security · Tools & Frameworks · UX · Web

TIMESLOTS	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Workshop
09:00 - 10:00	Keynote								
	Aral Balkan								
10:00 - 10:20	Break								
10:20 - 11:20	Getting Agile with Scrum	WebGL What?	Inside Mono for Android	Win8	Living with an evolving architecture	Ideas for efficient BDD with SpecFlow through examples	Just cause it's JavaScript, doesn't give you a license to write rubbish	Making an awesome Open-Source Project	Tutorial: Enterprise development with NServiceBus
	Mike Cohn	Rob Ashton	Andreia Gaita	Giorgio Sardo	Dan North	Gáspár Nagy	Hadi Hariri	Paul Betts	Andreas Öhlund
11:20 - 11:40	Break								
11:40 - 12:40	Leading a Self-Organizing Team	HTML5 Game Development	Patterns of Mobile Application Development	Win8	MicroService Architecture	Roslyn... hmmmm... what?	Professional Software Development	NuGet: Zero to DONE in no time.	Tutorial: Enterprise development with NServiceBus
	Mike Cohn	David Geary	Dino Esposito	Giorgio Sardo	Fred George	Shay Friedman	Robert C. Martin	Phil Haack	Andreas Öhlund
12:40 - 13:40	Lunch								
13:40 - 14:40	Agile Estimating	JavaScript All Over - Sticking Your Big Toe in Node.js	Mobile, How Do I Develop Thee? Let Me Count the Ways	Introduction to XAML in Windows 8/ Metro	Object Orientation Revisited. Simplicity and power with DCI.	Five Things You Didn't Know About PostGresSQL	Building External DSLs for Fun and Profit	Hacking .Net Applications: The Black Arts	Tutorial: Enterprise development with NServiceBus
	Mike Cohn	Sara Chipps	James Hughes	Billy Hollis	Trygve Reenskaug	Rob Conery	Jimmy Bogard	Jon Mccoy	Andreas Öhlund
14:40 - 15:00	Break								
15:00 - 16:00	Advanced Topics in Agile Planning	The rise of server-side JavaScript	One Service, Any Device, Any Experience: ASP.NET Web API, JSON, and Windows 8, Windows Phone, iOS, and Android	Metro Design Principles	Modeling Distributed Systems with NServiceBus Studio	WCF Extensibility: Tapping into the calls	Advances in Code Contracts for .NET	Authentication & Authorization in .NET 4.5 - Claims & Tokens become the standard Model	VIM for aspiring #ithipsters
	Mike Cohn	Guillermo Rauch	Ralph Squillace	Laurent Bugnion	Udi Dahan	Miguel Castro	Thomas Ball	Dominick Baier	Andreas Heim
16:00 - 16:20	Break								
16:20 - 17:20	Scaling Agile to Work with a Distributed Team	Learning to love HTML and CSS (for grumpy developers)	Windows Phone 7.5 Background workers	TBA	Crafting Wicked Domain Models	Moles, isolation framework for .NET	How we do language design at Microsoft: VB and C#	Mind Control Your Computer In C#: Natural User Interfaces Through The Power Of Thought	
	Mike Cohn	Jon Galloway	Gill Cleeren	Jonas Follesø	Jimmy Bogard	Jonathan "Peli" De Halleux	Lucian Wischik	Guy Smith-Ferrier	
17:20 - 17:40	Break								
17:40 - 18:40	User Stories for Agile Requirements	The Outliers of HTML5	The Joy of Push	In and out in WinRT	The Single Responsibility Principle.	Nemerle Programming Language	Continuous testing	Social Clairvoyance	
	Mike Cohn	Remy Sharp	Joe Pezzillo	Gill Cleeren	Robert C. Martin	Igor Tkachev	Svein Arne Ackenhausen	Gary Short	

PROGRAM – Thursday

Agile · Cloud · Craftsmanship · Design & Architecture · Devops · Interesting & Strange · Microsoft
Mobile · Programming Languages · Security · Tools & Frameworks · UX · Web

TIMESLOTS	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Workshop
09:00 - 10:00	When The Wheels Come Off Agile	TBA	Securing ASP.NET Web APIs	Developing hybrid solutions with Windows Azure Service Bus, WF and BizTalk Server 2010	Clean Architecture	Debugging .NET with WinDBG	What you don't know you don't know: How to become a better developer by breaking out of your comfort zone	The Process, Technology and Practice of Continuous Delivery	TBA
	Barry Hawkins	Damian Edwards	Dominick Baier	Paolo Salvatori	Robert C. Martin	Cory Foy	Roy Osherove	David Farley	Scott Allen
10:00 - 10:20	Break								
10:20 - 11:20	Embracing Uncertainty	Microsoft's Modern Web Stack, Starring ASP.NET Web API	Not a Mobile Developer? Not a Developer!	Hell has frozen over: writing Node.js apps for Windows Azure	Design Patterns for .NET Programmers	Dataflow networks in .NET 4.5	I Can't Believe It's Not Roslyn--Using Nemerle Macros to extend the C# Language	Continuous Delivery story with FIFA	TBA
	Dan North	Brad Wilson	Dino Esposito	Yavor Georgiev	Venkat Subramaniam	Alon Fliess	Philip Laureano	Mirosław Jedynak	Scott Allen
11:20 - 11:40	Break								
11:40 - 12:40	Scaling Agile Teams	Real World NodeJS - Creating the Tekpub API	Interactive user experience: natural user interfaces	Extending XAML To Overcome Pretty Much Any Limitation	Commands, Queries, and Consistency	Git and GitHub for Developers on Windows	Reinventing software quality	A Thousand deliveries a Day at AppHarbor	CyberDojo - genuine coding practice
	Esther Derby	Rob Conery	Alisa Smerdova & Felipe Longé	Miguel Castro	Udi Dahan	Phil Haack	Gojko Adzic	Michael Friis	Jon Jagger & Olve Maudal
12:40 - 13:40	Lunch								
13:40 - 14:40	Reintroducing Business Analysis into the Agile Stream and The Need for Structuring the Conversation with Stakeholders	HTML5 WebSockets and Socket.IO	Creating User Experiences: Unlocking the Invisible Cage	Windows Azure Access Control - Outsourcing Security to the cloud	What is OO? Where did it come from? Where is it going?	Full-text search with Lucene and neat things you can do with it	Dynamic .NET Demystified	TBA	CyberDojo - genuine coding practice
	Howard Podeswa	Guillermo Rauch	Billy Hollis	Dominick Baier	Robert C. Martin	Itamar Syn-Hershko	Keith Dahlby	Tore Vestues	Jon Jagger & Olve Maudal
14:40 - 15:00	Break								
15:00 - 16:00	The surprising science behind agile leadership	CSS in the 4th dimension: Not your daddy's CSS animations	Designing for Touch	Cloud Computing: More Than Just Hosting	RavenDB data modeling walkthrough	Git More Done	Branch-per-Feature in the realm of Agile and .NET development	Patterns of Effective Delivery	
	Jonathan Rasmusson	Lea Verou	Billy Hollis	John Sheehan	Itamar Syn-Hershko	Keith Dahlby	Adam Dymitruk	Dan North	
16:00 - 16:20	Break								
16:20 - 17:20	Kanban: Why it matters	Mind-blowing Apps with HTML5 Canvas	Building Applications with ASP.NET MVC	How to think cost when programming Cloud Computing apps?	Dealing with Dynamically-Typed Legacy Code	Introduction to Rx	A Better Way To Learn Refactoring	Ten Web Performance Tuning Tricks in 60 Minutes	Script your phone on your phone with touchdevelop.
	Cory Foy	David Geary	Scott Allen	Ole – André Johansen	Michael Feathers	Paul Betts	Philip Laureano	Richard Campbell	Jonathan "Peli" De Halleux
17:20 - 17:40	Break								
17:40 - 18:40	How to get productive in a project in 24h	Making Test Automation with Web Applications Work	.NET Rocks! Panel	The Cloud Security Rules	Deep Design Lessons	Debugging the Web with Fiddler	Revealing the SQL Server Magic	SignalR: Awesome in Real-Time with ASP.NET	
	Greg Young	Jeremy D. Miller	.NET Rocks!	Kai Roer	Michael Feathers	Ido Flatow	Mark S. Rasmussen	Damian Edwards	

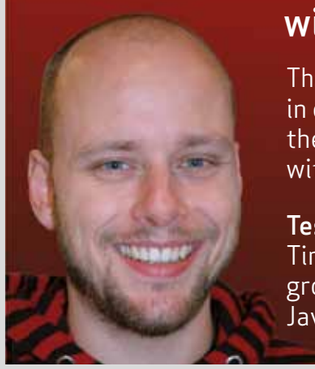
PROGRAM – Friday

Agile Cloud Craftsmanship Design & Architecture Devops Interesting & Strange Microsoft
Mobile Programming Languages Security Tools & Frameworks UX Web

TIMESLOTS	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Workshop
09:00 - 10:00	Developers: The Prima Donnas of the 21st Century	Hybrid Applications with MongoDB and RDBMS	MVVM Applied: From Silverlight to Windows Phone to Windows 8	A WIF Of Security In SharePoint And Azure	An architec-ture remake	Gamifying testing by accident with pex4fun.	Thinking in Functional Style using F# and (some) C#	What?!? C# Could Do That?!?	GOAL - Game Oriented Agile Learning
	Hadi Hariri	Chris Harris	Laurent Bugnion	Sahil Malik	Jimmy Nilsson	Jonathan "Peli" De Halleux	Venkat Subramaniam	Shay Friedman	Paul Goddard & Geoff Watts
10:00 - 10:20	Break								
10:20 - 11:20	Agile, Lean, and the Space Between	HTML5 and CSS3: does now really mean now?	Xamarin - State of the Union	Making your Application Cloud-ready	Beyond the compiler - going up to 11 with conventions in a statically typed language	RabbitMQ Hands On	F# in action: playing functional Conway's game of life	Gesture Recognition with Kinect for Windows	GOAL - Game Oriented Agile Learning
	Barry Hawkins	Chris Mills	Chris Hardy	Michael Friis	Krzysztof KoDmic	Alvaro Videla	Vagif Abilov	Carl Franklin	Paul Goddard & Geoff Watts
11:20 - 11:40	Break								
11:40 - 12:40	Programmer Anarachy	How to Destroy the Web	MonoTouch - C# + iOS = Good Times	SharePoint, Office 365, JavaScript And Azure	Neo4j in a .NET world	TBA	The Ajax Continuation Pattern	.NET in the physical and meta-physical world	GOAL - Game Oriented Agile Learning
	Fred George	Bruce Lawson	James Hughes	Sahil Malik	Tatham Oddie	Scott Allen	Jeremy D. Miller	Jonas Winje, Einar W. Høstand & Bjørn Einar Bjartnes	Paul Goddard & Geoff Watts
12:40 - 13:40	Lunch								
13:40 - 14:40	What Does Self-Organiz-ing Team Really Mean?	Accessibility doesn't exist!	Creating UX with Story Boarding	How GitHub Works	.NET On a Diet	Introduction to OWIN and Gate	Clojure for the Web	Async Part 1 -- new feature in Visual Studio 11 for responsive programming.	Learn how to build a modern browser applications using Backbone.js and the ASP. NET Web API
	Esther Derby	Chris Mills	Billy Hollis	Zach Holman	James Hughes	Dale Ragan	Bodil Stokke	Lucian Wischik	Anders Norås
14:40 - 15:00	Break								
15:00 - 16:00	10 things you can do to better lead your agile team	HTML5 Multimedia	High-Performance Notifications: Using Windows Azure Service Bus as an Internet-Scale Service Bus	Big time: introducing Hadoop on Azure	The era of tiny	SignalR: In browser two-way continuous communication	Pure JavaScript	Async Part 2 -- deep dive into the new language feature of VB/C#	
	Jonathan Rasmusson	Bruce Lawson	Ralph Squillace	Yaniv Rodenski	Jimmy Nilsson	Hadi Hariri	Christian Johansen	Lucian Wischik	
16:00 - 16:20	Break								
16:20 - 17:20	You had me at Halo	Metaprogram-ming with Nemerle	From Zero to Hero	Running SQL Server on Amazon EC2	Caring about Code Quality	No more magic	Develop mobile websites (not only) for the Windows Phone with ASP.NET MVC, HTML5 & jQuery	Continuous integration - char by char	
	Kari-Amelie Fiva Aasheim	Igor Tkachev	Sondre Bjellås	Mark S. Rasmussen	Venkat Subramaniam	Stefan Daugaard Poulsen	Max Knor	Harald S. Fianbakken	

Training for developers

Introductory JavaScript for programmers
with Christian Johansen



The JavaScript programming language is covered in depth. Gain the necessary understanding of the language to work actively and effectively with the applications front–end.

Test-Driven JavaScript
Tired of being a jQuery cowboy? Learn how to grow confident and productive working with JavaScript in this three day JavaScript TDD course.

Web Development with ASP.NET MVC, HTML5, CSS3 and jQuery/JavaScript
with Scott Allen

The web is evolving. The buzz around HTML5 is creating a lot of commotion. Developers need to know the new standards to create web applications that are running on all kinds of devices such as tablets, PCs and mobile devices. Microsoft offers two technologies today that developers can use for their development endeavors: WebForms and ASP.NET MVC. Knowing both technologies enables developers to choose the right technology for the job. In this 5 day course, students will get to learn both server–side technologies.

Using HTML5 and JavaScript to build Web Apps
with Remy Sharp

This three day workshop will introduce you to HTML5 with a brief backstory, before diving into the APIs one by one. As well as going through code and showing practical demonstrations, where possible, well also talk about the alternatives and polyfills for old browsers that dont support "awesome" out of the box.

ProgramUtvikling
OSLO - www.programutvikling.no

DeveloperFocus
LONDON - www.developerfocus.com