The **DEVELOPER** *2nd issue - Special Edition*

NDC MAGAZINE

For software developers and leaders

NDC motivates Unit 4, Agresso

see why at p.28

Scott Guthrie
Keynote Speaker at NDC 2011

see Agenda at p.42-43

The Big Wall: How to Bring a Huge Product Backlog Down to Size

by Mitch Lacey

Who needs a service bus, anyway?

by Udi Dahan

ALSO FEATURING:

IAN ROBINSON

ANDERS NORÅS

ROD PADDOCK

GASPAR NAGY

JONAS BANDI

JEFF WILCOX

GREG SHACKLES

DOMINICK BAIER

JON SKEET

GILL CLEEREN

GARY SHORT

BILLY HOLLIS

MARK SEEMANN

SCOTT BELLWARE

NORWEGIAN DEVELOPERS CONFERENCE 2011
Oslo Spektrum, Oslo
June 8 -10th

NDC 2011 NORWEGIAN DEVELOPERS CONFERENCE

EMC² VELOCITY² PARTNER PROGRAM

Microsoft Partner
Gold Portals and Collaboration

Navn: Hege
Lidenskap: Spreke løsningskonsepter
Jobb: Senior Business Consultant

Navn: Terje
Lidenskap: SharePoint
Jobb: Senior Software Engineer

FOTO: BO MATHISEN

Bli en del av vårt sterke fagmiljø innen samhandling og SharePoint

Steria er blitt kåret til en av Norges beste arbeidsplasser fire år på rad. Det er ikke uten grunn. Mye skyldes de gode utviklingsmulighetene. Mye skyldes de utfordrende prosjektene. Mye skyldes det sterke og gode fagmiljøet.

→ Velkommen til et av Europas største it-konsulentselskaper

Det har skjedd mye siden vi etablerte oss i 1969 i Paris. Vi er nå et av Europas 10 største it-konsulentselskaper med 20 000 medarbeidere i 16 land og en omsetning i 2010 på 14,2 milliarder. I Norge er vi 530 medarbeidere som sørger for en årlig omsetning på 835 millioner kroner. Vi holder til i et av Oslos beste lokaler på toppen av Posthuset.

GREAT PLACE TO WORK® INSTITUTE NORGE
Norges BESTE arbeidsplasser 2011

UNIVERSUM® TOP 10 IDEAL EMPLOYER 2009 STUDENT SURVEY

Vårt fagmiljø innen samhandling og informasjonshåndtering er trolig et av Nord-Europas sterkeste. Vi er EMC Documentum partner og den eneste leverandøren som har godkjent Noark 5 kjerne for Documentum med sakarkiv. I tillegg er vi Microsoft Gold Partner innen SharePoint Portals and Collaboration. Vi opplever nå økt tilgang på oppdrag og trenger konsulenter innen flere områder:

- **Erfarne samhandlings- og portalrådgivere**
  Du bør ha flere års erfaring med SharePoint som utvikler, rådgiver eller arkitekt, og et ønske om å jobbe som rådgiver.
- **Arkitekter, utviklere og rådgivere innen informasjonshåndtering**
  Du bør ha relevant utdanning som ing./siv.ing. eller innen arkivfag, gjerne praktisk erfaring med ledende ECM-plattformer som f. eks. Documentum og gjerne med Noark-standarden.
- **SharePoint rådgivere, utviklere og arkitekter**
  Du bør ha erfaring med Microsoft SharePoint samt utvikling på Microsofts .net plattform.

→ Spennende utfordringer

Steria er en kunnskapsbedrift. Vi lever av å gi gode råd om ledelse og it. Basert på våre sterke fagmiljøer bistår vi en rekke av Norges største virksomheter innen offentlig og privat sektor med modernisering og bedre bruk av teknologi, prosesser og mennesker.

→ Slik søker du

Send en søknad via Sterias jobbsøkesystem på www.steria.no/jobb. For generelle spørsmål, kontakt Marthe Outzen på tlf. 976 25 004. Har du faglige spørsmål relatert til den enkelte stilling, finner du kontaktpersonen i stillingsutlysningene på våre websider.

steria

→ Rådgivning → Systemutvikling → Infrastruktur → Drift

→ www.steria.no

Kjersti Sandberg, founder of ProgramUtvikling AS and Project Manager for the Norwegian Developers Conference.

Software significance

The software industry is becoming omnipresent in our lives and its growth is accelerating. It's a large and essential industry and we, ProgramUtvikling AS in partnership with Microsoft Norway, are proud to deliver one of the top conferences worldwide within .NET technologies and agile development. The vision behind the Norwegian Developers Conference is to offer applied knowledge and new ideas that inspire the software developers and project managers of this significant industry.

NDC is like a tapas table serving a variety of interesting software related topics. It's a three-day bonanza of top speakers, intriguing topics, mingling, and meet and greet with international software stakeholders. On Thursday we top it off with a huge party in Oslo Spektrum with music, food and dancing.

The liaison group for NDC.
From left: Jakob Bradford, Henriette Holmen, Kjersti Sandberg, Einar Ingebrigtsen, Petri Wilhelmsen and Jonas Follesø.

NDC is an international conference with principal speakers and participants from all corners of the world, from New Zealand and Australia to Hong Kong and Stockholm. Oslo has become an attractive and exotic conference location with an esteemed reputation, hosting NDC in the summer and JavaZone in the fall.

NDC offers you a wide-ranging insight into the many areas of technology application – from an agile perspective. Topics include architecture, design and technology suitable for all developer platforms. Choose the areas that suit you from our plentiful program offer, described in more detail in this magazine.

We welcome you to one of the most important events in the software developer industry and hope you enjoy it and leave inspired and motivated!

Publisher: Programutvikling AS
Organisation no.: 963 201 141
Address: Martin Linges vei 17-25, 1364 Fornebu, Norway
Phone: +47 67827235
E-mail: info@programutvikling.no

Print run: 15 000

Publication schedule 2011:
No. 3: 15 August
No. 4: 15 November

Print: Merkur Trykk AS

ARTICLES:

Foto: iStock • Design/layout: macsimum.no

Silverlight and Windows Phone 7 – a powerful combo

by Gill Cleeren

When Microsoft introduced Silverlight in its very first form – Silverlight 1.0 at Mix 2007 – I was among those for saw the power and flexibility of the new platform. It was then that Microsoft made a very important step in the world of RIAs (Rich Internet Applications). However, at that time, it didn't cross my mind that a few years later, I'd be using the same technology and tools for building apps for both the web and my mobile device (note that we were working with Windows Mobile 5.0). And yet, that's where we are today. In this article, I want to give you a small overview of the options and capabilities of Microsoft's new mobile offering.

Designing for Windows Phone 7

I assume that by now, you've all seen a Windows Phone 7. In case you haven't, take a look at the image to the right, which shows the WP7 start screen. The interface looks fresh and clean and with a simple glance at the screen, the user knows how many emails he/she has, how many calls he missed or what his friends are up to. Typography and simple, understandable icons are key in the design experience, dubbed Metro. Many things are animated (*which is hard to show in a printed magazine!*) but since these animations are subtle, they add to the experience instead of overloading the interface.

When building applications for a well-designed platform, we as developers must try to oblige to the design guidelines as well. By doing so, we make our applications easier to learn for end-users. Using large fonts, simple icons and transparency are easy things to do in our own apps.

Silverlight as the core of Windows Phone 7

As mentioned in the introduction, Silverlight is one of the two possible ways to create applications for WP7. The second option is XNA which targets game development. It's safe to say that for typical business applications, Silverlight will always be used. The framework available for developers on the phone is basically composed out of

- Silverlight 3
- Some Silverlight 4 features
- Windows Phone 7 framework

Silverlight itself is of course fully based on .NET, so we get all the things we know and love including threading, LINQ, service access... The Silverlight specifics such as the animation framework, navigation, most Silverlight controls and of course XAML markup are included as well. Finally, the WP7 framework extends this to include phone-specific features such as access to device hardware (camera, accelerometer...), specific phone controls, phone and SMS integration and much more.

> We as developers must try to oblige to the design guidelines

Visual Studio integration

To build Silverlight applications, both with Silverlight and XNA, Visual Studio is used. To make things easy for developers, Microsoft offers a free version, called Visual Studio 2010 Express for Windows Phone. If you have any version of Visual Studio, you can use the Windows Phone 7 tools which enable WP7 development from the IDE.

An important part of what's installed is the emulator. This emulator allows us to test our applications (both Silverlight and XNA) while having the debug experience from Visual Studio. We can also from Visual Studio deploy our code to a device and test applications straight on the device. One side note here though: to enable deploying applications to an actual device, you need a developer account. Once you're registered, you can "unlock" your device to become a developer device, on which you can then deploy and test.

Do-it-yourself Hub experiences

To show how easy it is to create compelling Silverlight applications on the phone, I'm going to finish of this article by showing how we can use the Panorama control to create a Hub-like experience. This begs of course for an explanation on the concept of a hub.

In the WP7 system, there are a few so-called hubs available. A hub can be considered as a landing page for a whole experience, be it your pictures, your music, the marketplace or Office. From there, you get a very rich overview of your content.

By including the Panorama control in the SDK, Microsoft makes it easy for Silverlight developers to create their own applications with the same design. The Panorama control mimics the behavior of a hub. Using this control, it's possible to flip horizontally through several "tabs" of content in a cyclical way (meaning that after you've scrolled through all tabs, you'll end up at the first one again). In the end, the user of your application will benefit from this as he'll have no difficulties figuring out how your application works: it's working in the exact same way as the operating system's hubs.

Take a look at the following XAML code

It shows the XAML markup of a Panorama control with a few defined tabs. Each tab can contain whatever content you need to place in it. So if you want to put a ListBox in there or a Grid with images, the Panorama will display them fine.

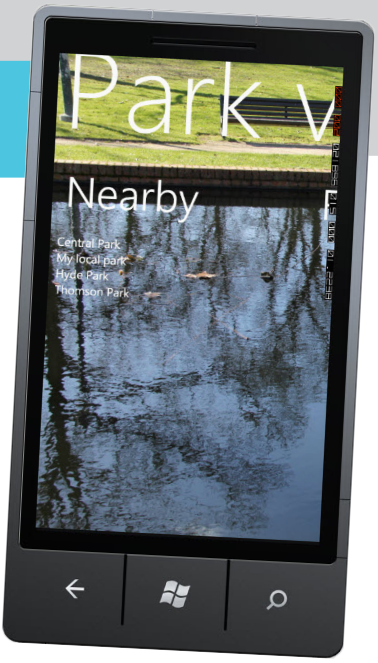
```
<Grid x:Name=”LayoutRoot”>
    <controls:Panorama Title=”Park viewer”>
        <controls:Panorama.Background>
            <ImageBrush ImageSource=”Background.jpg”></ImageBrush>
        </controls:Panorama.Background>
        <controls:PanoramaItem Header=”Nearby”>
            <ListBox>
                <ListBox.Items>
                    <ListBoxItem Content=”Central Park” />
                    <ListBoxItem Content=”My local park” />
                    <ListBoxItem Content=”Hyde Park” />
                    <ListBoxItem Content=”Thomson Park” />
                </ListBox.Items>
            </ListBox>
        </controls:PanoramaItem>

        <controls:PanoramaItem Header=”Far away”>
            <Grid/>
        </controls:PanoramaItem>
    </controls:Panorama>
</Grid>
```

You can see the result in the image to the right.

A powerful combo

If you already know Silverlight, the step to become a Windows Phone 7 developer is a very small one. Installing the tools and getting to know the phone specific features such as the controls and working with the hardware don't pose a steep learning curve. The power of Silverlight and the clean, fast and modern design of the WP7 platform make them a powerful combination while giving Silverlight developers an even wider target to apply their knowledge.



The Big Wall: Bringing a Huge Product Backlog Down to Size

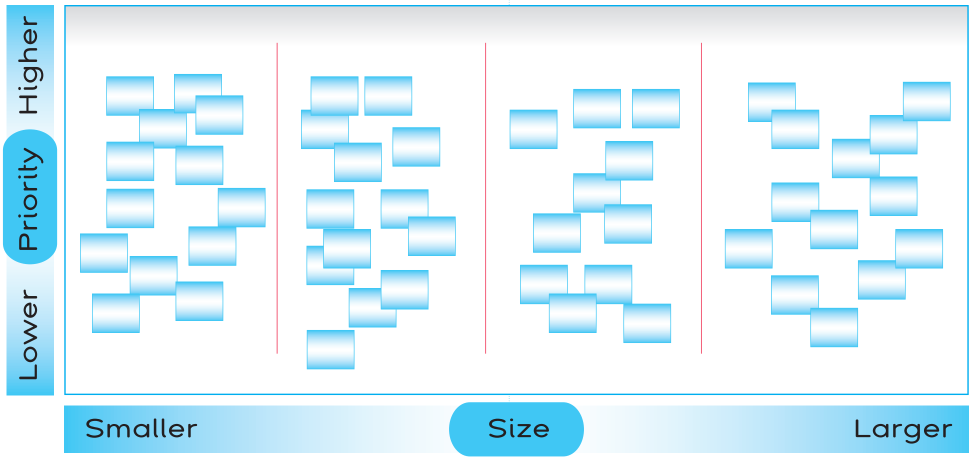
by Mitch Lacey

One of the biggest challenges for product owners on a new project is working with a raw (not estimated or prioritized) product backlog. While Planning Poker is a fantastic tool for estimating user stories, it's better suited for a small pile of stories than a stack a mile high. The sheer size of the effort can be overwhelming.

One of the first tools I turn to when faced with a raw product backlog is what I call "The Big Wall." The idea is to get all the user stories up on the wall in a way that generally reflects their size and importance relative to each other. You'll need two things: a stack of user stories and a big empty wall, about 14 feet long by 8-10 feet high. Then you need to gather your stakeholders and team and plan to spend a day or so getting physically involved with your stories.

If you already have an estimated backlog, you can just do the prioritization section of this exercise. If you already have a prioritized backlog, you can just do the estimation section of this exercise. (Your product owner will likely want to revisit the prioritization after the estimates are done. After all, cost has a big impact on priority.)

Figure 1. Stories are divided into size groups on the wall.



Estimation

Ideally, given a raw product backlog, you should start with estimation. For that, you'll need your team and your product owner. Estimation is expressed by putting the index cards in the correct place on the wall. Ask the team members to determine a story's proper placement with the following two poles in mind:

- The far left side of the wall holds the smallest possible stories
- The far right side contains the biggest possible stories.

Do not worry about the numbers at this point. Instead, have team members choose a story's position on the wall relative to the two poles and the other stories on the wall. Team members should feel free to move stories right or left as needed.

Once you've placed the stories on the wall, discuss with the team where the logical breaks are between story sizes. Put a line of tape where the team determines the stories separate into point groups. Soon you'll have a wall that looks something like the wall in Figure 1. Now that you've estimated your stories, you'll need to assign them a priority.

Prioritization

Your customers and stakeholders will look at the stories in a much different way than the team just did. They're not as interested in how big or small a story is; they are most interested in finding the stories that relate to them and making sure those stories get done. This will necessarily put the stakeholders in conflict with one another. This is ok –in fact, it's ideal because it reflects reality and helps the product owner make tough decisions.

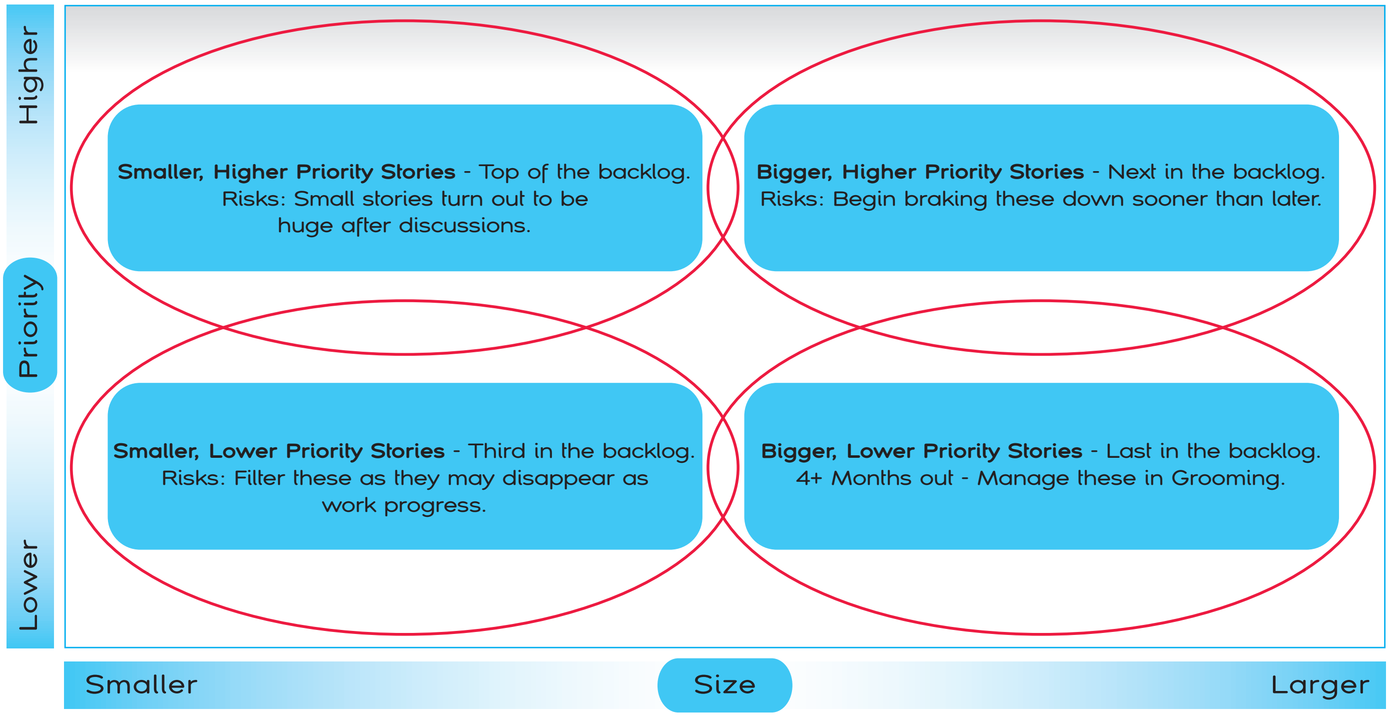
The first thing I do is explain to the stakeholders why we are here and what we want to accomplish. I tell them that the stories on the wall are in size order, with those to the left being the smallest and those to the right being largest. I then ask them to move the stories up or down inside the taped lines based on how crucial it is to the project. The higher up a story is on the wall, the higher its priority to the business. If a stakeholder places a story at the top, I ask for justification as to why it's up there. I encourage stakeholders to challenge each other as to why one story is more important than the other.

If someone moves a story lower on the wall than someone else did, the story should be marked with a yellow dot. This alerts everyone that we might need to have a conversation about the story's true priority. I also encourage/expect people to ask, "Who moved this one down (or up)?" or to say aloud, "I think this one needs to move. Who wants to disagree?" This enables a conversation to happen between the interested parties, without facilitation. If a discussion goes on too long without resolution, the facilitator (usually the product owner), should collect the card, note the two stakeholders who cannot agree, and make a note to meet with them privately later. I often invite the team to be in the room for prioritization, though they are not active participants. Instead, the team members observe, taking notes on the behavior, the interactions, and the reasons why certain stories are rising or falling in priority. They can also answer any stakeholder questions, if needed. For example, if there are stories that the team couldn't size with confidence, the team can clear up any confusion by asking specific stakeholders about those stories, as time allows.

In the end, all the participants will know how all the stories relate and have an idea of what functionality will be delivered when. The goal of prioritizing as a group is to help all stakeholders understand how all the stories relate to each other, be able to visualize the priorities of various stories and how the others view it, and have an idea of what functionality will be delivered and when.

Prioritization should take 2-6 hours, depending on the number of stories and the number of stakeholders.

Figure 2. The estimated and prioritized wall breaks into 4 quadrants.



Smaller, Higher Priority Stories - Top of the backlog. Risks: Small stories turn out to be huge after discussions.

Bigger, Higher Priority Stories - Next in the backlog. Risks: Begin braking these down sooner than later.

Smaller, Lower Priority Stories - Third in the backlog. Risks: Filter these as they may disappear as work progress.

Bigger, Lower Priority Stories - Last in the backlog. 4+ Months out - Manage these in Grooming.

Relating the Wall to the Product Backlog

When the dust has settled and the group steps back to look at what they've done, your wall will break down roughly into four quadrants, as shown in Figure 2.

The stories in the top left are high priority and small. They'll end up in the top of the product backlog. The stories in the top right are high priority and large. You should break most of those down into smaller stories because they'll be coming up in the first several sprints.

The bottom quadrants are less urgent. The lower left quadrant is made up of small stories that are lower in priority. The lower right quadrant is filled with large stories that are lower in priority. These stories are your epics or themes. They'll eventually need to be broken down into smaller, more manageable stories, but not until they rise in priority.

Leave some time to reexamine the wall as a whole. If stories are in the wrong quadrant, move them. If they need to be broken down, try to do it while everyone is in the room. By the time you are finished, the stakeholders will be able to see the start of a release plan. If you know the team's historical velocity, you can even give them a rough range of which stories in the upper left quadrant will be finished.

Don't be overwhelmed by a raw product backlog. You can estimate and prioritize even the most out-of-control product backlog fairly quickly if you have the right people, a lot of ink and paper, a little time, and a really big wall.

Leave some time to reexamine the wall as a whole

Tackling Technical Debt

by Gary Short

Technical Debt is the silent spectre that haunts software projects. Take your eye off of it, even for a moment, and it can build up so fast that it causes your project to fall so far behind or to go so far over budget, that the only sensible thing to do is to cancel it.

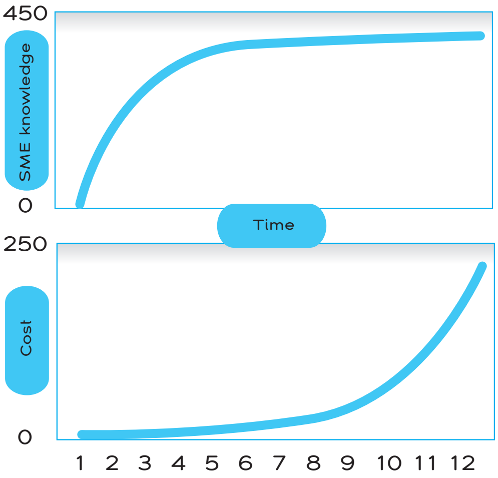
The term was first coined by Ward Cunningham while trying to describe this phenomenon to managers in the financial institution at which he was working. They understood the concept of financial debt, so he continued the financial metaphor: just as you borrow money now for financial gain, knowing you will have to pay back the original capital plus a little interest in the future, so you borrow time now to gain an advantage, knowing that you will have to pay back that original time with a little interest.

Of course, it is not the debt itself that is the problem. It is the management of such debt. The same can be said of Technical Debt. If you do not manage it correctly, or if you ignore it completely, then it is only a matter of time before it bankrupts your project.

There are a number of anti-patterns that can lead to the build-up of Technical Debt. Thankfully, if you can recognise the anti-patterns, there are fixes that you can apply to mitigate against them. One such anti-pattern is the waterfall methodology.

Although there are many sound reasons for utilising this method, lack of foresight within your waterfall project can lead to the build up of Technical Debt. The reason is a function of how the cost of change increases with time. As the process moves through each stage, so it becomes more expensive to make changes. This would not be a problem if it were not for the fact that your Subject Matter Expert goes through a learning curve of his own as the project moves forward. No one knows everything there is to know about a particular subject, and as his knowledge is tested, he will fill in the blanks that inevitably exist.

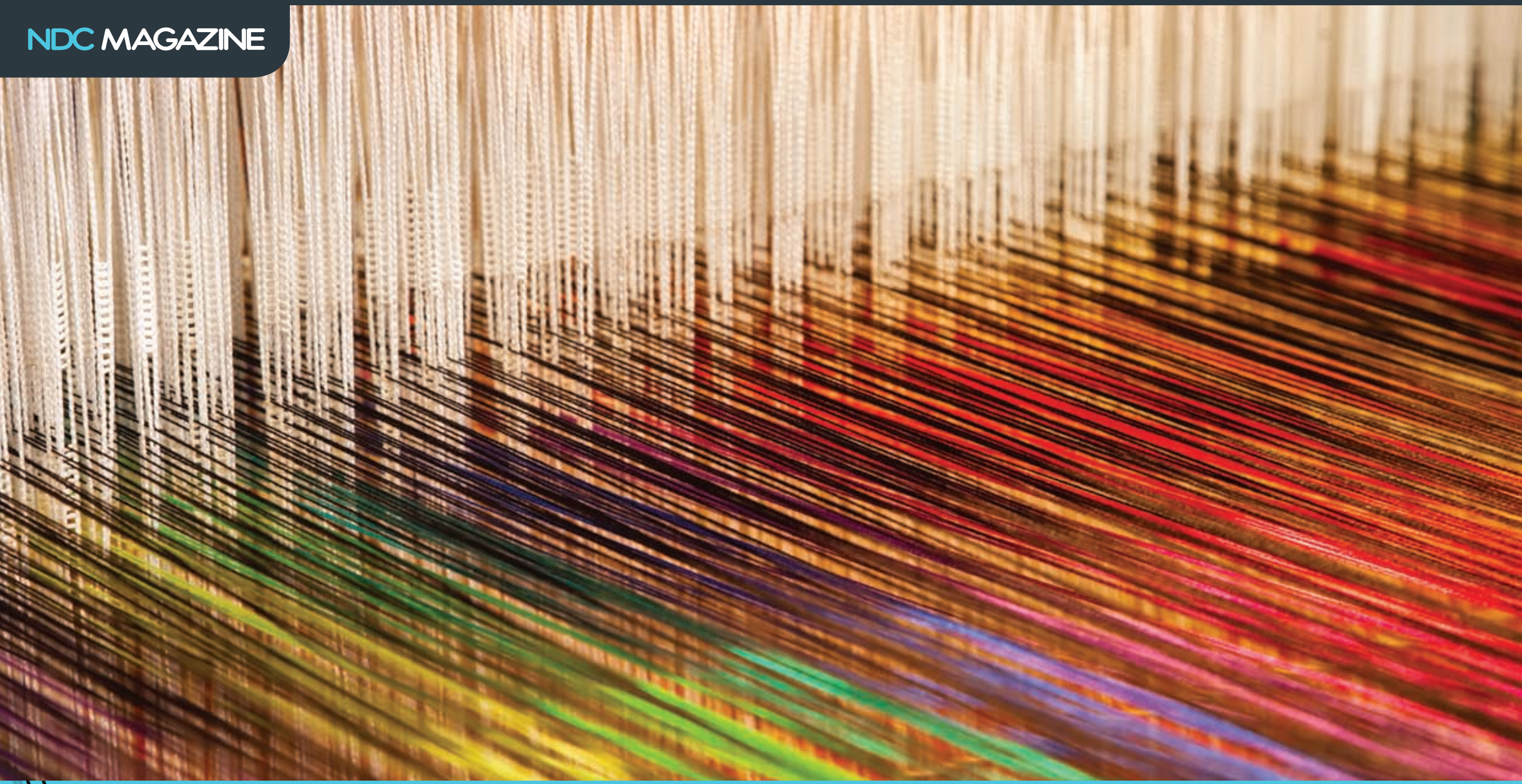
Unfortunately, this means that at the point where your SME knows the most about the subject and can suggest necessary changes, we reach the time at which it is most expensive to implement those changes. This expense presents the temptation for the business to ship the product, with the known shortcomings, instead of swallowing the cost, believing that they will come back in the future and fix things.



Continuously shipping software with known issues is what leads to Technical Debt in a project.

The two most effective weapons to fight this form of Technical Debt are to ensure that the SME is peer reviewed, in the hope of spotting shortfalls in their knowledge earlier, and to begin the testing phase as early as possible in the project.

Technical Debt need not be a cumbersome weight around the neck of your project. Like financial debt, if managed continually and effectively, it can be kept to such a level that, although being ever-present, its effects on your project go almost unnoticed.



Cross Platform Mobile Development with .NET

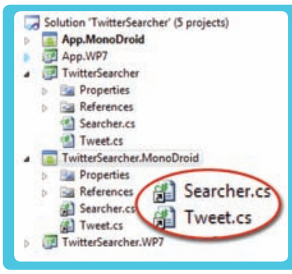
by Greg Shackles

It would be an understatement to say that the mobile space is booming right now, and the trend has yet to show any signs of slowing down. With so many dominant players in the market, each with its own technology stack, the thought of developing for all of them is daunting but unavoidable. There's the option of a mobile website, but it comes at the cost of losing the richness and improved user experience of a native application. Strange as it may seem, .NET developers are actually in the best position of all to tackle this problem. Windows Phone 7 runs .NET natively, and products like MonoTouch and Mono for Android allow developers to leverage the .NET framework on iOS and Android as well.

It's important to remember that although you can leverage the .NET framework on each platform, that doesn't mean the goal is write-once-deploy-everywhere. Instead, this allows you to separate your core application logic from the user interface, and share that core logic across platforms. The sharing isn't limited to mobile platforms, either. Since you're still working with .NET, the same code can be used in ASP.NET, Silverlight, WPF, etc. For each platform you essentially build a native user interface on top of that shared business layer.

That sounds great in theory, but how does it work out in practice? Let's talk a bit about compatibility. MonoTouch and Mono for Android both make use of Moonlight, Mono's implementation of Silverlight, for most of their core libraries. Since Silverlight is also used in Windows Phone 7, this makes a great foundation for building shared mobile libraries, and because it is a subset of the full .NET framework you get good compatibility with non-Silverlight platforms as well.
Your libraries still need to be compiled against each platform in order to work properly, though, so sharing a single DLL isn't an option. The best approach is to leverage the ability to link files across projects, which is supported by both Visual Studio and MonoDevelop. If you're using Visual Studio, there's a handy extension named Project Linker that helps make this process a little easier by automatically managing changes in linked projects for you.

Take a look at Figure 1 to see what a solution might look like that uses this approach. TwitterSearcher is a standard .NET library, and contains all the actual code files. TwitterSearcher.MonoDroid is an Android class library, links in the code files from the source project, and will compile that code against the Mono for Android profile. App.MonoDroid is an Android application, and references the Android class library. The same approach is used for WP7.



```
Solution 'TwitterSearcher' (5 projects)
  App.MonoDroid
  App.WP7
  TwitterSearcher
    Properties
    References
    Searcher.cs
    Tweet.cs
  TwitterSearcher.MonoDroid
    Properties
    References
    Searcher.cs
    Tweet.cs
  TwitterSearcher.WP7
```

Figure 1

As an added bonus, you'll often find out right at build time if there's a platform incompatibility. If you do encounter something that cannot be built on a particular platform, each target provides compilation symbols to let you #if away your troubles. Using directives can often be avoided, though, thanks to the common foundation across platforms.

The ability to share business logic across platforms is huge, especially when you can leverage the power of .NET. By sticking with a framework you already know you can concentrate on delivering a solid user experience, all while knowing that your foundation is stable. Life is good for the .NET developer, so get out there and write some apps!

Blog: http://gregshackles.com
Code: http://shackl.es/NDC-Code
Post: http://shackl.es/NDC-Post

Your trusted vendor of
productivity tools
for software developers
and teams

Application State in RESTful Distributed Applications

by Ian Robinson

One of the things that distinguishes RESTful distributed applications from many service-oriented distributed solutions is the way in which they deal with application state. In this article we'll look at how a typical service-oriented composition service handles application state, and then at how a RESTful application handles the same concerns. We'll also look at how the RESTful approach to application state impacts the design of both clients and servers.

Service orientation and application state

Many service-oriented distributed applications maintain application state on the server; that is, they make the server responsible for remembering how far a client has progressed through a multi-step process. With this knowledge, the server can ensure that a series of client requests are correctly ordered.

Figure 1 shows a simple procurement service in which the server is responsible for maintaining application state.

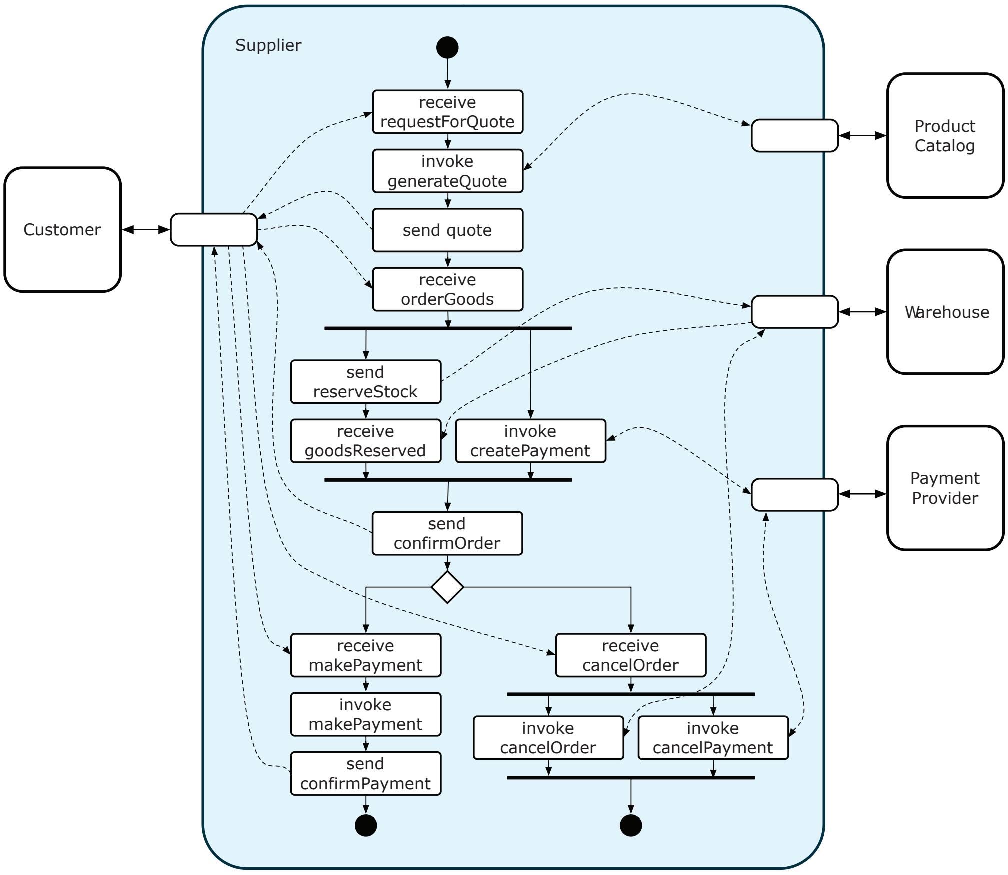


Figure 1. A service-oriented composition service for procurement

Using a composition engine, we implement the supplier service as a composite service. This supplier service is responsible for managing interactions with back-end services and applications on behalf of the client (the customer). As part of its public interface, the supplier service publishes a BPEL (Business Process Execution Language) protocol document. This protocol document describes the order in which the client must invoke service operations to execute the end-to-end process correctly. In addition to BPEL, the service uses WSDL (Web Services Description Language) to describe the capabilities of each message endpoint, XML Schema to describe the messages to be exchanged during the process, and WS-Policy to advertise its quality-of-service requirements.

In this service-based implementation of the procurement protocol, the server is responsible for maintaining application state. Of course, this application state isn't necessarily resident in memory at all times—most composition engines will persist the state of an application instance to a durable store in between requests. But irrespective of the mechanisms used to manage application instances, maintaining application state on the server is expensive—particularly when many clients are involved. In the long run, maintaining state on the server prevents the system from scaling beyond the constraints imposed by the backing store.

A RESTful approach to maintaining application state

Maintaining application state on the server is expensive, particularly when the server must deal with many clients simultaneously. To alleviate this burden, RESTful distributed applications shift the responsibility for maintaining application state from the server to the client. Each client then becomes responsible for the overall state of the application in which it is participating.

This reassignment of responsibilities is accompanied by a shift from a monolithic service model to a system in which a client interacts with a community of resources to achieve an application goal (Figure 2). These resources can be hosted and governed by a single server, or they can be distributed across the network. Either way, every resource implements the same uniform interface, which in the case of a web application is HTTP.

Implications for the client

Application state in a RESTful application is sometimes referred to as client state, or client application state. Irrespective of the terminology used, the state of the application in which a client is participating is a function of the responses received to date—status codes, headers, and the data encoded in the entity bodies.

As a maintainer of application state, the client in a RESTful distributed application is responsible for the integrity of a sequence of actions. In order to achieve an application goal the client progressively interacts with a set of resources. After each interaction the client is presented with one or more options to interact with additional resources. These options are encoded in a response using links and forms—otherwise known as hypermedia controls. The client decides which control to operate based on its understanding of the current state of the application. Figure 3 shows a client's view of how the state of an application changes as it operates controls and interacts with linked resources.

Implications for the server

This design—many resources, each of which supports the same uniform interface—is quite different from the service-based approach, where the client deals with a single, monolithic resource—the service—whose interface is composed of a proprietary set of operations. The artifacts used to establish the client-server contract are quite different too: whereas the service-oriented approach used abstract BPEL, WSDL, XML Schema and WS-Policy, the RESTful contract is based on media types, link relations and HTTP idioms. Media types describe the formats of the representations to be exchanged, together with any processing rules to be applied to representation elements. Link relations establish the semantic context for a piece of hypermedia; in describing how the resource at the other end of a link is related to the current context, a link relation helps a machine client understand why it might be worth operating the control annotated with the relation type. HTTP idioms describe the combinations of methods, headers and status codes used to coordinate the successful transfer of representations during an application's lifetime.

Conclusion

As should be clear by now, in a RESTful distributed application there's no one authoritative entity overseeing the progress of the application—or if there is, it's the client, not the server. In making the client responsible for maintaining application state, we facilitate breaking the application's problem space up into many independent, easily tested, uniquely identifiable resource state machines, each of which is responsible solely for its own simple lifecycle. In following the signposts—the links and forms—to be found in resource representations, the client drives the hypermedia system, and so progressively changes the state of the distributed application.

Figure 2. A client interacts with a set of resources to achieve its application goal

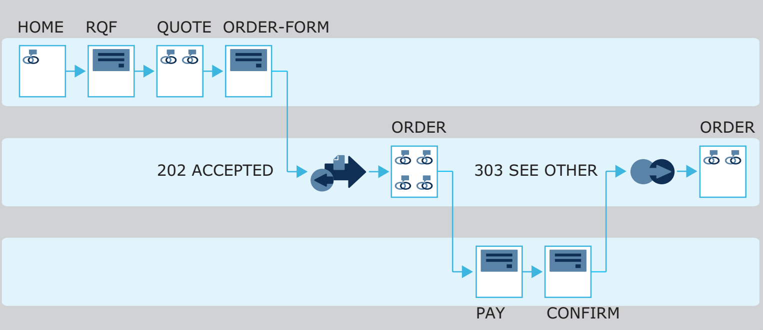
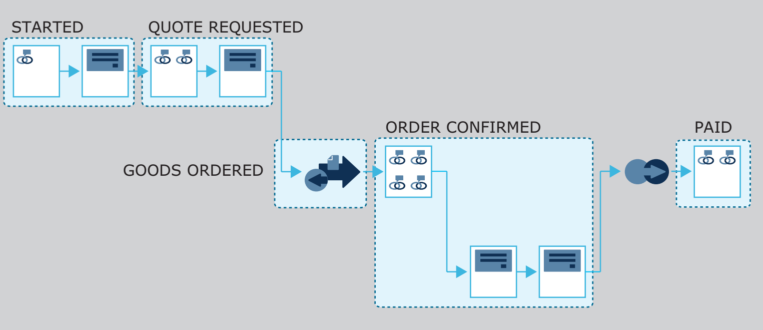


Figure 3. A client's view of the changes in the state of the application



Who needs a service bus, anyway?

by Udi Dahan

Although Enterprise Service Buses have been used in many larger companies, small and medium enterprises have often been put off by the high cost of these large middleware packages. These days we're seeing more open-source service buses gaining popularity and many developers are beginning to get curious - what would I use it for? What's wrong with keeping on using the same web services as before?

Web service reliability issues

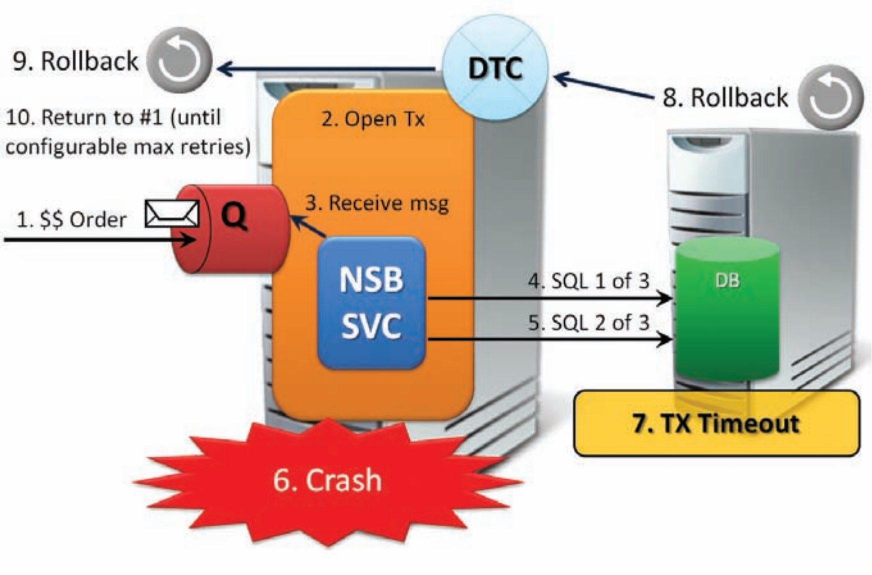
One of the primary things lacking in standard web-service-based communication is reliability. When invoking a web service over HTTP, if the client receives a timeout exception, there's no real way for it to differentiate between the case of the server processing the request successfully but that the response timed out and the case of the server not processing the request successfully (possibly due to the web server recycling at that time).

While a retry model could be useful to compensate for these kinds of problems when the client is performing a query, when it comes to submitting new data into the system (a command), we want to be very careful about not creating duplicates. While the client could try to query before retrying the command, often these queries are served off of somewhat stale caches (because going to the database for every query wouldn't scale), so a response telling the client that the data doesn't exist is no real guarantee.

In short, the client can't take responsibility for getting their data into the system, and servers, which may restart at any point in time, can't take that responsibility either - at least, not with standard HTTP web services.

Queuing and service buses to the rescue

The only way to guarantee reliability is to separate the delivery of data from the processing of data - and this is what queuing infrastructure provides. This infrastructure mediates the delivery of the command from client to server, writing it to durable and transactional storage. So if there's any kind of problem on the server while processing the command, the message can be rolled back and tried again - just like a database. This process is illustrated in the figure below.



It is the responsibility of the service bus infrastructure to manage the transactions around the queuing infrastructure, as well as to guarantee that a message whose processing fails constantly doesn't clog up the queue. This is handled by moving failed messages to some other queue where they can be examined (and corrected) by administrators before being returned to the source queue to be processed again. The failure may not have anything to do with the message - the database might be down, for example.

Architectural ramifications

The introduction of one-way fire-and-forget communication via a service bus has deep architectural ramifications. For example, if the client wants to send two messages one after the other like CreateParent and CreateChildOfParent, then the client would need to have the parent ID in order to send the second message. In a traditional architecture, this would require the client to wait for the success of the first message, so that the ID of the parent from the database could be used. When using a service bus, we'd change our ID strategy to one that can be generated client-side, like a GUID/UUID, which would impact our database design as well.

Queries wouldn't be well served by this fire-and-forget communication, but that doesn't mean we need to go back to traditional 3-tier approaches there. Instead, we could have servers publish events using the service bus to subscribed clients which would then keep their caches continuously up to date.

Also, we'd prefer much stronger client-side validation so that commands that arrive at the server will succeed most of the time. We could run business rules on the client using data from our almost-fresh cache so that users get immediate feedback on most errors.

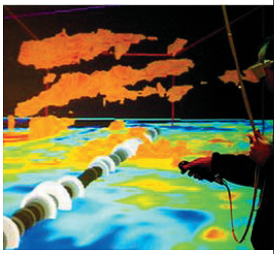
Summary

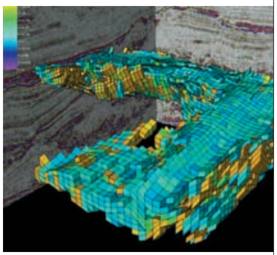
Service buses are not a straight-replacement of web service communication. On the one hand, they provide for much more robust and reliable message processing, but on the other, they lead us to a very different style of architecture - one that is more asynchronous and parallel, and therefore more scalable.

If you're a .NET developer and would like to see what using a service bus can be like, try out NServiceBus (www.NServiceBus.com), the most popular open-source service bus for .NET.

For more information on the architectural approach outlined above, see my blog post on CQRS.

Service buses are not a straight-replacement of web service communication





The birth of SpecFlow:

Chronicles of an open source project

by Jonas Bandi & Gaspar Nagy

SpecFlow is a prospering open-source project for .NET / Mono that was founded one and a half a years ago, and has attracted quite a lot of attention in the .NET community since. In this short self-interview, we talk about SpecFlow's birth and the challenges we faced providing it as an open-source project.

Jonas: *"Cucumber offers a lot of potential for Agile development."* This sentence started the SpecFlow story for me.

Christian Hassa, one of the managing partners at TechTalk said this to me one evening in Spring 2009 in a hotel lobby in Zürich. This was after my first job interview with TechTalk in their new office in Switzerland. After the interview, we kept on talking about our ideas and visions of Agile development and software testing. At the time, I was not familiar with Cucumber. I had seen demonstrations of RSpec Story Runner before, but this was like black magic to me at this point.

Gaspar: Cucumber was our third attempt to do specification-based functional testing and BDD. We were already beyond using annotated Selenium tables and building a proprietary language. Cucumber was helping us really concentrate on the specifications, while still being able to automate the application easily.

Jonas: After the interview I went home and eagerly began exploring Cucumber on the web. What I found was very interesting...

Fast forward five months. I decided to join TechTalk. My first real contact with the company was a company event in Sopron, Hungary. I arrived late at the hotel. Everybody was already well fed by a gigantic barbecue, and beers and drinks were circulating.

My SpecFlow story continued, again in a hotel lobby:

Attila, a developer whose mighty appearance honors his name, was kneeling in front of a salon table where he had placed his laptop and his beer. He was arguing fervently with a group of TechTalkers about using Cucumber for .NET development. He was typing fast in Vim and showing the output of his Ruby code in the console.

Most of the spectators standing around kneeling Attila were not really in the mood for discussing deep technical stuff, as they were much more interested in their drinks. But Gaspar was arguing deeply with Attila.

The main arguments concerned the accidental complexity and slow execution time that is added by using Cucumber under IronRuby for common .NET development.

I was fascinated by the depth of the argumentation between Gaspar, Attila and the other TechTalkers. Of course the discussions went on until deep into the night.

Gaspar: And not only the discussion... I had a technical concept ready by the end of the event.

Jonas: I returned to Switzerland. A week passed. Then Gaspar came forward with a first implementation of a tool that took Cucumber features and turned them into natively executable .NET code. Already integrated into Visual Studio!

SpecFlow was born... only we did not yet know the name of our baby.

Since then I have learned that this is Gaspar's very pragmatic way:

If he doesn't like something, he doesn't argue too much about it; but he works up a better solution.

Gaspar: Actually we had two choices: either we fail with BDD in this project because of the issues or I put together something very quickly (maybe I'm really pragmatic...). An interesting detail is that the very first version had its own parser, using the Oslo M grammar. And now we are in Oslo presenting SpecFlow – what a coincidence.

Jonas: For a month or two we tested the new tool on our own projects.

We were improving it and of course we were arguing...

Soon we decided that our new baby was ready to be shown to the world.

We thought we wanted to name it NCucumber, going with the naming fashion for .NET clones of projects from other platforms.

We contacted Aslak, the father of Cucumber, but he did not permit us to use the name for something that was not really grown from the real Cucumber seed.

At the time, there were several .NET Cucumber clones sprouting up, and Aslak was not fond of the idea of them using Cucumber's well-earned name .

Proud creators ourselves, we went on searching for an appropriate name for our beautiful baby.

We did not want the word "test" in it, because the BDD workflow is not about testing but about building a shared understanding.

Candidates were "ZinC – Zucchini is not Cucumber", "ScenarioSpec", "SpecNarrator" and "SpecFlow".

Finally the word Spec and the smooth motion of flowing (compared to the abrupt waterfall) seemed to fit our ideas perfectly... and the specflow.org domain was still available.

Finally we presented SpecFlow to the world: on October 22nd 2009 we committed SpecFlow on GitHub.

Gaspar: This was my first OSS project and it was a very strange feeling to put the source code online: being naked; putting up the pages of your diary on the wall – somehow like this. I did not know what would happen; how the contributors would join, whether anyone would abuse our achievements. It took me some time to get used to this, but I'm fine now.

Jonas: It seemed that we hit a nerve: very quickly we got quite a lot of attention. Only a few days after our first release, we got several interested users who contacted us with the intention of helping us grow the project.

However we found ourselves trapped between several other fresh open-source .NET BDD tools and the already major Cucumber project itself.

How should we proceed? Where should we look for alliances? Who were our competitors?

Gaspar: Especially interesting in this regard is the evolution of the Gherkin parser in SpecFlow. Gherkin is the plaintext language ("Given-When-Then") that Cucumber defined for writing specifications. SpecFlow had its own infrastructure for parsing the files (first with M Grammar and then with ANTLR), but remaining compatible with the "official" Cucumber Gherkin was hard. Other tools in the Cucumber-family had similar issues, and Aslak realized that there is a lot of value in the specification language itself (even without the Cucumber runner), so he gave a name for it (Gherkin) and extracted it from the core Cucumber project. The new "official" Gherkin parser supports several languages and provides a solid foundation for Cucumber-related tools. An interesting detail is that the Gherkin parser supports .NET through an OSS Java/.NET bridge called IKVM. We jumped on this opportunity and retired our own parser.

Jonas: We realized that long-term commitment was the key to success for an open source project. Cucumber demonstrated this commitment and we could build on that if we stayed close to Cucumber. We tried not to compete with other .NET BDD tools, and just concentrated on our own goals.

Another important aspect was community building. In the beginning we did not know where SpecFlow was heading: we had no experience in leading an open source project and we were not even familiar with the BDD community.

We were quickly able to establish good relations with some of the proponents of the Agile testing community in London, where most of the BDD mindset was coming from. Getting familiar with ideas and concepts like BDD, ATDD, specification by example and outside-in development was crucial for us to give the project a real vision. The focus on the theoretical foundations on which SpecFlow is built is visible in the SpecFlow Google Group, where discussions about BDD concepts and best practices outnumber the purely technical questions about SpecFlow.

Gaspar:
I'm always amazed by the level of discussions on the forum. When we started SpecFlow, I was afraid that all my time would be taken up answering support questions. Instead, it is rather a joy to read the forum posts, and most of the pure support questions are answered by somebody before I even see them.

Jonas: A positive side effect of our focus on the methodology was that we got more attention from the community. We were not perceived as "yet another tool" but as a serious proponent of the BDD mindset. Consequently we got mentioned several times on prominent sites like InfoQ and MSDN and we started to get invited to conferences...

Gaspar:
I cannot count the workshops, meetups and presentations anymore where SpecFlow was shown. But it is clear that NDC is an important milestone in SpecFlow's short life and it is a great honor to be here.

Gaspar and Jonas want to thank all the SpecFlow contributors. Special thanks go to Xerxes Battiwalla, Dale Ragan, Darren Cauthon and Marcus Hammarberg for their continuous effort. SpecFlow is sponsored and supported by TechTalk.

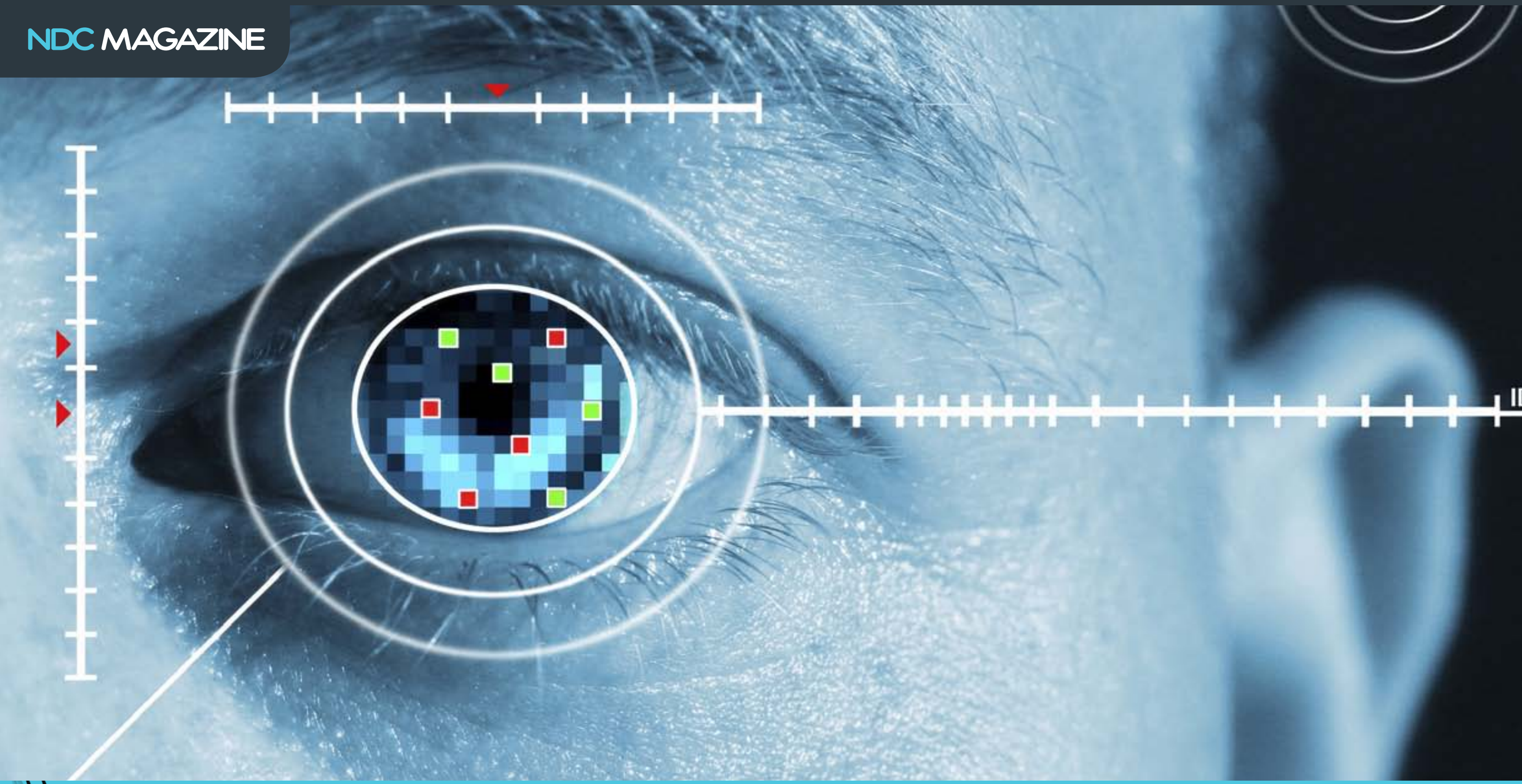
■

SpecFlow can be found at www.specflow.org and at http://github.com/techtalk/SpecFlow/

Gaspar and Jonas work for TechTalk (www.techtalk.at). They are the maintainers of the SpecFlow project.

You can find their blogs at http://gasparnagy.blogspot.com/ and http://blog.jonasbandi.net/

Twitter at http://twitter.com/gasparnagy and http://twitter.com/jbandi



The Windows Identity Foundation

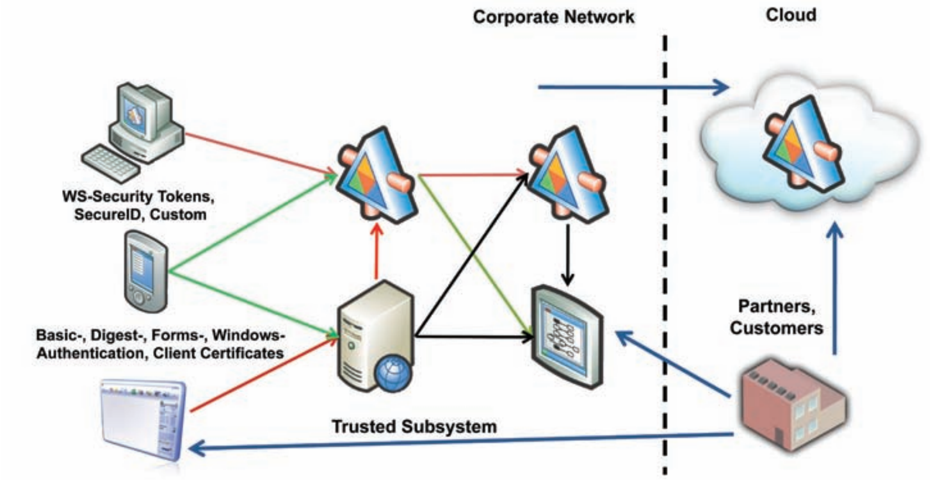
- The new Way to secure Web Applications and Services

by Dominick Baier

Security is pivotal for most distributed application scenarios. This applies to "normal" intranet applications, collaboration systems between partners and customers and ultimately cloud computing. After almost ten years of .NET, Microsoft updates its APIs to model distributed security and promotes claims-based identity to a first class citizen in the .NET/ Windows security ecosystem.

Where are we?

Even simple distributed applications have an enormous complexity these days: multiple client types (web, mobile, desktop) using different credential types (usernames, X.509 certificates, SAML tokens, Windows accounts) speaking different authentication protocols (HTTP authentication, SSL, WS-Security, Kerberos) talking to various backend services (web applications, SOAP & REST services, workflows and databases). These scenarios become even more interesting as soon as there is need to provision users to these applications that don't originate from the same security domain. Standard application hosting and outsourcing up to federation with partners and customer as well as cloud computing comes to mind here.

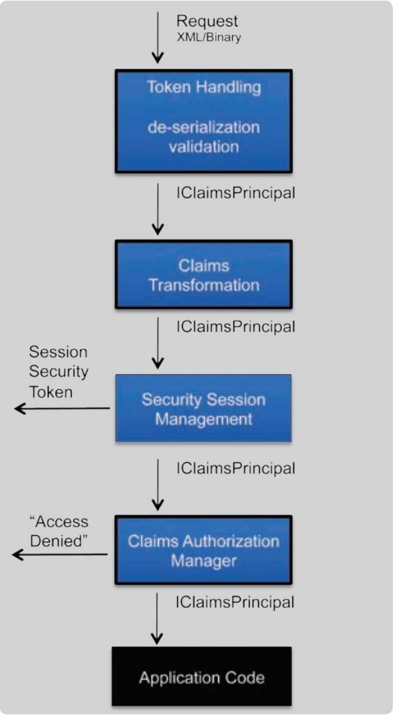

Corporate Network
Cloud
WS-Security Tokens, SecureID, Custom
Basic-, Digest-, Forms-, Windows-Authentication, Client Certificates
Partners, Customers
Trusted Subsystem

Unfortunately the .NET Framework, and especially the Base Class Library, only had limited built-in support to model such security scenarios and requirements. That led to an enormous amount of (error prone) custom code and home-grown implementations of security vital and complex protocols.
Up until November 2009 where Microsoft released the first version of the Windows Identity Foundation - a base API and framework that brings .NET applications up to speed with emerging and already established standards and protocols that allow implementing above scenarios in a standards compliant way and with much less friction and complexity.

The Windows Identity Foundation

WIF is a base API as well as a framework for bringing new authentication and federation protocols along with the claims-based identity paradigm to .NET applications. It has built-in support for protocols like WS-Trust and WS-Federation and the SAML token type. Upcoming updates will add support for the SAML 2.0p protocol family as well as OAuth and their corresponding toke types. While the core APIs allow to add that functionality to arbitrary applications, it comes with an easy to use out of the box integration for ASP.NET and WCF.

One of the major design goals was to simplify and unify the security programming model of both application frameworks. When you enable WIF in your applications, it wires up a common pipeline into the runtimes that gives you important extensibility points for separating your business logic from the actual low level security and protocol details.



The ultimate mission of the WIF pipeline is to turn a credential (or security token in security speak) into a collection of claims. These claims are packaged up in a new principal type called IClaimsPrincipal. Regardless of the credential and authentication protocol details, application code will always program against the IClaimsPrincipal type. That gives you application a good abstraction from the ever changing security details.
The following paragraphs summarize the most important steps in this pipeline.

Token Handling

In this stage the raw security tokens on the wire get turned into an IClaimsPrincipal. WIF's extensible security token handler infrastructure takes care of that.

Claims Transformation

The claims from the preceding stage a very "technical" and don't necessary reflect the data needed by your business domain. In this stage you have the chance to transform the incoming claims to whatever makes sense for the application at hand. This is a very important step in establishing a client identity the business logic can rely on.

Claims-based Authorization

The standard .NET IsInRole infrastructure is useful for coarse grained authorization but has some shortcoming. First it encourages you to mix business logic with authorization code which makes code hard to maintain. Next - the power of claims enables much more powerful authorization scenarios than simple true/false results. WIF's ClaimsAuthorizationManager allows for more complex authorization policies and encourages much better separation of authorization enforcement from business code.

Summary

In short - you want to use WIF when it comes to any authentication/authorization/personalization related task in distributed applications. It is the new .NET security system for these concerns and will gradually trickle into all major Microsoft products (e.g. Active Directory Federation Services 2, SharePoint 2010, CRM and Reporting Services are now all based on WIF). If you have any questions regarding WIF or distributed security in general feel free to contact me via:

dominick.baier@thinktecture.com
or have a look at my blog at
http://www.leastprivilege.com.

DX Video Rental
Rentals
DX Video Rental
North
DXGrid
DXPivotGrid

Café Del NDC

by Anders Norås

"It's a group of islands. It's sand in your foreskin but not caring. It's Clark's comfy shoes. It's corduroy. It's a lazy place. It's warm. It's Wellington boots. It's knowledge. It's Moonboots. It's carpet not laminate. It's Van Halen not Europe. It's council not Hilton. It's Jason Boardman. It's borrowing not buying. It's me, it's you, it's everybody. It's bollocks. It's great.",
these are the words of Barry Devan describing what Ibiza was all about. The isle, which still has a sort of mythical hold in most peoples imagination, has had massive influence on many perfect summer soundtracks throughout the years. Café Del Mar is better known as well-formatted, commercialized chill-out compilation albums than a bar in San Antoni de Portmany.

The island is a tourist magnet with as many Britons on benders as seagull-screams on all CDs in the series combined. Neo-hippies in search of a bohemian heyday on their very own Garlandian beach fled the labyrinth streets of Elvissa and the famed hippie market of Sant Carles for more exotic destinations in Congo or along trails only reachable by camelback in Southern Turkey.

Ibiza is a tolerant place, it has to be having endured so many visitors of thousands of years. It got its name from the Phoenicians god of safety, dance and protection, and it has been ruled by Romans, Vandals, Arabs and Catalans. In the nineteenth century, Jews fleeing persecution, Americans fleeing the Nam draft, fugitives from justice and injustice along with the hippies and jet-set found their haven on this Balearic isle. From this odd mix of people sprung a more or less self-governed community far away from the packaged tours destined to San Antonio. Inspired by the seminal hippie movie "More", shot at location in northern Ibiza and featuring a Pink Floyd soundtrack, bohemians flocked to the northern parts of Ibiza where they could hang out in the open. Drawing on the legacy of previous settlers they formed a commune sharing interests, property, knowledge and resources.

"It's lazy, yet intense ... It's undercurrents not mainstream "

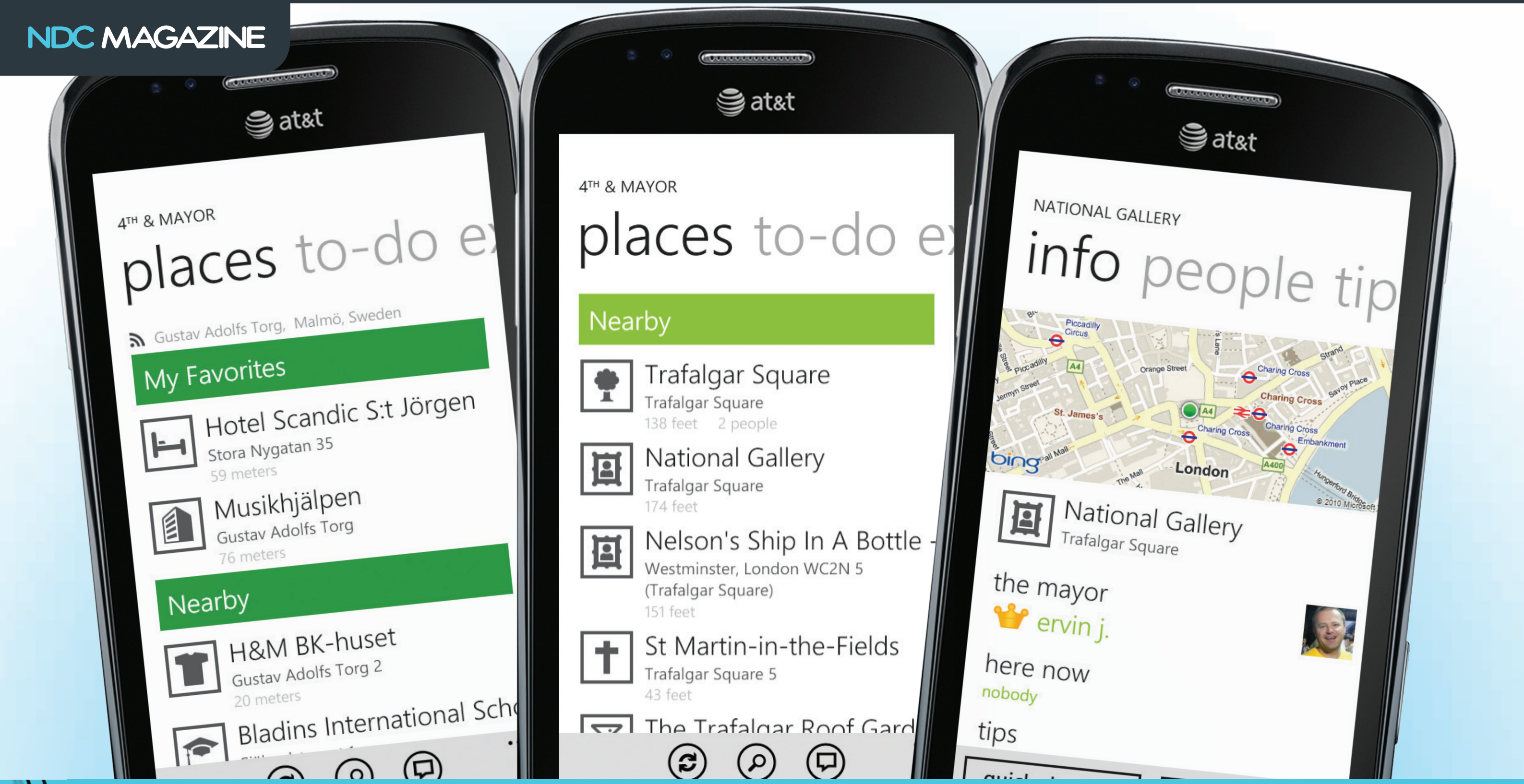
Roots of Ibizan chill-out, known as Balearic Beat, was a mish-mash of Italian disco mixed with indie guitar rock, pop twelve inches by Madonna and Phil Collins, quirky jazz and reggae numbers amongst other oddities. Music connoisseurs still argue whether this was a genre of its own or simply a jumble sale of crap masquerading as gold.

Regardless of discussions, it is inevitable that a few free-thinking minds who got together in Elvissa during the nineteen seventies has changed clubland ever after.

Experiencing the blend of progressive and established ideas, commercial and open source software, alongside up-and-coming thinkers and thought leaders present at the Norwegian Developers Conference gives me a feeling of being part of something big that shapes the future of the software industry. The people getting together here form an international community of forward-thinkers who share their passion for challenging status-quo, never resting on laurels. The eclectic blend of technologies presented is refreshing, creating a breeding-ground for platform and culture cross-pollination yielding results far more interesting than formulated spa-music.

When the doors open for a summer themed conference in June, developer communities from all over the world will join the Norwegian community at their every own Café Del NDC. I hope to witness heated discussions on the coolness of vintage code, just like JavaScript has become a programmer's answer to Phil Collins. And even if I don't, I'll still be one happy camper.

"It's a bunch of tracks. It's enterprisyness in your agile but not caring. It's vintage sneakers. It's denim. It's lazy, yet intense. It's warm. It's vendor merchandise. It's knowledge. It's Scott Guthrie. It's uncomfy seating and comfy beanbags. It's undercurrents not mainstream. It's Oslo Spectrum not Vegas. It's Scott Bellware, It's sharing and engaging. It's me, it's you, it's everybody. It's great. It's NDC 2011"



Passion. Functionality. Design.

Help your Windows Phone app shine.

by Jeff Wilcox

As of writing this, my app is the top-rated app in the Windows Phone Marketplace. I know it won't stay there forever, and I need to innovate and continue delivering continuous updates to keep my user-base happy. However, I do know of 3 distinct reasons that have contributed to an average rating of 5 stars by over 1000 users, and thought these would be worth going into some detail on – maybe you can improve your WP7 app's reception as well. The categories are:

- *Concept, Market Timing, Pricing*
- *Delivering first and foremost on functionality*
- *Very strong emphasis on the Windows Phone "metro" design*

The app I am talking about is simply a client for the popular foursquare social check-in service, but building such an involved app has taught me a lot about building an app, the platform, and what matters to end users. Eye-opening.

Concept, Market Timing, Pricing.

The first category is honestly a little luck: as a passionate user, I started building my own foursquare client in December of 2010, mostly to learn about the platform I was a part of at Microsoft. Over the long vacation, I built almost all the core functionality, learning about the good and bad of the dev platform. At the time I didn't really think it would end up getting all the buzz it has received – and it's clear the buzz is a direct result of my passion into the project, the code, the branding, everything all wrapped up together. I built the experience I wanted as a user, and clearly it's appreciated.

I shared early builds with friends, came up with a creative name, but didn't honestly plan on shipping it at all. However, just at the start of January, the foursquare service extended the iPhone and Android apps quite a bit, with new social features like commenting and photos. I took the few hours to add these to my app, and again, didn't plan on shipping anything beyond my circle of friends.

You'll never know if it is the right time to ship an app, but it's possible that you can get the right buzz going – I have found Twitter to be a very powerful connector.

I soon realized that I had a valuable program and focused on adding all the functionality that the iPhone and Android had ... something that Foursquare had not yet gotten to yet for the Windows Phone. I decided that I would offer my app as a "trial", and for paying users (maybe $1.99 US?), I'd add the extra goodies – commenting, photos, etc.

Flash forward to late March 2011, and some sort of login issue caused the official app to stop working for a lot of people. My app wasn't completely ready, but I saw a market opportunity, a need to help fellow foursquare users, and in an amazing moment, I decided: it will be FREE! I figured it would be a good learning experience, sharing the app with users everywhere at no cost.

Basic economics on the marketplace are probably an interesting study, but making it free removed any and all barrier to using the app, immediately skyrocketing downloads in the first day. Over 40,000 people daily have been using my app, and I'm convinced that had I offered a paid-trial, or pay-only app, that I would not have this strong of a user base within days.

I can always figure out monetizing the thing later. In the meantime, I'm getting valuable feedback, learning about my users, and interacting with other developers who have done well on the platform.

Deliver first and foremost on functionality

Most users start with an expectation that your application will provide a positive net gain to them, so it must deliver functionality. Bugs that impact core functionality are inexcusable, and regardless of any pending update, you can safely assume that you may lose the user forever if you regress significant functionality.

Deliver on the promise of good functionality: clearly call out core functionality in your app's description. Respect your users by keeping the app up, around, alive, and not crashing. Remember to polish with the right tombstoning code, and they'll reward you with great reviews: "the app just works!", "good features, everything worked", vs. "feature X did not work", "feature Y crashes".

You can always iterate on your work, adding functionality, but shipping with broken functionality is a negative connotation that will be a challenge to rectify in the future.

If you are building on existing concepts and apps, you really must match functionality, and the only differentiator will be bugs, errors, omissions, and yeah the occasional value-add.

Strong emphasis on Windows Phone "metro" design

The other way to differentiate on the marketplace really is in design. Windows Phone is pretty easy to design for. The platform enables a great compromise between developers and designers: by focusing on a simple grid system, rich text, and deemphasizing other concepts like gradients and chrome, developers can often get the "metro" effect done right within a few tries.

Here's what all developers need to know, regardless of developer experience:

> *The magic number is 12px. Developers love rules, and so that's the rule: remember to align everything, and look for off-by-12px errors.*

You should always be using the official phone styles for text blocks: PhoneTextNormalStyle, PhoneTextLargeStyle, these are designed to incorporate the 12px horizontal margins, and so if you don't use these styles, your margins will be off and things will not be properly aligned – and users can easily spot what "doesn't fit" in the Windows Phone world. You do not want that to be your app!

A large part of building a "beautiful" Windows Phone app is about stepping back, focusing on the basics, not cluttering the screen, and trying to deliver a simple, consistent story to your end users.

I'd also suggest that you try not to fall into the black & white trap: use the PhoneAccentBrush when appropriate to add some color to the experience, and the side benefit is, the color is the user's preference, so there's a nice attachment there.

So what's that emphasis on design buy you? It makes even the most independent sole developer offer a professional-looking, Windows Phone-themed experience, it's really something worth celebrating.

So in closing, I'm really talking about polish. Bring out the passion you have for your product as a developer, and do that by sharing your passion about your work. Ensure the functionality is everything you would expect as a user. And focus on the Windows Phone design, as that's something everyone will expect.

Good luck on the Marketplace!

The app I'm talking about is called 4th & Mayor, and it's my foursquare™ app for Windows Phone. More info at *www.4thandmayor.com.*

Moving from Possible to Easy with Design Principles

by Billy Hollis

As software developers, our first responsibility is to provide the capabilities that users need. Without relevant functionality, software is worthless.

I wrote applications in the 1970s that would look comically primitive today, but they looked good compared to index cards and manila folders. Of course, we've seen some progress. We started talking about "user friendly" software in the late 1980s, and developed an entire discipline for usability.

The average software developer, though, is still more concerned with making tasks possible than with making them easy. Even in companies producing software for sale as a package, in which ease of use should take a high priority, it's common to see badly designed user interfaces.

In December of 2010, I reviewed about a dozen medical packages at a conference. Perhaps nine or ten of them had a screen with a common problem: a large block of equally sized buttons, with seemingly random icons on them. Combining and generalizing what I saw in those packages, I created a sample screen in Figure 1 that shows the problem.

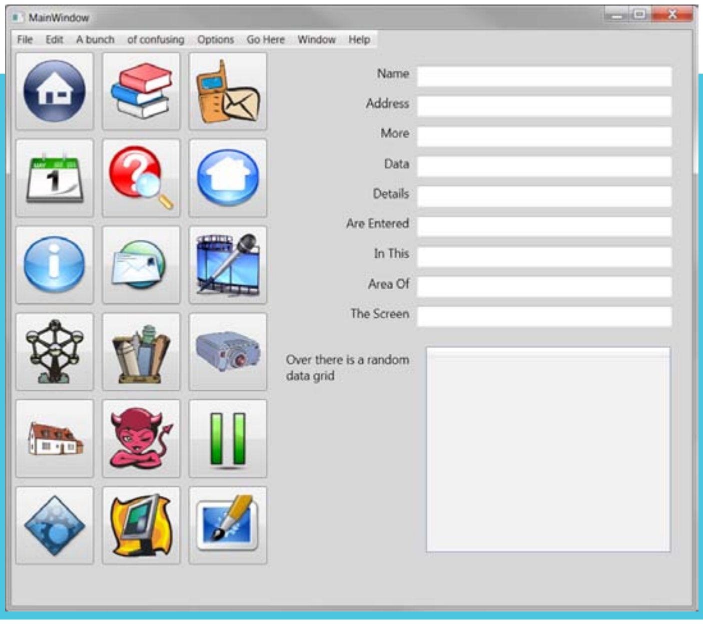


Figure 1.

In those applications, every one of their buttons is on the screen for a reason. The users needed a new function, the developer created the function, and then a button was added to access the new function. Repeat this enough, and you end up with the screen in Figure 1.

> At this point, I'd like you to stop reading for a moment, and write down what you think is wrong with the screen in Figure 1

Done? You probably wrote down something along the lines of "confusing" or "too crowded", and those are certainly apt descriptions. However, those observations don't really help us know exactly where the screen went wrong and what to do about it.

To do that, it helps to understand the general area of design principles. Just as we use design patterns in our code to help organize it effectively, we can use design principles to gain a deeper understanding of the factors that make a user interface effective.

For the screen in Figure 1, I can name several design principles that help us understand what is wrong with it, and suggest what to do about making it better. Let's look at three in brief, and see how each one suggests improvements.

The 80/20 Rule

The 80/20 Rule, also known as the Pareto Principle, describes a tendency in real world statistical distribution. For example, in many companies, roughly 20% of the users cause roughly 80% of the issues. In software systems, it's common for 20% of the features of an application to provide 80% of the value and be used 80% of the time. If that's true for the applications I looked at, then only about 20% of the buttons are needed on a routine basis. The others are certainly used, but they are probably used far less often.

This suggests that a better design would emphasize those commonly used buttons, and hide or move the others. A common design for that purpose is the "Advanced" option to expand a screen area and show less commonly used capabilities.

Fitt's Law

The design principle of Fitt's Law summarizes key factors concerning how long it takes a user to select an option. For example, the amount of time required for a user to locate and use an option is smaller if the option is larger.

Sounds obvious, right? It is, but this principle is violated regularly in user interfaces. Fitt's Law suggests that commonly used buttons should be larger, and there is no technical limitation preventing larger buttons for some options. Yet I rarely see user interfaces with differently-size buttons.

The consistency and symmetry of equally-sized buttons is also a factor, and that's probably why developers often use them. However, design principles are not absolutes, and sometimes they trade off against one another. The cosmetic appeal of equally-sized buttons is often outweighed by ease of use for different sizes. I think this design could benefit from larger buttons for common options.

Hick's Law

Hick's Law is perhaps the most important one I will discuss for this case. Hick's Law says that the amount of time to locate and use an option is larger if the user must choose from many options.

To state it another way, there is a price every time you add a new option to an existing group of options. This price is not often considered. Increasing the number of options slows down the user, and makes all the other options harder to use.

This is an even stronger argument that the commonly used buttons should be prominent, and the others should be hidden in an advanced area.

Learning More About Design Principles

An excellent book to start investigating design principles is Universal Principles of Design, by William Lidwell, Kritina Holden, and Jill Butler. This book summarizes about 125 design principles in one page each, and then contains another page for each one showing examples of the principle in use or the principle being violated. Some principles do not apply to software, but most of them do.

I'll be doing a session on Fundamental Design Principles for UI Developers at NDC2011, and I hope to see you there. If you want a more in-depth treatment, I'll also be back in Norway this fall to teach a class on design principles and advanced topics for XAML.

NDC-the motivational conference

by Linn Therese Skulstad

For the fourth consecutive year, UNIT4 AGRESSO is attending the Norwegian Developers Conference in Oslo Spektrum. According to the participants, this is a seminar that offers a wide range and is professionally conducted.

Arve Hansen, Section Manager UNIT4 Research and Development

"It is inspiring to attend this seminar. It's a seminar that keeps a wide range by focusing on different themes. It is not just Microsoft that is in focus, but we get other inputs as well," says Arve Hansen, section manager of UNIT4 Research and Development.

UNIT4 Agresso AS delivers ERP system, financial systems and HR System for Payroll and Personnel with corresponding adviser and costumer services to the Norwegian market. Together with its partners, the company serves approximately 800 customers in Norway. The development of Agresso Business World happens in its sister company UNIT4 Research & Development AS, which is participating in the Norwegian Developers Conference. It is the fourth year in a row that UNIT4 R&D is involved, and they are once again ready to be motivated and engaged.

Professionally conducted

For Hansen and the rest of the staff, the seminar is something they have great enjoyment from and they are very happy with the arrangements. According to them, it is professionally conducted:
"In general it's a great seminar. But if I have to mention three things that stand out, and which are good, the first would be the fact that they invite well known international speakers to the seminar. The second thing is that we are free to buy tickets and decide which days we want to attend. If required we can attend all three, or we can juggle the tickets as we want. We can pick and choose, which is an advantage. The third thing is that it's a good professional performance and there are good technical solutions," says Hansen.

> "For UNIT4 R&D, attending the seminar is an energy boost where they get value for their money and they can come across new things and remain inspired."

"The benefit is that we get a better look at and overview of new technologies. It is important to get a wider perspective. In addition, we learn much about the actual development process. And then there's the good entertainment spread over three days. In the middle, there is also a large party, including dinner and a band," he says with a smile.

From motivation to pride

“Another thing is that it’s a kick-off for us to join the conference. People sense the pride in their work and profession by being motivated into using the technologies presented. A motivation that is catching,” says Erik Marcussen, chief software engineer at UNIT4 R&D.

The level the software company wants to achieve is to gain a technology that is easy to maintain. And since technology is constantly changing, and you need good discipline to produce, NDC is an excellent seminar that shares their level.

“At this seminar we get suggestions on where to learn more and how to do software development correctly. Software development is a young profession, and you do not really know what works best at all times, but the NDC shares examples, ideas, recommendations and experiences. It is difficult to develop and maintain software, but NDC points us in the right direction,” says Hansen

Bigger participation

For UNIT4 D&R, it is not always easy to participate in seminars worldwide. Such things are often a financial cost, and unfortunately it is too expensive to send all 130 employees to another country. The fact that the NDC is located in Norway is a big plus for Hansen and the rest of the staff.

“Unfortunately, it’s true that we have a focus on economy and the cost of transporting the whole team to attend a seminar abroad. The fact that NDC is located in Oslo means that more people can participate, something we are very happy with.” says Hansen.

> *“We get as much benefit from NDC as we get from PDC or Tech-Ed, but a large seminar in Oslo for the fourth year in a row, means that even more people benefit from it.”*

The importance of C# 5's async support

by Jon Skeet

If you don't write client code and you don't write server-side code, you can stop reading now.

Still with me? I thought so. Okay, so it's probably an exaggeration to claim that *every* server, *every* client app will benefit from C# 5, but it's very likely that at *some* point you'll work on a project performing operations which could be handled asynchronously.

A little bit of history...

Over the last ten years or so, many people have written an awful lot about how to squeeze the most out of multicore machines, and we've all become reasonably comfortable with the idea of firing off background tasks to keep our user interfaces responsive, instead of resorting to calls to the reviled Application.DoEvents method.

Slightly less attention has been paid to performing general operations asynchronously – taking the idea of "some arbitrary piece of work" and only continuing with our own logic when that task has completed. That task may be taking place in the same application, somewhere else on the same machine, or perhaps on another machine on the network. For the most part, you don't really care – but you want to be able to *react* when the task has completed, or timed out, or failed.

It's not that Microsoft hasn't provided APIs to make this possible. First we had the BeginXXX and EndXXX pattern, exposed by classes such as WebRequest and Stream. Then we had the event-based asynchronous pattern, as implemented by classes such as WebClient. Each of these approaches had its pros and cons, but they were still fundamentally tricky to use properly.

The problem is with us, really – the developers. You see, over time we've become *just about competent* at writing synchronous code, on a good day with a bit of luck. But as soon as we start having to work asynchronously, our brains explode. At least, mine does – the code quickly resembles spaghetti, and the possibilities for getting it wrong are endless. At this point a good day consists of everything working in the *success* path... but trying to catch all the error possibilities without introducing race conditions becomes really tricky.

C# 5 to the rescue

Fortunately, the aspects of asynchrony which are really hard for *humans* to deal with properly are relatively straightforward for a *machine* to deal with... once you frame the question in the right way.

The trick is to think of a piece of code as a state machine, where the relevant state is "all the data we have access to" and "where the instruction pointer is". Normally we just move through this state machine in a way which computers find natural, executing one instruction after another, calling into sub-state-machines (other methods etc) and "finishing" when we reach a terminal state. We understand what can happen at each point, and it's easy to see how the control can flow.

The tricky bit is trying to work asynchronously. Essentially we want to be able to say: "When the current asynchronous task has completed, I want to get back to the same point in the state machine, so I can keep going with my logic." Unfortunately representing that "same point" is hard when writing the code by hand, and we *also* have to consider what to do if the asynchronous task fails.

Writing state machines manually is difficult and error-prone... but computers are really good at it. We've known that the C# compiler can handle state machines for years – iterator blocks (methods with "yield return" and/or "yield break" statements) have used that technique since C# 2.0. Add in a bit of useful behaviour (based on the Task Parallel Library from .NET 4.0) to allow the state machine to continue in an appropriate thread, and return a Task<T> to allow the *caller* of the state machine code to wait for it (potentially asynchronously), check it for completion and so on, and you've got a really powerful mechanism.

Essentially, C# 5 lets you write synchronous-looking methods which the compiler transforms into complicated state machines to work asynchronously. Anywhere you want to "wait" for a result but you don't want to block a thread, you simply use the new "await" expression, and let the compiler do the hard work. As a developer, you get the best of both worlds: the simplicity of the synchronous code flow, and the efficiency of asynchronous handling.

Async in client applications

Typically client applications want to offload CPU-intensive tasks to a different thread, and make other asynchronous tasks occur in the background *somehow*... while it's certainly feasible to wait for a web request synchronously in a background thread, it feels like a bit of a waste of a relatively valuable resource. The important thing is that when the asynchronous task *has* completed, we want to be able to display the results in the UI – which means getting back to the UI thread. In the past this has usually involved using BackgroundWorker, Control.Invoke/BeginInvoke or Dispatcher.Invoke/BeginInvoke, but again that can end up getting pretty messy... the resulting code often spends far more space describing the *machinery* of how we want the code to execute than the real logic.

When you write an async method in C# 5 which starts off in a UI thread, it will (by default) *continue* running on the UI thread... but while it's waiting for other tasks to complete, it won't block the UI thread from reacting to other events. When a task finishes (successfully or not), a continuation is placed on the UI message queue as normal, and your async method will continue from where it left off.

Async in server-side code

Personally, I'm a server-side man. This is mostly because I'm simply *awful* at writing user interfaces, but it also happens to be the area I've spent most of my career working in. These days, any server-side code worth its salt has to make calls to at least a few other services – for example authentication control, databases, and of course web services. Even just reading a large file from disk can be regarded as a service call in many ways. Some servers do little else *but* make calls to a few different services, combine the results in a simple way and return them to the client.

The "thread per request" model has kept developers in a relatively cosy situation for many years, but we've always been aware that it often means we can have huge numbers of threads just waiting for other things to happen... and using up memory while doing nothing really useful. Often service calls can be made in parallel, but even doing that has been *relatively* painful. The Task Parallel Library has helped to a large extent, but it's still not as simple as writing synchronous code. Language support for asynchronous methods should make it pretty easy to make service calls happen in the way we'd always have *liked* them to before, but were too afraid to try.

Conclusion

Everything above has been somewhat rose-tinted. I'm not going to pretend that C# 5's async support will magically mean you can get everything right without a *bit* of thought... but it makes it all considerably simpler. You can think of it as removing many of the mechanical-but-error-prone parts of asynchronous programming, leaving only the *inherent* difficulties which we'll never really be able to get away from.

Want to know more? Look at the Visual Studio Async home page (http://msdn.microsoft.com/vstudio/async) or come to my sessions at NDC!

IntelliNonsense

by Scott Bellware

I ran into a former co-worker a number of years ago at an electronics shop in Austin. This was not long before .NET was first released in 2001. We had both worked at a FoxPro shop in Austin in the late 90's.

We quickly caught each other up on what we had been up to. My friend had seen the writing on the wall and shifted his skillset and career from FoxPro toward Visual Basic. I had spent the previous year working for a first generation dot com startup in Silicon Valley. We were a Java shop. I worked in professional services building customer apps with out (at the time) innovative Javascript APIs, and doing some tinkering with Java. I was still fond of FoxPro, but was relieved to finally be working with "real" programming languages.

After catching up, the inevitable geek bravado could no longer be kept at bay, and the inevitable pissing contest ensued:

"I hear that FoxPro might finally get IntelliSense," he jabbed. *"Really?"* I replied wryly, sensing an impending master stroke, *"I hear that Visual Basic might finally get objects."*

Expecting him to fall down clutching his temples like the victim of an indefensible psychic assault, he just stood there with a triumphant look on his face, apparently oblivious to having just been utterly decimated at his own game. A rude awakening for me. One that would be repeated many times in the many years of Microsoft community work that followed.

In the summer of 2006, many of the Smalltalk-come-Java developers who had introduced the first wave of .NET adopters to Agile Development had begun to call attention to Ruby on Rails. A handful of us who had begun our journey in C# in 2001 - and who had been students of these guys - took notice of the change of subject. We began to look into Rails as well since our teachers hadn't steered us wrong yet.

Slowly but surely, over a period of a couple of years, I went from tinkering with Rails on off-hours to working full time as a Rails developer. My historical self would have found it hard to imagine: years of full-time web programming work without the "productivity" of IntelliSense or static typing. And yet I've never been more productive. I wouldn't have believed it had I not experienced it for myself. But then, direct experience of something "counter-intuitive" is the only way to get past "counter" and move on to the "intuitive".

In the past, I'd not stopped to wonder if IntelliSense was part of my productivity equation. It just wasn't something to question. IntelliSense was so entwined with the notion of productivity, that it didn't occur to me to wonder about it. And it certainly hadn't occurred to me to challenge it outright. It's not an issue of bravado or geek heroism to code without IntelliSense. There are platforms where IntelliSense makes sense, or is even a necessity. And there are platforms where it's not a necessity; that make other tradeoffs.

Clarity is the common denominator in platforms where tools like IntelliSense are not seen as a necessity; clarity in the code, clarity in the design, clarity in the architecture. Everything is knowable at a glance - to everyone involved. It's a usability concern; something that developers are often far too removed from to notice when its imperatives go unfulfilled.

Even small increases in clarity have great cumulative effects on productivity. An outsider looking in on development in Ruby, for example, might dismiss the clarity advantages inherent in Ruby as "negligible". And yet, .NET developers who have moved to Ruby and Rails and who have made the effort to have direct experience will tell the same story: it turns out that IntelliSense wasn't as much of a big deal as we once thought. Not from the perspective of productivity, anyway - not once a larger spectrum of productivity factors are taken into account.

Try this thought experiment: if IntelliSense offers you one unit of productivity, and another environment without IntelliSense offers you five units of productivity, and if the learning cost wasn't a factor, would you stick with IntelliSense?

The reality is that there are learning costs, but not insurmountable ones. The point, though, isn't that you should go platform hopping. The most salient point here is that some of the strongest justifications offered for .NET today can be readily challenged by more contemporary platforms, even if - or perhaps especially if - they seem counter-intuitive to your current perspective.

I don't know what my friend from the electronics shop does these days. Like many work relationships, we fell out of touch as our paths diverged. I look back on that exchange and wonder about where he is in his career. He's probably doing well for himself in the industry, after all there are opportunities for a great many different kinds of software people irrespective of their perspectives. But I wonder about an industry at large where the majority of developers know the difference between the productivity that can be had from tool assists, and the productivity that can be had by recognizing which subtle, seemingly-negligible decisions will create the obstructions that leave us few choices beyond tool assists.

The most powerful tool of all in a software developer's arsenal is an in-depth understanding of the relationship between productivity and design. Learning costs are indeed a factor, but the organizations where you'll find people pursuing a mastery of software development productivity are organizations that are also possessed of a profound ability to transfer knowledge. Because, you see, knowledge transfer is the raw material of productivity in knowledge work like software development. And the whole purpose of design is creating clarity.

Without clarity, the full extent of productivity will be limited to only that which tools can provide, which is really not terribly much in the end.

COURSES

COURSE								
MOBILE APPLICATIONS	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Advanced Android Application Development - Mark Murphy	3	4		24		26	IT Fornebu	17 900
*Android Application Development (Intro + Advanced) - Mark Murphy	5	2		22		24	IT Fornebu	22 900
*Aral Balkan's iOS SDK Dojo	3		15				IT Fornebu	17 900
*Intro to Android Application Development - Mark Murphy	2	2		22		24	IT Fornebu	13 900
*Silverlight for Windows Phone 7 - Gill Cleeren	3	4					IT Fornebu	17 900
MANAGEMENT	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Agile Management - Jurgen Appelo	2	30					IT Fornebu	13 900
*Lead Better - Essential Skills for Software Team Leadership - Roy Osherove	2			1			IT Fornebu	13 900
Effective Concurrency	**Days**	mai	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
Effective Concurrency - Herb Sutter	3					12	Felix, Oslo	17 900
LEAN - AGILE	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Kanban - Sergey Dmitriev	2		23		22		IT Fornebu	13 900
*Lean QA: Smart Kvalitetssikring - Tom & Kai Gilb	2	12					IT Fornebu	13 900
SCRUM	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Certified Scrum Product Owner - CSPO - Geoff Watts	2		16				IT Fornebu	13 900
Certified Scrum Product Owner - CSPO - Mike Cohn	2	4			28		Telenor Arena	13 900
*Certified ScrumMaster - CSM - Geoff Watts	2		14				IT Fornebu	13 900
Certified ScrumMaster - CSM - Mike Cohn	2	2			26		Felix, Oslo	13 900
*Smidig utvikling med Scrum med Arne Laugstøl	1				9		IT Fornebu	5 900
*Verdi Smidig: nye praktiske metoder for å styre smidige utviklingslag til å levere målbar verdi - Tom & Kai Gilb	3	9					IT Fornebu	17 900
TESTDRIVEN DEVELOPMENT - TESTING	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Agile Testing: A Roadmap to Success with Janet Gregory	3	11					IT Fornebu	17 900
*Clean Code: Agile Software Craftsmanship - Robert C. Martin	2						IT Fornebu	13 900
*From User Stories to Acceptance Tests - Gojko Adzic	3					24	IT Fornebu	17 900
*Legacy Code -Test Writing Techniques - Michael Feathers	2						IT Fornebu	13 900
*Test-Driven Development & Refactoring Techniques - Robert C. Martin	3						IT Fornebu	17 900
*Test-Driven Development in .NET (TDD in .NET Course) - Roy Osherove	3			3			IT Fornebu	17 900
*Test-Driven JavaScript -Christian Johansen	3		29				IT Fornebu	17 900
DESIGN - ANALYSIS- ARCHITECTURES	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Advanced Distributed Systems Design using SOA & DDD - Udi Dahan	5			15			IT Fornebu	22 900
*Agile Architecture and Design - Robert C. Martin	4				26		IT Fornebu	22 900
*Agile Design and Modeling for Advanced Object Design with Patterns - Craig Larman	4					31	IT Fornebu	20 900
*Analyse og design med SCRUM	4			29			IT Fornebu	19 900
*Architecture Skills - Kevlin Henney	3						IT Fornebu	17 900
*Arkitekturer med C#.NET	4		21		19		IT Fornebu	19 900
*Domain-Driven Design (DDD) - Course design Eric Evans - Instructor Kristian Nordal	4		21				IT Fornebu	20 900
*The Architect`s Master Class - Juval Lowy	5				19		IT Fornebu	23900
DATABASE	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Advanced T-SQL kurs - Itzik Ben-Gan	5				5		IT Fornebu	22 900
*Databasedesign, -implementering og SQL-programmering - Dag Hoftun Knutsen	4	10			27		IT Fornebu	19 900
*Oracle SQL- programmering - Dag Hoftun Knutsen	3						IT Fornebu	17 900
PROGRAMMING	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Browsers with Wings: HTML5 APIs - Remy Sharp	3		27				IT Fornebu	17 900
*C++-501: C++ for Embedded Developers - Mike Tarlton	5				26		IT Fornebu	22 900
*Objektorientert utvikling	4	3				24	IT Fornebu	19 900
*Programming in C++ with Mike Tarlton	4				26		IT Fornebu	19 900

JAVA	Days	May	Jun	Aug	Sep	Oct	Location	Price
*Avansert Maven 2	2	18			27		IT Fornebu	13 900
*Core Spring Framework 3.0	4	2	27		5	17	IT Fornebu	20 900
*Effective JPA - Industrial strength Java Persistence with JPA 2.0 and Hibernate	2	26			1		IT Fornebu	13 900
*Effective Scala	3				5		IT Fornebu	17 900
*Programming (Java SE 6) with Peet Denny	5	9					IT Fornebu	22 900
*Spring and Hibernate Development	5	2	27		5	17	IT Fornebu	22 900
XML	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Exchanging and Managing Data using XML and XSLT - Espen Evje	3	18					IT Fornebu	17 900
MICROSOFT	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*70-513 Windows Communication Foundation (WCF) 4 - Dominick Baier	4	2			5		IT Fornebu	19 900
*70-513 Windows Communication Foundation (WCF) 4 - Maurice de Beijer	4						IT Fornebu	19 900
*ASP.NET MVC	3				14		IT Fornebu	17 900
*C#.NET: Utvikling av applikasjoner i .NET med C# - Arjan Einbu	5	9		15	19	31	IT Fornebu	22 900
*Cloud Computing with the Windows Azure Platform - Michele Leroux Bustamante	3			17			IT Fornebu	16 900
*Design Concepts, Architecture, and Advanced Capabilities for Silverlight and WPF - Billy Hollis	3				28		IT Fornebu	17 900
*MS 70-515 Web-utvikling med ASP.NET - Arjan Einbu	5		27	29		24	IT Fornebu	22 900
*MS 70-516 / 10265A Developing Data Access Solutions with Microsoft Visual Studio 2010 - Einar Ingebrigtsen	5			22			IT Fornebu	22 900
*Silverlight 4.0 - Dave Wheeler	5		20				IT Fornebu	22 900
*WPF - 70-511 / 10262A Windows Presentation Foundation (WPF) - Arne Laugstøl	4		14	22		10	IT Fornebu	17 900
*WPF and Silverlight Liftoff - Billy Hollis	2				26		IT Fornebu	13 900
SHAREPOINT	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*SharePoint 2010: For the .NET Developer and Architect - Sahil Malik	5			22		10	IT Fornebu	22 900
*SharePoint 2010: User Experience and Design - Sahil Malik	3	23				17	IT Fornebu	18 900
*SharePoint in the cloud: Office365, Azure, and Multitenant environments	3				12		IT Fornebu	17 900
BUSINESS INTELLIGENCE - BI	**Days**	**May**	**Jun**	**Aug**	**Sep**	**Oct**	**Location**	**Price**
*Microsoft Business Intelligence BI: SQL Server 2008 R2 and Office 2010 - Peter Myers	5	23					IT Fornebu	22 900

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

IN-HOUSE TRAINING?

In addition to our scheduled courses, we offer a number of courses on demand. These can be designed as in-house courses in your company, or, if there is sufficient interest, as open courses. To find out about availability, please contact:
info@programutvikling.no

ENROLMENT OPTIONS

www.programutvikling.no

info@programutvikling.no

Tel. +47 67 82 72 35

Fax +47 67 82 72 31

PROGRAM
UTVIKLING AS

A Cool jQuery Ajax Function You Might Not Know About

by Rod Paddock

One of the best features of the jQuery library its Ajax functionality. jQuery does a great job of abstracting away the complexity of dealing with Ajax services calls. What you may not know is that jQuery handles provides a set of global Ajax events that you can plug into. The global Ajax functions provided by jQuery are:

Event name	Purpose
ajaxSend	Called before an Ajax request is sent
ajaxStart	Called when an Ajax request starts
ajaxComplete	Called when a single Ajax call completes
ajaxStop	Called when ALL Ajax calls are complete
ajaxSuccess	Called when an Ajax call processes successfully
ajaxError	Called when an Ajax call errors out

In this article I am going to discuss one of the functions I have found most useful in my day to day jQuery programming:

ajaxSend

ajaxSend

One application I worked on recently was a payment processing application for a banking call center. During the development of this application we determined that in nearly every Ajax call we needed to send the employees identifier to the server for auditing reasons. This application makes numerous calls to the server and we didn't want to add parameters to each and every Ajax call. This is where we found ajaxSend.

One of the most useful features of ajaxSend is its ability to attach data to each and every jQuery Ajax call. The code sample below shows how you can attach data to every Ajax POST call (opposite page).

```
$(document).ajaxSend(function(event, request, settings) {

  if ( settings.type == ‘POST’ ) {
   settings.data = (settings.data ? settings.data + “&” : “”)
            + “NDC=” + encodeURIComponent(“NDC”);
   request.setRequestHeader(“Content-Type”,
      “application/x-www-form-urlencoded”);
            }
   });
```

There is a lot going on in that code:

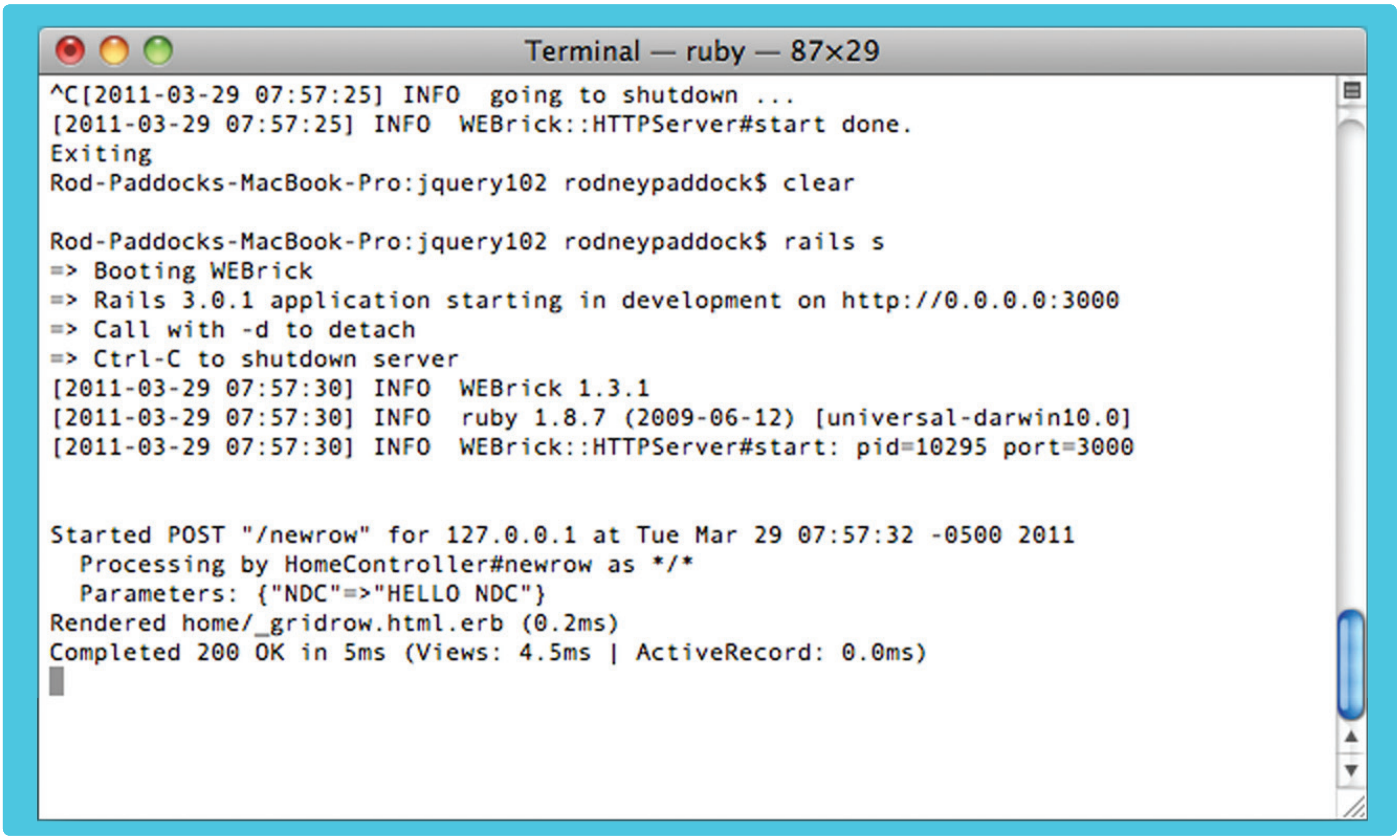
The first thing the code does is to check and see what type of call is being made to the server. In our code the preferred mechanism is to use POST commands.

After checking the type of request we are making the code then checks to see if there are already any parameters attached to the Ajax call. If there are it simply append them to the request. If not it populates the data option on the settings parameter.

Data appended to the Ajax call must be properly encoded. This is done using the encodeURIComponent command built into the browser’s JavaScript implementation.

Finally the code sets the request header’s Content-Type to “application/x-www-form-urlencoded”. This is a requirement when posting content to the server.

Now let’s take a look at this code in action. The following screen shot shows the HTTP log of a jQuery Ajax request using code illustrated above.



```
^C[2011-03-29 07:57:25] INFO  going to shutdown ...
[2011-03-29 07:57:25] INFO  WEBrick::HTTPServer#start done.
Exiting
Rod-Paddocks-MacBook-Pro:jquery102 rodneypaddock$ clear

Rod-Paddocks-MacBook-Pro:jquery102 rodneypaddock$ rails s
=> Booting WEBrick
=> Rails 3.0.1 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2011-03-29 07:57:30] INFO  WEBrick 1.3.1
[2011-03-29 07:57:30] INFO  ruby 1.8.7 (2009-06-12) [universal-darwin10.0]
[2011-03-29 07:57:30] INFO  WEBrick::HTTPServer#start: pid=10295 port=3000


Started POST "/newrow" for 127.0.0.1 at Tue Mar 29 07:57:32 -0500 2011
  Processing by HomeController#newrow as */*
  Parameters: {"NDC"=>"HELLO NDC"}
Rendered home/_gridrow.html.erb (0.2ms)
Completed 200 OK in 5ms (Views: 4.5ms | ActiveRecord: 0.0ms)
█
```

As you can imagine this functionality is very useful for a number of applications. You could use this to manage sessions from the client to the server, you could use this functionality to send authentication tokens from server to client and back.

The possibiliteies are endless.

The Summer Party of the Year

by Inger Renate Moldskred

Two performers. Two bands. Lots of Summer hits.

Welcome to the summer party of the year at NDC 2011.This year we are proud to present two of Norways most popular and skilled musicians, from two of Norway's best selling bands of all time – on the same stage – performing together.

Arne Hurlen

Hans Petter Aaserud

Thursday 9th of June, Arne Hurlen, lead vocalist in "Postgirobygget" and singer and guitar player in the popular band "Trang fødsel", Hans Petter Aaserud, come together to perform a great selection of timeless Norwegian summer hits.

Hans Petter Aaserud co-founded the band "Trang Fødsel" as a duo in 1992 together with Jørund Fluge Samuelsen. When the first album was released the in 1995, the duo had grown to a five man band. The band was, with songs like "Kursiv" (Italic) and "Drømmedame" (Dream woman), the leading band within the genre defined as "student rock" in the 90's.

The rock band "Postgirobygget" also started in the 90's, and their debut album "Melis" (Icing sugar) from 1996 sold 160.000 copies. During the years 1996–1998 the album held a spot in the popular hit list "VG-lista" a grand total of a whopping 70 weeks.

The music from these bands is characterized by humorous, sarcastic lyrics and fun rhythms. Both bands are popular and brilliant live bands, and the crowd often sings along when well known classics like "23 tommer" (23 inches) and "Livet er helt ålreit" (Life is pretty all-right) flow from the speakers.

One thing you can be sure of, is that these two amazing musicians are bound to set the mood for a wonderful night of catchy music, singing and maybe even dancing. The choice is yours!

80's party with Loveshack

To keep the party going, we continue the success from previous years with Loveshack. This year, the gig will be held from the main stage inside the conference hall. The boys in Loveshack bring a selection of 80's hits to the floor. Get ready to party!

The Flying Culinary Circus bring their Magic to NDC2011

by Thomas Nilsen

Ladies and Gentlemen, we are once again proud to present The Flying Culinary Circus!

These guys carry the knowhow and passion to enthrall the world through their well-honed recipes. Flying around the globe on broomsticks made of celery or whatnot, they make stops to entertain the big stars. From a selection of great names we mention Elton John, Sarah Ferguson, Perez Hilton and American pop star Ke$ha.

Each day the participants at NDC will be served breakfast, lunch and an afternoon bite based on the concept of the day.

Speaking from personal experience and utter bedazzlement during a sampling session, I ensure you that our lovely quartet will conjure the most delightful treats.

The concepts are described below:

Day 1
A Taste of Summer

For day one FCC focus their energy to bring you "A Taste of Summer" with fruit salad, noodles, smoked pork chops and salmon.

Day 2
Health and Vigor

On day two they put their strength into "Health and Vigor" with fruit enriched cottage cheese, spicy soups and a very interesting version of corn on the cob.

Day 3
Western Style

With a flick of the wrist they bring out the big guns and go "Western Style" on day three. Saddle up for a genuine cowboy breakfast, spicy salad and a juicy burger buffet.

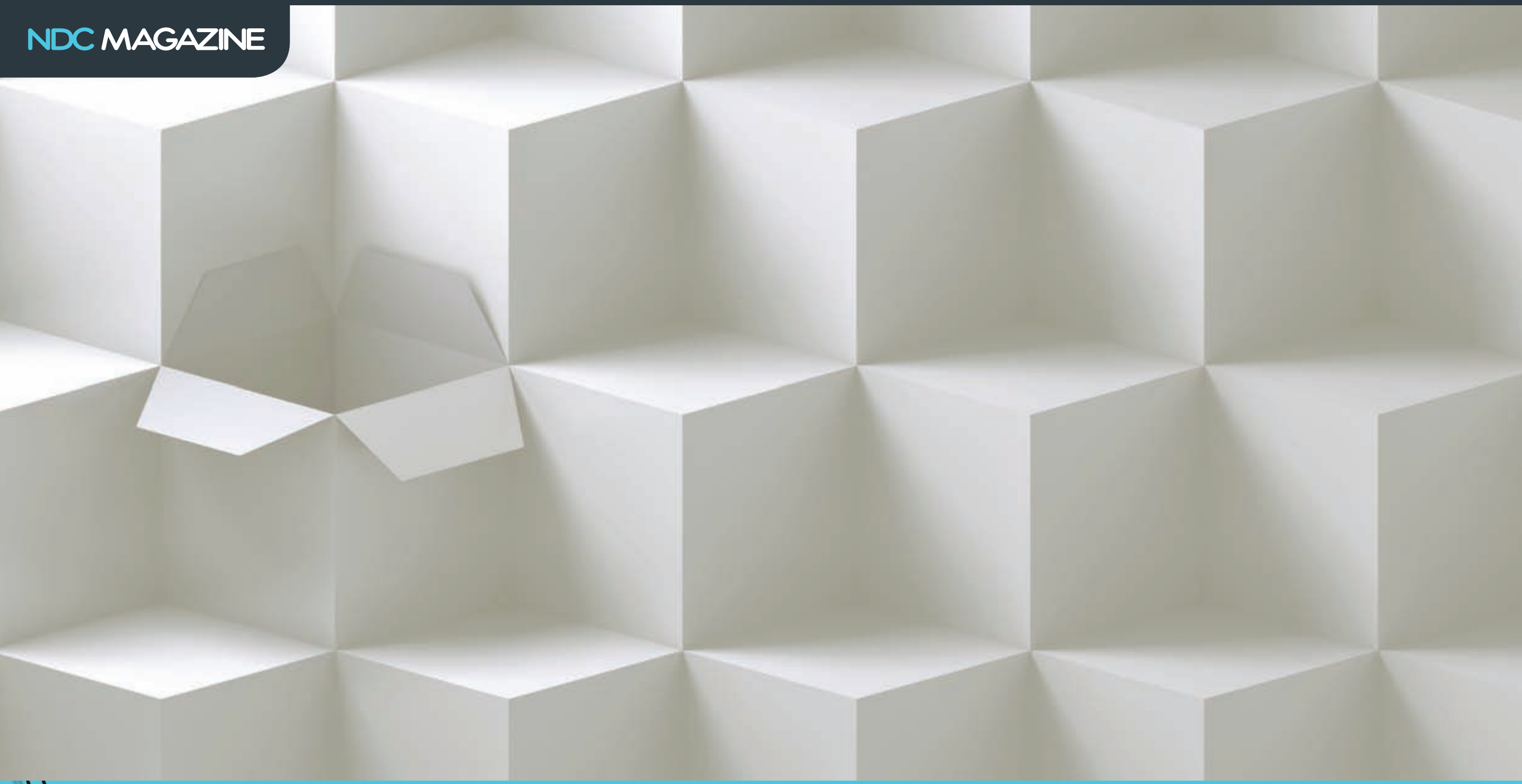
Trond Svendgård
Fish and Seafood Expert

Hans Kristian Larsen
Meat Expert

Mathias Spieler Bugge
Sauce & Soup Expert

Tor Jørgen Kramprud Arnesen
Herb & Vegetable Expert

See you at NDC2011 !



Deterministic unit tests with non-deterministic input

by Mark Seemann

One very fundamental rule of unit testing is that all tests must be deterministic. It's a good rule, but following it rigidly may lead to maintainability issues, so I prefer to bend it. Unit tests should be deterministic, but I drive them with non-deterministic input. This certainly isn't the most accepted advice for unit testing you'll read all day, but there's a rationale behind it – and it also works very well in practice. In this article I will outline why this technique works so well.

The motivating reason behind the rule of deterministic unit tests is that we use test suites for regression testing. This means that it's extremely important that *the same code path* is exercised every time a given test case executes. While deterministic input certainly makes this possible, it's not always the only way to make tests deterministic. In fact, we can often use non-deterministic input and still always execute the same code path. This enables us to make unit tests look more like *specifications* and less like *examples*. I'll show you an example (sic!) that demonstrates how this works.

Let's a assume that we have a simple, immutable Name class with FirstName and LastName properties and that we want to use tests to drive forward the implementation of a ChangeFirstNameTo method. Although we want to use a test-first approach, I'll begin by showing you the final result:

```
public Name ChangeFirstNameTo(string newFirstName)
{
    if (newFirstName == null)
    {
        throw new ArgumentNullException(“newFirstName”);
    }

    return new Name(newFirstName, this.LastName);
}
```

There are only two paths the execution can take through this method: Either newFirstName is null and an exception is thrown, or a new Name instance is returned. This means that we should be able to cover the method with only two tests. However, if we try to drive this method forward with deterministic input, it takes more than two test cases. The first test case might look like this:

```
var sut = new Name(“John”, “Doe”);
var result = sut.ChangeFirstNameTo(“Jim”);
Assert.Equal(“Jim”, result.FirstName);
```

A test like this is not a *specification* – it's an *example*, and the problem with these is that you can't generalize from a single example. Thus, the *simplest possible* implementation of the method looks like this:

```
return new Name(“Jim”, “Doe”);
```

That’s obviously not the implementation we wish to drive forward, so we can create another example:

```
var sut = new Name(“Jane”, “Doe”);
var result = sut.ChangeFirstNameTo(“Jill”);
Assert.Equal(“Jill”, result.FirstName);
```

However, the simplest possible implementation now gives us this:

```
return new Name(newFirstName, “Doe”);
```

This is better, but we must obviously also write a test that drives us towards using this.LastName for the last name. This process confirms Robert C. Martin’s statement that “As the tests get more specific, the code gets more generic.” However, the disadvantages of this type of *example-driven development* are:

- *It forces us to write duplicate test code.*
- *It becomes harder to understand what the tests verify because we, as test readers, have to make a mental effort of collecting and correlating input and output example data across many different test cases.*

The first issue is fairly easy to deal with, but the second represents a serious problem. In the example I use here it may be straightforward because it’s so simple, but just a bit of added complexity can make it really hard to extract the information from the noise.

A step in the direction of writing the test case as a specification is to get rid of the hardcoded values. Hiding the value of the new first name effectively forces us to make explicit the relationship between input and expected output:

```
var newFirstName = GetFirstName();
var sut = new Name(“John”, “Doe”);
var result = sut.ChangeFirstNameTo(newFirstName);
Assert.Equal(newFirstName, result.FirstName);
```

Notice how replacing the hardcoded string with a variable forces us to use the newFirstName variable as both input and expected output. This makes the tests easier to read and understand because *a single test is sufficient* to state the requirements. This is called a *Derived Value* in the excellent book “xUnit Test Patterns”.

The implementation of the GetFirstName method might simply return a hardcoded value such as “Jim”, but that doesn’t give us the appropriate level of confidence in the implementation. In fact, the first naïve implementation with hardcoded values would pass the test. This means that hardcoded values are inadequate when it comes to driving forward implementations.

Using a *Generated Value* solves this problem nicely. This implementation of the GetFirstName method makes the actual value truly unknown:

```
return Guid.NewGuid().ToString();
```

Such a truly unknown value forces us to deal with the symbolic representation of the value in the unit test as well as in the implementation. Instead of going through Robert C. Martin’s Transformation Priorities we can drive and complete a single code path with a single *specification*.

Even though the input value is non-deterministic it has no influence on the execution path of the code. This is true for a surprisingly large set of code. As long as we constrain the non-deterministic value to stay within the same *Equivalence Class* for the input in question, the unit test is guaranteed to be deterministic even in the face of non-deterministic input. I call this principle *Constrained Non-determinism* and it’s a fundamental design philosophy underlying my open source library AutoFixture.

Agenda NDC2011

June 8th

TRACK 1	10:20	Windows Azure - Under the hood	Maarten Balliauw
	11:40	Architecting for a cost effective Windows Azure solution	Maarten Balliauw
	13:40	A journey to the other side Windows Azure + Java	Magnus Mårtensson
	15:00	Let me introduce my Moncai!	Dale Ragan
	16:20	Build Friendly Applications that like to Chit-Chat	Magnus Mårtensson
	17:40	Cloud computing Guidance Part 1 and 2	Shy Cohen
TRACK 2	10:20	TBA	TBA
	11:40	Don't get stung - an introduction to web security	Barry Dorrans
	13:40	Document databases with ASP.NET MVC	Rob Ashton
	15:00	ASP.NET MVC 3 Vs. Ruby on Rails 3	Shay Friedman
	16:20	Why Web Performance Matters	Richard Campbell
	17:40	Continuous Deployment for ASP.net	Ben Hall
TRACK 3	10:20	Fundamental Design Principles for UI Developers	Billy Hollis
	11:40	Radically Advanced Templates for in Silverlight	Billy Hollis
	13:40	Develop Business Application Screens in WPF	Paul Sheriff
	15:00	WPF Styles, Resources and Templates, Oh My!	Paul Sheriff
	16:20	Switch on the LightSwitch	Gill Cleeren
	17:40	Modernizing a large desktop application through vertical porting	Westlye Larsen/Aqrawi/Flugsrud
TRACK 4	10:20	Kanban Explained! A Counter-intuitive Approach to Creating a Lean Technology Organization	David Anderson
	11:40	How to survive on an agile project?	Sergey Dmitriev
	13:40	Cognitive Biases and Effects You Should Know About	Kevlin Henney
	15:00	The Mistake at the Heart of Agile (and how to get past it)	Michael Feathers
	16:20	Business Architecture Foundations of IT	Ian Robinson
	17:40	Evolutionary Design Illustrated	James Shore
TRACK 5	10:20	Learn Your Tools. NUnit	Daniel González García
	11:40	The benefits of simple, automated performance testing	Kristoffer Dyrkorn
	13:40	SpecFlow: Pragmatic BDD for .NET	Gaspar Nagy/Jonas Bandi
	15:00	Building .NET applications with BDD	Gaspar Nagy/Jonas Bandi
	16:20	Test-driven JavaScript in practice	Christian Johansen
	17:40	The Transformation Priority Premise	Robert C. Martin
TRACK 6	10:20	Things You Can Learn from 101 Things I Learned in Architecture School	Kevlin Henney
	11:40	Designing Software with SOLID principles	Daniel González García
	13:40	TBA	TBA
	15:00	When less is more - Agile web architecture	Tore Vestues/Jonas Follesø
	16:20	Introducing The FLUID Principles	Kevlin Henney/Anders Norås
	17:40	Practical OData with and without Entity Framework	Vagif Abilov
TRACK 7	10:20	Clojure - The Last Programming Language	Robert C. Martin
	11:40	TBA	TBA
	13:40	TBA	TBA
	15:00	ECMAScript 5: The New Parts	Douglas Crockford
	16:20	Dynamic in a Static World	Hadi Hariri
	17:40	Async 101	Jon Skeet

June 9th

TRACK 1	09:00	TBA	TBA
	10:20	Silverlight, MVVM & WCF RIA Services: an architectural story	Kevin Dockx
	11:40	Bubble Gum and Dynamite: WebMatrix Is More Powerful Than You Thin	Rob Conery
	13:40	Blend for developers	Gill Cleeren
	15:00	Develop Business Application Screens in WPF	Paul Sheriff
	16:20	Architecting Claims-aware Applications (with the Windows Identity Foundation and Active Directory Federation Services)	Dominick Baier
	17:40	Securing REST-Services and Web-APIs	Dominick Baier
TRACK 2	09:00	HTML5: Why, what, who, when, how?	Bruce Lawson
	10:20	HTML5 before your very eyes	Bruce Lawson
	11:40	jQuery: in with the new, out with the old and busted	Remy Sharp
	13:40	HTML5: Huh - what is it good for?	Remy Sharp
	15:00	HTML5 - Offline Business Applications for Desktops, Tablets and Phones	Ingo Rammer
	16:20	jQuery 101	Rod Paddock
	17:40	jQuery 102	Rod Paddock
TRACK 3	09:00	Driving a Kaizen Culture using regular Operations Reviews	David Anderson
	10:20	We Deliver Business Value	James Shore
	11:40	Framing the Problem	Kevlin Henney
	13:40	Metrics for measuring agile team maturity	Geoff Watts
	15:00	Are we nearly there yet?	Geoff Watts
	16:20	Dealing With Defects on an Agile Team	Janet Gregory
	17:40	The Land that Scrum Forgot	Robert C. Martin
TRACK 4	09:00	TBA	Aral Balkan
	10:20	Peeking at the source of a successful Windows Phone app	Jeff Wilcox
	11:40	MonoTouch State of the Union	Chris Hardy
	13:40	Making a beautiful and rock-solid Windows Phone application	Jeff Wilcox
	15:00	Mono and Mobile	Chris Hardy
	16:20	Cross platform .NET in practice. An end-to-end example of the same app implemented across iPad, iPhone, WP7 and Android	Jonas Follesø
	17:40	Mobile Panel Discussion - TBA	TBA

Agenda NDC2011

June 9th

TRACK 5	09:00	Effective Specifications and Tests for Agile Projects	Gojko Adzic
	10:20	Winning Big with Specification by Example - Lessons Learned from 50 Successful Projects	Gojko Adzic
	11:40	Test-Driven Development for Web UI with Selenium	Scott Bellware
	13:40	Quality without Testing!!	Jason Bryant
	15:00	Test-Driven Development as if You Meant It	Keith Braithwath
	16:20	Quality	Douglas Crockford
	17:40	Best Practices for writing First Class Unit Tests	Dennis Doomen
TRACK 6	09:00	IronRuby FTW!!!	Shay Friedman
	10:20	Clojure: Lisp for the Real World	Stuart Sierra
	11:40	Rethinking Object-Oriented: Clojure and the Expression Problem	Stuart Sierra
	13:40	Functional Programming in F#	Oliver Sturm
	15:00	WTF is a Monad?	Robert C. Martin
	16:20	Wake Up and Smell The Coffee	Anders Norås
	17:40	JavaScript the worlds most important language?	Trygve Lie
TRACK 7	09:00	Hardcore .NET Production Debugging	Ingo Rammer
	10:20	TBA	Corey Haines
	11:40	Out of Comfort	Roy Osherove
	13:40	The 10 habits of highly effective programmers	Dennis Doomen
	15:00	Discovering Startling Things From Your Version Control History	Michael Feathers
	16:20	A developers guide to encryption	Barry Dorrans
	17:40	MetaProgramming using T4	Mårten Rånge

June 10th

TRACK 1	09:00	TBA	TBA
	10:20	NuGet in a caramel coated nutshell	Damian Edwards
	11:40	Designing a JavaScript application	Christian Johansen
	13:40	Ruby on Rails for .NET Developers	Scott Bellware
	15:00	Avoiding Cross site scripting - not as easy as you might think	Erlend Oftedal
	16:20	Progressive EPiServer Development	Joel Abrahamsson
TRACK 2	09:00	Introduction to RavenDB	Rob Ashton
	10:20	Getting Things Done with REST	Ian Robinson
	11:40	Dr. CQRS or: How I Learned to Stop CRUDing and Love the Domain Model	Fredrik Kalseth
	13:40	Entity Framework 4 - Get started now	Alf Kåre Lefdal
	15:00	Raven DB by Example	Emil Cardell
	16:20	Being reliable without transactions	Szymon Pobiega/Joost van de Sande
TRACK 3	09:00	Danger! Craftsmen ahead!	Gil Zilberfeld
	10:20	How to succeed with the Product Owner role	Mitch Lacey
	11:40	How to architect a codebase-wide refactoring project	Gary Short
	13:40	Refactoring Legacy Code Bases	Hadi Hariri
	15:00	Agile Patterns: Agile Estimation	Stephen Forte
	16:20	TBA	TBA
TRACK 4	09:00	Producing great video and audio on a budget	Carl Franklin
	10:20	Async Deep Dive	Jon Skeet
	11:40	TEST	Rob Conery
	13:40	Top 10 things to learn from Clojure that will	Martin Jul
	15:00	Developing .NET Applications for the Mac App Store	Geoff Norton
	16:20	Asynchronous programming made simple through messaging	Svein Arne Ackenhausen
TRACK 5	09:00	Produce Cleaner Code with Aspect-Oriented Programming (AOP)	Gael Fraiteur
	10:20	Advanced Aspect Oriented Programming	Donald Belcham
	11:40	Ten Simple Rules for Rewriting IL on the CLR	Philip Laureano
	13:40	Inversion of Control containers: patterns and anti-patterns	Krzysztof Kozmic
	15:00	Because You Suck at Design Patterns	Gary Short
	16:20	NDepend and good practices for code analysis	Rune Sundling
TRACK 6	09:00	JavaScript max-out with Sharepoint 2010	Sahil Malik
	10:20	Introduction to Analysis Services Data Mining	Peter Myers
	11:40	Taking Your Application to the Next Level with Data Mining	Peter Myers
	13:40	Maintainable custom code strategies in Sharepoint 2010	Sahil Malik
	15:00	To Rest Or Not To Rest	Miguel Castro
	16:20	No-Config WCF	Miguel Castro
TRACK 7	09:00	Package Management deep-dive with OpenWrap	Sebastien Lambla
	10:20	TBA	TBA
	11:40	TBA	TBA
	13:40	Vim for Victory	Roy Osherove
	15:00	Unit testing with AutoFixture	Mark Seeman
	16:20	SharePoint 2010 automation using PowerShell	Harald Fianbakken

See ndc2011.no for updates

webnodes

Semantic ASP.NET CMS

Get more out of your content!

Built for developers

- Use your exsisting ASP.Net skills
- Compact and easy to learn API
- Built-in ORM system
- Use LINQ for all data access
- Powerful semantic data model
- More than just page publishing
- Expose all content as ODATA
- Content types are actual classes

Some sites running on Webnodes CMS:

Save the children - Norway
www.reddbarna.no

Tomra
www.tomra.com

Diplom-Is
www.diplom-is.no

Rom Eiendom
www.rom.no

Norwegian Property
www.norwegianproperty.no

Datatilsynet
www.slettmeg.no

Econa
www.econa.no

Akademikerne
www.akademikerne.no

Accident Investigation Board Norway
www.aibn.no

Try it out! www.webnodes.com/try-it