1

An Inside Look at the Evolution of DotNetNuke

As much as I would like people to believe that DotNetNuke was intentionally created as a premier open source project for the Microsoft platform, it is unfortunately not the case. As is true with many open source projects, the software was created with commercial intentions in mind, and only when it was discovered that its true purpose would not be realized was it reconsidered as an open source project.

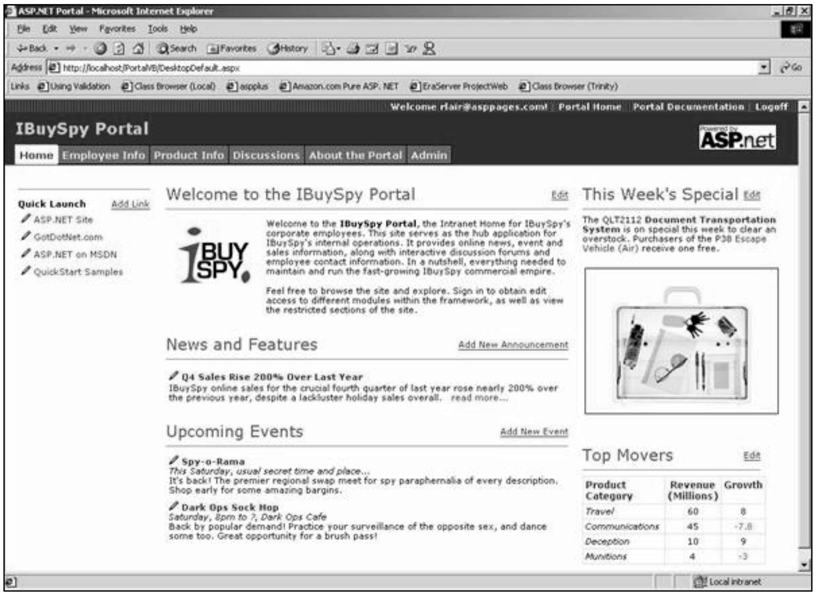
In 2001–2002 I was working for a medium-sized software consulting company that was providing outsourced software development services to a variety of large U.S. clients specializing primarily in e-Learning initiatives. The internal push was to achieve CMM 3.0 on a fairly aggressive schedule so that we could compete with the emerging outsourcing powerhouses from India and China. As a result there was an incredible amount of focus on process and procedure and somewhat less focus on the technical aspects of software engineering. Because the majority of the client base was interested in the J2EE platform, the company had primarily hired resources with Java skills — leaving myself with my legacy Microsoft background to assume more of an internal development and project management role. The process improvement exercise consumed a lot of time and energy for the company; attempting to better define roles and responsibilities and ensuring proper documentation throughout the project life cycle. Delving into CMM and the PMBOK were great educational benefits for me — skills that would prove to be invaluable in future endeavors. Ultimately the large U.S. clients decided to test the overseas outsourcing options anyway, which resulted in severe downsizing for the company. It was during these tumultuous times that I recognized the potential of the newly released .NET Framework (beta) and decided that I would need to take my own initiative to learn this exciting new platform in order to preserve my long-term employment outlook.

For a number of years I had been maintaining an amateur hockey statistics application as a sideline hobby business. The client application was written in Visual Basic 6.0 with a Microsoft Access backend, and I had augmented it with a simplistic web publishing service using Active Server Pages 3.0 and SQL Server 7.0. However, better integration with the World Wide Web was quickly becoming the most highly requested enhancement and I concluded that an exploration into

ASP.NET was the best way to enhance the application, while at the same time acquire the skills necessary to adapt to the changing landscape. My preferred approach to learning new technologies is to experience them firsthand rather than through theory or traditional education. It was during a Microsoft Developer Days conference in Vancouver, British Columbia in 2001 that I became aware of a reference application known as the IBuySpy Portal.

IBuySpy Portal

Realizing the educational value of sample applications, Microsoft had built a number Source Projects, which were released with the .NET Framework 1.0 Beta to encourage developers to cut their teeth on the new platform. These projects included full source code and a very liberal End User License Agreement (EULA) that provided nearly unrestricted usage. Microsoft co-developed the IBuySpy Portal with Vertigo Software and promoted it as a "best practice" example for building applications in the new ASP.NET environment. Despite its obvious shortcomings, the IBuySpy Portal had some very strong similarities to both Microsoft Sharepoint as well as other open source portal applications on the Linux/Apache/mySQL/PHP (LAMP) platform. The portal allowed you to create a completely dynamic web site consisting of an unlimited number of virtual "tabs" (pages). Each page had a standard header and three content panes — a left pane, a middle pane, and a right pane (a standard layout for most portal sites). Within these panes the administrator could dynamically inject "modules" — essentially mini-applications for managing specific types of web content. The IBuySpy Portal application shipped with six modules designed to cover the most common content types — (announcements, links, images, discussions, html/text, XML) as well as a number of modules for administrating the portal site. As an application framework the IBuySpy Portal (see Figure 1-1) provided a mechanism for managing users, roles, permissions, tabs, and modules. With these basic services, the portal offered just enough to whet the appetite of many aspiring ASP.NET developers.


ASP.NET Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/PortalVB/DesktopDefault.aspx
Go
Links Using Validation Class Browser (Local) aspplus Amazon.com Pure ASP. NET EraServer ProjectWeb Class Browser (Trinity)
Welcome rlair@asppages.com! | Portal Home | Portal Documentation | Logoff
IBuySpy Portal
Home Employee Info Product Info Discussions About the Portal Admin
Powered by ASP.net
Quick Launch Add Link
ASP.NET Site
GotDotNet.com
ASP.NET on MSDN
QuickStart Samples
Edit
Add New Announcement
Add New Event
Local intranet

Welcome to the IBuySpy Portal

Welcome to the **IBuySpy Portal**, the Intranet Home for IBuySpy's corporate employees. This site serves as the hub application for IBuySpy's internal operations. It provides online news, event and sales information, along with interactive discussion forums and employee contact information. In a nutshell, everything needed to maintain and run the fast-growing IBuySpy commercial empire.

Feel free to browse the site and explore. Sign in to obtain edit access to different modules within the framework, as well as view the restricted sections of the site.

News and Features

Q4 Sales Rise 200% Over Last Year
IBuySpy online sales for the crucial fourth quarter of last year rose nearly 200% over the previous year, despite a lackluster holiday sales overall. read more...

Upcoming Events

Spy-o-Rama
This Saturday, usual secret time and place...
It's back! The premier regional swap meet for spy paraphernalia of every description. Shop early for some amazing bargins.

Dark Ops Sock Hop
Saturday, 8pm to ?, Dark Ops Cafe
Back by popular demand! Practice your surveillance of the opposite sex, and dance some too. Great opportunity for a brush pass!

This Week's Special

The QLT2112 **Document Transportation System** is on special this week to clear an overstock. Purchasers of the P38 Escape Vehicle (Air) receive one free.

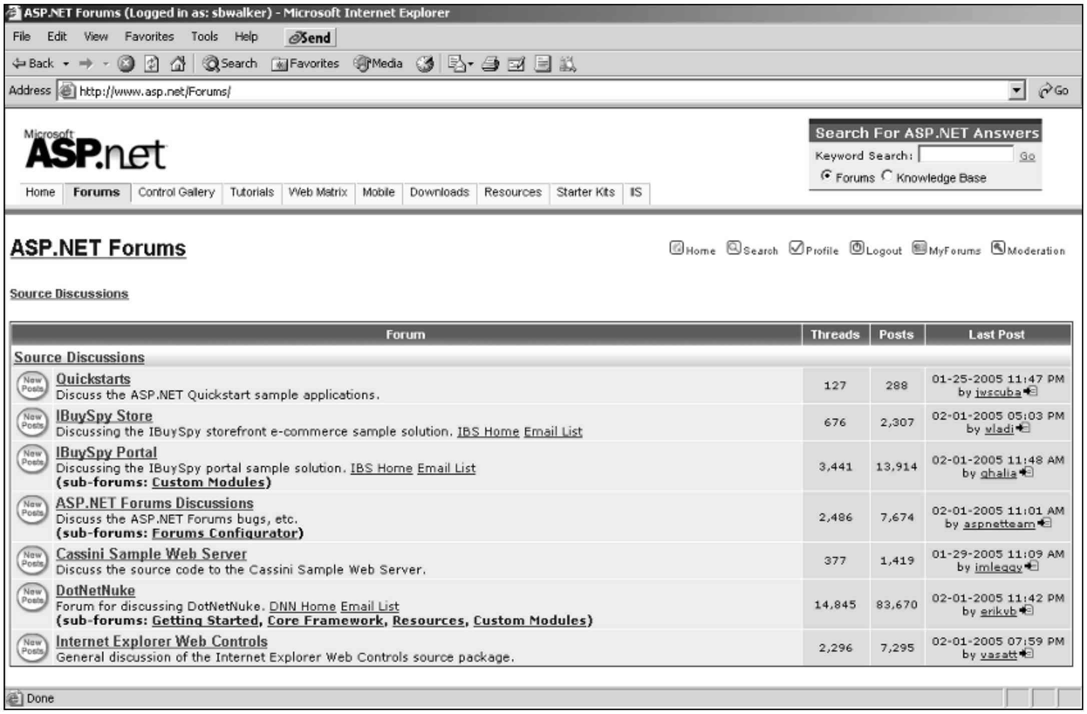
Top Movers

Product Category	Revenue (Millions)	Growth
Travel	60	8
Communications	45	-7.8
Deception	10	9
Munitions	4	-3

Figure 1-1

ASP.NET

The second critical item that Microsoft delivered at this point in time was a community Forums page on the `www.asp.net` web site (see Figure 1-2). This Forum provided a focal point for Microsoft developers to meet and collaborate on common issues in an open, moderated environment. Prior to the release of the Forums on `www.asp.net` there was a real void in terms of Microsoft community participation in the online or global sphere, especially when compared to the excellent community environments on other platforms.


ASP.NET Forums (Logged in as: sbwalker) - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://www.asp.net/Forums/ Go
Microsoft ASP.net
Home Forums Control Gallery Tutorials Web Matrix Mobile Downloads Resources Starter Kits IIS
Search For ASP.NET Answers
Keyword Search: Go
Forums Knowledge Base
Home Search Profile Logout MyForums Moderation
Done

ASP.NET Forums

Source Discussions

Forum	Threads	Posts	Last Post
Source Discussions			
Quickstarts Discuss the ASP.NET Quickstart sample applications.	127	288	01-25-2005 11:47 PM by jwscuba
IBuySpy Store Discussing the IBuySpy storefront e-commerce sample solution. IBS Home Email List	676	2,307	02-01-2005 05:03 PM by vladi
IBuySpy Portal Discussing the IBuySpy portal sample solution. IBS Home Email List **(sub-forums: Custom Modules)**	3,441	13,914	02-01-2005 11:48 AM by ghalia
ASP.NET Forums Discussions Discuss the ASP.NET Forums bugs, etc. **(sub-forums: Forums Configurator)**	2,486	7,674	02-01-2005 11:01 AM by aspnetteam
Cassini Sample Web Server Discuss the source code to the Cassini Sample Web Server.	377	1,419	01-29-2005 11:09 AM by imleggy
DotNetNuke Forum for discussing DotNetNuke. DNN Home Email List **(sub-forums: Getting Started, Core Framework, Resources, Custom Modules)**	14,845	83,670	02-01-2005 11:42 PM by erikvb
Internet Explorer Web Controls General discussion of the Internet Explorer Web Controls source package.	2,296	7,295	02-01-2005 07:59 PM by vasatt

Figure 1-2

One discussion forum on the `www.asp.net` site was dedicated to the discussion of the IBuySpy Portal application, and it soon became a hotbed for developers to discuss their enhancements, share source code enhancements, and debate IT politics. I became involved in this Forum early on and gradually increased my community participation as my confidence in ASP.NET and the IBuySpy Portal application grew.

In order to appeal to the maximum number of community stakeholders, the IBuySpy Portal was available in a number of different source code release packages. There were VB.NET and C#.NET language versions, each containing their own VS.NET and SDK variants. Although Microsoft was aggressively pushing the newly released C# language, I did not feel a compelling urge to abandon my familiar Visual Basic roots. In addition, my experience with classic ASP 3.0 allowed me to conclude that the new code-behind model in VS.NET was far superior to the inline model of the SDK. As luck would have it, I was able to get access to Visual Studio .NET through my employer. So as a result, I moved forward with the

VB.NET/VS.NET version as my baseline framework. This decision would ultimately prove to be extremely important in terms of community acceptance, as I will explain later.

When I first started experimenting with the IBuySpy Portal application I had some very specific objectives in mind. In order to support amateur sports organizations, I had collected a comprehensive set of end user requirements based on actual client feedback. However after evaluating the IBuySpy Portal functionality, it quickly became apparent that some very significant enhancements were necessary if I hoped to achieve my goals. My early development efforts, although certainly not elegant or perfectly architected, proved that the IBuySpy Portal framework was highly adaptable for building custom applications and could be successfully used as the foundation for my amateur sports hosting application.

The most significant enhancement I made to the IBuySpy Portal application during these early stages was a feature that is now referred to as " multi-portal " or "site virtualization." Effectively, this was a fundamental requirement for my amateur sports hosting model. Organizations wanted to have a self-maintained web site but they also wanted to retain their individual identity. A number of vendors had emerged with semi-self-maintained web applications but nearly all of them forced the organization to adopt the vendor's identity (that is, `www.vendor.com/clientname` rather than `www.clientname.com`). Although this may seem like a trivial distinction for some, it has some major effects in terms of brand recognition, site discovery, search engine ranking, and so on. The IBuySpy Portal application already partitioned its data by portal (site) and it had a field in the Portals database table named PortalAlias, which was a perfect candidate for mapping a specific domain name to a portal. It was as if the original creators (Microsoft/Vertigo) had considered this use case during development but had not had enough time to complete the implementation, so they had simply left the "hook" exposed for future development. I immediately saw the potential of this concept and implemented some logic that allowed the application to serve up custom content based on domain name. Essentially, when a web request was received by the application, it would parse the domain name from the URL and perform a lookup on the PortalAlias field to determine the content that should be displayed. This site virtualization capability would ultimately become the "killer" feature that would allow the application to achieve immediate popularity as an open source project.

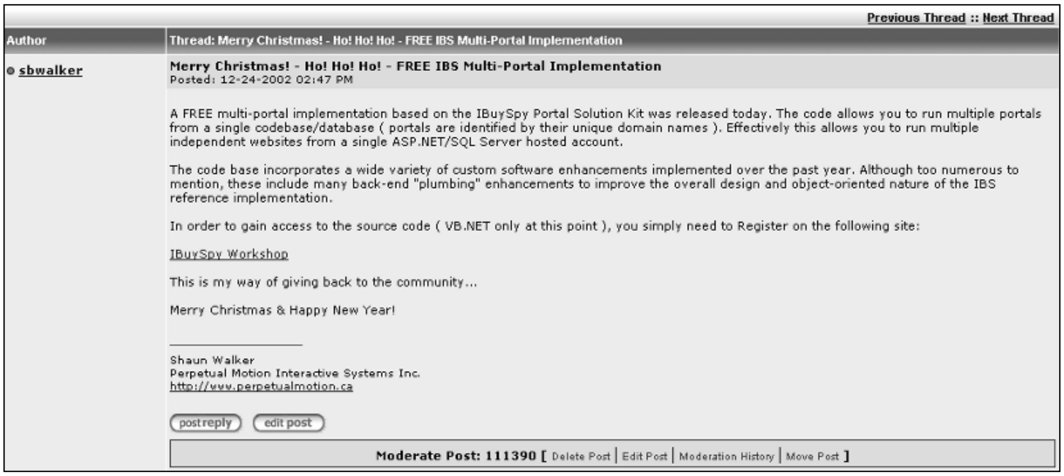
Over the next 8 to 10 months, I continued to enhance and refactor the IBuySpy Portal application as I created my own custom implementation (now codenamed SportsManager.Net). I added numerous features to improve the somewhat limited portal administration and content management aspects. At one point I enlisted the help of another developer, John Lucarino, and together we steadily improved the framework using whatever spare time we were able to invest. Unfortunately, since all of this was going on outside of regular work hours, there was very little time to focus on building a viable commercial venture. So at the end of 2002, it soon became apparent that we did not have enough financial backing or a business model to take the amateur sports venture to the next level. This brought the very commercial nature of the endeavor under scrutiny. If the commercial intentions were not going to succeed, I at least wanted to feel that my efforts had not been in vain. This forced me to evaluate alternative non-commercial uses of the application. Coincidentally, I had released the source code for a number of minor application enhancements to the `www.asp.net` community Forum during the year and I began to hypothesize that if I abandoned the amateur sports venture altogether, it was still possible that my efforts could benefit the larger ASP.NET community.

The fundamental problem with the IBuySpy Portal community was the fact that there was no central authority in charge of managing its growth. Although Microsoft and Vertigo had developed the initial code base, there was no public commitment to maintain or enhance the product in any way. Basically the product was a static implementation, frozen in time, an evolutionary dead-end. However, the IBuySpy Portal EULA was extremely liberal, which meant that developers were free to enhance, license, and redistribute the source code in an unrestricted manner. This led to many developers creating their own customized versions of the application, sometimes sharing discrete patches with the general community, but more often keeping their enhancements private; revealing only their public-facing web sites for community recognition (one of the most popular threads at this time was titled "Show me your Portal"). In hindsight, I really don't understand what each developer was hoping to achieve by keeping their enhancements private. Most probably thought there was a commercial opportunity in building a portal application with a richer feature set than their competitor. Or perhaps individuals were hoping to establish an expert reputation based on their public-facing efforts. Either way, the problem was that this mindset was really not conducive to building a community but rather to fragmenting it — a standard trap that tends to consume many things on the Microsoft platform. The concept of sharing source code in an unrestricted manner was really a foreign concept, which is obviously why nobody thought to step forward with an organized open source plan.

I have to admit I had a very limited knowledge of the open source philosophy at this point since all of my previous experience had been in the Microsoft community — an area where "open source" was simply equated to the Linux operating system movement. However, there had been chatter in the Forums at various times regarding the organized sharing of source code, and there was obviously some interest in this area. Coincidentally, a few open source projects had recently emerged on the Microsoft platform to imitate some of the more successful open source projects in the LAMP community. In evaluating my amateur sports application, I soon realized that nearly all of my enhancements were generic enough that they could be applied to nearly any web site — they were not sports related whatsoever. I concluded that I should release my full application source code to the ASP.NET community as a new open source project. So, as I mentioned earlier, the initial decision to open source what would eventually become DotNetNuke happened more out of frustration of not achieving my commercial goals rather than predicated philanthropic intentions.

IBuySpy Portal Forum

On December 24, 2002, I released the full open source application by creating a simple web site with a zip file for download. The lack of foresight of what this would become was extremely evident when you consider the casual nature of this original release. However, as luck would have it, I did do three things right. First, I thought I should leverage the "IBuySpy" brand in my own open source implementation so that it would be immediately obvious that the code base was a hybrid of the original IBuySpy Portal application, an application with widespread recognition in the Microsoft community. The name I chose was IBuySpy Workshop because it seemed to summarize the evolution of the original application — not to mention the fact that the "IBSW" abbreviation preferred by the community contained an abstract personal reference ("SW" are my initials). Ironically, I did not even have the domain name resolution properly configured for `www.ibuyspyworkshop.com` when I released (the initial download links were based on an IP address, `http://65.174.86.217/ibuyspyworkshop`). The second thing I did right was require people to register on my web site before they were able to download the source code. This allowed me to track the actual interest in the application at a more granular level than simply by the total number of downloads. Third, I publicized the availability of the application in the IBuySpy Portal Forum on `www.asp.net` (see Figure 1-3). This particular forum was extremely popular at this time; and as far as I know, nobody had ever released anything other than small code snippet enhancements for general consumption. The original post was made on Christmas Eve, December 24, 2002, which had excellent symbolism in terms of the application being a gift to the community.



Previous Thread :: Next Thread

Author | Thread: Merry Christmas! - Ho! Ho! Ho! - FREE IBS Multi-Portal Implementation

sbwalker

Merry Christmas! - Ho! Ho! Ho! - FREE IBS Multi-Portal Implementation
Posted: 12-24-2002 02:47 PM

A FREE multi-portal implementation based on the IBuySpy Portal Solution Kit was released today. The code allows you to run multiple portals from a single codebase/database (portals are identified by their unique domain names). Effectively this allows you to run multiple independent websites from a single ASP.NET/SQL Server hosted account.

The code base incorporates a wide variety of custom software enhancements implemented over the past year. Although too numerous to mention, these include many back-end "plumbing" enhancements to improve the overall design and object-oriented nature of the IBS reference implementation.

In order to gain access to the source code (VB.NET only at this point), you simply need to Register on the following site:

IBuySpy Workshop

This is my way of giving back to the community...

Merry Christmas & Happy New Year!

Shaun Walker
Perpetual Motion Interactive Systems Inc.
http://www.perpetualmotion.ca

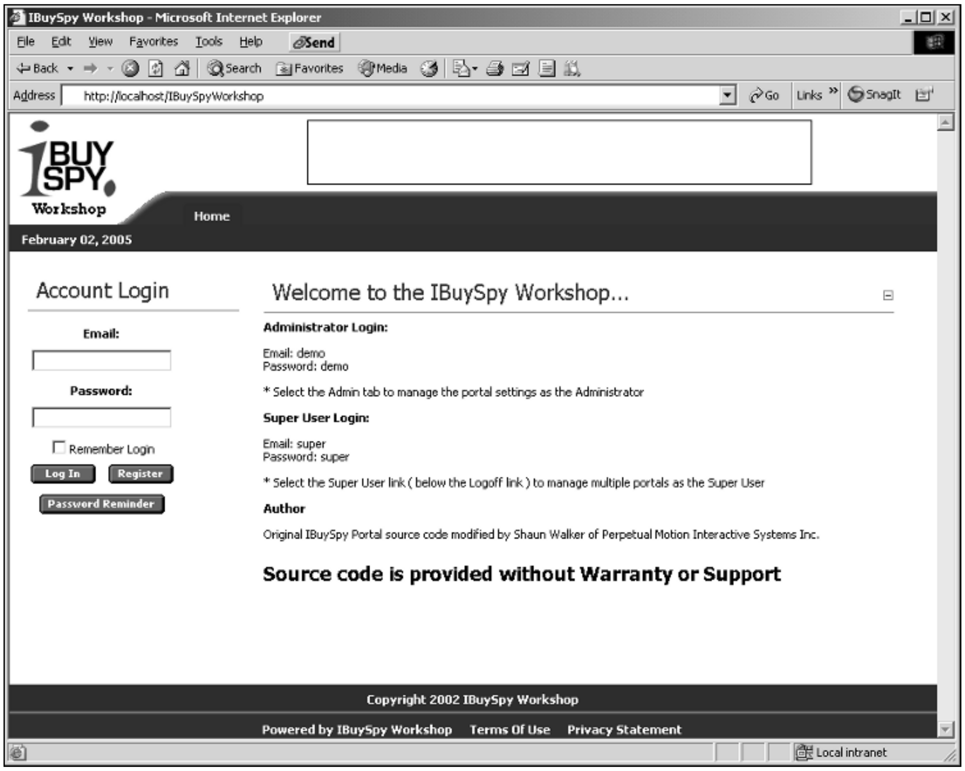
postreply
edit post
Moderate Post: 111390 [Delete Post | Edit Post | Moderation History | Move Post]

Figure 1-3

IBuySpy Workshop

The public release of the IBuySpy Workshop (see Figure 1-4) created such a surge in Forum activity that it was all I could do to keep up with the feedback, especially since this all occurred during the Christmas holidays. I had a family vacation booked for the first two weeks of January, and I left for Mexico on January 2, 2003 (one week after the initial IBuySpy Workshop release). At the time, the timing of this family vacation seemed very poor as the groundswell of interest in the IBuySpy Workshop seemed like it could really use my dedicated focus. However in hindsight, the timing could not have been better, because it proved that the community could support itself — a critical element in any open source project. When I returned home from vacation I was amazed at the massive response the release had achieved. The IBuySpy Portal Forum became dominated with posts about the IBuySpy Workshop and my Inbox was full of messages thanking me for my efforts and requesting me for support and enhancements. This certainly validated my decision to release the application as an open source project, but also emphasized the fact that I had started a locomotive down the tracks and it was going to take some significant engineering to keep it on the rails.

Over the coming months I frantically attempted to incorporate all community suggestions into the application while at the same time keep up with the plethora of community support questions. Because I was working a day job that prevented effort on the open source project, most of my evenings were consumed with work on the IBuySpy Workshop, which definitely caused some strain on my marriage and family life. Four hours of sleep per night is not conducive to a healthy lifestyle but, like I said, the train was rolling and I had a feeling the project was destined for bigger things.


IBuySpy Workshop - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/IBuySpyWorkshop
IBUYSPY Workshop
Home
February 02, 2005
Local intranet

Account Login

Email:

Password:

Remember Login

Log In · Register · Password Reminder

Welcome to the IBuySpy Workshop...

Administrator Login:

Email: demo
Password: demo

* Select the Admin tab to manage the portal settings as the Administrator

Super User Login:

Email: super
Password: super

* Select the Super User link (below the Logoff link) to manage multiple portals as the Super User

Author

Original IBuySpy Portal source code modified by Shaun Walker of Perpetual Motion Interactive Systems Inc.

Source code is provided without Warranty or Support

Copyright 2002 IBuySpy Workshop

Powered by IBuySpy Workshop · Terms Of Use · Privacy Statement

Figure 1-4

Supporting a user base through upgrades is fundamental in any software product. This is especially true in open source projects where the application can evolve very quickly based on community feedback and technical advancements. The popular open source expression is that "no user should be left on an evolutionary dead-end." As luck would have it, I had designed a very reliable upgrade mechanism in the original sports management application, which I included in the IBuySpy Workshop code base. This feature allowed users of the application to easily migrate from one release version to the next — a critical factor in keeping the community engaged and committed to the evolution of the product.

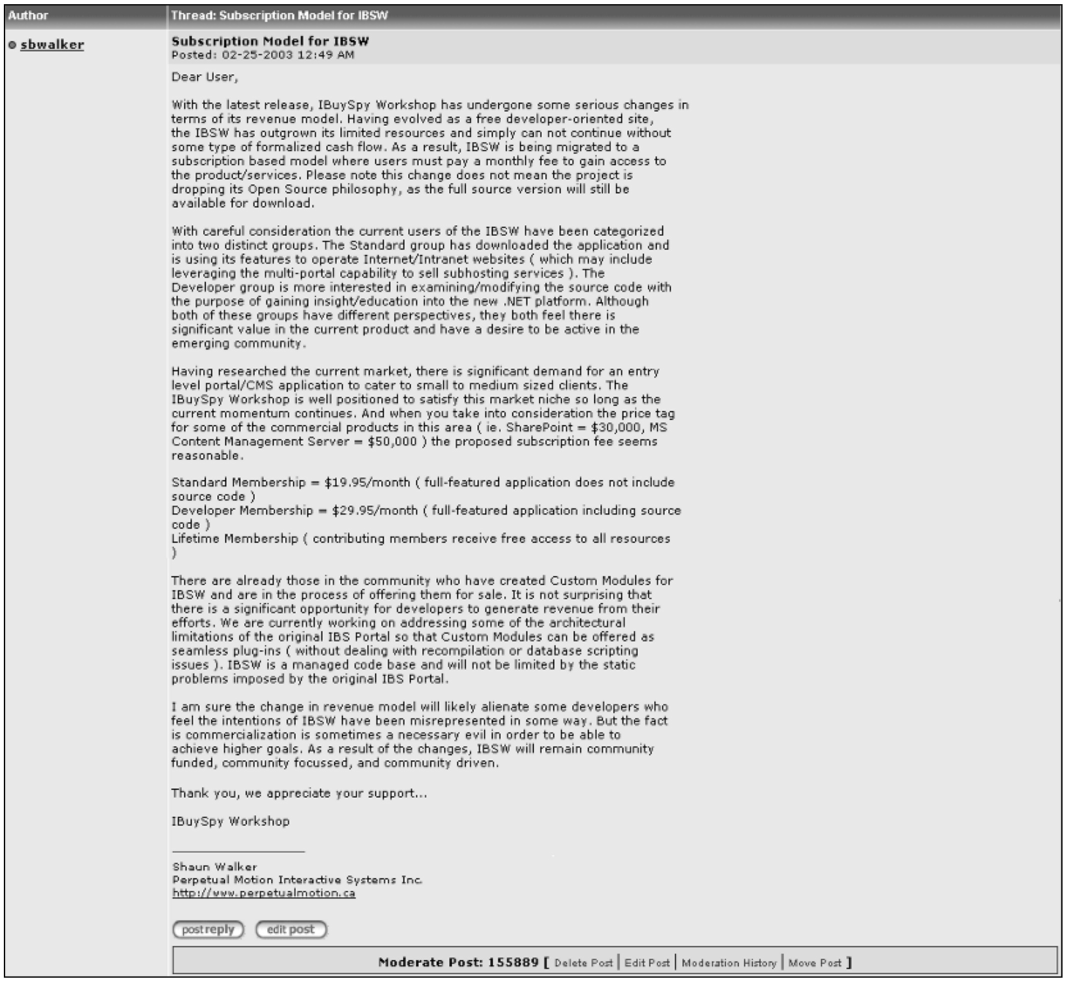
In February 2003, the IBuySpy Portal Forum had become so congested with IBuySpy Workshop threads that it started to become difficult for the two communities to co-exist peacefully. At this point, I sent an e-mail to the anonymous alias posted at the bottom of the Forums page on the `www.asp.net` site with a request to create a dedicated forum for the IBuySpy Workshop. Because the product functionality and source code of the two applications had diverged so significantly, my intent was to try and keep the

Forum posts for the two applications separate; providing both communities the means to support their membership. I certainly did not have high hopes that my e-mail request was even going to be read — let alone granted. But to my surprise, I received a positive response from none other than Rob Howard (an ASP.NET icon), which proved to be a great introduction to a long-term partnership with Microsoft. Rob created the forum and even went a step further to add a link to the Source Download page of the `www.asp.net` site, an event that would ultimately drive a huge amount of traffic to the emerging IBuySpy Workshop community.

There are a number of reasons why the IBuySpy Workshop became so immediately popular when it was released in early 2003. The obvious reason is because the base application contained a huge number of enhancements over the IBuySpy Portal application that people could immediately leverage to build more powerful web sites. From a community perspective, the open source project provided a central management authority, which was dedicated to the ongoing growth and support of the application framework; a factor that was definitely lacking in the original IBuySpy Portal community. This concept of open source on the Microsoft platform attracted many developers; some with pure philosophical intentions, and others who viewed the application as a vehicle to further their own revenue-generating interests. Yet another factor, which I think is often overlooked, relates to the programming language on which the project was based. With the release of the .NET Framework 1.0, Microsoft had spent a lot of energy promoting the benefits of the new C# programming language. The C# language was intended to provide a migration path for C++ developers as well as a means to entice Java developers working on other platforms to switch. This left the Visual Basic and ASP 3.0 developer communities feeling neglected and somewhat unappreciated. The IBuySpy Workshop, with its core framework in VB.NET, provided an essential community ecosystem where legacy VB developers could interact, learn, and share.

In late February 2003, the lack of sleep, family priorities, and community demands finally came to a head and I decided that I should reach out for help. I contacted a former employer and mentor, Kent Alstad, with my dilemma and we spent a few lengthy telephone calls brainstorming possible outcomes. However, my personal stress level at the time and my urgency to change direction on the project ultimately caused me to move too fast and with more aggression than I should have. I announced that the IBuySpy Workshop would immediately become a subscription service where developers would need to pay a monthly fee in order to get access to the latest source code. From a personal perspective the intent was to generate enough revenue that I could leave my day job and focus my full energy on the management of the open source project. And with 2000 registered users, a subscription service seemed like a viable model (see Figure 1-5).

However, the true philosophy of the open source model immediately came to light and I had to face the wrath of a scorned community. Among other things I was accused of misleading the community, lying about the open source nature of the project, and letting my personal greed cloud my vision. For every one supporter of my decision there were 10 more who publicly crucified me as the evil incarnate. Luckily for me Kent had a trusted work associate named Andy Baron, a senior consultant at MCW Technologies and a Microsoft Most Valuable Professional since 1995, who has incredible wisdom when it comes to the Microsoft development community. Andy helped me craft a public apology message (see Figure 1-6), which managed to appease the community while at the same time restore the IBuySpy Workshop to full open source status.



Author | Thread: Subscription Model for IBSW

sbwalker

Subscription Model for IBSW
Posted: 02-25-2003 12:49 AM

Dear User,

With the latest release, IBuySpy Workshop has undergone some serious changes in terms of its revenue model. Having evolved as a free developer-oriented site, the IBSW has outgrown its limited resources and simply can not continue without some type of formalized cash flow. As a result, IBSW is being migrated to a subscription based model where users must pay a monthly fee to gain access to the product/services. Please note this change does not mean the project is dropping its Open Source philosophy, as the full source version will still be available for download.

With careful consideration the current users of the IBSW have been categorized into two distinct groups. The Standard group has downloaded the application and is using its features to operate Internet/Intranet websites (which may include leveraging the multi-portal capability to sell subhosting services). The Developer group is more interested in examining/modifying the source code with the purpose of gaining insight/education into the new .NET platform. Although both of these groups have different perspectives, they both feel there is significant value in the current product and have a desire to be active in the emerging community.

Having researched the current market, there is significant demand for an entry level portal/CMS application to cater to small to medium sized clients. The IBuySpy Workshop is well positioned to satisfy this market niche so long as the current momentum continues. And when you take into consideration the price tag for some of the commercial products in this area (ie. SharePoint = $30,000, MS Content Management Server = $50,000) the proposed subscription fee seems reasonable.

Standard Membership = $19.95/month (full-featured application does not include source code)
Developer Membership = $29.95/month (full-featured application including source code)
Lifetime Membership (contributing members receive free access to all resources)

There are already those in the community who have created Custom Modules for IBSW and are in the process of offering them for sale. It is not surprising that there is a significant opportunity for developers to generate revenue from their efforts. We are currently working on addressing some of the architectural limitations of the original IBS Portal so that Custom Modules can be offered as seamless plug-ins (without dealing with recompilation or database scripting issues). IBSW is a managed code base and will not be limited by the static problems imposed by the original IBS Portal.

I am sure the change in revenue model will likely alienate some developers who feel the intentions of IBSW have been misrepresented in some way. But the fact is commercialization is sometimes a necessary evil in order to be able to achieve higher goals. As a result of the changes, IBSW will remain community funded, community focussed, and community driven.

Thank you, we appreciate your support...

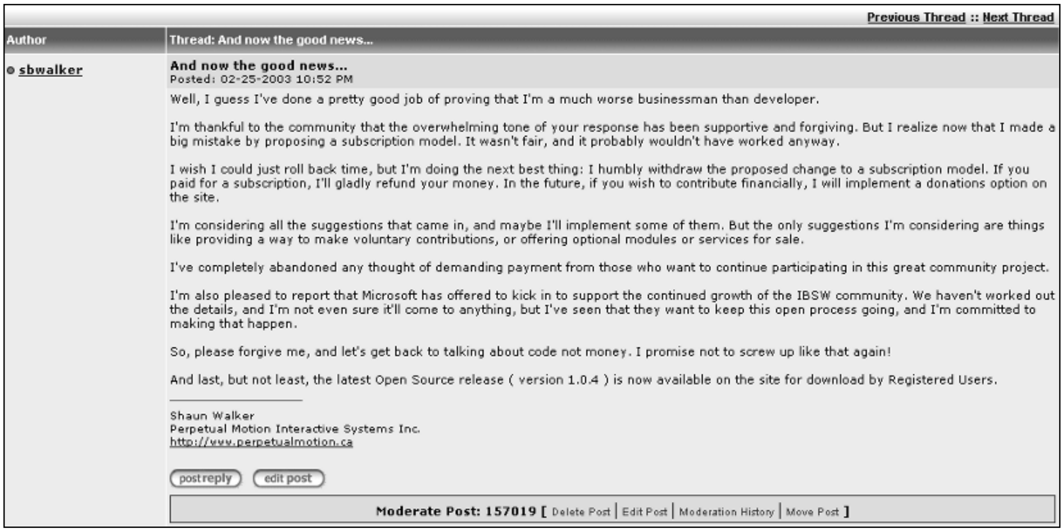
IBuySpy Workshop

Shaun Walker
Perpetual Motion Interactive Systems Inc.
http://www.perpetualmotion.ca

post reply
edit post
Moderate Post: 155889 [Delete Post | Edit Post | Moderation History | Move Post]

Figure 1-5

Coincidentally, the political nightmare I created in the IBuySpy Workshop Forum with my subscription announcement resulted in some direct attention from the Microsoft ASP.NET product team (the maintainers of the `www.asp.net` site). Still trying to recover from the damage I had incurred, I received an e-mail from none other than Scott Guthrie (co-founder of the Microsoft ASP.NET Team), asking me to reexamine my decision on the subscription model and making suggestions on how the project could continue as a free, open source venture. It seemed that Microsoft was protective of its evolving community and did not want to see the progress in this area splinter and dissolve just as it seemed to be gaining momentum. Scott Guthrie made no promises at this point but he did open a direct dialogue that ultimately led to some fundamental discussions on sponsorship and collaboration. In fact, this initial e-mail led to a number of telephone conversations and ultimately an invitation to Redmond to discuss the future of the IBuySpy Workshop.


Previous Thread :: Next Thread
Author
Thread: And now the good news...
sbwalker

And now the good news...
Posted: 02-25-2003 10:52 PM

Well, I guess I've done a pretty good job of proving that I'm a much worse businessman than developer.

I'm thankful to the community that the overwhelming tone of your response has been supportive and forgiving. But I realize now that I made a big mistake by proposing a subscription model. It wasn't fair, and it probably wouldn't have worked anyway.

I wish I could just roll back time, but I'm doing the next best thing: I humbly withdraw the proposed change to a subscription model. If you paid for a subscription, I'll gladly refund your money. In the future, if you wish to contribute financially, I will implement a donations option on the site.

I'm considering all the suggestions that came in, and maybe I'll implement some of them. But the only suggestions I'm considering are things like providing a way to make voluntary contributions, or offering optional modules or services for sale.

I've completely abandoned any thought of demanding payment from those who want to continue participating in this great community project.

I'm also pleased to report that Microsoft has offered to kick in to support the continued growth of the IBSW community. We haven't worked out the details, and I'm not even sure it'll come to anything, but I've seen that they want to keep this open process going, and I'm committed to making that happen.

So, please forgive me, and let's get back to talking about code not money. I promise not to screw up like that again!

And last, but not least, the latest Open Source release (version 1.0.4) is now available on the site for download by Registered Users.

Shaun Walker
Perpetual Motion Interactive Systems Inc.
http://www.perpetualmotion.ca

postreply
edit post
Moderate Post: 157019 [Delete Post | Edit Post | Moderation History | Move Post]

Figure 1-6

I still remember the combination of nerves and excitement as I drove from my home in Abbotsford, British Columbia to the Microsoft head office in Redmond, Washington (about a three-hour trek). I really did not know what to expect and I had tried to strategize all possible angles. Essentially all of my planning turned out to be moot — my meeting with Scott Guthrie turned out to be far more laid back and transparent than I could have ever imagined. Scott took me to his unassuming office and we spent the next three hours brainstorming ideas of how the IBuySpy Workshop fit into the current ASP.NET landscape. Much of this centered on the evolving vision of ASP.NET 2.0 — an area in which I had little or no knowledge prior to the meeting (the Whidbey Alpha had not even been released at this point).

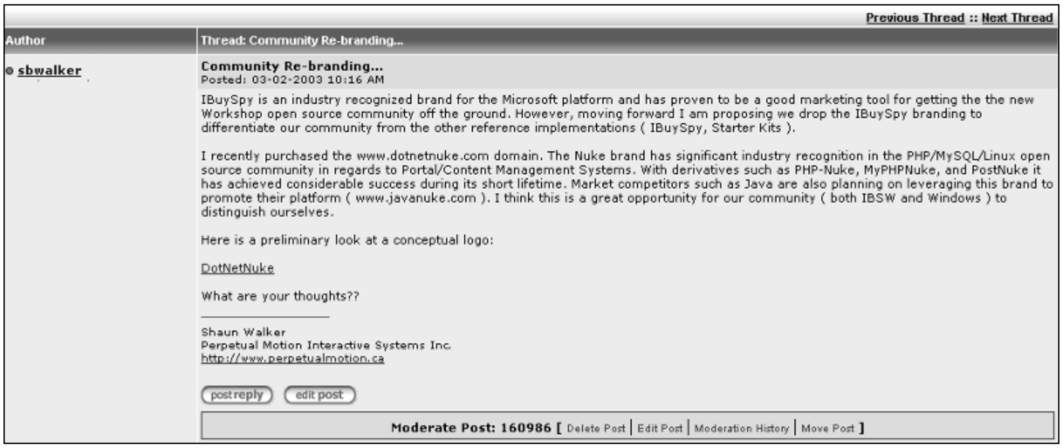
At the beginning of the meeting, Scott had me demo the current version of the IBuySpy Workshop, explaining its key features and benefits. We also discussed the long-term goals of the project as well as my proposed roadmap for future enhancements. Scott's knowledge of both the technical and community aspects of the ASP.NET platform really amazed me — I guess that's why he is the undisputed Father of ASP.NET. In hindsight I can hardly believe my good fortune to have received three dedicated hours of his time to discuss the project — it really changed my "ivory tower" perception of Microsoft and forged a strong relationship for future collaboration.

Upon leaving Redmond, I had to stifle my excitement as I realized that, regardless of the direct interaction with Microsoft, I personally was still in the exact same situation as before the subscription model announcement. Since the subscription model had failed to generate the much-needed revenue that would have allowed me to devote 100% of my time to the project, I was forced to examine other possible alternatives. There had been a number of suggestions from the community and the concept that seemed to have the most potential was related to web hosting.

In these early stages, there were very few economical Microsoft Windows hosting options available that offered a SQL Server database — a fundamental requirement for running the IBuySpy Workshop application. Coincidentally, I had recently struck up a relationship with an individual from New Jersey who was very active in the IBuySpy Workshop forums on `www.asp.net`. This individual had a solid background in web hosting and proposed a partnership whereby he would manage the web hosting infrastructure and I would continue to enhance the application and drive traffic to the business. Initially, a lot of community members signed up for this service; some because of the low-cost hosting option, others because they were looking for a way to support the open source project. It soon became obvious that the costs to build and support the infrastructure were consuming the majority of the revenue generated. And over time the amount of effort to support the growing client base became more intense. Eventually it came to a point where it was intimated that my contributions to the web hosting business were not substantial enough to justify the current partnership structure. I was informed that the partnership should be dissolved. This is where things got complicated because there had never been any formal agreement signed by either party to initiate the partnership. Without documentation, it made the negotiation for a fair settlement difficult and resulted in some bad feelings on both sides. This was unfortunate because I think the relationship was formed with the best intentions, but the demands of the business had resulted in a poor outcome. In any case, this ordeal was an important lesson I needed to learn; regardless of the open source nature of the project, it was imperative to have all contractually binding items properly documented.

One of the topics that Scott Guthrie and I discussed in our early conversations was the issue of product branding. IBuySpy Workshop had achieved its early goals of providing a public reference to the IBuySpy Portal community. This had resulted in an influx of ASP.NET developers who were familiar with the IBuySpy Portal application and were interested in this new open source concept. But as the code bases diverged there was a need for a new project identity — a unique brand that would differentiate the community and provide the mechanism for building an internationally recognized ecosystem. Research of competing portal applications on other platforms revealed a very strong tendency toward the "nuke" slogan.

The "nuke" slogan had originally been coined by Francisco Burzi of PHP-Nuke fame (the oft-disputed pioneer of open source portal applications). Over the years, a variety of other projects had adopted the slogan as well; so many that the term had obtained industry recognition in the portal application genre. To my surprise a WHOIS search revealed that dotnetnuke.com, .net, and .org had not been registered and, in my opinion, seemed to be the perfect identity for the project. Again emphasizing the bare bones resources under which the project was initiated, my credit card transaction to register the three domain names was denied and I was only able to register dotnetnuke.com (in the long run an embarrassing and contentious issue, as the .net and .org domain names were immediately registered by other individuals). Equally as spontaneous, I did an Internet search for images containing the word "nuke" and located a three-dimensional graphic of a circular gear with a nuclear symbol embossed on it. Contacting the owner of the site, I was given permission to use the image (it was in fact, simply one of many public domain images they were using for a fictitious store front demonstration). A new project identity was born — Version 1.0.5 of the IBuySpy Workshop was rebranded as DotNetNuke, which the community immediately abbreviated to DNN for simplicity (see Figure 1-7).


Previous Thread :: Next Thread
Author
Thread: Community Re-branding...
sbwalker
post reply
edit post
Moderate Post: 160986 [Delete Post | Edit Post | Moderation History | Move Post]

Community Re-branding...
Posted: 03-02-2003 10:16 AM

IBuySpy is an industry recognized brand for the Microsoft platform and has proven to be a good marketing tool for getting the the new Workshop open source community off the ground. However, moving forward I am proposing we drop the IBuySpy branding to differentiate our community from the other reference implementations (IBuySpy, Starter Kits).

I recently purchased the www.dotnetnuke.com domain. The Nuke brand has significant industry recognition in the PHP/MySQL/Linux open source community in regards to Portal/Content Management Systems. With derivatives such as PHP-Nuke, MyPHPNuke, and PostNuke it has achieved considerable success during its short lifetime. Market competitors such as Java are also planning on leveraging this brand to promote their platform (www.javanuke.com). I think this is a great opportunity for our community (both IBSW and Windows) to distinguish ourselves.

Here is a preliminary look at a conceptual logo:

DotNetNuke

What are your thoughts??

Shaun Walker
Perpetual Motion Interactive Systems Inc.
http://www.perpetualmotion.ca

Figure 1-7

A secondary issue that had not been addressed during the early stages of the project was licensing. The original IBuySpy Portal had been released under a very liberal Microsoft EULA license, which allowed for unrestricted usage, modification, and distribution. However, the code base had undergone such a major transformation, it could hardly be compared with its predecessor. Therefore, when the IBuySpy Workshop application was released I had not included the original Microsoft EULA; nor had I included any copyright or license of my own. Essentially this meant that the application was in the public domain. This is certainly not the most accepted approach to an open source project and eventually some of the more legal-savvy community members brought the issue to a head. I was forced to take a hard look at open source licensing models to determine which license was most appropriate to the project.

In stark contrast to the spontaneous approach taken to finding a project identity, the licensing issue had much deeper ramifications. Had I not performed extensive research on this subject, I would have likely chosen a GPL license because it seemed to dominate the vast majority of open source projects in existence. However, digging beneath the surface, I quickly realized that the GPL did not seem to be a good candidate for my objectives of allowing DotNetNuke to be used in both commercial and non-commercial environments. Ultimately, the selection of a license for an open source project is largely dependent upon your business model, your product architecture, and understanding who owns the intellectual property in your application. The combination of these factors prompted me to take a hard look at the open source licensing options available.

If you have not researched open source software, you would be surprised at the major differences between the most popular open source licensing models. It is true that these licenses all meet the standards of the Open Source Definition, a set of guidelines managed by the Open Source Initiative (OSI) at `www.open-source.org`. These principles include the right to use open source software for any purpose, the right to make and distribute copies, the right to create and distribute derivative works, the right to access and use source code, and the right to combine open source and other software. With such fundamental rights shared between all open source licenses it probably makes you wonder why there is need for more than one license at all. Well, the reason is because each license has the ability to impose additional rights or restrictions on top of these base principles. The additional rights and restrictions have the effect of altering the license so that it meets the specific objectives of each project. Because it is generally bad practice to create brand new licenses (based on the fact that the existing licenses have gained industry acceptance as well as a proven track record), people generally gravitate toward either a GPL or BSD license.

The GPL (or GNU Public License) was created in 1989 by Richard Stallman, founder of the Free Software Foundation. The GPL is what is now known as a "copyleft" license, a term coined based on its controversial reciprocity clause. Essentially this clause stipulates that you are allowed to use the software on condition that any derivative works that you create from it and distribute must be licensed to all under the same license. This is intended to ensure that the software and any enhancements to it remain in the public domain for everyone to share. While this is a great humanitarian goal, it seriously restricts the use of the software in a commercial environment.

The BSD (or Berkeley Software Distribution) was created by the University of California and was designed to permit the free use, modification, and distribution of software without any return obligation whatsoever on the part of the community. The BSD is essentially a "copyright" license, meaning that you are free to use the software on condition that you retain the copyright notice in all copies or derivative works. The BSD is also known as an "academic" license because it provides the highest degree of intellectual property sharing.

Ultimately I settled on a standard BSD license for DotNetNuke; a license that allows the maximum licensing freedom in both commercial and non-commercial environments — with only minimal restrictions to preserve the copyright of the project. The change in license went widely unnoticed by the community because it did not impose any additional restrictions on usage or distribution. However, it was a fundamental milestone in establishing DotNetNuke as a true open source project.

```
DotNetNuke - http://www.dotnetnuke.com
Copyright (c) 2002-2005
by Shaun Walker (sales@perpetualmotion.ca) of Perpetual Motion Interactive Systems
Inc. (http://www.perpetualmotion.ca)

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

The next major milestone in the project's open source evolution occurred in the summer of 2002. Up until this point I had been acting as the sole maintainer of the DotNetNuke code base; a task that was consuming 110% of my free time as I feverishly fixed bugs and enhanced the framework based on community feedback. Yet still I felt more like a bottleneck than a provider, in spite of the fact I was churning out at least one significant release every month leading up to this point. The more active community members were becoming restless due to a lack of direct input into the progress of the project. In fact, a small faction of these members even went so far as to create their own hybrid or "fork" of the DotNetNuke code base, which attempted to forge ahead and add features at a more aggressive pace than I was capable of on my own. These were very challenging times from a political standpoint because I was eventually forced to confront all of these issues in a direct and public manner — flexing my "Benevolent Dictator" muscles for the first time; an act I was not the least bit comfortable performing. Luckily for me, I had a number of very loyal and trustworthy community members who supported my position and ultimately provided the backing to form a very strong and committed Core Team.

As a result of the single-threaded issues I mentioned earlier, most successful open source projects are comprised of a number of community volunteers who earn their positions of authority within the community based on their specific expertise or community support activities. This is known as a meritocracy; a term which means that an individual's influence is directly proportional to the ability that the individual demonstrates within the project. It's a well-observed fact that individuals with more experience and skills will have less time to devote to volunteer activities; however, their minimal contributions prove to be incredibly valuable. Similarly, individuals with less experience may be able to invest more time but may only be capable of performing the more repetitive, menial tasks. Building a healthy balance of these two roles is exactly what is required in every successful open source project; and in fact, is one of the more challenging items to achieve from a management perspective.

The original DotNetNuke Core Team was selected based on their participation and dedication to the DotNetNuke project in the months leading up to the team's formation. In most cases this was solely based on an individual's public image and reputation that had been established in the DotNetNuke Forum on the `www.asp.net` web site. And in fact, in these early stages, the online persona of each individual proved to be a good indicator of the specific skills they could bring to the project. Some members were highly skilled architects, others were seasoned developers, yet others were better at discussing functionality from an end-user perspective and providing quality support to their community peers.

In order to establish some basic structure for the newly formed Core Team, I attempted to summarize some basic project guidelines. My initial efforts combined some of the best Extreme Programming (XP) rules with the principles of other successful open source projects. This became the basis of the DotNetNuke Manifest document:

- ❑ Development is a Team Effort: The whole is exponentially greater than the sum of its parts. Large-scale open source projects are only viable if a large enough community of highly skilled developers can be amassed to attack a problem. Treating your users as co-developers is your most effective option for rapid code improvement and effective debugging.
- ❑ Build the right product before you build the product right: Focus should be directed at understanding and implementing the high-level business requirements before attempting to construct the perfect technical architecture. Listen to your customers.
- ❑ Incremental Development: Every software product has infinite growth potential if managed correctly. Functionality should be added in incremental units rather than attempting a monolithic implementation. Release often but with a level of quality that instills confidence.
- ❑ Law of Diminishing Return: The majority of the effort should be invested in implementing features that have the most benefit and widest general usage by the community.

DotNetNuke version 1.0.10 was the proving grounds for the original Core Team. The idea was to establish the infrastructure to support disconnected team development by working on a stabilization release of the current product. A lot of debate went into the selection of the appropriate source control system because, ironically enough, many of the Core Team had never worked with a formal source control process in the past (a fact that certainly emphasized the varied professional background of the Core Team members). The debate centered on whether to use a CVS or VSS model.

CVS is a source control system that is very popular in the open source world that allows developers to work in an unrestricted manner on their local project files and handles any conflicts between versions when you attempt to commit your changes to the central repository. Visual SourceSafe (VSS) is a Microsoft source control system that is supported by the Microsoft development tool suite, which requires developers to explicitly lock project files before making modifications to prevent version conflicts. Ultimately, the familiarity with the Microsoft model won out and we decided to use the free WorkSpaces service on the GotDotNet web site (a new developer community site supported by Microsoft). GotDotNet also provided a simplistic Bug Tracker application that provided us with a means to manage the tracking of issues and enhancement requests. With these infrastructure components in place we were able to focus on the stabilization of the application; correcting known defects and adding some minor usability enhancements. It was during this time that Scott Willhite stepped forward to assume a greater role of responsibility in the project; assisting in management activities, communication, prioritization, and scheduling.

A significant enhancement that was introduced in this stabilization release came from a third party that had contacted me with some very specific enhancements they had implemented and wished to contribute. The University of Texas at El Paso had done extensive work making the DotNetNuke application compliant with the guidelines of the American Disabilities Association (ADA) and Section 508 of the United States Rehabilitation Act. The United States government had made compliancy mandatory for most public organizations; therefore, this was a great enhancement for DotNetNuke because it allowed the application to be used in government, educational, and military scenarios. Bruce Hopkins became the Core Team owner of this item in these early stages; a role that required a great deal of patience as the rest of the team came to grips with the new concept.

Establishing and managing a team was no small challenge. On one hand there were the technical challenges of allowing multiple team members, all in different geographic regions, to communicate and collaborate in a cost-effective, secure environment. Certainly this would have never been possible without the Internet and its vast array of online tools. On the other hand there was the challenge of identifying different personality types and channeling them into areas where they would be most effective. Because there are limited financial motivators in the open source model, people must rely on more basic incentives to justify their volunteer efforts. Generally this leads to a paradigm where contributions to the project become the de facto channel for building a reputation within the community — a primary motivator in any meritocracy. As a result of working with the team, it soon became obvious that there were two extremes in this area — those who would selflessly sacrifice all of their free time (often to their own detriment) to the open source project, and those who would invest the minimal effort and expect the maximum reward. As the creator and maintainer of the project it was my duty to remain objective and put the interests of the community first. This often caused me to become frustrated with the behavior of specific individuals, but in nearly all cases these issues could be resolved without introducing any hard feelings on either side. This was true in all cases except one.

Early in the project history I had been approached by an individual from Germany with a request to maintain a localized DotNetNuke site for the German community. I was certainly not naïve to the dangers of forking at this point and I told him that it would be fine so long as the site stayed consistent with the official source code base, which was under my jurisdiction. This was agreed upon and in the coming months I would have periodic communication with this individual regarding his localization efforts. However, as time wore on, he became critical of the manner in which the project was being managed; in particular the sole maintainer aspect, and began to voice his disapproval in the public Forum. There was a group who believed that there should be greater degree of transparency in the project; that developers should be able to get access to the latest development source code at anytime, and that the maintenance of the application should be conducted by a team rather than an individual. He was able to convince a number of community members to collaborate with him on a modified version of DotNetNuke; a version that integrated a number of the more popular community enhancements available, and called it DotNetNuke XXL.

Now I have to admit that much of this occurred due to my own inability to respond quickly and form a Core Team. In addition, I was not providing adequate feedback to the community regarding my goals and objectives for the future of the project. The reality is, the background management tasks of creating the DotNetNuke Core Team and managing the myriad other issues had undermined my ability to deliver source code enhancements and support to the. The combination of these factors resulted in an unpleasant situation; one that I should have mitigated sooner but was afraid to act upon due to the fragility of the newly formed community. And you also need to remember that the creator of the XXL variant had broken no license agreement by creating a fork; it was completely legal based on the freedom of the BSD open source license.

Eventually the issue came to a head when members of the XXL group began promoting their full source code hybrid in the DotNetNuke Forum. Essentially piggy-backing on the primary promotion channel for the DotNetNuke project, they were able to convince many people to switch to the XXL code base. This had some very bad consequences for the DotNetNuke community. Mainly it threatened to splinter the emerging community on territorial boundaries; an event I wanted to avoid at all costs. This situation was the closest attempt of project hijacking that I can realistically imagine. The DotNetNuke XXL fork seemed to be fixated on becoming the official version of DotNetNuke and assuming control of the future project roadmap. The only saving grace was that I personally managed the DotNetNuke infrastructure and therefore had some influence over key aspects of the open source environment.

In searching for an effective mechanism to protect the integrity of the community and prevent the XXL fork from gaining momentum, some basic business fundamentals came into play. Any product or service is only as successful as its promotion or marketing channel. The DotNetNuke Forum on the `www.asp.net` web site was the primary communication hub to the DotNetNuke community. Therefore it was not difficult to realize that restricting discussion on XXL in the Forum was the simplest method to mitigate its growth. In probably the most aggressive political move I have ever been forced to make, I introduced some bold changes to the DotNetNuke project. I established some guidelines for Core Team conduct that included, among other things, restrictions on promoting competing open source hybrids of the DotNetNuke application. I also posted some policies on the DotNetNuke Forum that emphasized that the Forum was dedicated solely to the discussion of the official DotNetNuke application and that discussion of third-party commercial or open source products was strictly forbidden. This was an especially difficult decision to make from a moral standpoint because I was well aware that the DotNetNuke application had been introduced to the community via the IBuySpy Portal Forum. Nonetheless, the combination of these two announcements resulted in both the resignation of the XXL project leader from the Core Team as well as the end of discussion of the XXL fork in the DotNetNuke Forum.

The unfortunate side effect, about which I had been cautioning members of the community for weeks, was that users who had upgraded to the XXL fork were effectively left on an evolutionary dead-end — a product version with no support mechanism or promise of future upgrades. This is because many of the XXL enhancements were never going to be integrated into the official DotNetNuke code base (either due to design limitations or inapplicability to the general public). This situation, as unpleasant as it may have been for those caught on the dead-end side of the equation, was a real educational experience for the community in general as they began to understand the longer term and deeper implications of open source project philosophy. In general, the community feedback was positive to the project changes, with only occasional flare ups in the weeks following. In addition, the Core Team seemed to gel more as a result of these decisions because it provided some much-needed policies on conduct, loyalty, and dedication, as well a concrete example of how inappropriate behavior would be penalized.

Emerging from the XXL dilemma, I realized that I needed to establish some legal protection for the long-term preservation of the project. Because standard copyright and the BSD license offered no real insurance from third-party threats, I began to explore intellectual property law in greater detail. After much research and legal advice, I decided that the best option was to apply for a trademark for the DotNetNuke brand name. Registering a trademark protects a project's name or logo, which is often a project's most valuable asset. Once the trademark was approved it would mean that although an individual or company could still create a fork of the application, they legally could not refer to it by the DotNetNuke trademark name. This appeared to be an important distinction so I proceeded with trademark registration in Canada (since this is the country in which Perpetual Motion Interactive Systems Inc. is incorporated).

I must admit the entire trademark approval process was quite an educational experience. Before you can register your trademark you need to define a category and description of your wares and/or services. This can be challenging, although most trademark agencies now provide public access to their database where you can browse for similar items that have been approved in the past. You pay your processing fee when you submit the initial application, but the trademark agency has the right to reject your application for any number of reasons; whereby, you need to modify your application and submit it again. Each iteration can take a couple of months, so patience is indeed a requirement. Once the trademark is accepted it must be published in a public trademark journal for a specified amount of time, providing third parties the opportunity to contest the trademark before it is approved. If it makes it through this final stage, you can pay your registration fee for the trademark to become official. To emphasize the lengthy process involved, the DotNetNuke trademark was initially submitted on October 9, 2003 and was finally approved on November 15, 2004 (TMA625,364).

In August 2003, I finally came to terms on an agreement with Microsoft regarding a sponsorship proposal for the DotNetNuke project. In a nutshell, Microsoft wanted DotNetNuke to be enhanced in a number of key areas; the intent being to use the open source project as a means of demonstrating the strengths of the ASP.NET platform. Because these enhancements were completely congruent with the future goals of the project, there was little negative consequence from a technical perspective. In return for implementing the enhancements, Microsoft would provide a number of sponsorship benefits to the project including web hosting for the `www.dotnetnuke.com` web site, weekly meetings with an ASP.NET Team representative (Rob Howard), continued promotion via the `www.asp.net` web site, and more direct access to Microsoft resources for mentoring and guidance. Please note that it took five months for this sponsorship proposal to come together, which demonstrates the patience and perseverance required to collaborate with such an influential partner as Microsoft. Nonetheless, this was potentially a one-time offer and at such a critical stage in the project evolution, it seemed too important to ignore.

An interesting perception that most people have in the IT industry is that Microsoft is morally against the entire open source phenomenon. In my opinion, this is far from the actual truth — and the reality is so much more simplistic. Like any other business that is trying to enhance its market position, Microsoft is merely concerned about competition. This is nothing new. In the past, Microsoft faced competitive challenges from many other sources: companies, individuals, and governments. However, the current environment makes it much more emotional and newsworthy to suggest that Microsoft is pitted against a grass roots community movement rather than a business or legal concern. So in my opinion, it is merely a coincidence that the only real competition facing Microsoft at this point in time is coming from the open source development community. And there is no doubt it will take some time and effort for Microsoft to adapt to the changing landscape. But the chances are probably high that Microsoft will eventually embrace open source to some degree in order to remain competitive.

When it comes to DotNetNuke, many people probably question why Microsoft would be interested in assisting an open source project for which they receive no direct benefit. And it may be perplexing why they would sponsor a product that competes to some degree with several of their own commercial applications. But you do not have to look much further than the obvious indirect benefits to see why this relationship has tremendous value. First and foremost, at this point in time the DotNetNuke application is only designed for use on the Microsoft platform. This means that in order to use DotNetNuke you must have valid licenses for a number of Microsoft infrastructure components (Windows operating system, database server, and so on). So this provides the financial value. In addition, DotNetNuke promotes the benefits of the .NET Framework and encourages developers to migrate to this new development platform. This provides the educational value. Finally, it cultivates an active and passionate community — a network of loyal supporters who are motivated to leverage and promote Microsoft technology on an international scale. This provides the marketing value.

So in September 2003, with the assistance of the newly formed Core Team, we embarked on an ambitious mission to implement the enhancements suggested by Microsoft. The problem at this point was that in addition to the Microsoft enhancements, there were some critical community enhancements, which I ultimately perceived as an even higher priority if the project should hope to grow to the next level. So the scope of the enhancement project began to snowball, and estimated release dates began to slip. The quality of the release code was also considered to be so crucial a factor that early beta packages were not deemed worthy of distribution. Ultimately the code base evolved so much that there was little question the next release would need to be labeled version 2.0. During this phase of internal development, some members of the Core Team did an outstanding job of supporting the 1.x community and generating excitement about the next major release. This was critical in keeping the DotNetNuke community engaged and committed to the evolving project.

A number of excellent community enhancements for the DotNetNuke 1.0 platform also emerged during this stage. This sparked a very active third-party reseller and support community; establishing yet another essential factor in any largely successful open source project. Unfortunately at this point the underlying architecture of the DotNetNuke application was not particularly extensible, which made the third-party enhancements susceptible to upgrade complications and somewhat challenging to integrate for end users. As a Core Team we recognized this limitation and focused on full modularity as a guiding principle for all future enhancements.

Modularity is an architecture principle that basically involves the creation of well-defined interfaces for the purpose of extending an application. The goal of any framework should be to provide interfaces in all areas that are likely to require customization based on business requirements or personalization based on individuality. DotNetNuke provides extensibility in the area of modules, skins, templates, data providers, and localization. And DotNetNuke typically goes one step beyond defining the basic interface; it actually provides the full spectrum of related resource services including creation, packaging, distribution, and installation. With all of these services available, it makes it extremely easy for developers to build and share application extensions with other members of the community.

One of the benefits of working on an open source project is the fact that there is a very high priority placed on creating the optimal solution or architecture. I think it was Bill Gates who promoted the concept of "magical software" and it is certainly a key aspiration in many open source projects. This goal often results in more preliminary analysis and design, which tends to elongate the schedule but also results in a more extensible and adaptable architecture. This differs from traditional application development, which often suffers from time and budget constraints, resulting in shortcuts, poor design decisions, and delivery of functionality before it is has been validated. Another related benefit is that the developers of open source software also represent a portion of its overall user community, meaning they actually "eat their own dog food," so to speak. This is really critical when it comes to understanding the business requirements under which the application needs to operate. Far too often you will find commercial vendors that build their software in a virtual vacuum; never experiencing the fundamental application use cases in a real-world environment.

One of the challenges in allowing the Core Team to work together on the DotNetNuke application was the lack of high-quality infrastructure tools. Probably the most fundamental elements from a software development standpoint were the need for a reliable source code control system and issue management system. Because the project had little to no financial resources to draw upon, we were forced to use whatever free services were available in the open source community. And although some of these services are leveraged successfully by other open source projects, the performance, management, and disaster recovery aspects are sorely lacking. This led to a decision to approach some of the more successful commercial vendors in these areas with requests for pro-bono software licenses. Surprisingly, these vendors were more than happy to assist the DotNetNuke open source project — in exchange for some minimal sponsorship recognition. This model has ultimately been carried on in other project areas to acquire the professional infrastructure, development tools, and services necessary to support our growing organization.

As we worked through the enhancements for the DotNetNuke 2.0 project, a number of Core Team members gained considerable respect within the project based on their high level of commitment, unselfish behavior, and expert development skills. Joe Brinkman, Dan Caron, Scott McCulloch, and Geert Veenstra sacrificed a heap of personal time and energy to improve the DotNetNuke open source project. And the important thing to realize is that they did so because they wanted to help others and make a difference, not because of self-serving agendas or premeditated expectations. The satisfaction of working with other highly talented individuals in an open, collaborative environment is reward enough for some developers. And it is this particular aspect of open source development that continues to confound and amaze people as time goes on.

In October 2003, there was a Microsoft Professional Developers Conference (PDC) in Los Angeles, California. The PDC is the premier software development spectacle for the Microsoft platform; an event that only occurs every two years. About a month prior to the event, I was contacted by Cory Isakson, a developer on the Rainbow Portal open source project, who informed me that "Open Source Portals" had been nominated as a category for a "Birds of Feather" session at the event. I posted the details in the DotNetNuke Forum and soon the item had collected enough community votes that it was approved as an official BOF session. This provided a great opportunity to meet with DotNetNuke enthusiasts and critics from all over the globe. It also provided a great networking opportunity to chat with the most influential commercial software vendors in the .NET development space (contacts made with SourceGear and MaximumASP at this event proved to be very important to DotNetNuke as time would tell).

In January 2004 another interesting dilemma presented itself. I received an e-mail from an external party, a Web Application Security Specialist, who claimed to have discovered a severe vulnerability in the DotNetNuke application (version 1.0). Upon further research I confirmed that the security hole was indeed valid and immediately called an emergency meeting of the more trusted Core Team members to determine the most appropriate course of action. At this point we were fully focused on the DotNetNuke 2.0 development project but also realized that it was our responsibility to serve and protect the growing DotNetNuke 1.0 community. From a technical perspective, the patch for the vulnerability proved to be a simple code modification. The more challenging problem was related to communicating the details of the security issue to the community. On the one hand we needed the community to understand the severity of the issue so that they would be motivated to patch their applications. On the other hand, we did not want to cause widespread alarm that could lead to a public perception that DotNetNuke was an insecure platform. Exposing too many details of the vulnerability would be an open invitation for hackers to try and exploit DotNetNuke web sites; but revealing too few details would downplay the severity. And the fact that the project is open source meant that the magnitude of the problem was amplified. Traditional software products have the benefit of tracking and identifying users through restrictive licensing policies. Open source projects have licenses that allow for free redistribution; which means the maintainer of the project has no way to track the actual usage of the application and no way to directly contact all community members who are affected. The whole situation really put security issues into perspective for me. It's one thing to be an outsider, expressing your opinions on how a software vendor should or should not react to critical security issues in their products. It's quite another thing to be an insider, stuck in the vicious dilemma between divulging too much or too little information, knowing full well that both options have the potential to put your customers at even greater risk. Ultimately, we created a new release version and issued a general security alert that was sent directly to all registered users of the DotNetNuke application by e-mail and posted in the DotNetNuke Forum on `www.asp.net`.

```
Subject: DotNetNuke Security Alert

Yesterday we became aware of a security vulnerability in DotNetNuke.

It is the immediate recommendation of the DotNetNuke Core Team that all
users of DotNetNuke based systems download and install this security patch
as soon as possible. As part of our standard security policy, no further
detailed information regarding the nature of the exploit will be provided to
the general public.

This email provides the steps to immediately fix existing sites and mitigate
the potential for a malicious attack.

Who is vulnerable?

-- Any version of DotNetNuke from version 1.0.6 to 1.0.10d

What is the vulnerability?

A malicious user can anonymously download files from the server. This is not
the same download security issue which has been well documented in the past
whereby an anonymous user can gain access to files in the /Portals directory
if they know the exact URL. This particular exploit bypasses the file
security machanism of the IIS server completely and allows a malicious user
to download files with protected mappings (ie. *.aspx).
```

```
The vulnerability specifically *does not* enable the following actions:

-- A hacker *cannot* take over the server (e.g. it does not allow hacker
code to be executed on the server)

How to fix the vulnerability?

For Users:

{ Instructions on where to download the latest release and how to install }

For Developers:

{ Instructions with actual source code snippets for developers who had diverged
from the official DotNetNuke code base and were therefore unable to apply a general
release patch }

Please note that this public service announcement demonstrates the
professional responsibility of the Core Team to treat all possible security
exploits as serious and respond in a timely and decisive manner.

We sincerely apologize for the inconvenience that this has caused.

Thank you, we appreciate your support...

DotNetNuke - The Web of the Future
```

The security dilemma brings to light another often misunderstood paradigm when it comes to open source projects. Most open source projects have a license that explicitly states that there is no support or warranty of any kind for users of the application. And while this may be true from a purely legal standpoint, it does not mean that the maintainer of the open source application can ignore the needs of the community when issues arise. The fact is, if the maintainer did not accept responsibility for the application, the users would quickly lose trust and the community would dissolve. This implicit trust relationship is what all successful open source communities are based upon. So in reality, the open source license acts as little more than a waiver of direct liability for the maintainer. The DotNetNuke project certainly conforms to this model because we take on the responsibility to ensure that all users of the application are never left on an evolutionary dead-end and security issues are always dealt with in a professional and expedient manner.

After six months of development, including a full month of public beta releases and community feedback, DotNetNuke 2.0 was released on March 23, 2004. This release was significant because it occurred at VS Live! in San Francisco, California — a large-scale software development event sponsored by Microsoft and Fawcette publications. Due to our strong working relationship with Microsoft, I was invited to attend official press briefings conducted by the ASP.NET Team. Essentially this involved 6–8 private sessions with the leading press agencies (Fawcette, PC Magazine, Computer Wire, Ziff Davis, and others) where I was able to summarize the DotNetNuke project, show them a short demonstration, and answer their specific questions. The event proved to be spectacularly successful and resulted in a surge of new traffic to the community (now totaling more than 40,000 registered users).

DotNetNuke 2.0 was a hit. We had successfully delivered a high quality release that encapsulated the majority of the most requested product enhancements from the community. And we had done so in a manner that allowed for clean customization and extensibility. In particular the skinning solution in DotNetNuke 2.0 achieved widespread critical acclaim.

In DotNetNuke 1.x the user interface of the application allowed for very little personalization; essentially all DNN sites looked very much the same, a very negative restriction considering the highly creative environment of the World Wide Web. DotNetNuke 2.0 removed this restriction and opened up the application to a whole new group of stakeholders — web designers. As the popularity of portal applications had increased in recent years, the ability for web designers to create rich, graphical user interfaces had diminished significantly. This is because the majority of portal applications were based on platforms that did not allow for clear separation between form and function, or were architected by developers who had very little understanding of the creative needs of web designers. DotNetNuke 2.0 focused on this problem and implemented a solution where the portal environment and creative design process could be developed independently and then combined to produce a stunningly harmonious end-user experience. The process was not complicated and did not require the use of custom tools or methodologies. It did not take very long before we began to see DotNetNuke sites with richly creative and highly graphical layouts emerge — proving the effectiveness of the solution and creating a “can you top this” community mentality for innovative portal designs.

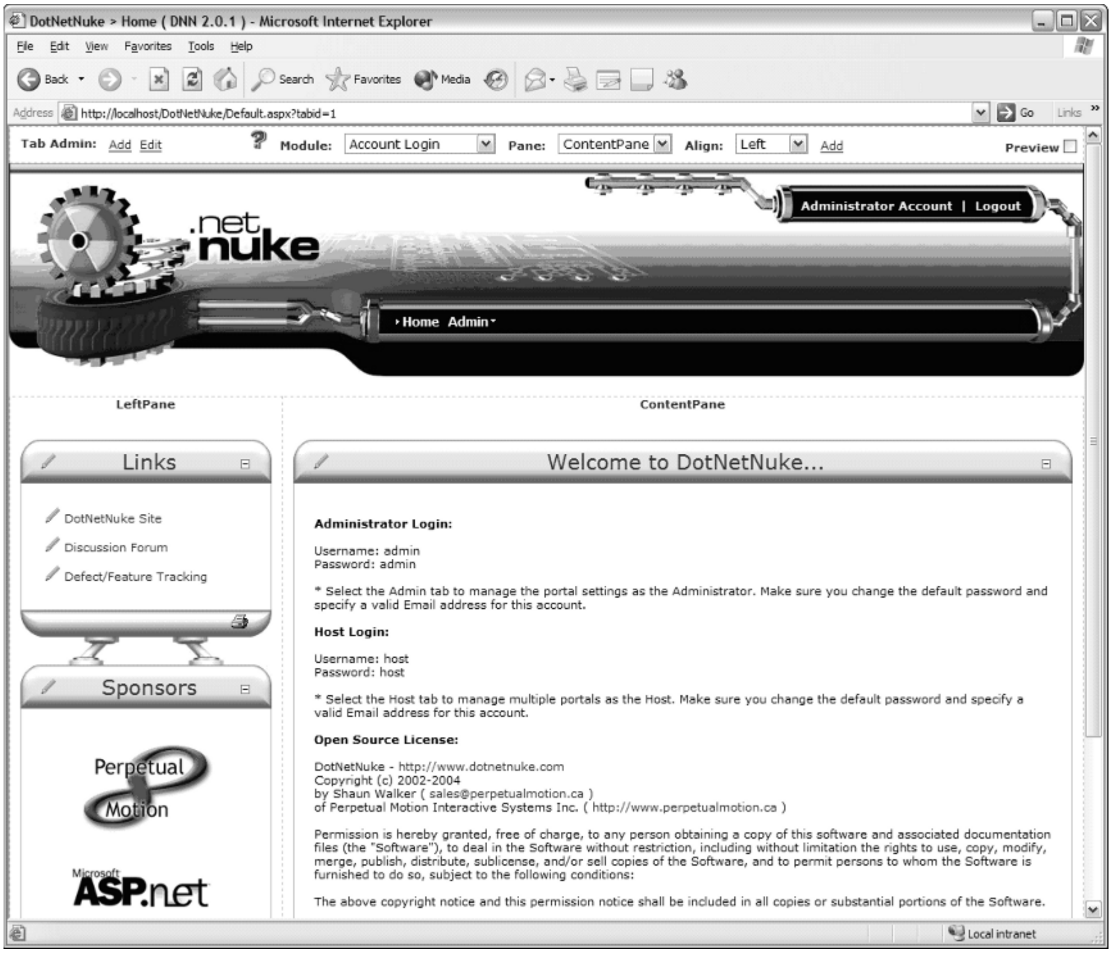
DotNetNuke (DNN) Web Site

To demonstrate the effectiveness of the skinning solution, I commissioned a local web design company, Circle Graphics, to create a compelling design for the `www.dotnetnuke.com` web site (see Figure 1-8). As an open source project, I felt that I could get away with an unorthodox, somewhat aggressive site design and I had been impressed by some of Circle Graphics’ futuristic, industrial concepts I had viewed in the past. It turned out that the designer who had created these visuals had since moved on but was willing to take on a small contract as a personal favor to the owner. He created a skin that included some stunning 3-D imagery including the now infamous “nuke-gear” logo, circuit board, and plenty of twisted metallic pipes and containers. The integration with the application worked flawlessly and the community was wildly impressed with the stunning result. Coincidentally, the designer of the DotNetNuke skin, Anson Vogt, has since gone on to bigger and better things, working with rapper Eminem as Art Director — 3-D Animation on the critically acclaimed Mosh video.

One of the large-scale enhancements that Microsoft had insisted upon for DotNetNuke 2.0 also proved to be very popular. The data access layer in DotNetNuke had been re-architected using an abstract factory model, which effectively allowed it to interface with any number of relational databases. Microsoft had coined the term “provider model” and emphasized it as a key component in the future ASP.NET 2.0 framework. Therefore, getting a reference implementation of this pattern in use in ASP.NET 1.x had plenty of positive educational benefits for Microsoft and DotNetNuke developers. DotNetNuke 2.0 included both a fully functional SQL Server and MS Access version, and the community soon stepped forward with mySQL and Oracle implementations as well. Again the extensibility benefits of good architecture were extremely obvious and demonstrated the direction we planned to pursue in all future product development.

Upon review of the DotNetNuke 2.0 code base it was very obvious that the application bore very little resemblance to the original IBuySpy Portal application. This was a good thing because it raised the bar significantly in terms of n-tiered, object-oriented, enterprise-level software development. However, it was also bad in some ways because it alienated some of the early DotNetNuke enthusiasts who were in fact "hobby programmers," using the application more as a learning tool than a professional product. This is an interesting paradigm to observe in many open source projects. In the early stages, the developer community drives the feature set and extensibility requirements which, in turn, results in a much higher level of sophistication in terms of system architecture and design. However, as time goes on, this can sometimes result in the application surpassing the technical capabilities of some of its early adopters. DotNetNuke had ballooned from 15,000 lines of managed code to 46,000 lines of managed code in a little over six months. The project was getting large enough that it required some serious effort to understand its organizational structure, its dependencies, and its development patterns.

When researching the open source phenomenon, I found that there are a few fundamental details are often ignored in favor of positive marketing rhetoric. I would like to take the opportunity to bring some of these to the surface because they provide some additional insight into some of the issues we face in the DotNetNuke project.



Administrator Login:

Username: admin
Password: admin

* Select the Admin tab to manage the portal settings as the Administrator. Make sure you change the default password and specify a valid Email address for this account.

Host Login:

Username: host
Password: host

* Select the Host tab to manage multiple portals as the Host. Make sure you change the default password and specify a valid Email address for this account.

Open Source License:

DotNetNuke - http://www.dotnetnuke.com
Copyright (c) 2002-2004
by Shaun Walker (sales@perpetualmotion.ca)
of Perpetual Motion Interactive Systems Inc. (http://www.perpetualmotion.ca)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Figure 1-8

The first myth surrounds the belief that open source projects basically have an unlimited resource pool at their immediate disposal. While this may be true from a purely theoretical perspective, the reality is you still require a dedicated management structure to ensure all of the resources are channeled in an efficient and productive manner. An army of developers without some type of central management authority will never consistently produce a cohesive application; and more likely, their efforts will result in total chaos. As much as the concept is often despised by hard core programmers, dedicated management is absolutely necessary to set expectations and goals, ensure product quality, mitigate risk, recognize critical dependencies, manage scope, and assume ultimate responsibility. You will find no successful open source project that does not have an efficient and highly respected management team.

Further to the unlimited resourcing myth, there are in fact very few resources who become involved in an open source project that possess the level of competency and communication skills required to earn a highly trusted position in the meritocracy. More often, the resources who get involved are capable of handling more consumer-oriented tasks such as testing, support, and minor defect corrections. And this is not to say that these resources do not play a critical role in the success of the project; every focused ounce of volunteer effort certainly helps sustain the health of the project. But my point is that there is usually a relatively small group on most open source projects who are responsible for the larger scale architectural enhancements.

Yet another myth is related to the belief that anyone can make a direct and immediate impact on an open source project. While this may be true to some degree, you generally need to build a trusted reputation within the community before you are granted any type of privilege. And there are very few individuals who are ever awarded direct write access to the source code repository. Anyone has the ability to submit a patch or enhancement suggestion; however, this does not mean it is guaranteed to be added to the open source project code base. In fact, all submissions are rigorously peer reviewed by trusted resources, and only when they have passed all validation criteria are they introduced to the source code repository. From a control standpoint, this is not much different than a traditional software project. However, the open source model does significantly alter this paradigm in the fact that everyone is able to review the source code. As a result, the sheer volume of patches submitted to this process can be massive.

There are also some interesting interpretations of open source philosophy that occasionally result in differences of opinion and, in the worst cases, all-out community revolts. This generally occurs because the guidelines for open source are quite non-explicit and subjective. One particularly hot topic that relates to DotNetNuke is related to source code access.

Some open source projects provide anonymous read-only access to the development source code base at all times. This full transparency is appreciated by developers who want to stay abreast of the latest development efforts — even if they are not trusted members of the inner project team. These developers accept the fact that the development code may be in various stages of stability on any given day; yet they appreciate the direct access to critical fixes or enhancements. Although this model does promote more active external peer review it can often lead to a number of serious problems. If a developer decides to use pre-release code in a production environment, they may find themselves maintaining an insecure or unstable application. This can lead to a situation where the community is expected to support many hybrid variants rather than a consistent baseline application. Another issue that can happen is that a developer who succumbs to personal motivations may be inclined to incorporate some of the development enhancements into the current production version and release it as a new application version. While the open source license may allow this, it seriously affects the ability for official project maintainer to support the community. It is the responsibility of the project maintainer to always ensure a managed migration path from one version to the next. This model can only be supported if people are forced to use the official baseline releases offered by the project maintainer. Without these constants to build from, upgrades become a manual effort and many users are left on evolutionary dead-ends. For these reasons, DotNetNuke chooses to restrict anonymous read access to the development source code repository. Instead we choose to issue periodic point releases that allow us to provide a consistent upgrade mechanism as the project evolves.

Following the success of DotNetNuke 2.0 we focused on improving the stability and quality of the application. Many production issues had been discovered after the release that we would have never anticipated during internal testing. As an application becomes more extensible, people find new ingenious ways to apply it, which can often lead to unexpected results. We also integrated some key Roadmap enhancements, which had already been developed in isolation by Core Team members. These enhancements were actually quite advanced as they added a whole new level of professional features to the DotNetNuke code base, transforming it into a viable enterprise portal application.

It was during this time that Dan Caron single-handedly made a significant impact on the project. Based on his experience with other enterprise portals, he proceeded to add integrated exception handling and event logging to the application. This added stability and "auditability"; two major factors in most professional software products. He also added a complex, multithreaded scheduler to the application. The Scheduler was not just a simple hard-coded implementation like I had seen in other ASP.NET projects, but rather it was fully configurable via an administration user interface. This powerful new feature could be used to run background housekeeping jobs as well as long-running tasks. With this in place, the extensibility of the application improved yet again.

An interesting concern that came to our attention at this time was related to our dependence on external components. In order to provide the most powerful application, we had leveraged a number of rich third-party controls for their expert functionality. Because each of these controls was available under their own open source license, they seemed to be a good fit for the DotNetNuke project. But the fact is, there are some major risks to consider. Some open source licenses are viral in nature and have the potential to alter the license of the application they are combined with. In addition, there is nothing that prevents a third party from changing their licensing policy at any time. If this situation occurred, then it is possible that all users of the application who reference the control could be in violation of the new license terms. This is a fairly significant issue and certainly not something that can be taken lightly. Based on this knowledge, we quickly came up with a strategy that was aimed at minimizing our dependency on third-party components. We constructed a policy whereby we would always focus on building the functionality ourselves before considering an external control. And in the cases where a component was too elaborate to replicate, we would use a provider model, much like we had in the database layer, to abstract the application from the control in such a way that it would allow for a plug-in replacement. This strategy protects the community from external license changes and also provides some additional extensibility for the application.

With the great publicity on the `www.asp.net` web site following VS Live! and the consistent release of powerful new enhancements, the spring of 2004 brought a lot of traffic to the dotnetnuke.com community web site. At this point, the site was very poorly organized and sparse on content due to a lack of dedicated effort. Patrick Santry had been on the Core Team since its inception and his experience with building web sites for the ASP.NET community became very valuable at this time. We managed to make some fairly major changes to improve the site, but I soon realized that a dedicated resource would be required to accomplish all of our goals. But without the funding to secure such a resource, many of the plans had to unfortunately be shelved.

The summer of 2004 was a restructuring period for DotNetNuke. Thirty new community members were nominated for Core Team inclusion and the Core Team itself underwent a reorganization of sorts. The team was divided into an Inner Team and an Outer Team. The Inner Team designation was reserved for those original Core Team individuals who had demonstrated the most loyalty, commitment, and value to the project over the past year. The Outer Team represented individuals who had earned recognition for their community efforts and were given the opportunity to work toward Inner Team status. Among other privileges, write access to the source code repository is the pinnacle of achievement in any source code project, and members of both teams were awarded this distinction to varying degrees.

In addition to the restructuring, a set of Core Team guidelines was established that helped formulize the expectations for team members. Prior to the creation of these guidelines it was difficult to penalize non–performers because there were not objective criteria by which they could be judged. In addition to the new recruits, a number of inactive members from the original team were retired; mostly to demonstrate that Core Team inclusion was a privilege — not a right. The restructuring process also brought to light several deficiencies in the management of intellectual property and confidentiality among team members. As a result, all team members were required to sign a retroactive nondisclosure agreement as well as an intellectual property contribution agreement. All of the items exemplified the fact that the project had graduated from its "hobby" roots to a professional open source project.

During these formative stages, I was once again approached by Microsoft with an opportunity to showcase some specific ASP.NET features. Specifically, a Membership API had been developed by Microsoft for Whidbey (ASP.NET 2.0), and they were planning on creating a backported version for ASP.NET 1.1, which we could leverage in DotNetNuke. This time the benefits were not so immediately obvious and required some thorough analysis. This is because DotNetNuke already had more functionality in these areas than the new Microsoft API could deliver. So in order to integrate the Microsoft components without losing any features, we would need to wrap the Microsoft API and augment it with our own business logic. Before embarking on such an invasive enhancement, we needed to understand the clear business benefit provided.

Well, you can never discount Microsoft's potential to impact the industry. Therefore, being one of the first to integrate and support the new Whidbey APIs would certainly be a positive move. In recent months there had been numerous community questions regarding the applicability of DotNetNuke with the early Whidbey Beta releases now in active circulation. Early integration of such a core component from Whidbey would surely appease this group of critics. From a technology perspective, the Microsoft industry had long been awaiting an API to converge upon in this particular area; making application interoperability possible and providing best practice due diligence in the area of user and security information. Integrating the Microsoft API would allow DotNetNuke to "play nicely" with other ASP.NET applications; a key factor in some of the larger scale extensibility we were hoping to achieve. Last, but not least, it would further our positive relationship with Microsoft; a factor that was not lost on most as the key contributor to the DotNetNuke project's growth and success.

The reorganization of the Core Team also resulted in the formation of a small group of highly trusted project resources which, for lack of a better term, we named the Board of Directors. The members of this group included Scott Willhite, Dan Caron, Joe Brinkman, Patrick Santry, and myself. The purpose of this group was to oversee the long-term strategic direction of the project. This included discussion on confidential issues pertaining to partners, competitors, and revenue. In August 2004, we scheduled our first general meeting for Philadelphia, Pennsylvania. With all members in attendance, we made some excellent progress on defining action items for the coming months. This was also a great opportunity to finally meet in person some of the individuals whom we had only experienced Internet contact with in the past. With the first day of meetings behind us, the second day was dedicated to sightseeing in the historic city of Philadelphia. The parallels between the freedom symbolized by the Liberty Bell and the software freedom of open source were not lost on any of us that day.

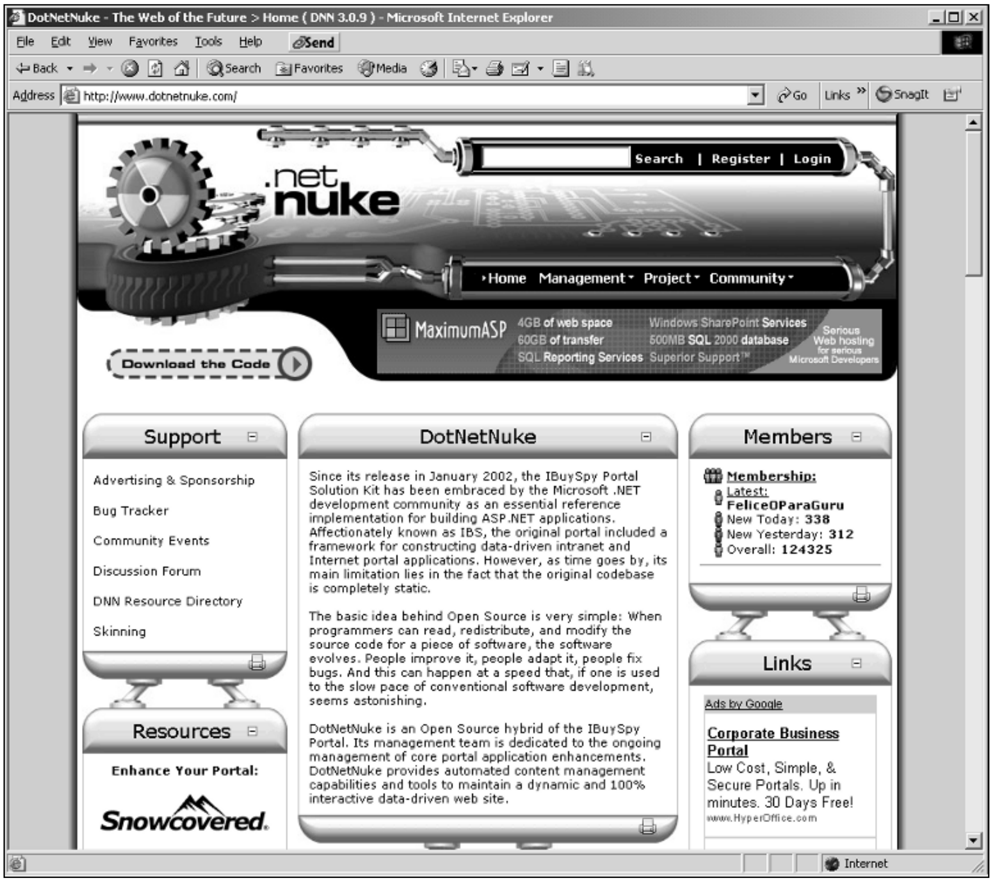
Returning from Philadelphia, I knew that I had some significant deliverables on my plate. We began the Microsoft Membership API integration project with high expectations of completion within three months. But as before there were a number of high-priority community enhancements that had been promised prior to the Microsoft initiative, and as a result the scope snowballed. Scope management is an extremely difficult task to manage when you have such an active and vocal community.

The snowball effect soon revealed that the next major release would need to be labeled version 3.0. This is mostly because of "breaking" changes; modifications to the DotNetNuke core application that changed the primary interfaces to the point that plug-ins from the previous version 2.0 release would not integrate without at least some minimal changes. The catalyst for this was due to changes in the Membership API from Microsoft, but this only led to a decision of "if you are forced to back compatibility then introduce all of your breaking changes in one breaking release." The fact is, there had been a lot of baggage that had been preserved from the IBuySpy Portal, which we had been restricted from removing due to legacy support considerations. DotNetNuke 3.0 provided the opportunity to reexamine the entire project from a higher level and make some of the fundamental changes we had been delaying for, in some cases, years. This included the removal of a lot of dead code and deprecated methods as well as a full namespace reorganization, which finally accurately broke the project API into logical components.

DotNetNuke 3.0 also demonstrated another technical concept that would both enrich the functionality of the portal framework as well as improve the extensibility without the threat of breaking binary compatibility. Up until version 3.0 the service architecture for DotNetNuke was completely uni-directional. Custom modules could consume the resources and services offered by the core DotNetNuke framework but not vice versa. So although the application managed the secure presentation of custom modules within the portal environment, it could not get access to the custom module content information. Optional interfaces allow custom modules to provide plug-in implementations for defined core portal functions. They also provide a simple mechanism for the core framework to call into third-party modules, providing a bi-directional communication channel so that modules can finally offer resources and services to the core (see Figure 1-9).

Along with its many technological advances, DotNetNuke 3.0 was also being groomed for use by an entirely new stakeholder, Web Hosters. For a number of years the popularity of Linux hosting has been growing at a far greater pace than Windows hosting. The instability arguments of early Microsoft web servers were beginning to lose their weight as Microsoft released more resilient and higher quality server operating systems. Windows Server 2003 had finally shed its clunky Windows NT 4.0 roots and was a true force to be reckoned with. So aside from the obvious economic licensing reasons, there was another clear reason why Hosters were still favoring Linux over Windows for their clients — the availability of end-user applications.

The Linux platform had long been blessed with a plethora of open source applications running on the Apache web server, built with languages such as PHP, Perl, and Python and leveraging open source databases such as mySQL. The Windows platform was really lacking in this area and was desperately in need of applications to fill this void.


DotNetNuke - The Web of the Future > Home (DNN 3.0.9) - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://www.dotnetnuke.com/
Go Links SnagIt
.net nuke
Search | Register | Login
Home Management Project Community
MaximumASP 4GB of web space 60GB of transfer SQL Reporting Services Windows SharePoint Services 500MB SQL 2000 database Superior Support™ Serious Web hosting for serious Microsoft Developers
Download the Code
Internet

Support

Advertising & Sponsorship
Bug Tracker
Community Events
Discussion Forum
DNN Resource Directory
Skinning

Resources

Enhance Your Portal:

Snowcovered.

DotNetNuke

Since its release in January 2002, the IBuySpy Portal Solution Kit has been embraced by the Microsoft .NET development community as an essential reference implementation for building ASP.NET applications. Affectionately known as IBS, the original portal included a framework for constructing data-driven intranet and Internet portal applications. However, as time goes by, its main limitation lies in the fact that the original codebase is completely static.

The basic idea behind Open Source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.

DotNetNuke is an Open Source hybrid of the IBuySpy Portal. Its management team is dedicated to the ongoing management of core portal application enhancements. DotNetNuke provides automated content management capabilities and tools to maintain a dynamic and 100% interactive data-driven web site.

Members

Membership:
Latest: FeliceOParaGuru
New Today: 338
New Yesterday: 312
Overall: 124325

Links

Ads by Google

Corporate Business Portal
Low Cost, Simple, & Secure Portals. Up in minutes. 30 Days Free!
www.HyperOffice.com

Figure 1-9

In order for DotNetNuke to take advantage of this opportunity, it needed a usability overhaul to transform it from a niche developer–oriented framework to a polished end-user product. This included a usability enhancement from both the portal administration as well as the web host perspectives. Since Rob Howard had left Microsoft in June 2004, my primary Microsoft contact had become Shawn Nandi. Shawn did a great job of drawing upon his usability background at Microsoft to come up with suggestions to improve the DotNetNuke end-user experience. Portal administrators received a multi-lingual user interface with both field-level and module-level help. Enhanced management functions were added in key locations to improve the intuitive nature of the application. Web Hosters received a customizable installation mechanism. In addition, the application underwent a security review to allow it to run in a Medium Trust — Code Access Security (CAS) environment. The end result is a powerful open source portal framework that can compete with the open source products on other platforms and offer Web Hosters a viable Windows alternative for their clients.

DotNetNuke is an evolving open source platform, with new enhancements being added constantly based on user feedback. The organic community ecosystem that has grown up around DotNetNuke is vibrant and dynamic, establishing the essential support infrastructure for long-term growth and prosperity. You will always be able to get the latest high-quality release including full source code from `www.dotnetnuke.com`.