1

An Inside Look at the Evolution of DotNetNuke

By Shaun Walker
Project Founder

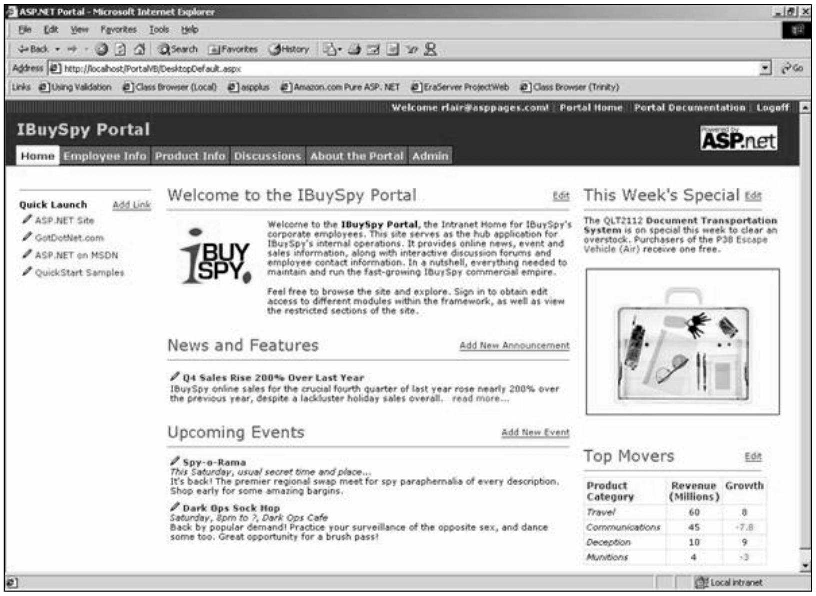
As much as DotNetNuke is an open source software application written for the Microsoft ASP.NET platform, it is also a vibrant community with developers, end users, vendors, and volunteers — all working together collaboratively in a rich and diverse ecosystem. This chapter attempts to capture the essence of the project, expose its humble beginnings, provide insight into its evolution, and document its many achievements, but not shy away from some of the hard lessons learned in the process. The lifeblood of any community is its people; therefore, it is a distinct honor and privilege to be able to share some of the emotion and passion that has gone into the DotNetNuke project so that you may be able to establish a personal connection with the various stakeholders and perhaps precipitate your own decision to join this burgeoning ecosystem.

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 primarily hired resources with Java skills — leaving me 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 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 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, and 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 built a number of source projects that 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 liberal End User License Agreement (EULA), which 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 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, middle pane, and 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, and 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
Powered by ASP.net
Home Employee Info Product Info Discussions About the Portal Admin
Local intranet

Quick Launch Add Link
- ASP.NET Site
- GotDotNet.com
- ASP.NET on MSDN
- QuickStart Samples

Welcome to the IBuySpy Portal Edit

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 Add New Announcement

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 Add New Event

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 Edit

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 Edit

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

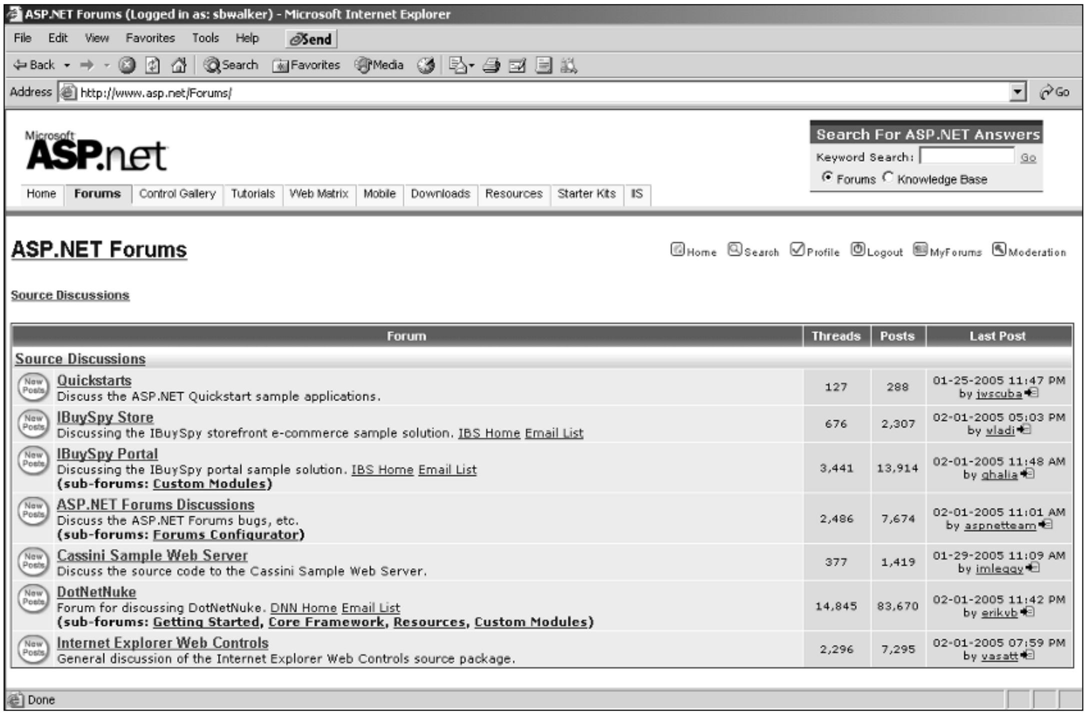
Figure 1-1

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.

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.

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'll explain later.


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

When I first started experimenting with the IBuySpy Portal application, I had some specific objectives in mind. 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 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 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 that was a perfect candidate for mapping a specific domain name to a portal. It was as if the original creators (Microsoft and Vertigo) considered this use case during development but did not have enough time to complete the implementation, so they 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.

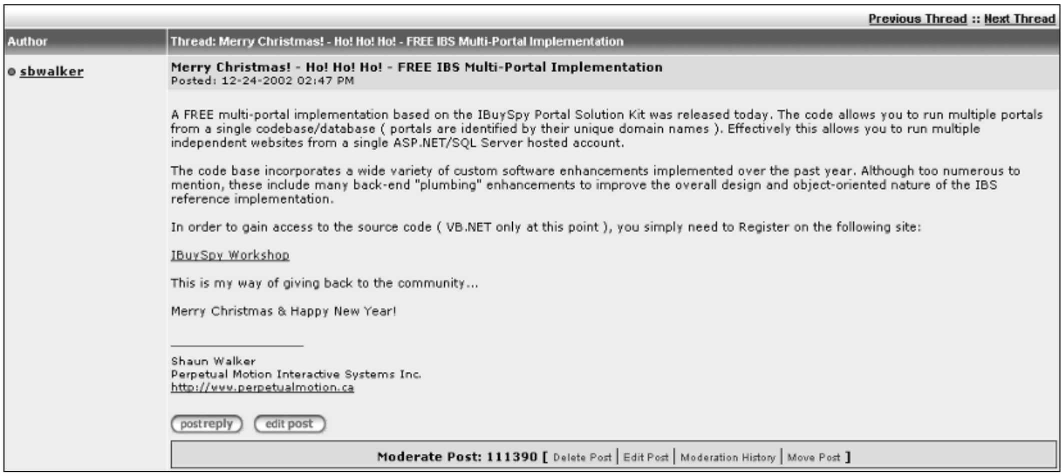
Over the next 8 to 10 months, I continued to enhance and refactor the IBuySpy Portal application as I created my own custom implementation (now code-named SportsManager.Net). I added numerous features to improve the somewhat limited portal administration and content management aspects. At one point, 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, because all of this was going on outside of regular work hours, there was little time that could be focused 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 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 were not 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 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 his 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 limited knowledge of the open-source philosophy at this point because all of my previous experience was in the Microsoft community — an area where "open source" was simply equated to the Linux operating system movement. However, there was chatter in the forums at various times regarding the organized sharing of source code, and there was obviously some interest in this area. The concept of incorporating the best enhancements into a rapidly evolving open-source application made a lot of sense because it benefited the entire community and created a wealth of opportunities for everyone. 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 a matter of fact, the initial decision to open source that 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 to 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

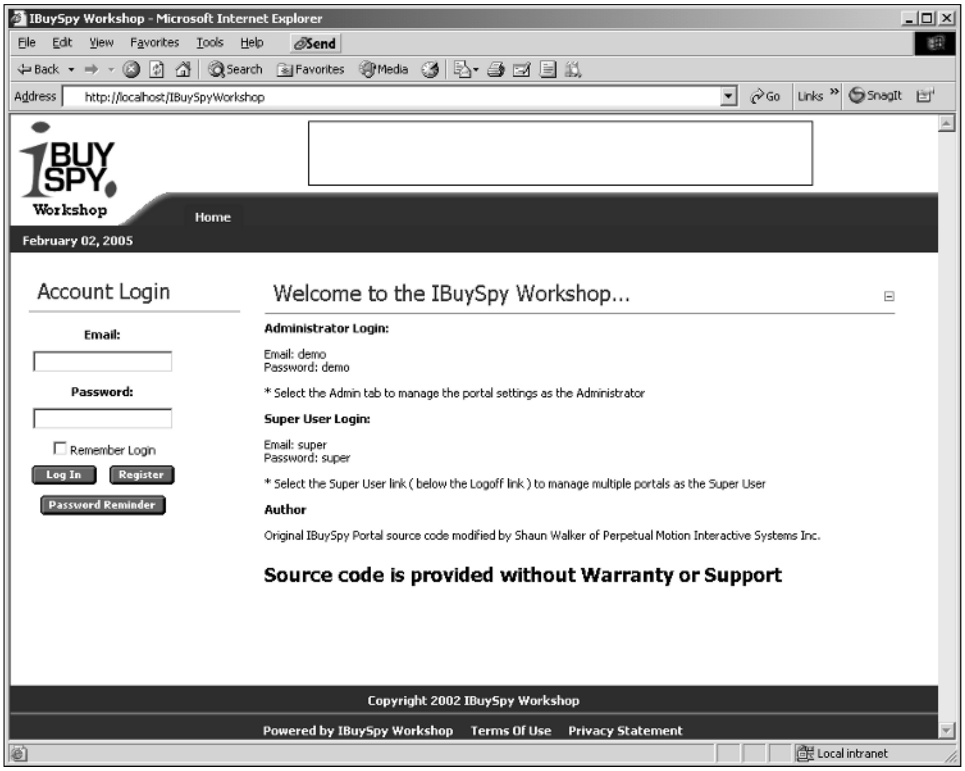
postreply
edit post
Moderate Post: 111390 [Delete Post | Edit Post | Moderation History | Move Post]

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 because 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 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 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 to provide 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 next few 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 Send
Back Search Favorites Media
Address http://localhost/IBuySpyWorkshop Go Links SnagIt
IBUY SPY Workshop
Home
February 02, 2005
Account Login
Email:
Password:
Remember Login
Log In
Register
Password Reminder
Local intranet

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 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 reliable upgrade mechanism in the original sports management application that I included in the IBuySpy Workshop code base. This feature enabled 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 diverged so significantly, my intent was to try and keep the forum posts for the two applications separated, 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 and added 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.

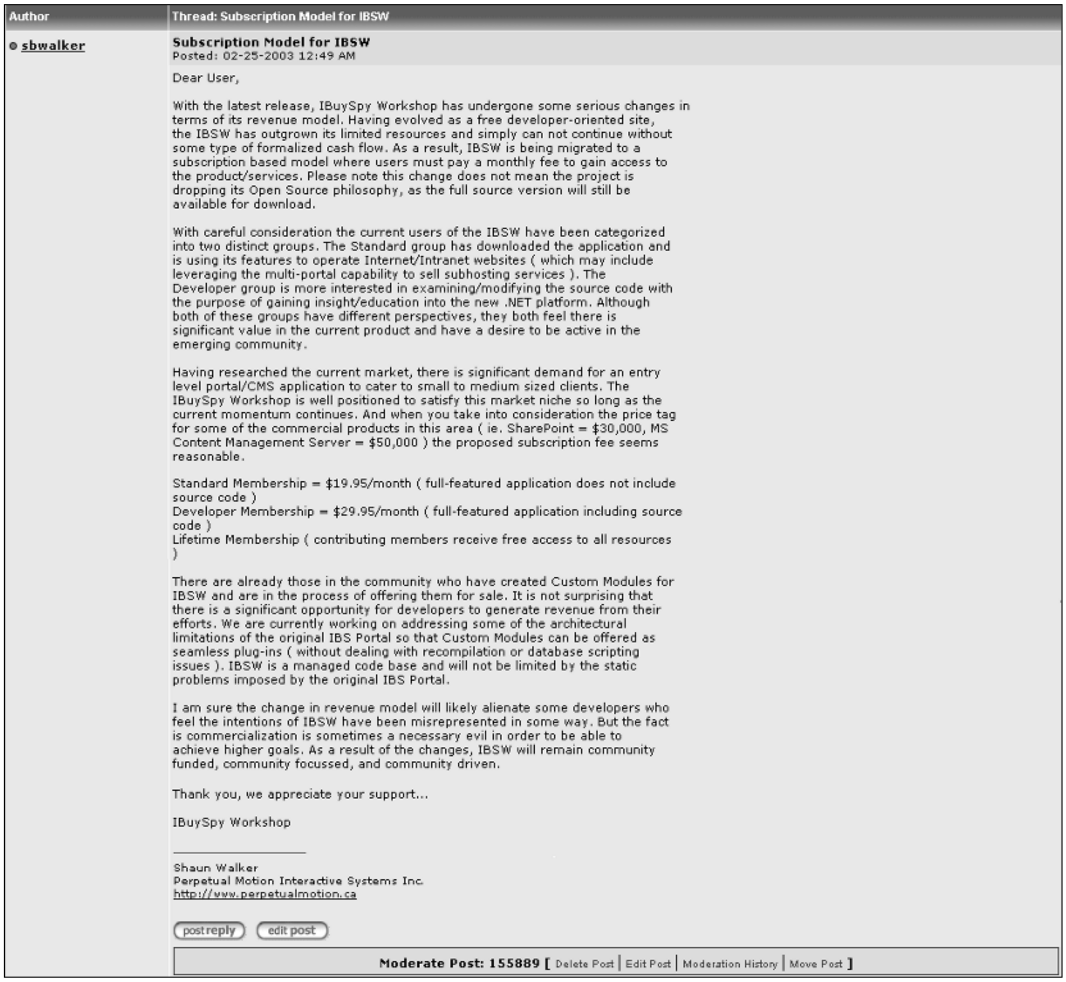
There are a number of reasons why the IBuySpy Workshop became so immediately popular when it was released in early 2003. The obvious reason is because the base application contained a huge number of enhancements over the IBuySpy Portal application, and people could immediately leverage them to build more powerful web sites. From a community perspective, the open source project provided a central management authority that 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 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.

Subscription Fiasco

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 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) that managed to appease the community and 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.

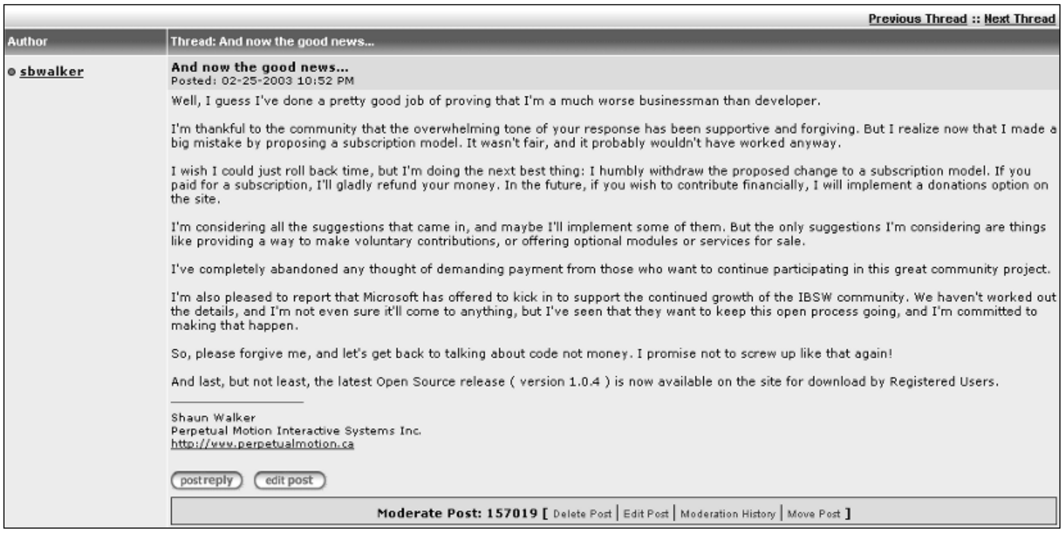
Thank you, we appreciate your support...

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



Previous Thread :: Next Thread
Author
Thread: And now the good news...
sbwalker
post reply
edit post
Moderate Post: 157019 [Delete Post | Edit Post | Moderation History | Move Post]

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

Figure 1-6

Microsoft

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 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.

I still remember the combination of nerves and excitement as I drove from my home in Abbotsford, British Columbia to Microsoft's head office in Redmond, Washington (about a three-hour trek). I really did not know what to expect, and I tried to strategize all possible angles. Essentially all of my planning turned out to be moot, because 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 about 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 where 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 demonstrate 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.

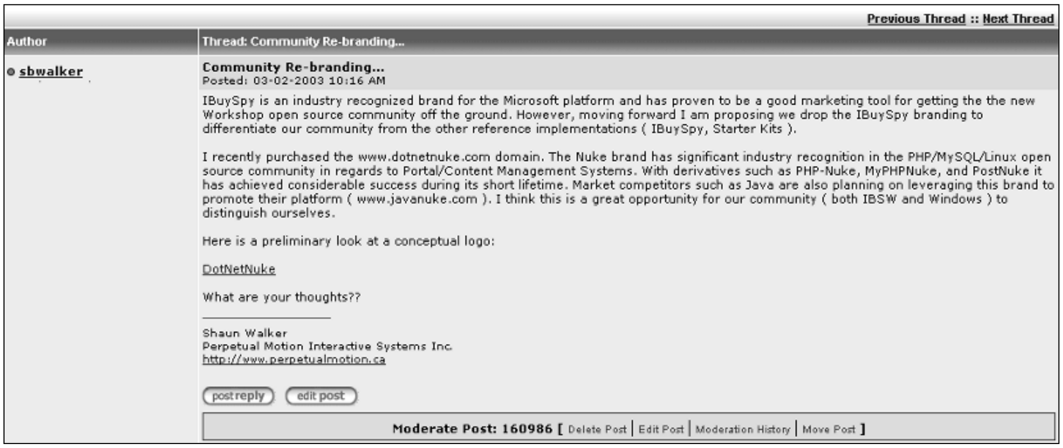
Upon leaving Redmond, I had to stifle my excitement as I realized that, regardless of the direct interaction with Microsoft, I personally was still in the same situation as before the subscription model announcement. Because the subscription model failed to generate the much-needed revenue that would have allowed me to devote 100% of my time to the project, I was forced to examine other possible alternatives. There were 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 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 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 there were a lot of community members who 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 was never 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 resulted in a poor outcome. Regardless, 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.

DotNetNuke

One of the topics that Scott Guthrie and I discussed in our early conversations was the issue of product branding. IBuySpy Workshop achieved its early goals of providing a public reference to the IBuySpy Portal community. This 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 strong tendency toward the "nuke" slogan.

The "nuke" slogan was originally 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 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` were not 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. I contacted the owner of the site and was given permission to use the image (it was in fact, simply one of many public domain images they were using for a fictitious storefront demonstration). A new project identity was born — Version 1.0.5 of the IBuySpy Workshop was re-branded 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

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

post reply
edit post
Moderate Post: 160986 [Delete Post | Edit Post | Moderation History | Move Post]

Figure 1-7

Licensing

A secondary issue that was not addressed during the early stages of the project was licensing. The original IBuySpy Portal was released under a liberal Microsoft EULA license that allowed for unrestricted usage, modification, and distribution. However, the code base underwent such a major transformation that it could hardly be compared with its predecessor. Therefore, when the IBuySpy Workshop application was released, I did not include the original Microsoft EULA, nor did I include 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 for 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.

For those of you who 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 the 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. Although 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 on the part of the community. The BSD is essentially a "copyright" license, meaning that you are free to use the software on the 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(r) - http://www.dotnetnuke.com
Copyright (c) 2002-2006
by 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.
```

Core Team

The next major milestone in the project's open source evolution occurred in the summer of 2003. 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. Still I felt more like a bottleneck than a provider, in spite of the fact that 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 that attempted to forge ahead and add features at a more aggressive pace than I was capable of on my own. These were 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 loyal and trustworthy community members who supported my position and ultimately provided the backing to form a 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 that 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 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 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, and others were better at discussing functionality from an end-user perspective and providing quality support to their community peers.

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 popular in the open source world that enables 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 who had contacted me with some 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 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 is true in all cases except one.

XXL Fork

Early in the project history, I was 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 had 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 that 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 community. 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 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 as 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. It is important to note that such a defensive move would not have been possible without the loyalty and support of the rest of the Core Team in terms of enforcing the guidelines.

The unfortunate side effect, one 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.

Trademarks

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 name. Registering a trademark protects a project's name or logo, which is often a project's most valuable asset. After 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 name. This appeared to be an important distinction so I proceeded with trademark registration in Canada (because 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. After 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).

Sponsorship

In August 2003, I finally came to 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. 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 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 sources — companies, individuals, and governments. However, the current environment makes it much more emotional and newsworthy to suggest that Microsoft is pitted against a grassroots 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 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 to remain competitive.

When it comes to DotNetNuke, many people probably question why Microsoft would be interested in assisting an open source project where it receives no direct benefit. And it may be perplexing why Microsoft would sponsor a product that competes to some degree with several of its 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 the DotNetNuke application is only designed for use on the Microsoft platform. This means that 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.

Enhancements

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 an 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 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 that tends to elongate the schedule but also results in a more extensible and adaptable architecture. This differs from traditional application development that often suffers from time and budget constraints, resulting in shortcuts, poor design decisions, and delivery of functionality before it is 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 find commercial vendors who 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 lot 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 occurs only every two years. About a month prior to the event Cory Isakson, a developer on the Rainbow Portal open source project, contacted me, saying 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 important to DotNetNuke, as time would tell).

Security Flaw

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, which 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 that 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 mechanism 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.

DotNetNuke 2.0

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 up to eight private sessions with the leading press agencies (Fawcette, PC Magazine, Computer Wire, Ziff Davis, and so on) 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).

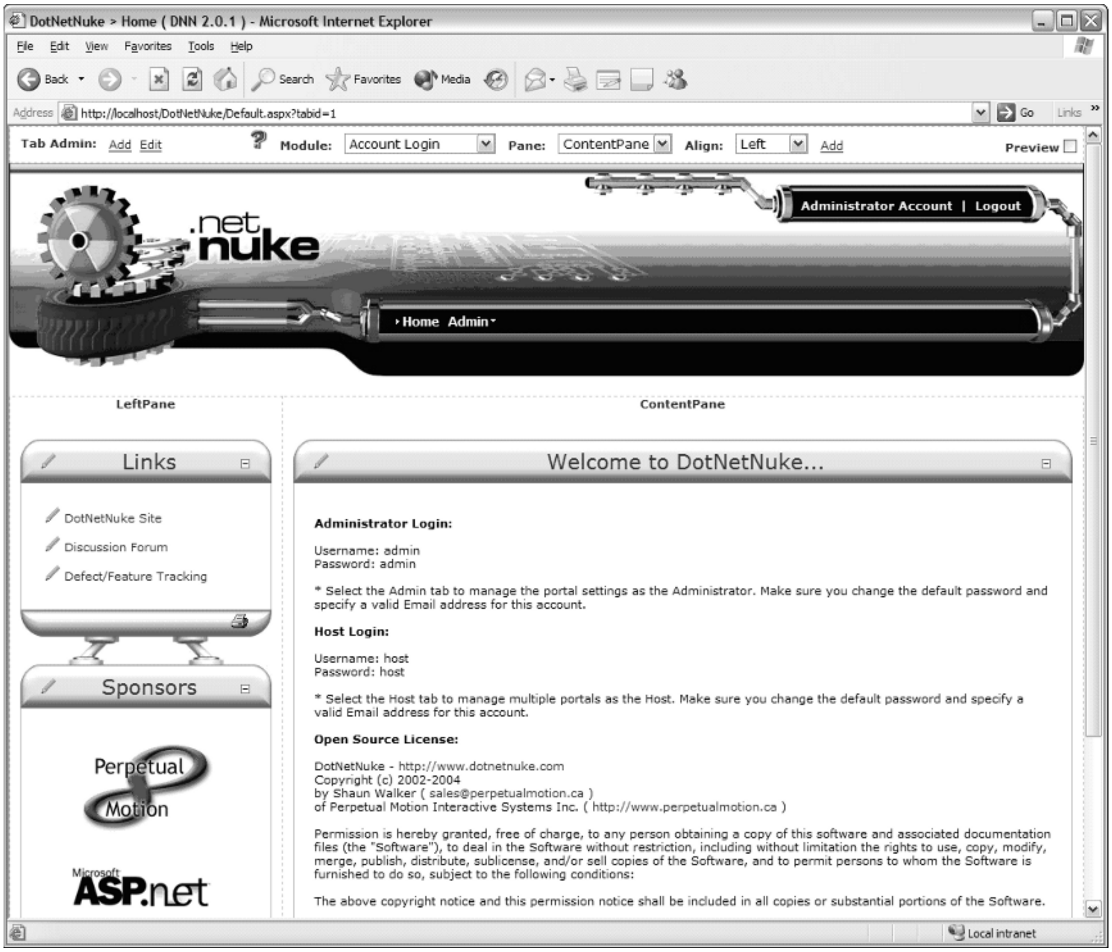
DotNetNuke 2.0 was a hit. We had successfully delivered a high-quality release that encapsulated the majority of the most requested product enhancements from the community. And we had done so in a manner that allowed for clean customization and extensibility. In particular, the skinning solution in DotNetNuke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

In DotNetNuke 1.X, the user interface of the application allowed for little personalization — essentially all DNN sites looked much the same, a 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 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 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 was impressed by some of Circle Graphic's futuristic, industrial concepts I had seen.

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 the Art Director for 3-D animation on the critically acclaimed Mosh video.



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

Provider Model

One of the large-scale enhancements that Microsoft insisted on for DotNetNuke 2.0 also proved to be popular. The Data Access Layer in DotNetNuke had been re-architected using an abstract factory model that effectively allowed it to interface with any number of relational databases. Microsoft 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 obvious that the application bore 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 that, 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 more than six months. The project was getting large enough that it required some serious effort to understand its organizational structure, dependencies, and development patterns.

Open Source Philosophy

When researching the open source phenomenon, there are a few fundamental details that 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.

The first myth surrounds the belief that open source projects basically have an unlimited resource pool at their immediate disposal. Although this may be true from a purely theoretical perspective, the reality is that you still require a dedicated management structure to ensure that 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.

Also with regards to the unlimited resourcing myth, there are in fact 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. 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. Although 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 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, there's no guarantee that it will 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. In addition, although a specific submission may appear to be quite useful when judged in isolation, there may be higher-level issues to consider in terms of upgrade support (a situation that can lead to submitter frustration if the issues are not fully explained). From a control standpoint, this is not much different than source control management on a traditional software project. However, the open source model does significantly alter this paradigm in 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 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 developers decide 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 in which the community is expected to support many hybrid variants rather than a consistent baseline application. Another possible issue 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. Although 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.

Stabilization

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

It was during this time that Dan Caron single-handedly made a significant impact on the project. Based on his experience with other enterprise applications, he proceeded to add integrated exception handling and event logging to DotNetNuke. This provided stability and "auditability" — two major factors in most professional software products. He also added a complex, multi-threaded 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 administrative 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.

Third-Party Components

An interesting concern that came to our attention at this time was related to our dependence on external components. 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 its 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 third parties from changing their licensing policy at any time. If this situation occurs, then it is possible that all users of the application who reference the control could be in violation of the new license terms. That's 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 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 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. Without the funding to secure such a resource, many of the plans had to unfortunately be shelved.

Core Team Reorganization

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 formalize the expectations for team members. Prior to the creation of these guidelines, it was difficult to isolate non–performers because there were no 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 non-disclosure 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.

Microsoft Membership API

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 that 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 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.

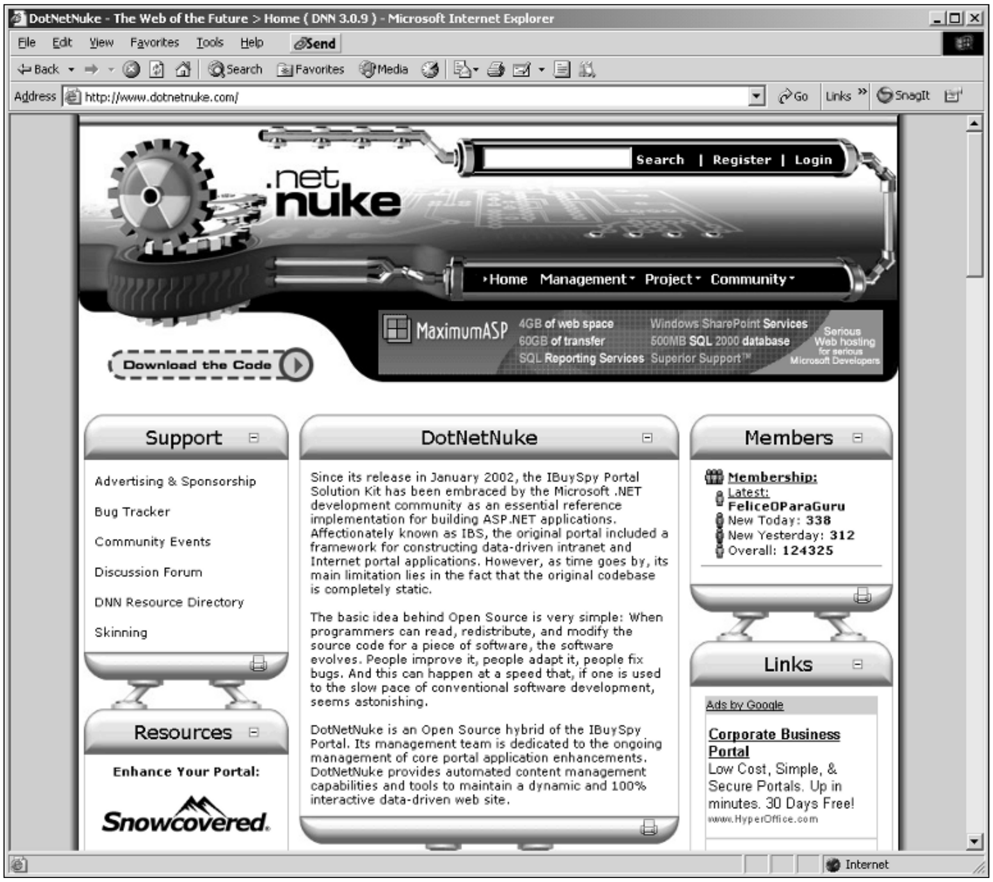
The reorganization of the Core Team also resulted in the formation of a small group of highly trusted project resources that, for lack of a better term, we named the Board of Directors. The members included Scott Willhite, Dan Caron, Joe Brinkman, Patrick Santry, and me. The purpose of this group was 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 with whom we had only experienced Internet contact 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 when you have such an active and vocal community.

"Breaking" Changes

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 break compatibility, introduce all of your breaking changes in one breaking release." The fact is there was a lot of baggage preserved from the IBuySpy Portal that we were 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 years in some cases. This included the removal of a lot of dead code and deprecated methods as well as a full namespace reorganization that 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 application 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 enable 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 could finally offer resources and services to the core (see Figure 1-9).


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
Support
Advertising & Sponsorship
Bug Tracker
Community Events
Discussion Forum
DNN Resource Directory
Skinning
Resources
Enhance Your Portal:
Snowcovered
DotNetNuke
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
Internet

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.

Figure 1-9

Web Hosters

Along with its many technological advances, DotNetNuke 3.0 was also being groomed for use by entirely new stakeholders: 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. 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 combination of these technologies is commonly referred to as LAMP.) The Windows platform was really lacking in this area and was desperately in need of applications to fill this void.

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 left Microsoft in June 2004, my primary Microsoft contact was 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 enable it to run in a Medium Trust — Code Access Security (CAS) environment. The end result was a powerful open source, web-application framework that could compete with the open source products on other platforms and offer Web Hosters a viable Windows alternative for their clients.

DotNetNuke 3.0

Much of the integration work on the Membership API and usability improvements were fueled by a much larger hosting initiative that Microsoft was preparing to unleash in May 2005. This initiative included a comprehensive program aimed at increasing awareness for Windows-based hosting solutions on an international level. Based on its strength as a framework for building consumer web sites, Microsoft invited DotNetNuke to participate in the program as long as it could meet a defined set of technical criteria, including Membership API integration, Medium Trust CAS compliance, localization, and usability improvements. Nearly all of the enhancements were already identified on the product roadmap, so the opportunity to be included in the hosting program was really a win-win proposition for the project and the community. In addition, we believed that the benefit of participating in such a large-scale initiative would be enormous in terms of lending credibility to the DotNetNuke product, introducing the project to influential new stakeholders, and helping to build brand equity.

Core Team members made significant contributions during the development of DotNetNuke 3.0. Scott McCulloch, with the assistance of Jeremy White, implemented a full-featured URL rewriting component that allowed DotNetNuke to use standard URLs. Vicenc Masanas was instrumental in working on localization, templating, and stabilization tasks. Joe Brinkman implemented search-engine architecture, enabling content indexing across all modules in a portal instance. Jon Henning introduced a Client API library, enabling powerful client-side behavior in DotNetNuke modules. Perhaps the greatest code contributions were made by Charles Nurse. Realizing the massive amount of work that would be required to deliver the enhancements for the hosting program (and knowing that using only volunteer efforts would not hit the schedule deadlines), I hired the first full-time DotNetNuke contract resource. Charles was immediately put to work abstracting all of the core modules into independent private assemblies. At the same time, he reorganized entry fields in all application user interfaces and added full localization capabilities, including field-level online help.

The concept of localization was one the most commonly requested enhancements for the DotNetNuke application. Localization actually has multiple meanings when it comes to software applications because there is a distinct difference between static and dynamic content. Static content is information that is delivered as part of the core application typically implemented by developers. Dynamic content is information that is provided by users of the application and is typically entered by knowledge workers or webmasters. In DotNetNuke 3.0, we delivered full static localization for all administrative interfaces. This meant that all labels, messages, and help text could be translated and displayed in different languages based on the preference of the user. Developing a scalable architecture in this area turned out to be a challenging task because the solutions offered by Microsoft as part of the ASP.NET 1.x framework were better suited for desktop applications and had serious deficiencies and limitations for web applications. Instead, we decided to target the ASP.NET 2.0 localization architecture, which better addressed the web scenario. However, due to the specific business requirements of DotNetNuke, we soon realized that we were going to have to take some liberties with the proposed ASP.NET 2.0 localization architecture to enable us to achieve our goals for runtime updatability and scalability in a shared hosting environment. In the end, we were able to deliver a powerful solution that satisfied our business needs and provided forward compatibility to the upcoming ASP.NET 2.0 release.

The optional interface architectural model described earlier reaped rewards in DotNetNuke 3.0 in a number of key application areas. Registration of module actions in earlier versions of DotNetNuke was always less than optimal because they were dependent on page life-cycle events that were difficult to manage in a variety of scenarios. Optional interfaces finally provided a clean mechanism for the core framework to programmatically call into modules and retrieve their module actions. Other new features based on optional interfaces included content indexing, import, and export. In each of these cases, the core framework could rely on modules to provide content in a specific format that then allowed the core framework to provide advanced portal services.

After multiple beta releases (some of which were deemed not fit for public consumption), DotNetNuke 3.0 was officially released on March 12, 2005. Although there were breaking changes between DotNetNuke 2.0 and DotNetNuke 3.0, a number of modules were immediately available for DotNetNuke 3.0 due to the success of a pilot program named “30 for 3.0.” This program was the shrewd strategy of Scott Willhite, and allowed a serious group of commercial module developers to have early access to beta releases of the DotNetNuke 3.0 product, enabling them to deal with any compatibility issues before the core framework became publicly available. Aside from the obvious benefits of having “applications” immediately available for the new platform, this program also provided some excellent business intelligence. It proved one of Scott’s earlier assumptions that the vocal forums community represented only a small portion of the overall DotNetNuke user community. It also exposed the fact that DotNetNuke had found its way into Fortune 500 companies, military applications, government web sites, international software vendors, and a variety of other high-profile installations.

DotNetNuke 3.0 was released with two supported languages: English and German. Delivering two complete language packs adhered to one of our newer philosophies of always attempting to provide multiple functional examples to prove the effectiveness of a particular extensibility model. Before long, community members began submitting new language packs in their native dialects that were posted on the `dotnetnuke.com` site for download. The total number of supported language packs soon surpassed 30. This resulted in incredible growth and adoption for the DotNetNuke framework on an international basis.

Release Schedule

A common open source concept is referred to as "release early, release often." The justification is that the sooner you release, the sooner the open source community can validate the functionality, and the sooner you get feedback — good and bad — which helps improve the overall product. This concept is often combined with a "public daily build" paradigm, where continuous integration is used to automatically build, package, and publish a new application version every day. These concepts make a lot of sense for single-purpose applications; that is, applications that have closed APIs and have no external dependencies. But plug-in framework applications such as DotNetNuke possess a different set of requirements, many of which are not complementary with the "release early, release often" model.

Consider the case of any entity that has developed plug-in resources for the DotNetNuke framework. These could include modules, language packs, skins, or providers. Every time a new core version is released, each of these resources needs to be validated to ensure that it functions correctly. In many cases, this involves extensive testing, packaging a new version of the specific resource, publishing compatibility information, updating related documentation, communicating availability and/or issues to users, servicing compatibility support requests, updating commercial product listings, and so on. You must also consider the issues for the resource consumer. Consumers need to feel confident in the acquisition and installation of application resources. They are not keen on analyzing complicated compatibility matrices to manage their investment. And resellers such as Hosters represent an even larger superset of application consumers. The effort involved to perform application upgrades becomes more complicated and costly as the release frequency increases. This is clearly a case where "release early, release often" can lead to issues for framework consumers and suppliers.

For these reasons, DotNetNuke has always tried to follow a fairly well-structured release cycle. This has resulted in fewer major public releases but a much higher quality, more stable, core application. In general, it has enabled DotNetNuke resource suppliers and consumers to participate in a functional product ecosystem. However, as the number of serious platform adopters increased, so did the demands for better core-release communication.

DotNetNuke Projects

One of the goals of the DotNetNuke 3.0 product release that had tremendous value for the community at large was the abstraction of the modules that were traditionally bundled with the core framework. The core modules were neglected in favor of adding more functionality to the core framework services. This resulted in a set of modules that demonstrated limited functionality and were not evolving at the same pace as the rest of the project. The abstraction of the modules from the core framework led to the formation of the DotNetNuke Projects program: a new organizational concept modeled after the Apache Foundation that allowed many complementary open source projects to thrive within the DotNetNuke ecosystem. From a technical perspective, the modules were abstracted in a manner that conformed to our extensibility model for building "private assembly" modules and allowed each module to be managed as its own independent project. The benefit was that each module could form its own team of developers, with its own roadmap for enhancements, and its own release schedule. As a governing entity, DotNetNuke would provide infrastructure services such as a source code repository, issue tracker, project home page, and e-mail services for the project as well as a highly visible and respected distribution and marketing channel.

Obviously there are tradeoffs that need to be accepted when decomposing a monolithic system into its constituent components, but the overall benefits of this approach reaped substantial rewards for the project. For one thing, it provided a new opportunity for developer participation — basically providing a sandbox where developers could demonstrate their skills and passion for the DotNetNuke project. This helped promote the "meritocracy" model and aided in our Core Team recruitment efforts. The community benefited through the availability of powerful, free, open source components that were licensed under the standard DotNetNuke BSD license. It also allowed the modules to evolve much more rapidly and with more focus than they ever received as part of the monolithic DotNetNuke application. Abstracting the core set of modules was a good start; however, the platform was lacking some other essential modules — modules that were well integrated and provided the common functionality required by most consumer web sites. These items included a discussion forum, blog, and photo gallery.

Early in the DotNetNuke 3.0 life cycle, there were discussions with a high-profile third-party software development company that was actively developing an integrated suite of components with forum, blog, and gallery functionality. Although early indications seemed to be positive regarding collaboration, they unfortunately did not comprehend the opportunity of working with the DotNetNuke community and ultimately decided to instead focus their efforts on constructing their own proprietary solution. Because this decision was not communicated to us until late in the DotNetNuke 3.0 development cycle, it meant that we had to scramble to find a suitable alternative. Luckily, two of our own Core Team members — Tam Tram Minh of TTT Corporation and Bryan Andrews of AppTheory — had been collaborating on a comparable set of modules and had already been offering them for free download to the DotNetNuke community. Discussions with them led to the creation of three powerful new DotNetNuke Projects: the DotNetNuke Forums, Blog, and Gallery.

Integrating third-party modules is not without its share of challenges. An "incubation" period is required to make the module conform to the official DotNetNuke project standards. An official marketing name must be defined for the project and all references to the old module name need to be updated. This includes namespaces, folder names, filenames, code comments, database object names, release package metadata, and documentation. To allow legacy users of the contributed module to be able to migrate to the new DotNetNuke project, a robust upgrade mechanism must be created. The module also needs to be reviewed to ensure that it does not contain any security flaws or serious defects that could affect the general community. From an infrastructure perspective, the code needs to be uploaded to a dedicated source code repository, an issue tracker project must be created, and a project home page complete with discussion forum and blog needs to be created on `dotnetnuke.com`. These tasks represent the technical integration issues that need to be addressed; but an item of even greater importance for third-party modules is management of the associated intellectual property.

Intellectual Property

There are two main contributing factors when it comes to intellectual property: copyright and licensing. The copyright holder is the person who owns the rights to the intellectual property. Normally this is the creator; however, copyright can also be transferred to other individuals or companies. The copyright holder has the right to decide how his intellectual property can be used by others. When it comes to software, these usage details are generally published as a license agreement. License agreements can vary a great deal depending on the environment, but they generally resemble a standard legal contract, explicitly outlining the rights and responsibilities of each party. The copyright holder also has the right to change the license for the intellectual property at their discretion. It is this scenario that requires the most due diligence when dealing with third-party contributions.

Anybody who contributes source code to the DotNetNuke project must submit a signed Contributor License Agreement. This document ensures that the individual has the right to contribute intellectual property to the project without any type of encumbrance. It also transfers copyright for any contributed intellectual property to the project. This is important because DotNetNuke needs to be able to ensure all of its intellectual property is licensed consistently throughout the entire application. It protects the community from a situation where an individual copyright holder could change the license restrictions for a specific piece of intellectual property, forcing the entire community into a reactive situation (a situation we have already seen multiple times in the still nascent Microsoft open source community).

In the case of third-party modules that are fully functional applications with an existing and active user base, the intellectual property rights are owned by the external party. Under this scenario, we cannot adopt the intellectual property into the DotNetNuke project because it would mean that we would have no control over its licensing. Even if the contributor agreed to license the intellectual property under a complementary BSD open source license, the original copyright holder would still have the ability to change the license at any time in the future, which would put all users of the module in jeopardy. To mitigate this risk, we require that DotNetNuke must have sufficient rights to the intellectual property so that the community is adequately protected. However, we do not feel it would be fair to force a contributor to release all of the rights to their own intellectual property. Therefore, we have a Software Grant Agreement — a contract that provides both parties with full copyright to the specified intellectual property. Essentially this means that the intellectual property has been split into two independent versions. The contributor owns one version and is allowed to license it or modify it as they see fit. DotNetNuke owns the other version and licenses it under the standard DotNetNuke BSD License for distribution and enhancement. The end result is a win-win situation for both parties as well as the community.

Marketing

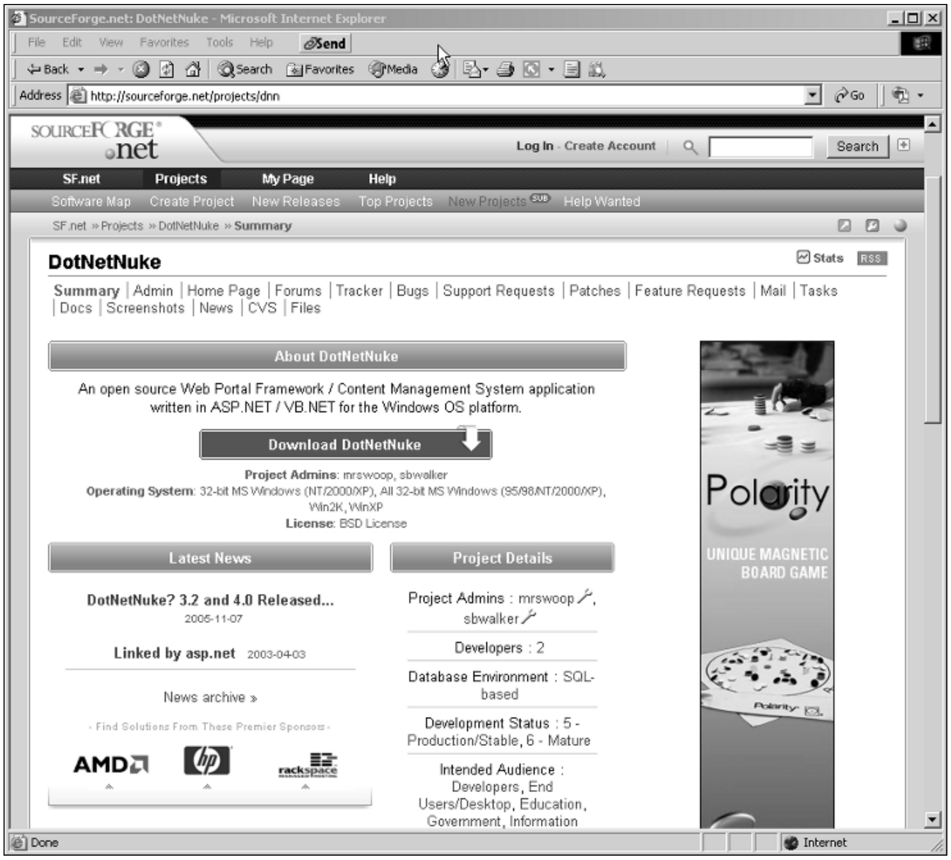
The success of any serious initiative must begin with the formulation of specific goals and the ability to measure progress as you work toward those goals. In terms of measuring the growth of the DotNetNuke project, we had traditionally monitored the total number of registered users on the `dotnetnuke.com` web site, the number of new users per month, and the number of downloads per month. These metrics revealed some definite trends but were rather myopic in terms of providing a relative comparison to other open source or commercial products. As a result, we looked for some other indicators that we could use to measure our overall market impact.

Alexa is a free service provided by Amazon that can be used to judge the popularity of an Internet web site. Popularity is an interesting metric because traffic distribution on the Internet conforms to a 90/10 rule: 10% of web sites account for 90% of the overall traffic, and 90% of web sites share the other 10%. This logarithmic scale means that it gets progressively more difficult to make substantial gains in your Alexa ranking as your web site popularity increases. Although the Alexa ranking is not a conventional progress indicator, we decided to use it as one of our key progress indicators (KPI) in determining the impact of our marketing efforts. The `dotnetnuke.com` web site had an Alexa ranking of 19,000 in April 2005.

SourceForge is the world's largest development and download repository of open source code and applications. Early in its project history, DotNetNuke had established a presence on SourceForge.Net (`http://sourceforge.net/projects/dnn` as shown in Figure 1-10) and continued to leverage its mirrored download infrastructure and bandwidth for hosting all project release packages. Because

SourceForge.Net contained listings for all of largest and most successful open source projects in existence, it also provided a variety of comparison and ranking statistics that could be used to judge activity and popularity. This seemed to be another good KPI to measure our impact in the open source realm. In April 2005, the DotNetNuke project had an overall project ranking of 1,271.

One of the items that had been neglected over the life of the project was the `dotnetnuke.com` web site. It had long been a goal to build this asset into a content-rich communication hub for the DotNetNuke community. Patrick Santry made some early progress in this area but recently found his volunteer time diminishing due to personal and family commitments. Because a web site is largely an extension of product marketing (another function that had long been ignored) the `dotnetnuke.com` web site suffered from sparse content, poor organization, and inconsistent focus. After the release of DotNetNuke 3.0, a significant effort was invested in improving all aspects of the web site. Much of the initial improvements came as a result of evaluating web sites of other open source projects. After extensive deliberation, we decided to organize the site information into three functional areas: user-oriented information, community collaboration, and developer information. New "sticky" content areas were added for project news and community events. The Home Page was completely revamped to provide summary marketing information and project metrics.


SourceForge.net: DotNetNuke - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://sourceforge.net/projects/dnn Go
SOURCEFORGE.net
Log In - Create Account Search
SF.net Projects My Page Help
Software Map Create Project New Releases Top Projects New Projects Help Wanted
SF.net » Projects » DotNetNuke » Summary
Stats RSS
Summary | Admin | Home Page | Forums | Tracker | Bugs | Support Requests | Patches | Feature Requests | Mail | Tasks | Docs | Screenshots | News | CVS | Files
About DotNetNuke
Download DotNetNuke
Latest News
Project Details
News archive »
- Find Solutions From These Premier Sponsors -
Polarity
UNIQUE MAGNETIC BOARD GAME
Done
Internet

DotNetNuke

An open source Web Portal Framework / Content Management System application written in ASP.NET / VB.NET for the Windows OS platform.

Project Admins: mrswoop, sbwalker
Operating System: 32-bit MS Windows (NT/2000/XP), All 32-bit MS Windows (95/98/NT/2000/XP), Win2K, WinXP
License: BSD License

DotNetNuke? 3.2 and 4.0 Released... 2005-11-07

Linked by asp.net 2003-04-03

Project Admins : mrswoop, sbwalker
Developers : 2
Database Environment : SQL-based
Development Status : 5 - Production/Stable, 6 - Mature
Intended Audience : Developers, End Users/Desktop, Education, Government, Information

Figure 1-10

In March 2005, another significant milestone occurred in DotNetNuke history. Dan Egan, a passionate DotNetNuke community member, wrote a book for PackT Publishing entitled *Building Websites with VB.NET and DotNetNuke 3.0*. This was the first book published about DotNetNuke and was essential in proving the demand for the product, paving the way for future DotNetNuke books from a variety of other publishers. In addition, a handful of Core Team members, including me, were also collaborating on a book for WROX Press during this time frame, but the demands of getting the DotNetNuke 3.0 product ready for release forced us to slip the publication date. Regardless, any technical content that makes it to mass publication through traditional channels lends an incredible amount of credibility and equity to the project or technology for which it is written. In addition, books can have a positive marketing impact; especially if they reach wide circulation through online retailers and brick-and-mortar bookstores.

In May of 2005, Core Team member Jim Duffy was successful in securing a DotNetNuke session on DotNetRocks!, an Internet radio talk show hosted by Carl Franklin and Richard Campbell. This was our second appearance on the show (the first being in August of 2004), and it was a lot of fun to talk about DotNetNuke in such a relaxed and open atmosphere. The show focused on the recent DotNetNuke 3.0 release and proved to be great way to promote some of the incredible new application features. It is hard to estimate the impact of the appearance on the DotNetRocks! show, but it certainly made me a firm believer in the benefits of podcasting as a powerful broad distribution marketing medium.

Microsoft Hosting Program

Throughout the month of May 2005, Microsoft launched the aforementioned Hosting program. The purpose of the program was to encourage shared hosting providers to take advantage of Windows technology to grow their hosting businesses. The primary benefit of this program was the Service Provider License Agreement (SPLA), which allowed hosting companies to avoid large capital expenditures and pay their licensing fees based on actual usage. This lowered the barrier of entry in terms of cost and provided a risk-free model to test the demand for services. In addition to the SPLA, Microsoft recognized the value of end-user applications and included substantial promotion of DotNetNuke in the hosting seminars encompassing thirty cities around the globe. I was fortunate enough to attend the first seminar in Redmond, Washington, which provided an excellent opportunity to network with the Microsoft Hosting Evangelists, a group of hard-working individuals who were dedicated to the growth of Windows web hosting on an international basis. At the beginning of June, I was also privileged to attend a WSHA seminar in Amsterdam, Netherlands. The invitation was extended by Microsoft Europe, which was especially interested in the localization capabilities of the DotNetNuke application. This trip gave me a deeper understanding of the localization challenges of the international community and also provided me the opportunity to meet Geert Veenstra and Leigh Pointer — two Core Team members who actively participated in and evangelized DotNetNuke since its creation.

Although the Microsoft Hosting program did not reap any direct financial rewards for DotNetNuke, it provided a number of powerful benefits. It exposed the application to an influential group of organizations: large-scale web hosting companies that dominate the shared hosting market in terms of customer base and annual revenues. Companies such as GoDaddy, Pipex, and 1and1 began offering DotNetNuke as part of their Windows hosting plans. The hosting program also caught the attention of the largest hosting control panel vendors. Companies such as SW-Soft (Plesk), WebHostAutomation (Helm), and Ensim added integrated installation support for the DotNetNuke application within their control panel applications. All of these strategic partnerships exposed DotNetNuke to a much larger consumer audience and would not have been possible had it not been for the Microsoft Hosting program.

Collaboration with web hosts also resulted in new application features that were added to satisfy some of their specific business requirements. The ability for DotNetNuke to run in a web farm environment was one such feature that really addressed the application scalability questions beyond a single web server configuration. Dan Caron stepped up yet again to champion these enhancements, producing an architecture with two different caching providers to satisfy the widest array of use cases. Charles Nurse also completed the abstraction of all modules into isolated components that could be optionally installed and uninstalled from the core framework. This change provided additional flexibility for Web Hosters in terms of being able to customize their offering for clients.

Infrastructure

One of the benefits of the original sponsorship agreement with Microsoft was a free shared hosting account on the servers managed by the ASP.NET team at OrcsWeb. This arrangement served us well in the early stages but the fact that we had extremely limited access (that is, FTP) to the account and absolutely no control over the associated infrastructure services eventually created some challenges for the project. In addition, we had long been leveraging services from PortalWebHosting for back office items such as DNS, source control, issue tracking, and e-mail, but a recent change in ownership created some friction in regards to legacy promises and agreements. Approaching premium hosting provider MaximumASP in the fall of 2004, we were able to secure a generous formal sponsorship agreement that paved the way for a more centralized and professionally managed project infrastructure.

Initially, MaximumASP provided us with two dedicated servers and a Virtual Private Server (VPS) account on a shared server. One of the dedicated servers was configured as a SQL Server database server and the other as a back office server. The VPS account was provisioned as a web account for our public web site. This configuration served us well initially, but the rapid growth of membership and the lack of control over the web server soon forced us to look for other options. Further discussions with MaximumASP resulted in the allocation of a dedicated web server for our public web site. The combination of a dedicated web server and a dedicated database server proved flexible enough to handle our full web site requirements. It was not until we added discussion forums to our site and pushed our traffic past 4 million page views a month that we felt the need to consider a web farm configuration.

The physical abstraction of the core application into a more modular organization had a direct impact on our back office project infrastructure. Rather than simply managing a single source code repository and issue tracking database, we now had to deal with many Project sandboxes — each with their own membership and security considerations. In addition, establishing effective communication channels for different stakeholder groups was critical for managing the project. This is one of the reasons why the DotNetNuke Forums Project played such a significant role in the evolution of the DotNetNuke projects. It allowed for a variety of discussion forums to be created, some public and some private, providing focused communication channels for project members.

During 2005, Scott Willhite also made some huge contributions to the project in terms of infrastructure management. In a project of this size with so many active participants, there is an incredible amount of administrative work that goes on behind-the-scenes to keep the project moving forward. As most people know, administrative tasks are largely unappreciated and only seem to get attention when there is a problem. Scott does his best to keep the endless stream of infrastructure tasks flowing; receiving little or no recognition for his efforts, but playing an instrumental role in the success of the DotNetNuke project.

Branding

One of the things that became obvious during the writing of *Professional DotNetNuke ASP.NET Portals* (Wiley Publishing, Inc.) was that our branding message was not clear. Although our trademark and domain name reflected "DotNetNuke," our logo contained an abbreviated terminology of ".netnuke." This led to confusion for authors of the book as well as the publisher in terms of what was the correct product branding. As I mentioned earlier in this chapter, the initial branding was constructed with little or no foresight; therefore, it came as no surprise that a major overhaul was necessary.

Initial conversations within the Core Team offered some interesting and sometimes surprising opinions on the DotNetNuke brand. When discussion came to a stalemate, the topic was raised in the public forums that resulted in a similar scenario. Some folks considered the "nuke" term to be too offensive, unprofessional, or shocking to be used as a serious brand name. Others placed a significant metaphorical value in the current logo, which contained a gear embossed with a nuclear symbol. Some preferred a transition to the "DNN" acronym that was often used as a shorthand reference in various communication channels. Further debate ensued over the category we occupied (portal, content management system, framework, and so on) and the clear marketing message we wished to convey.

As the project founder, I had my own opinions on the brand positioning and ultimately decided to resort to an authoritarian model rather that a committee model so that we could make a decision and move forward. From my perspective, when it comes to technology companies, there is a lot of acceptance for non-traditional brand names (consider Google, Yahoo!, Go Daddy, and so on). In addition, due to the press coverage of the Microsoft Hosting program, the DotNetNuke name achieved a significant amount of exposure; therefore, a complete change in brand would impose a serious setback in terms of brand acceptance and market reach. Taking into consideration the valued perspectives of the Core Team and community, I felt there should be a way to provide a win-win solution for everyone.

I first tried working with a local design company (the same company that produced the DotNetNuke 2.0 site skin), and although they had a real talent for brand identity services, there were no concepts produced that really grabbed my attention or satisfied my goals for the project. Perhaps I was being overly critical in my judgment of various designs, but I knew that I absolutely did not want to settle for a concept unless I thought it met 100% of my criteria. Although Nik Kalyani had been on the Core Team for eight months and had even expressed a serious interest in the marketing activities of the project, it was not until the re-branding exercise where his talents were truly exemplified.

Nik and I started an offline dialog where we quickly established some complementary goals, at least at a conceptual level. The basic decision was that we wanted to retain the full "DotNetNuke" brand name and strengthen rather than dilute its brand emphasis. We also wanted to reduce or eliminate the negative imagery associated with the nuclear warning symbol in the current logo. Although the abbreviated form of the word "nuke" tended to evoke a negative response from the general population (relating it to bombs and radiation), the expanded form of "nuclear" and "nucleus" had a much more positive response (related to science, energy, and power). The word "nucleus" also had some complementary terms associated with it such as "core," "kernel," and so on that worked well with the open-source project philosophy. The trick was to find a way to emphasize one aspect over another.

Nik spent countless hours designing alternative logo concepts. From a typeface perspective, he suggested using the Neuropol font, and I really liked the fact that it had a strong technical overtone but not so much that it could not be used effectively in other mainstream media applications. To achieve a uniform appearance for the typeface, we decided to use all capital letters — even though the standard format for the brand name in regular print would continue to be mixed case. Nik included a unique customization for the "E" and the "T" letters that resulted in a distinctive, yet professional styling for the word-mark contained within the logo.

Creating the graphical element for the logo was a much bigger challenge because we were looking for a radically new design that exemplified so many diverse project attributes. To summarize some of the more important criteria, we were looking for something simple yet distinctive, with at least some elements that provided a visual reference to the old logo for continuity. It needed to be scalable and adaptable to a wide range of media (both digital and print) and cost-effective to reproduce. And perhaps the most subjective item I promoted was that the logo should be stylish — with my acceptance criteria being, "Would my wife permit me to wear clothing embossed with the logo when we went out in public together?" Nik created more than 40 unique logo concepts before arriving at a design that seemed to catch the full essence of what we were trying to accomplish (see Figure 1-11). After working at this for so long and dealing with the discouragement and frustration, it was a euphoric moment to discover the proverbial "love at first sight."

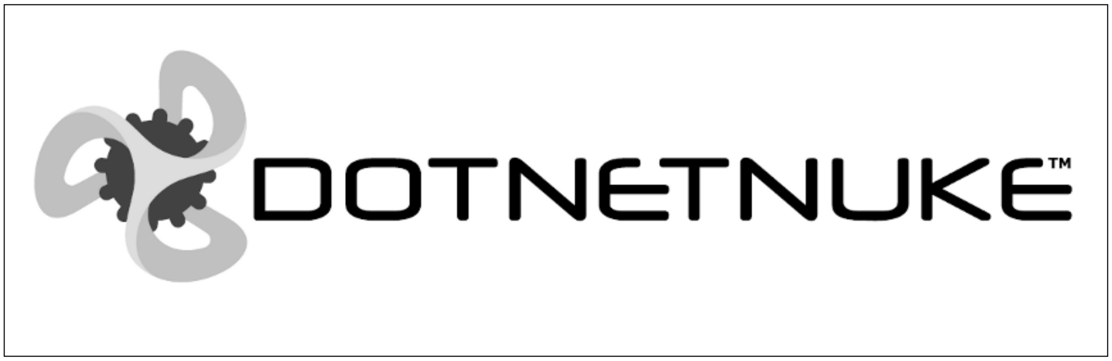


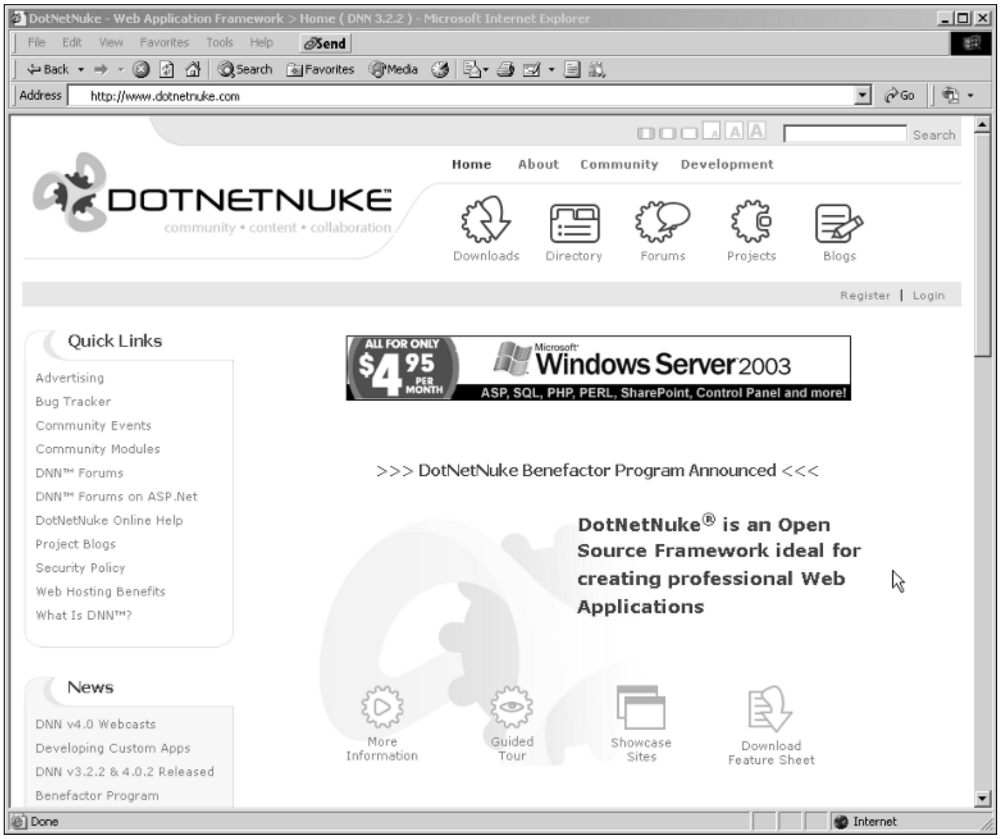
Figure 1-11

It is amazing how many diverse concepts can be represented in a single image. The saying "a picture is worth a thousand words" is cliché, but in this case, it certainly summarized the final product. The new logo had the basic shape of a nuclear atom. The nucleus of the atom was shaped like a gear to retain its heritage to the previous project logo. The logo was two basic colors — red and black (using shades of grey to achieve a 3-D effect) — making it much more adaptable and simple to reproduce in a wide variety of media formats than the previous logo (which used shadows and gradients for 3-D effects). The gear had twelve teeth (a number considered to be lucky in many cultures). The intersection of the three revolving electron trails (referred to as the "triad") could still be subtly viewed as a nuclear symbol reference. With some creative inference, they could also be viewed as the three-letter project acronym: DNN. Later, someone on the Core Team mentioned that the triad bore some resemblance to the Perpetual Motion Interactive Systems Inc. "infinity" logo — a reference I had never formally recognized but something that I am sure played a subliminal role in my selection.

In terms of brand acceptance, we realized there may be significant community backlash related to the new creative brand, especially from companies who were currently leveraging the existing DotNetNuke branding in their marketing materials. Therefore we were pleasantly surprised at the overwhelming positive feedback we received regarding the new brand identity. Our goal was to roll out the brand in progressive stages with the DotNetNuke 3.1 product release representing the official brand launch to the general community.

With the creative elements out of the way, it was time to finalize the rest of the branding process. Because DotNetNuke serves many different stakeholder groups, it was difficult to come up with a product category that was focused but not too limiting in scope. From a marketing perspective, the board agonized over the optimal brand message. "Content management" was a powerful industry buzzword, but if you compared the capabilities of DotNetNuke in this area with other enterprise software offerings, it became obvious that it would be some time before we could be considered a market leader. The term "portal" had been so overused in recent years that it became severely diluted and lost its clarity as an effective marketing message. Conversely, the emerging term "framework" began to surface more regularly and was starting to gain industry acceptance with both developers and management groups as a powerful software development category. Because DotNetNuke's architectural principles were predicated on simplicity and extensibility, the framework category seemed to be a natural fit. The next step involved clarifying the type of framework. DotNetNuke was primarily designed for use in a web environment and its breadth of features made it well-suited for building advanced data-driven Internet applications. The resulting "web application framework" was an emerging industry category in which DotNetNuke could take an immediate leadership role. Where applicable, we could also leverage our "open source" classification to emphasize our community philosophy and values.

One of the toughest parts of any re-branding exercise involves updating all existing brand references to reflect the new identity. In DotNetNuke's case, this affected the content and design of the `dotnetnuke.com` web site, the marketing references in the DotNetNuke release package, and all technical and user documentation. Compared to the time it took to construct the new logo, the time it took Nik Kalyani to create a new site design was minimal (which is truly amazing considering the amount of time and effort that typically goes into a custom site design). I had long been a fan of Nik's minimalist style, which emphasized clean presentation, lightweight graphics. and plenty of whitespace. Nik's expert grasp of the DotNetNuke skinning architecture enabled him to create a combination of skin and containers that were applied in a matter of minutes to completely transform the entire web site. The new site design was creative yet professional and eliminated the "cartoonish" criticisms of the previous site design (see Figure 1-12). Nik also created our first professional document templates that would provide consistency and emphasis of our branding elements within our technical and user documentation.


DotNetNuke - Web Application Framework > Home (DNN 3.2.2) - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://www.dotnetnuke.com
Go
Search
Home About Community Development
DOTNETNUKE
community • content • collaboration
Downloads Directory Forums Projects Blogs
Register | Login
Quick Links
Advertising
Bug Tracker
Community Events
Community Modules
DNN™ Forums
DNN™ Forums on ASP.Net
DotNetNuke Online Help
Project Blogs
Security Policy
Web Hosting Benefits
What Is DNN™?
ALL FOR ONLY $4.95 PER MONTH
Microsoft Windows Server 2003
ASP, SQL, PHP, PERL, SharePoint, Control Panel and more!
News
DNN v4.0 Webcasts
Developing Custom Apps
DNN v3.2.2 & 4.0.2 Released
Benefactor Program
More Information
Guided Tour
Showcase Sites
Download Feature Sheet
Done
Internet

>>> DotNetNuke Benefactor Program Announced <<<

DotNetNuke® is an Open Source Framework ideal for creating professional Web Applications

Figure 1-12

Tech Ed

At the beginning of June, there was a massive Microsoft technology conference, Tech Ed, in Orlando, Florida. Based on a generous invitation from the International .NET Association (INETA), Scott Willhite and I were provided with an opportunity to attend the event as their special guests. The timing was perfect because *Professional DotNetNuke ASP.NET Portals* was officially released at this event, as was the new project branding. Joe Brinkman and Dan Caron were able to attend some aspects of the book launch festivities, and we managed to jam a substantial amount of marketing activities into the five-day event. We had a dedicated Birds of Feather session, two community focus sessions at the INETA booth, a guest appearance at an INETA User Group workshop related to building effective web sites (where we learned 90% of .NET user groups were already using DotNetNuke), and a number of book signings scheduled by WROX Press at the Tech Ed bookstore. The DotNetNuke book was the top-selling developer book at the Tech Ed bookstore for the event — a fact that emphasized the growing popularity of the project. We also distributed official DotNetNuke T-shirts that showcased the new project branding, a popular item amid all the typical free swag provided at these events.

Seizing the opportunity of having the majority of the DotNetNuke Board of Directors together in one place, we had our second official board meeting — an all-day session in the conference room of our hotel in Orlando. On the agenda was a serious discussion related to Core Team reorganization and key project roles. For quite some time, we had realized that the current flat organizational structure was somewhat dysfunctional and that we ultimately needed more dedicated management resources to accomplish our goals. However, to support these resources, we needed a sufficient financial model. Discussion focused on the pros and cons of various revenue opportunities, their revenue potential, and their perceived effect on the community ecosystem. We also talked about what it would take for the current Board members to commit to full-time dedicated roles in the organization and the associated financial and security implications. A lot of really deep discussion ensued, which gave us a much better mental picture of the challenges that lay ahead if we truly wanted to take the project to the next level.

Following the publication of *Professional DotNetNuke ASP.NET Portals*, there was a bit of a media frenzy around the relationship between Microsoft and the open source phenomenon. Some of my personal opinions and quotes from the book found their way into an article published on CNET (one of the leading mainstream news sites), resulting in a lot of additional exposure for the project. It was interesting to see the power of the media at work, where a reference in a highly visible and trusted journalism channel can lead to broad distribution of a particular message (much like a stone in a pond leads to a concentric series of expanding ripples). For the most part, large companies are the most successful at leveraging these medial channels, but special-interest organizations also have the opportunity to make a significant impression.

Credibility

Although DotNetNuke had experienced a healthy growth rate through its open source philosophy, it had largely done so by appealing to the needs of grass-roots developers. Although these stakeholders represent an integral part of the high-tech marketplace, there is another group that is far more influential in terms of market impact. The so-called "decision-makers" represent the management interests in serious enterprise-level business organizations. For DotNetNuke to make the transition from a developer-oriented open source project to a serious enterprise software contender, it needed to appeal to the decision-maker mentality.

Where developers think in terms of short-term technical decisions (that is, "What tool can I use to get this job done as quickly as possible so that I can impress my boss?"), decision-makers think in terms of long-term business decisions. They are interested in the future support of a platform or product. They consider solutions in terms of "investments," "security," and how much "risk" is associated with adopting a particular technology as part of their company infrastructure. And regardless of the technical superiority of a software solution, the adoption criteria always come down to basic trust and consumer confidence. So the challenge for an open source project like DotNetNuke is establishing the necessary level of credibility to be taken seriously.

In the commercial world, customers get a sense of confidence based on the fact that they have paid licensing fees to a vendor that generally provides them with a certain level of future support. Obviously nothing is guaranteed, but this financial model provides both parties with a sense of security and responsibility. Another thing that the financial model affords is the ability to market the product through traditional channels — channels that "decision-makers" tend to monitor on a regular basis.

In the open source world, there are no licensing fees, which helps contribute to the lower cost of ownership but also leaves the investment/security aspect somewhat lacking. If you look at Linux, for example, you will notice that the broad industry buy-in for the operating system did not occur until after some serious market vendors (Sun and IBM) pledged their support. As soon as this happened, many medium-large companies began to take Linux more seriously. And this was not because Linux received any product improvements through these relationships, but rather because it reduced its risk perception in the general marketplace. And without traditional licensing fees, open source products generally do not have the budget to leverage traditional marketing channels and must instead rely on grassroots and viral marketing techniques.

So let's consider some of the ways in which an open source product can improve its credibility and reduce its risk perception for decision-makers. Clearly one way is that it can align itself with large, respected vendors who lend credibility (that is, "If vendor X thinks its good, then so do we."). Another way is to have mainstream books, magazines, and mass media distributors publish information about the product, contributing to the overall community knowledge base and providing recognition. Yet another option is to identify reference implementations that exemplify the best qualities of the product and impress people with their performance, elegance, or extensibility. Another way is to demonstrate a proven track record and history for supporting the community, especially through platform transitions where the likelihood of project failure is high. The overall size of the community ecosystem, including the open source participants, consumers, and third-party service providers, is another critical aspect in demonstrating credibility.

DotNetNuke definitely made some significant advancements in credibility in 2005. The strong working relationship with Microsoft reaped rewards with the Hosting program. The publication of *Professional DotNetNuke ASP.NET Portals* by Wiley Publishing, Inc. and *Building Websites with VB.NET and DotNetNuke 3.0* by PackT Press provided some excellent recognition through traditional publishing channels. Articles and references in mainstream magazines such as *Visual Studio Magazine*, *ASP.NET Pro*, *CoDe Magazine*, and *.NET Developers Journal* also provided some great benefits. The showcase on `dotnetnuke.com` contained many diverse reference implementations and we had proven through three years of product upgrades that we were committed to supporting the community. The membership and download metrics continued to grow exponentially, as did the number of independent software vendors (ISVs) providing products or services within the DotNetNuke ecosystem.

Trademark Policy

Unfortunately, an unexpected issue arose in the summer of 2005 that immediately put the project into crisis mode. Based on some invalid assumptions, a software consultant from Australia recommended that their client register a trademark for the DotNetNuke name in Australia. Aside from the obvious ethical implications, the immediate reaction was that this move was based on ulterior motives that could potentially hold the entire Australian DotNetNuke community hostage. Further communication revealed that the Australian company had concerns over the official trademark registered in Canada; specifically in regards to the fact it was embedded within the application source code and binaries, and that their business investment could be compromised if restrictions were ever put on trademark usage. Ultimately this whole situation revealed a number of critical issues when it comes to trademarks. First, the holder of the trademark must publish a policy that clearly defines the allowable usage of the mark under a wide range of use cases. Second, the trademark holder must make every attempt to enforce the policy so that the mark does not become a common term and lose its value as a protected asset. Third, a trademark must be registered in every jurisdiction where it intends to be used.

To satisfy the first requirement, I firmly believe in the philosophy of "standing on the shoulders of giants." Research revealed that Mozilla had recently gone through a similar project challenge, so we decided to use their recently published trademark policy as a template for our own. The political ramifications of introducing the policy at this point seemed controversial, but absolutely necessary if we intended to protect our brand. After extensive research, review, and legal advice, we finally announced the trademark policy in conjunction with the logo guidelines in July 2005. The overall community feedback was quite positive, because the policy made every effort to emphasize our open source roots and strong community ideals.

To satisfy the second requirement, all marketing materials were updated to reflect the trademark policy guidelines, and many community sites made changes to bring their use of the trademark into compliance. We also obtained legal advice on the creation of a Trademark License Agreement to be used in situations where third parties required the right to use the DotNetNuke trademarks for specific business purposes.

The third requirement was somewhat more challenging to deal with because it had substantial financial implications. The cost to register an individual trademark in a specific jurisdiction (country) can cost anywhere from $2000.00 to $5000.00. As an organization, we simply do not have the financial means to support such a large expenditure. So instead of considering all jurisdictions, we instead decided to focus on those jurisdictions that had a large project following. These included the United States, Canada, Australia, Japan, and the European Union. This whole experience gave me a much deeper understanding of the financial commitment required by large multinational companies who wish to protect their brand around the world.

ASP.NET 2.0

In July 2005, we recognized that we had approximately four months to prepare for the launch of Microsoft's next-generation software development platform. ASP.NET 2.0 had been under development for three years and had finally reached the point where it was ready for public release. Aside from reading the standard marketing propaganda in the various trade magazines catering to the Windows platform, I had not done significant research into the specific challenges DotNetNuke faced as a product related to this platform upgrade. And, as is usually the case, we quickly found out it was going to be some of the unpublicized platform changes that were going to cause us the most difficulty.

Based on early community feedback for the ASP.NET 1.0 release, Microsoft decided to completely overhaul the way web projects operated, including substantial changes to the underlying compilation model. Because DotNetNuke's advanced modular architecture strayed so far from the traditional monolithic ASP.NET application model, these platform changes had a significant impact on the project. Our solid working relationship with Microsoft reaped benefits in that we were able to engage in some focused dialog and onsite meetings in Redmond with the Microsoft Product Managers who understood the nuances of the new ASP.NET 2.0 platform better than anyone. Scott Guthrie, Simon Calvert, Omar Khan, and a number of other key Microsoft resources got personally involved in assisting us to find a suitable migration path.

I have to admit I was a vocal critic during these early discussions, because I could not understand the business cases that precipitated some of the major architectural changes. But after working closely with the Microsoft Product Managers, I began to warm up to the benefits of the new model and started to envision how we could leverage its capabilities to expose some powerful new options to the DotNetNuke community. But before we could focus on these new options, our most critical requirement was that we could not have breaking changes in the DotNetNuke framework in our ASP.NET 2.0 release. The main business criteria driving this requirement was the fact we had just had a major release with significant breaking changes in March 2005, and we could not risk an all-out community revolt (or product fork) based on compatibility issues.

Research and discussion proceeded throughout the months of July and August as we worked with Microsoft to find an optimal solution. Feedback from the community seemed to be mixed. People who were victims of the Microsoft propaganda machine seemed to think that the release of ASP.NET 2.0 would signal the end of DotNetNuke, because it promised to deliver so many overlapping application features. Other people who had adopted DotNetNuke as part of their business infrastructure expressed apprehension and fear regarding ASP.NET 2.0, based on their past experience that a significant platform upgrade usually resulted in a costly migration effort. Surprisingly, out of all the feedback collected, it appeared that nobody was making a serious attempt to perform the upgrade on their own, and that they were waiting for us to provide a migration path (as we had always done in the past). This element of trust was not lost on me, and I did my best to blog on a regular basis to provide public communication of our progress.

Reorganization

Throughout the summer and fall of 2005 there was ongoing discussion related to Core Team reorganization. Based on the guidelines that had been created when individuals were invited to join the team in the summer of 2004, there was clearly a group of members who had not lived up to their commitments. The list of responsibilities included staying involved in Core Team business through the private discussion forum; participating in weekly Core Team chats; contributing bug fixes, enhancements, or documentation to the core product; and being active in community support channels. There were many legitimate reasons, both personal and business-related, which led to inactivity for team members. However, the unfortunate side effect is that it led to a community perception that based on the total number of Core Team members, we were underachieving in terms of our capabilities as a whole. The Core Team reorganization meant that a number of team members needed to be retired to make way for some new members who had earned the right to participate based on their community accomplishments over the past year. The project had never had to deal with a situation like this in the past, and it's safe to say that as software developers, we are much more adept at solving technical problems than human-resources issues. So the dilemma was how to break the news to the inactive members in a professional and courteous manner that still respected their past accomplishments and left the door open for future participation. It was Scott Willhite who demonstrated the most experience and wisdom in this area, as we worked on establishing effective human resources processes for the organization.

Since the original formation of the Core Team, all members had received equal rights in terms of project participation. This included not only communication channels but also permissions to the product source code repository. This model worked well when the team was small and all members were on equal footing in terms of their technical abilities. However, it proved to be a challenge when the team grew in size and members were added with varying technical backgrounds. DotNetNuke had grown into a mission-critical web application framework that many businesses now relied on for rock solid performance and reliability. We could no longer accept the risk of inexperienced team members checking in code that could compromise the stability of the application. As a result, we needed to re-factor our project roles to reflect the new project requirements.

A common theme that helped drive the re-factoring of the project roles was accountability. In the past, we had witnessed the fact that without accountability, an individual would not exhibit the same level of commitment, dedication, or passion for the project. As a result, it was important to provide Core Team members with areas of accountability where their contributions would be highly visible and easily recognized by the general public. This public aspect provided them with a much greater benefit in terms of visibility in the community, but it also made them a target for criticism if they were inactive because they were personally responsible for specific areas of the project.

Using the Apache Foundation as a meritocracy reference, we made some significant changes to the organizational model of the project. The old "Inner Team" designation was abolished in favor of a new "Core Team Trustee" role. Scott Willhite came up with this new name based on the desire for industry-accepted terminology and the fact that this innermost project role assumed the highest level of trust from a development perspective. Core Team Trustees had multiple years of experience on the project, had successfully demonstrated their technical aptitude, and as a result were granted write access to the core repository. The old "Outer Team" designation was simplified to "Core Team Member" — a role that was able to participate in all Core Team communication channels, but was only provided read access to the source code repository. In addition, we added a role for the DotNetNuke Projects of "Project Team Lead." This role was responsible for managing the project infrastructure and communicating project status to the Core Team.

Conferences

The month of September 2005 began with the Professional Developer Conference (PDC) in Los Angeles, California. Based on a kind gesture from Microsoft, a large number of Core Team members were provided with free registration for the event in exchange for analysis of key ASP.NET 2.0 features that could be used in the DotNetNuke framework. Scott Willhite, Dan Caron, Nik Kalyani, Jon Henning, John Mitchell, Charles Nurse, and I were all able to attend the event, bringing together in one place the largest group of Core Team members ever. It was an excellent opportunity to get to know one another and we spent a lot of time hanging out together, exploring the exhibitor area, hosting a Birds of Feather session, visiting Universal Studios, and attending a variety of conference sessions.

The DotNetNuke Board, with the recent inclusion of Nik Kalyani, also took the opportunity to have some serious meetings regarding the progress of the revenue opportunities discussed at Tech Ed. The summer had not been productive in getting any programs launched other than Advertising and Sponsorship, and Nik took a lead role in attempting to clarify both our marketing and financial initiatives for the next 12 months. Specific board members were assigned to each major opportunity, and projections were presented and discussed in terms of assumptions, benefits, and execution tasks. We had a lot of work ahead of us, including a major platform transition, now firmly scheduled for November 7, 2005.

Later in September, Microsoft hosted a three-day summit for its Most Valuable Professional (MVP) community members. Based on public achievements, a number of DotNetNuke Core Team members earned this award of distinction in 2005. Bruce Hopkins (Georgia, USA), Phil Beadle (Australia), Cathal Connelly (Ireland), Jim Duffy (USA), and I (Canada) were all able to attend the private summit in Redmond, Washington. The summit provided the opportunity to get to know these Core Team members on a more personal level, including their appetite for social festivities. I was also able to spend some time with a number of prominent ASP.NET personalities and DotNetNuke evangelists whom I greatly respected in terms of their contributions to the community. In addition, there was also a large representation of Microsoft employees at the MVP summit that resulted in some excellent networking opportunities and offline discussions. Steve Balmer's keynote address provided some valuable insight into the roadmap for Microsoft's products and revealed areas where DotNetNuke could focus its efforts to strengthen its market position in the coming year.

Directly following the MVP summit, I had the privilege of attending my first ASPInsiders summit as well. The ASPInsiders represent a group of well-respected industry leaders in the Microsoft ASP.NET community. I had recently been inducted as an official member and appreciated the opportunity to be included in such an elite group of professionals. Perhaps the most important benefit of being an ASPInsider was that it provided representation for the DotNetNuke development community and validation of our extensive contributions to the industry. Due to its small focused membership, the ASPInsiders summit had a personal and direct interaction with Microsoft employees, allowing its members to provide feedback on a number of exciting new technologies. The networking opportunity was incredible, and the intricate dynamics of the various personalities and companies represented was especially interesting.

DotNetNuke 4.0

Throughout the months of September and October, Charles Nurse was instrumental in working on the migration to the ASP.NET 2.0 platform. He invested a massive amount of time researching compatibility issues, creating various proof of concepts, and communicating regularly with Microsoft. He actually pursued two different agendas simultaneously: the upgrade of DotNetNuke 3.0 to ASP.NET 2.0 from a runtime perspective, and the creation of a new web project model for DotNetNuke 4.0 that provided a development strategy for the future.

To support the community, we concluded that we would need to support two parallel code bases for an undetermined period of time: DotNetNuke 3.x (ASP.NET 1.1) and DotNetNuke 4.0 (ASP.NET 2.0). Obviously, a more optimal solution would have been a single code base that worked on both platforms; however, this simply was not possible based on the platform compilation changes in ASP.NET 2.0. In addition, we did not know what to expect in terms of the adoption rate for the new Microsoft platform. Therefore, it seemed natural that we focus on developing for both ASP.NET 1.1 and 2.0 in the short term. An unfortunate side effect of this model involved a general recommendation to develop to the lowest common denominator (that is, not leverage ASP.NET 2.0-specific technology) and synchronizing all fixes and enhancements across the two code bases.

One of the greatest achievements in the platform migration was that we were able to fully satisfy our business requirement for no breaking changes. DotNetNuke modules and skins developed on ASP.NET 1.1 could be installed directly into the ASP.NET 2.0 environment without any changes whatsoever. This had massive benefits for the commercial DotNetNuke ecosystem because vendors could continue developing their modules as a single code base on the ASP.NET 1.1 platform but offer their packaged products for sale in both channels.

The only item that remained outstanding right up until the week before the November 7th launch was how to develop DotNetNuke 4.0 modules on the ASP.NET 2.0 platform. The new dynamic compilation model in ASP.NET 2.0 created some challenges for many of our runtime extensibility features, especially where they relied on object instantiation through reflection. As is often the case with technical problems, the answer is out there — it's just a matter of finding the right person to ask. As luck would have it, a Microsoft developer (Ting-Hao Yang) who was copied on some of the communication between our team and the Microsoft ASP.NET Product Manager group finally responded with details on a new ASP.NET 2.0 framework method that ultimately solved all of our remaining reflection issues. In the end, all that was required was a change to a single method in the DotNetNuke 4.0 core framework (to use BuildManager.GetType).

One of the benefits of the new ASP.NET 2.0 platform was that Microsoft had put a lot of focus on making the technology more accessible to the general developer community. A key deliverable in this strategy was the release of an entire suite of free "Express" tools. Included in the Express line was a tool named "Visual Web Developer" that provided a functional Integrated Development Environment (IDE) for ASP.NET 2.0. Leveraging the benefits of this powerful new tool, we created a DotNetNuke 4.0 Starter Kit that enabled a developer to configure a fully functional development environment within minutes. This had significant implications on the DotNetNuke development community because it lowered the barrier of entry and now made it possible for any aspiring software developer, from beginner to advanced, to be instantly productive with the DotNetNuke web application framework. Combine this with the free SQL Server 2005 Express database engine and you have a zero cost development environment. Visual Web Developer could not be used to develop server controls or class libraries; however, the fact that the DotNetNuke extensibility architecture was based on user controls made it a perfect fit.

Not wanting to neglect the existing DotNetNuke 3.0 community by focusing solely on ASP.NET 2.0 migration, we decided to integrate a few powerful new features that had long been requested by the general community. Core Team member Tam Tran Minh had been developing an Active Directory integration component for a number of years and agreed to contribute it as a fully supported core framework component. Additionally, Jon Henning had been busy working on a full-featured JavaScript API that would allow developers to leverage powerful client-side behavior in their modules. This included a new menu control, the DNN Menu, and an implementation of the popular Asynchronous JavaScript for XML (or AJAX) technology. AJAX technology had become one of the hottest new trends for web development, and it is important to note that DotNetNuke included a powerful AJAX library well before the announcement of ATLAS by Microsoft. The combination of these features offered benefits to both platform consumers and application developers, and further strengthened our core platform offering.

The official Microsoft launch date for ASP.NET 2.0 was set for November 7, 2005. We knew if we could release DotNetNuke 4.0 to coincide with this event, we would be able to ride the huge marketing wave created by Microsoft. Because we had always advocated "releasing software when it is ready," this hard deadline imposed some serious challenges on our meager project resources. Aside from the obvious technical deliverables, we had communication and marketing deliverables that also needed to roll out in unison. Nik Kalyani and Bill Walker showed their agility to pull things together on a tight schedule, and we launched our first monthly newsletter to the entire DotNetNuke registered user base (now 200,000 registered users) on November 7. The response was overwhelmingly positive as the significance of the achievement began to sink in. In the month of November, we recorded 165,000 downloads, far eclipsing any previous monthly download total in the history of the project.

An interesting aspect to consider in the ASP.NET 2.0 migration was that we delivered a fully managed upgrade to users of the DotNetNuke web application framework. Anyone who has ever attempted a major platform upgrade on their own should recognize the incredible value of this accomplishment. We had effectively eliminated a budget line item of considerable cost and effort from thousands of IT departments and business entities around the world. Compare this to scenarios where companies create their own custom ASP.NET 1.1 applications. In these cases, each company would need to invest significant resources and funding to work out their own web application migration strategy. Or compare this to another scenario where you adopt another web application framework, commercial or open source, which had not even considered the upgrade challenges posed by ASP.NET 2.0 and were going to force you to postpone your upgrade until it fit their own release schedule. In either case, the decision to adopt DotNetNuke as part of an organization's business infrastructure had certainly paid dividends worthy of the attention of any business decision maker.

Immediately following the DotNetNuke 4.0 release, we focused on stabilization issues that were exposed through testing by a larger community audience. Another area that received dedicated focus was the Module Item Template feature of the DotNetNuke 4.0 Starter Kit. Through research and persistence, we were able to construct a DotNetNuke Module Template that could automatically create all of the development resources required to build a fully functional module in DotNetNuke 4.0. It even had some parameterization capabilities so that the template could be customized at runtime to meet the needs of the developer. I wrote an article describing the Starter Kit and Module Template and posted it on the public forums on `www.asp.net`. The article proved to be popular, with nearly 30,000 views recorded in the six weeks following its publication. It turned out that the changes in ASP.NET 2.0 resulted in some decent productivity benefits for module developers, further improving the capabilities of the DotNetNuke framework.

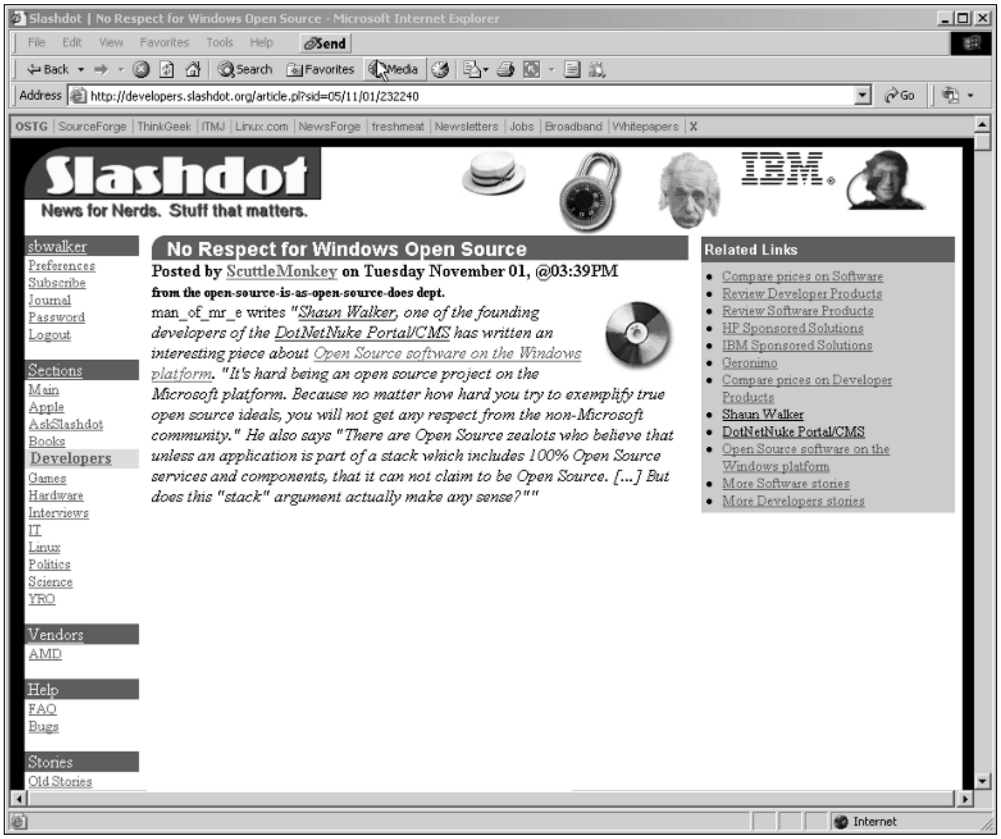
An interesting event occurred in December 2005, well after the official launch of ASP.NET 2.0. Based largely on the feedback that we provided Microsoft during our product migration efforts, Microsoft announced some add-ons for Visual Studio 2005 that added back ASP.NET 1.1 development support through Web Application Projects as well as compilation and merge support through Web Deployment Projects. Based on its superior architecture and incredible popularity, DotNetNuke was able to unite a significant portion of the Microsoft developer community and create a much stronger voice and more compelling argument in favor of specific platform features than would have ever been possible for individual developers. Beside the fact that these add-ons provided some critical options for web application developers, it was really gratifying to see that our direct feedback could have such an immediate and influential effect on the industry.

Slashdotted

In October 2005, I wrote a blog titled "No Respect for Windows Open Source." The blog was a political rant based on the fact that because DotNetNuke did not run on a fully open source stack of software components (that is, Linux/Apache/MySQL/PHP or LAMP), it did not get any respect from the general open source community. Further, it argued that all open source projects regardless of platform should be judged solely on the validity of their open source license and ideals. The blog was picked up by Slashdot, the largest independent news site for information technology, and resulted in a lot of exposure for the project (see Figure 1-13). The posting on Slashdot generated more than 500 comments, each with their unique perspective on the Windows open source paradigm.

In October, we were approached by *.NET Developers Journal* (*.NETDJ*) to do a series of articles on the DotNetNuke project. This was an excellent opportunity to showcase various aspects of the project in a mainstream magazine. A number of Core Team members were identified as potential authors and the first article in a series of six was published in the November edition of *.NETDJ*. Forging relationships with publishers is a great way to raise the profile of the project and open doors for future opportunities. In this case, working with SYS-CON (the publisher of *.NETDJ*) reaped rewards in terms of being approved as a featured speaker in the upcoming SYS-CON Enterprise Open Source conference in June 2006.

By the end of 2005, the `dotnetnuke.com` web site had achieved an impressive Alexa ranking of 6,741 and our SourceForge.Net ranking had climbed to #75 (out of all the open source projects in the world). We were consistently getting 15,000 new registered users per month and our project downloads averaged 120,000 per month. The `dotnetnuke.com` site was now serving 4.5 million page views per month, and every indication was pointing to even more improvement in 2006.


Slashdot | No Respect for Windows Open Source - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://developers.slashdot.org/article.pl?sid=05/11/01/232240
Go
OSTG | SourceForge | ThinkGeek | ITMJ | Linux.com | NewsForge | freshmeat | Newsletters | Jobs | Broadband | Whitepapers | X
Slashdot
News for Nerds. Stuff that matters.
sbwalker
Preferences
Subscribe
Journal
Password
Logout
Sections
Main
Apple
AskSlashdot
Books
Developers
Games
Hardware
Interviews
IT
Linux
Politics
Science
YRO
Vendors
AMD
Help
FAQ
Bugs
Stories
Old Stories
Related Links
Compare prices on Software
Review Developer Products
Review Software Products
HP Sponsored Solutions
IBM Sponsored Solutions
Geronimo
Compare prices on Developer Products
Shaun Walker
DotNetNuke Portal/CMS
Open Source software on the Windows platform
More Software stories
More Developers stories
Internet

No Respect for Windows Open Source

Posted by ScuttleMonkey on Tuesday November 01, @03:39PM

from the open-source-is-as-open-source-does dept.

man_of_mr_e writes "*Shaun Walker, one of the founding developers of the DotNetNuke Portal/CMS has written an interesting piece about Open Source software on the Windows platform. "It's hard being an open source project on the Microsoft platform. Because no matter how hard you try to exemplify true open source ideals, you will not get any respect from the non-Microsoft community." He also says "There are Open Source zealots who believe that unless an application is part of a stack which includes 100% Open Source services and components, that it can not claim to be Open Source. [...] But does this "stack" argument actually make any sense?""*

Figure 1-13

Benefactor Program

As much as there is a romantic notion regarding a distributed group of purely volunteer resources working together in their free time to produce an enterprise-level software product, it does not represent reality. To effectively manage all of the aspects of a professional software product, dedicated management is an absolute requirement. This does not just entail the standard project management principles for software development, but also the legal and marketing aspects of managing a high-profile technology asset. Since the project inception, I had been able to commit 100% of my time to the project only because there was a sufficient stream of project revenue to support my needs. And throughout the life of the project, a number of team members had been financially compensated for various deliverables so that we could meet obligations and scheduled deadlines. The financial resources came from a variety of sources, including third-party sponsorship, advertising, and custom consulting opportunities. Unfortunately, the revenue streams were not sizable or stable in terms of securing multiple resources for long-term engagements. Essentially, we were trying to operate a product company without any direct product revenue. And with the constant growth of the project, the demands were increasing rather than decreasing, putting even more pressure on the minimal set of project resources.

Back in July 2005, I concluded that without a dedicated sales effort, the `dotnetnuke.com` web site was never going to reach its full potential as a revenue-generating asset. (We had published ad rates on the site months earlier and had not received many serious inquiries.) I decided it was time to more actively cultivate our advertising and sponsorship revenue streams and that it was going to require spending some money to make money. Armed with a huge number of industry contacts collected at Tech Ed, I hired a full-time resource to actively manage the advertising and sponsorship program. Due to major content improvements made in the previous four months, the `dotnetnuke.com` web site became a targeted channel for the Microsoft development community. By simplifying the advertising rate sheet and employing traditional sales techniques, we were successfully able to grow this revenue stream in a relatively short time frame.

In the fall of 2005, while driving home from a business trip, I spent some dedicated time immersing myself in the revenue model dilemma. Over the years, I did a lot of research on business models for open source projects, and the big question was, "How do you sustain an open source organization while still adhering to its open source ideals?" There were obviously a number of companies that had demonstrated their ability to succeed in this area by employing a variety of financial options; however, I was keenly aware that each model had its own set of disadvantages.

One of the other recurring themes I kept thinking about is "who we serve." In a traditional business model, you serve your customers — but this generally assumes that some money is changing hands. For DotNetNuke, I would like to think that our open source community is who we serve — but because they are essentially using the product for free, it becomes challenging when other stakeholders step forward with financial support. Examining each of the more popular open source revenue models based on this theme proved to be a useful exercise.

The Pure Volunteer option has no revenue model. As a result, it has no resource cost — but at the same time it has no accountability, responsibility, or dedicated management. It could be argued that although it is supposed to serve the open source community, it really does not because there are no motivating factors driving the development and support.

The Dual License model has become popular in recent years because it allows for an open source version as well as a commercial version of the same product. The commercial version provides traditional licensing revenue that helps sustain dedicated management and developer resources, resulting in improved accountability. Unfortunately, it tends to lead to a number of conflict-of-interest scenarios within the ecosystem that can be quite damaging. For one thing, the open source version of the product is often stripped of its more valuable features in favor of promoting sales of the commercial version. The open source license is often tarnished to protect the intellectual property rights of the company in the commercial version. Extensibility options are throttled as the company attempts to control the financial ecosystem around the product. And the company typically shows favoritism through support and marketing channels to its paying commercial customers over the organic open source community. In the worst-case scenario, the company can be accused of taking the most valuable intellectual property from the community and using it for their own financial gain.

The Sponsorship model involves utilizing a revenue stream from one or more third-party funding sources. Although this revenue model results in funding for dedicated management, it often compromises the project ideals as the sponsor attempts to exert their influence over the project roadmap and marketing goals. It also results in a revenue stream that is variable, creating challenges in terms of cash-flow requirements. In addition, the project needs to be extremely diligent regarding the ownership of the intellectual property so as not to put itself in a situation where the third party could sue the project for copyright infringement or affect the open source project licensing.

The Professional Services model is based on a concept where the platform maintainer does a significant amount of custom consulting for a third-party client. The revenue from the custom consulting is used to fund the dedicated management for the open source product. Unfortunately, this model tends to consume a high level of resources to qualify leads, formulate contracts, manage accounts, obtain signoff, and keep the pipeline full of revenue opportunities. The revenue stream is variable, affecting cash flow, and key project resources are often required to focus on specific client requirements rather than supporting and improving the open source product.

The Charitable Donations model is popular in the traditional open source world because it involves voluntary community financial support of the project. The problem is that it does not generate a consistent, sustainable revenue stream, which means it is unable to secure dedicated management resources. In addition, there is a tendency for community members to assume that other members are making financial donations, when in reality the project is receiving no financial contributions from anyone.

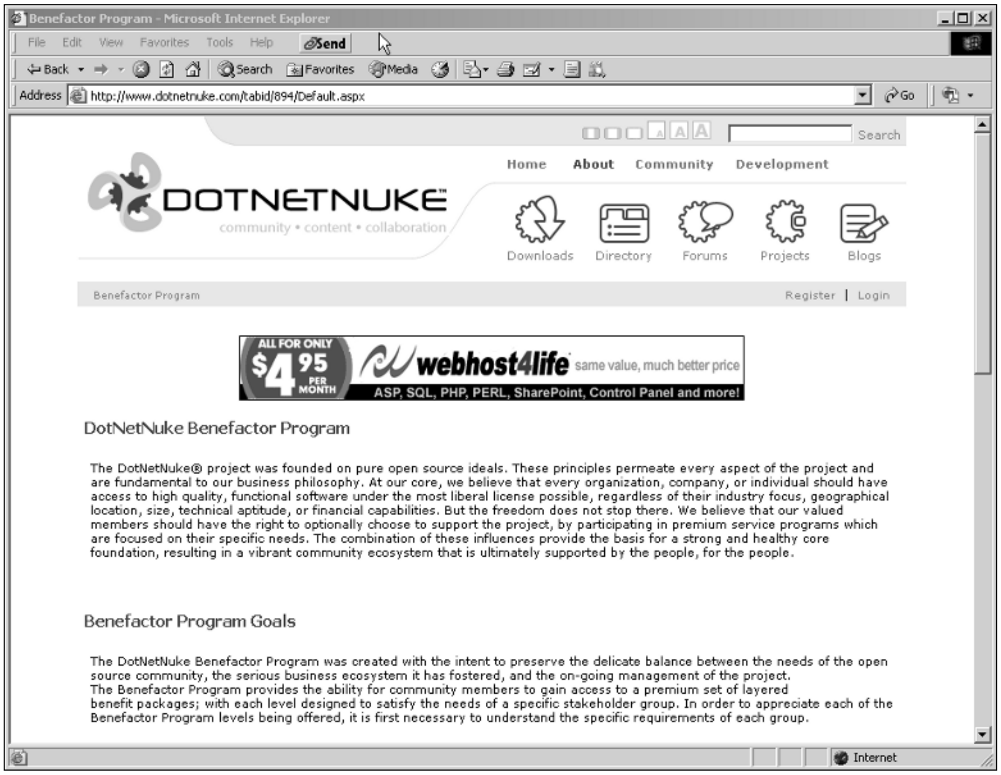
The Vertical Application model leverages the open source product to create a highly specialized, commercial, vertical market application. The vertical market application typically generates revenue through as application service provider (ASP) revenue model, which contributes funding back to the open source project. The challenge is that it requires focused management and marketing in the vertical market, complete with domain challenges, competition, legal considerations, and political constraints. The open source application also tends to cater the product roadmap to the needs of the vertical market application, resulting in a less robust application framework.

Because each of the common revenue models seem to have their own set of issues, it made me brainstorm what I would consider to be an optimal open source revenue model. The main criterion is that the project should serve the open source community ("by the people, for the people"). It should be objective and open, avoiding conflict of interest and adhering to open source ideals. Finally, the revenue stream must be consistent and sustainable, capable of sustaining multiple dedicated resources.

An interesting economics philosophy that Scott Willhite turned me on to was the concept of the "abundance mentality." In terms of business value, an "abundance mentality" refers to an attitude of growth. Essentially, it means that the overall size of the ecosystem becomes larger as the number of opportunities within the ecosystem increases. By working together with various stakeholders in the ecosystem, all members of the collective group benefit through a greater abundance of revenue-generating opportunities. The opposite of the "abundance mentality" is the "scarcity mentality," where participants consider the size of the ecosystem to be constant and the goal is to capture as much of the market share as possible (choking out the smaller competitors in the process). DotNetNuke's extensible architecture and open source philosophy constantly pushes the envelope in terms of creating new business opportunities within the community. It was another principle that needed to be adhered to in our quest for a suitable revenue model.

With all of these ideas swirling in my head, I concluded that a Membership concept would be an effective revenue model for advancing our goals. It would mean that the open source project was funded by the community. It would also mean that the project was accountable and responsible to the community. Through the creation of new benefits, we would be able to provide more opportunities for community members to participate in the project ecosystem. From a public perspective, it would provide a defined method for any supporter, big or small, to contribute to the project. And we would not need to compromise any of our open source ideals. Membership would be available by subscription that would create on ongoing, consistent revenue stream.

The DotNetNuke Benefactor Program (see Figure 1-14) was officially launched in December 2005. Nik Kalyani came up with the marketing term "benefactor" because it clearly communicated the financial support goal of the program. The program had four levels of participation to cater to the needs of various stakeholders in the community, from individual developers to enterprise business organizations. The initial set of benefits was targeted to each program level and the administrative aspects of the program were automated as much as possible to provide a seamless user experience. The overall response to the program was positive and paved the way for future revenue opportunities.


Benefactor Program - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://www.dotnetnuke.com/tabid/894/Default.aspx Go
Search
Home About Community Development
DOTNETNUKE
community • content • collaboration
Downloads Directory Forums Projects Blogs
Benefactor Program
Register | Login
ALL FOR ONLY $4.95 PER MONTH webhost4life same value, much better price
ASP, SQL, PHP, PERL, SharePoint, Control Panel and more!
Internet

DotNetNuke Benefactor Program

The DotNetNuke® project was founded on pure open source ideals. These principles permeate every aspect of the project and are fundamental to our business philosophy. At our core, we believe that every organization, company, or individual should have access to high quality, functional software under the most liberal license possible, regardless of their industry focus, geographical location, size, technical aptitude, or financial capabilities. But the freedom does not stop there. We believe that our valued members should have the right to optionally choose to support the project, by participating in premium service programs which are focused on their specific needs. The combination of these influences provide the basis for a strong and healthy core foundation, resulting in a vibrant community ecosystem that is ultimately supported by the people, for the people.

Benefactor Program Goals

The DotNetNuke Benefactor Program was created with the intent to preserve the delicate balance between the needs of the open source community, the serious business ecosystem it has fostered, and the on-going management of the project.
The Benefactor Program provides the ability for community members to gain access to a premium set of layered benefit packages; with each level designed to satisfy the needs of a specific stakeholder group. In order to appreciate each of the Benefactor Program levels being offered, it is first necessary to understand the specific requirements of each group.

Figure 1-14

DotNetNuke Marketplace

The extensibility model in DotNetNuke spawned an active commercial ecosystem. By January 2006, there were hundreds of commercial modules and skins available for the DotNetNuke application. In addition, there were many companies who were providing business services exclusively to the DotNetNuke market. This dynamic ecosystem was helping propel the growth of the project, but it was not without its share of issues.

Early in project's history, a third party created a reseller environment that allowed developers to sell their DotNetNuke products to consumers. This made it extremely easy for anyone, from a hobbyist developer to a serious independent software vendor, to get involved in the DotNetNuke commercial ecosystem. In the early stages, the existence of an established business environment for commercial components was critical to the growth of the project and promotion of the "abundance mentality." However, one of the most common types of negative feedback that we received related to this environment as time went on was about the quality of third-party products and services.

Based on the low barrier of entry of the reseller environment, the quality of commercial DotNetNuke components was extremely inconsistent. Some vendors were providing high-quality components, with professional support, and explicit licensing terms. Others were essentially providing untested code snippets with no support or licensing considerations. The combination of these polar opposites in terms of product quality posed some real issues in terms of our goals to promote DotNetNuke as an enterprise-level framework. Effectively, the existing reseller environment was supporting a "buyer beware" mentality that was not complementary with our goals for taking the project to the next level. Some of the more serious independent software vendors told us that for them to get involved in the ecosystem, a more professional reseller channel would need to be made available.

One of the issues that we struggled with for quite some time was the creation of some review criteria for DotNetNuke components. This became increasingly more critical as many of the DotNetNuke Projects entered the release pipeline. Leveraging the revenue from the Benefactor Program, I was finally able to fund this effort, and Joe Brinkman was selected as the best resource for the task. The review criteria played a fundamental role in our internal organizational process, but we also had another strategic goal in mind.

The current reseller environment was managed by a third party. We had approached them a number of times in the past with hopes that we could form a business partnership. The critical points were that we were not receiving any revenue from the DotNetNuke ecosystem we created, and we were not collecting any business intelligence related to the users of the product. For us to effectively manage the product roadmap, it was becoming increasingly more important that we get in touch with our true user community. The discussion forums represented a small-but-vocal group of community members who offered feedback, but there was a much larger group of users with whom we had absolutely no contact. Unfortunately, the reseller was not interested in working with us in this capacity, which left us with a single alternative: establishing our own reseller channel.

Combining the concepts of the review criteria with a reseller channel seemed to be a great way to satisfy a variety of project goals. Our reseller channel would only sell components that passed our objective review criteria. This improved the overall perception of quality and confidence in the community, provided us with the necessary business intelligence, and exposed a new revenue stream to help us secure more dedicated management resources.

Summary

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`.