If it is decided that our system needs an overhaul, what is the best way to go about it?

Asked 16 years, 3 months ago Modified 15 years, 6 months ago

Viewed 440 times

3

We are mainting a web application that is built on Classic ASP using VBScript as the primary language. We are in agreement that our backend (framework if you will) is out dated and doesn't provide us with the proper tools to move forward in a quick manner. We have pretty much embraced the current webMVC pattern that is all over the place, and cannot do it, in a reasonable manner, with the current technology. The big missing features are proper dispatching and templating with inheritance, amongst others.

Currently there are two paths being discussed:

1. Port the existing application to Classic ASP using JScript, which will allow us to hopefully go from there to .NET MSJscript without too much trouble, and eventually end up on the .NET platform (preferably the MVC stuff will be done by then, ASP.NET isn't much better than were we are on now, in our opinions). This has been argued as the safer path

with less risk than the next option, albeit it might take slightly longer.

2. Completely rewrite the application using some other technology, right now the leader of the pack is Python WSGI with a custom framework, ORM, and a good templating solution. There is wiggle room here for even django and other pre-built solutions. This method would hopefully be the quickest solution, as we would probably run a beta beside the actual product, but it does have the potential for a big waste of time if we can't/don't get it right.

This does not mean that our logic is gone, as what we have built over the years is fairly stable, as noted just difficult to deal with. It is built on SQL Server 2005 with heavy use of stored procedures and published on IIS 6, just for a little more background.

Now, the question. Has anyone taken either of the two paths above? If so, was it successful, how could it have been better, etc. We aren't looking to deviate much from doing one of those two things, but some suggestions or other solutions would potentially be helpful.

python **asp-classic** **vbscript**

edited Sep 17, 2008 at 20:55

asked Sep 17, 2008 at 20:48

neouser99

1,827 ● 1 ● 10 ● 24

What is '.NET MSJscript'? ASP.NET does not have JScript as a server-side language option (outside of Silverlight) – John Sheehan Sep 17, 2008 at 20:51

I would like to know too, we got two or three situations like that. They all got stuck in ASP... nobody wants to touch them or redo them from scratch. – Sklivvz Sep 17, 2008 at 20:51

JScript is Microsoft's implementation of the ECMA standard, like JavaScript, but each has variations on each other. People have forgotten it's called JScript because it is used in the same context as standard JS, but JScript.NET is compiled to IL and uses the .NET runtime. – TheXenocide Sep 17, 2008 at 20:55

9 Answers

Sorted by: Highest score (default)

7

Don't throw away your code!

It's the single worst mistake you can make (on a large codebase). See Things You Should Never Do, Part 1.

You've invested a lot of effort into that old code and worked out many bugs. Throwing it away is a classic developer mistake (and one I've done many times). It makes you feel "better", like a spring cleaning. But you don't need to buy a new apartment and all new furniture to outfit your house. You can work on one room at a

time... and maybe some things just need a new paintjob. Hence, this is where refactoring comes in.

For new functionality in your app, write it in C# and call it from your classic ASP. You'll be forced to be modular when you rewrite this new code. When you have time, refactor parts of your old code into C# as well, and work out the bugs as you go. Eventually, you'll have replaced your app with all new code.

You could also write your own compiler. We wrote one for our classic ASP app a long time ago to allow us to output PHP. It's called Wasabi and I think it's the reason Jeff Atwood thought Joel Spolsky went off his rocker. Actually, maybe we should just ship it, and then you could use that.

It allowed us to switch our entire codebase to .NET for the next release while only rewriting a very small portion of our source. It also caused a bunch of people to call us crazy, but writing a compiler is not that complicated, and it gave us a lot of flexibility.

Also, if this is an internal only app, just leave it. Don't rewrite it - you are the only customer and if the requirement is you need to run it as classic asp, you can meet that requirement.

answered Sep 18, 2008 at 2:19

Michael Pryor

25.3k ● 18 ● 73 ● 92

3

Use this as an opportunity to remove unused features! Definitely go with the new language. Call it 2.0. It will be a lot less work to rebuild the 80% of it that you really need.

Start by wiping your brain clean of the whole application. Sit down with a list of its overall goals, then decide which features are needed based on which ones are used. Then redesign it with those features in mind, and build.

(I love to delete code.)

Share Improve this answer

Follow

answered Sep 17, 2008 at 20:53

easeout

8,736 ● 5 ● 45 ● 51

3

It works out better than you'd believe.

Recently I did a large reverse-engineering job on a hideous old collection of C code. Function by function I reallocated the features that were still relevant into classes, wrote unit tests for the classes, and built up what looked like a replacement application. It had some of the original "logic flow" through the classes, and some classes were poorly designed [Mostly this was because of a subset of the global variables that was too hard to tease apart.]

It passed unit tests at the class level and at the overall application level. The legacy source was mostly used as

a kind of "specification in C" to ferret out the really obscure business rules.

Last year, I wrote a project plan for replacing 30-year old COBOL. The customer was leaning toward Java. I prototyped the revised data model in Python using Django as part of the planning effort. I could demo the core transactions before I was done planning.

Note: It was quicker to build a the model and admin interface in Django than to plan the project as a whole.

Because of the "we need to use Java" mentality, the resulting project will be larger and more expensive than finishing the Django demo. With no real value to balance that cost.

Also, I did the same basic "prototype in Django" for a VB desktop application that needed to become a web application. I built the model in Django, loaded legacy data, and was up and running in a few weeks. I used that working prototype to specify the rest of the conversion effort.

Note: I had a working Django implementation (model and admin pages only) that I used to plan the rest of the effort.

The best part about doing this kind of prototyping in Django is that you can mess around with the model, unit tests and admin pages until you get it **right**. Once the model's right, you can spend the rest of your time fiddling around with the user interface until everyone's happy.

Share Improve this answer

Follow

answered Sep 17, 2008 at 21:44

S.Lott

391k ● 82 ● 517 ● 788

2

Whatever you do, see if you can manage to follow a plan where you do not have to port the application all in one big bang. It is tempting to throw it all away and start from scratch, but if you can manage to do it gradually the mistakes you do will not cost so much and cause so much panic.

Share Improve this answer

Follow

answered Sep 17, 2008 at 21:08

Hallgrim

15.5k ● 11 ● 48 ● 54

1

Half a year ago I took over a large web application (fortunately already in Python) which had some major architectural deficiencies (templates and code mixed, code duplication, you name it...).

My plan is to eventually have the system respond to WSGI, but I am not there yet. I found the best way to do it, is in small steps. Over the last 6 month, code reuse has gone up and progress has accelerated.

General principles which have worked for me:

1. Throw away code which is not used or commented out
2. Throw away all comments which are not useful

3. Define a *layer hierarchy* (models, business logic, view/controller logic, display logic, etc.) of your application. This has not to be very clear cut architecture but rather should *help you think about the various parts of your application* and help you better categorize your code.
4. If something grossly violates this hierarchy, change the offending code. Move the code around, recode it at another place, etc. At the same time adjust the rest of your application to use this code instead of the old one. Throw the old one away if not used anymore.
5. **Keep you APIs simple!**

Progress can be painstakingly slow, but should be worth it.

Share Improve this answer

Follow

answered Sep 19, 2008 at 10:56

pi.

21.5k ● 8 ● 40 ● 59

0

I would not recommend JScript as that is definitely the road less traveled. ASP.NET MVC is rapidly maturing, and I think that you could begin a migration to it, simultaneously ramping up on the ASP.NET MVC framework as its finalization comes through. Another option would be to use something like ASP.NET w/Subsonic or NHibernate.

Share Improve this answer

Follow

answered Sep 17, 2008 at 20:55

Forgotten Semicolon

14.1k ● 2 ● 52 ● 61

0

Don't try and go 2.0 (more features then currently exists or scheduled) instead build your new platform with the intent of resolving the current issues with the code base (maintainability/speed/wtf) and go from there.

Share Improve this answer

Follow

edited Sep 18, 2008 at 2:42

answered Sep 18, 2008 at 2:23

myid

0

A good place to begin if you're considering the move to Python is to rewrite your administrator interface in Django. This will help you get some of the kinks worked out in terms of getting Python up and running with IIS (or to migrate it to Apache). Speaking of which, I recommend isapi-wsgi. It's by far the easiest way to get up and running with IIS.

Share Improve this answer

Follow

answered Feb 27, 2009 at 0:10

Jason Baker

198k ● 138 ● 382 ● 520

0

I agree with Michael Pryor and Joel that it's almost always a better idea to continue evolving your existing code base rather than re-writing from scratch. There are typically opportunities to just re-write or re-factor certain components for performance or flexibility.

Share Improve this answer

Follow

answered Jun 15, 2009 at 2:33

pbreitenbach

11.3k ● 3 ● 35 ● 24