Structuring projects & dependencies of large winforms applications in C#

Asked 16 years, 2 months ago Modified 12 years, 10 months ago

Viewed 7k times

24

UPDATE:
This is one of my most-visited questions, and yet I still haven't really found a satisfactory solution for my project. One idea I read in an answer to another question is to create a tool which can build solutions 'on the fly' for projects that you pick from a list. I have yet to try that though.

How do you structure a very large application?

- Multiple smallish projects/assemblies in one big solution?
- A few big projects?
- One solution per project?

And how do you manage dependencies in the case where you don't have one solution. Note: I'm looking for advice based on experience, not answers you found on Google (I can do that myself).

I'm currently working on an application which has upward of 80 dlls, each in its own solution. Managing the

dependencies is almost a full time job. There is a custom in-house 'source control' with added functionality for copying dependency dlls all over the place. Seems like a sub-optimum solution to me, but is there a better way? Working on a solution with 80 projects would be pretty rough in practice, I fear.

(Context: winforms, not web)

EDIT: *(If you think this is a different question, leave me a comment)*

It seems to me that there are interdependencies between:

- Project/Solution structure for an application
- Folder/File structure
- Branch structure for source control (if you use branching)

But I have great difficulty separating these out to consider them individually, if that is even possible.

I have asked another related question here.

visual-studio **winforms** **architecture** **projects-and-solutions**

Share

Improve this question

Follow

edited May 23, 2017 at 12:15

Community Bot

1 ● 1

asked Sep 30, 2008 at 7:13

Benjol

66.4k ● 55 ● 192 ● 275

Related: stackoverflow.com/questions/529907/… – Benjol Aug 13, 2010 at 10:59

9 Answers

Sorted by: Highest score (default)

5

Source Control

We have 20 or 30 projects being built into 4 or 5 discrete solutions. We are using Subversion for SCM.

1) We have one tree in SVN containing all the projects organised logically by namespace and project name. There is a .sln at the root that will build them all, but that is not a requirement.

2) For each actual solution we have a new trunks folder in SVN with SVN:External references to all the required projects so that they get updated from their locations under the main tree.

3) In each solution is the .sln file plus a few other required files, plus any code that is unique to that solution and not shared across solutions.

Having many smaller projects is a bit of a pain at times (for example the TortoiseSVN update messages get messy with all those external links) but does have the

huge advantage that dependancies are not allowed to be circular, so our UI projects depend on the BO projects but the BO projects cannot reference the UI (and nor should they!).

Architecture We have completely switched over to using MS SCSF and CAB enterprise pattern to manage the way our various projects combine and interact in a Win Forms interface. I am unsure if you have the same problems (multiple modules need to share space in a common forms environment) but if you do then this may well bring some sanity and convention to how you architect and assemble your solutions.

I mention that because SCSF tends to merge BO and UI type functions into the same module, whereas previously we maintained a strict 3 level policy:

FW - Framework code. Code whose function relates to software concerns. BO - Business Objects. Code whose function relates to problem domain concerns. UI - Code which relates to the UI.

In that scenario dependancies are strictly UI -> BO -> FW

We have found that we can maintain that structure even while using SCSF generated modules so all is good in the world :-)

answered Sep 30, 2008 at 8:07

Ewan Makepeace

5,416 ● 9 ● 33 ● 31

2

To manage dependencies, whatever the number of assemblies/namespaces/projects you have, you can have a glance at the tool NDepend.

Personnaly, I foster few large projects, within one or several solutions if needed. I wrote about my motivations to do so here: Benefit from the C# and VB.NET compilers perf

Share Improve this answer

Follow

answered Oct 27, 2008 at 18:12

Patrick from NDepend team

13.8k ● 6 ● 67 ● 106

INteresting blog post. When you say projects.. do you mean seperate projects or the solutions in the project? – Piotr Kula Feb 25, 2014 at 14:31

I mean distinct VS projects in one or several VS solutions. More has been written on this here: ndepend.com/WhiteBooks.aspx – Patrick from NDepend team Feb 25, 2014 at 17:35

1

I think it's quite important that you have a solution that contains all your 80 projects, even if most developers use other solutions most of the time. In my experience, I tend to work with one large solution, but to avoid the pain of rebuilding all the projects each time I hit F5, I go to Solution Explorer, right-click on the projects I'm not interested in right now, and do "Unload Project". That

way, the project stays in the solution but it doesn't cost me anything.

Having said that, 80 is a large number. Depending on how well those 80 break down into dicrete subsystems, I might also create other solution files that each contain a meaningful subset. That would save me the effort of lots of right-click/Unload operations. Nevertheless, the fact that you'd have one big solution means there's always a definitive view of their inter-dependencies.

In all the source control systems that I've worked with, their VS integration chooses to put the .sln file in source control, and many don't work properly *unless* that .sln file is in source control. I find that intriguing, since the .sln file used to be considered a personal thing, rather than a project-wide thing. I think the only kind of .sln file that definitely merits source control is the "one-big-solution" that contains all projects. You can use it for automated builds, for example. As I said, individuals might create their own solutions for convenience, and I'm not against those going into source control, but they're more meaningful to individuals than to the project.

Share Improve this answer Follow

answered Sep 30, 2008 at 7:26

Martin

5,452 ● 31 ● 40

The other options (perhaps better?) is to use the Build Configuration Manager and mark those projects to not build (uncheck the "Build" checkbox). Unloading projects is meant to be a temporary action to edit the project file as text. If you

have other projects that reference it, those refs break.
– Scott Dorman Oct 21, 2008 at 20:33

1

I think the best solution is to break it in to smaller solutions. At the company I currently work for, we have the same problem; 80 projects++ in on solution. What we have done, is to split into several smaller solutions with projects belonging together. Dependent dll's from other projects are built and linked in to the project and checked in to the source control system together with the project. It uses more disk space, but disk is cheap. Doing it this way, we can stay with version 1 of a project until upgrading to version 1.5 is absolutely necessary. You still have the job with adding dll's when deciding to upgrade to a other version of the dll though. There is a project on google code called TreeFrog that shows how to structure the solution and development tree. It doesn't contain mush documentation yet, but I guess you can get a idea of how to do it by looking at the structure.

Share Improve this answer

Follow

answered Sep 30, 2008 at 7:48

Fossmo

2,892 ● 4 ● 25 ● 47

A method that i've seen work well is having one big solution which contains all the projects, for allowing a project wide build to be tested (No one really used this to build on though as it was too big.), and then having smaller projects for developers to use which had various related projects grouped together.

These did have depencies on other projects but, unless the interfaces changed, or they needed to update the version of the dll they were using, they could continue to use the smaller projects without worrying about everything else. Thus they could check-in projects while they were working on them, and then pin them (after changing the version number), when other users should start using them.

Finally once or twice a week or even more frequently the entire solution was rebuild using pinned code only, thus checking if the integration was working correctly, and giving testers a good build to test against.

We often found that huge sections of code didn't change frequently, so it was pointless loading it all the time. (When you're working on the smaller projects.)

Another advantage of using this approach is in certain cases we had pieces of functionality which took months to complete, by using the above approach meant this could continue without interrupting other streams of work.

I guess one key criteria for this is not having lots of cross dependencies all over your solutions, if you do, this approach might not be appropriate, if however the dependencies are more limited, then this might be the way to go.

Share Improve this answer

Follow

answered Sep 30, 2008 at 8:04

Bravax

10.5k ● 8 ● 43 ● 68

The dependency graph on 'my' code looks like a plate of spaghetti :(– Benjol Sep 30, 2008 at 8:07

Possibly that is something to look into? I find if we start to get over 10 or so dependencies per project then something is not right. Typically in those cases we can remove some dependencies by refactoring code into our framework.
– Bravax Sep 30, 2008 at 8:20

1

For a couple of systems I've worked on we had different solutions for different components. Each solution had a common Output folder (with Debug and Release sub-folders)

We used project references within a solution and file references between them. Each project used Reference Paths to locate the assemblies from other solutions. We had to manually edit the .csproj.user files to add a $(Configuration) msbuild variable to the reference paths as VS insists on validating the path.

For builds outside of VS I've written msbuild scripts that recursively identify project dependencies, fetch them from subversion and build them.

Share Improve this answer Follow

answered Sep 29, 2009 at 5:56

forgot my open id login

1

I gave up on project references (although your macros sound wonderful) for the following reasons:

- It wasn't easy to switch between different solutions where sometimes dependency projects existed and sometimes didn't.
- Needed to be able to open the project by itself and build it, and deploy it independently from other projects. If built with project references, this sometimes caused issues with deployment, because a project reference caused it to look for a specific version or higher, or something like that. It limited the mix and match ability to swap in and out different versions of dependencies.
- Also, I had projects pointing to different .NET Framework versions, and so a true project reference wasn't always happening anyways.

(FYI, everything I have done is for VB.NET, so not sure if any subtle difference in behavior for C#)

So, I:

- I build against any project that is open in the solution, and those that aren't, from a global folder, like C:\GlobalAssemblies
- My continuous integration server keeps this up to date on a network share, and I have a batch file to sync anything new to my local folder.
- I have another local folder like C:\GlobalAssembliesDebug where each project has a post build step that copies its bin folder's contents to this debug folder, only when in DEBUG mode.
- Each project has these two global folders added to their reference paths. (First the C:\GlobalAssembliesDebug, and then C:\GlobalAssemblies). I have to manually add this reference paths to the .vbproj files, because Visual Studio's UI addes them to the .vbprojuser file instead.
- I have a pre-build step that, if in RELEASE mode, deletes the contents from C:\GlobalAssembliesDebug.
- In any project that is the host project, if there are non dlls that I need to copy (text files outputted to other project's bin folders that I need), then I put a prebuild step on that project to copy them into the host project.
- I have to manually specify the project dependencies in the solution properties, to get them to build in the correct order.

So, what this does is:

- Allows me to use projects in any solution without messing around with project references.
- Visual Studio still lets me step into dependency projects that are open in the solution.
- In DEBUG mode, it builds against open loaded projects. So, first it looks to the C:\GlobalAssembliesDebug, then if not there, to C:\GlobalAssemblies
- In RELEASE mode, since it deletes everything from C:\GlobalAssembliesDebug, it only looks to C:\GlobalAssemblies. The reason I want this is so that released builds aren't built against anything that was temporarily changed in my solution.
- It is easy to load and unload projects without much effort.

Of course, it isn't perfect. The debugging experience is not as nice as a project reference. (Can't do things like "go to definition" and have it work right), and some other little quirky things.

Anyways, that's where I am on my attempt to make things work for the best for us.

answered Feb 13, 2012 at 15:26

Mafu Josh
2,672 ● 1 ● 24 ● 27

Very interesting! I've no idea if my macros will work on VB projects, but it should be easy to tweak if not. – Benjol Feb 14, 2012 at 6:26

0

We have one gigantic solution on the source control, on the main branch.

But, every developer/team working on the smaller part of the project, has its own branch which contains one solution with only few projects which are needed. In that way, that solution is small enough to be easily maintenaced, and do not influence on the other projects/dlls in the larger solution.

However, there is one condition for this: there shouldn't be too much interconnected projects within solution.

Share Improve this answer Follow

answered Sep 30, 2008 at 8:43

Nenad Dobrilovic
1,545 ● 1 ● 16 ● 29

0

OK, having digested this information, and also answers to this question about project references, I'm currently working with this configuration, which seems to 'work for me':

- One big solution, containing the application project and all the dependency assembly projects

- I've kept all project references, with some extra tweaking of manual dependencies (right click on project) for some dynamically instantiated assemblies.
- I've got three Solution folders (_Working, Synchronised and Xternal) - given that my source control isn't integrated with VS (*sob*), this allows me to quickly drag and drop projects between _Working and Synchronised so I don't lose track of changes. The XTernal folder is for assemblies that 'belong' to colleagues.
- I've created myself a 'WorkingSetOnly' configuration (last option in Debug/Release drop-down), which allows me to limit the projects which are rebuilt on F5/F6.
- As far as disk is concerned, I have all my projects folders in just one of a few folders (so just one level of categorisation above projects)
- All projects build (dll, pdb & xml) to the same output folder, and have the same folder as a reference path. (And all references are set to Don't copy) - this leaves me the choice of dropping a project from my solution and easily switching to file reference (I've got a macro for that).
- At the same level as my 'Projects' folder, I have a 'Solutions' folder, where I maintain individual solutions for some assemblies - together with Test code (for example) and documentation/design etc specific to the assembly.

This configuration seems to be working ok for me at the moment, but the big test will be trying to sell it to my colleagues, and seeing if it will fly as a team setup.

Currently unresolved drawbacks:

- I still have a problem with the individual assembly solutions, as I don't always want to include all the dependent projects. This creates a conflict with the 'master' solution. I've worked around this with (again) a macro which converts broken project references to file references, and restores file references to project references if the project is added back.
- There's unfortunately no way (that I've found so far) of linking Build Configuration to Solution Folders - it would be useful to be able to say 'build everything in this folder' - as it stands, I have to update this by hand (painful, and easy to forget). (You can right click on a Solution Folder to build, but that doesn't handle the F5 scenario)
- There is a (minor) bug in the Solution folder implementation which means that when you re-open a solution, the projects are shown in the order they were added, and not in alphabetical order. (I've opened a bug with MS, apparently now corrected, but I guess for VS2010)
- I had to uninstall the CodeRushXPress add-in, because it was choking on all that code, but this was before having modified the build config, so I'm going to give it another try.

Summary - things I didn't know before asking this question which have proved useful:

1. Use of solution folders to organise solutions without messing with disk
2. Creation of build configurations to exclude some projects
3. Being able to manually define dependencies between projects, even if they are using file references

This is my most popular question, so I hope this answer helps readers. I'm still very interested in further feedback from other users.

Share Improve this answer

Follow

edited May 23, 2017 at 12:08

Community Bot

1 ● 1

answered Nov 18, 2008 at 11:14

Benjol

66.4k ● 55 ● 192 ● 275