AUTODESK UNIVERSITY 2014

SD6006: Extending Autodesk Products Like a Pro: A Crash Course in Software Development

Mark Perrott, Thomas Closs – Autodesk Inc.

Learning Objectives

At the end of this class, you will be able to:

- Learn how to apply modern techniques to maintain and back up scripts and source code
- Understand the fundamentals of the software development lifecycle
- Learn how to deploy additional tools to enable an easier coding workflow
- Learn how to create and understand the value of automated tests

About the Speaker

Mark Perrott is a solution architect in the Autodesk Consulting Team at Autodesk, Inc.

Coming from a software development and systems integrations background, Mark has experience in a range of Autodesk products and APIs. He specializes in utilities and data management, and has successfully architected and deployed Autodesk software into large organizations building on Vault, Infrastructure Map Server, Revit, and AutoCAD. Mark holds a master of information technology degree and a bachelor's degree in multimedia (networks and computing).

Tom Closs has been working with Autodesk products and technology since 1992.

He has cross industry experience in both manufacturing and architectural fields. As a Solution Architect for Autodesk Consulting, Tom uses his knowledge to help companies integrate different systems and technologies to improve their processes and lower the costs of doing business. The creative problem solving involved in solving these types of problems, more than trickles over into Tom's personal life as a lifelong Maker. When it comes down to fixing something at home or 3D printing a unique tool for a project Tom is always looking to use tools and technology to work smarter not harder.

The Ivory Tower

KISS - Keep it super simple

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

— Brian W. Kernighan and P. J. Plauger in The Elements of Programming Style.

The KISS principle states that most systems work best if they are kept simple rather than made complicated; therefore simplicity should be a key goal in design and unnecessary complexity should be avoided.

When adding functionality to an application we must always be mindful of the additional burden placed on the complexity of the system. As systems grow in complexity they become more difficult to maintain and debug.

Often this problem manifests itself in code when a developer is eager to use a new technology, framework or pattern without just cause for its implementation.

http://en.wikipedia.org/wiki/KISS_principle

Technical Debt

"a debt that you incur every time you avoid doing the right thing… letting the code quality deteriorate over time" — Martin Fowler

It is a rare occurrence to have the amount of time we would like to implement a perfect solution. Inevitably, the code we create will contain hacks and kludges. This should not be seen as an action of shame or something that must be covered up as it is something that every developer is guilty of. However, when we implement a less than optimal solution we must do so consciously and with an understanding that we are incurring technical debt.

The technical debt metaphor suggests that just like balancing a budget we can incur debt for a while, but it is not a sustainable state. As a system incurs more technical debt the ability to modify, reason with or extend the system becomes more and more difficult. Working with a system with technical debt requires additional effort and can be seen as "paying interest" in the form of additional effort.

We can "pay off" technical debt by cleaning up the code and addressing poor design decisions by refactoring, restructuring the code without changing its external behavior.

This metaphor can be useful to communicate why effort is being spent on maintaining code, or why a feature will be difficult to implement to non-technical colleagues.

Systems requiring little change can retain technical debt, but too much accumulated technical debt can lead to "technical bankruptcy", code re-write.

http://en.wikipedia.org/wiki/Technical_debt

YAGNI - You aren't going to need it

Often as software developers we look at a rapidly changing business and believe we can anticipate the future needs of users or the business and write our code accordingly. Unfortunately even the best guesses are often wrong. This leads to effort wasted on code that's never used and increased complexity of r no value.

http://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it

Standards & Conventions

When working with developing solutions it is important to develop and stick to rigorous standards for how we write and layout source code. These standards extend to how we name objects and constructs, format our comments, layout namespaces and directories and beyond.

It is important to work to standards and conventions as it not only enables the ease of sharing with other developers, but also an ease of understanding for ourselves.

Some examples of naming conventions can be:

- Method (Pascal Case): .ToString()
- Parameter (Camel Case): Array.Copy(Array sourceArray, …)
- Field (Camel Case usually prefixed): int _testNum, mTestNum

Choosing a specific convention over another is important, but the most important factor when naming variables is to ensure that your names are meaningful. Although you may save some initial keystrokes, you will waste hours trying to work out what your obscurely named variables mean later.

e.g.:

- du vs. defaultUser
- foo vs. selectedShape
- a1, a2, a3, a4 vs. Anything!

When choosing coding and conversions it is recommended to use industry standards. This will allow you to quickly consume the majority of community content. For C# the following links are Microsoft's recommendations:

- C# Coding Conventions (C# Programming Guide)
  - http://msdn.microsoft.com/en-us/library/ff926074.aspx
- MSDN Naming Guidelines
  - http://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx

DRY – Don't repeat yourself

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

— Andy Hunt and Dave Thomas, The Pragmatic Programmer

One of the most important paradigms in creating code it to not repeat yourself. If a piece of functionality exists it should only exist in one place. The lack of repetition leads to a single source of logical truth and less code to manage.

When a piece of logic must be updated if the code is repeated it must be updated in every place that it exists.

http://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Do one thing well

When designing and developing object orientated code each class, function, variable, etc. should define a single responsibility. When a class or function takes on multiple responsibilities the code becomes less modular and more fragile.

As a litmus test to determine if a class has too many responsibilities we should try to describe the functionality of the class. If we feel compelled to use the word "and" when describing the class, we should probably consider braking the class into two or more.

For example: "The class performs page layout and printing".

This class should probably be split into a class that does "page layout" and another that does "printing". This would mean that we can make changes to printing with a stronger degree of confidence that we won't be affecting "page layout". Additionally, our code becomes more "composeable" and if we wanted to develop "plotting" or a "export to file" we could do so by simply replacing our implementation of printing.

http://en.wikipedia.org/wiki/Single_responsibility_principle

Premature Optimization

"The root of all evil. Yet we should not pass up our opportunities in that critical 3% … after that code has been identified" — Donald Knuth

Optimizing code early often leads to wasted time. Optimizing code to make the program run more efficient and faster is a good thing to do but as a developer you need to ask yourself if the application is already running at an acceptable speed or if the amount of resources the application is using are acceptable. If spending a large amount of time optimizing the application

code only gains .01 seconds of processing time or only saves 8 bytes of memory is it really worth the time investment to optimize the code.

Breaking application code into smaller logical components that can be reused is not the same as optimizing application code. As mentioned when discussing the topic of not repeating yourself using small repeatable chunks of logic should be done as soon as these logical blocks are defined.

http://en.wikipedia.org/wiki/Program_optimization#When_to_optimize

Enforce Boundaries - Encapsulation

Your code should only know what it needs to know. Computers are fantastic at maintain the state of millions of variables at any given time in memory. Unfortunately, the human brain can only concentrate on a handful at one time. The more variables that your code has access to the more you need to think about when developing the code and the more difficult time you'll have when debugging.

One way we can bridge this gap is by encapsulation or scoping the access of variables to only the aspects of code that need it.

http://en.wikipedia.org/wiki/Encapsulation_%28object-oriented_programming%29

The Shop Floor

Source Control

Source control is essential for collaborating between to developers working on the same project. Source control allows the developers to work or the same source code with little worry about one developer overwriting the others code. When working as a single developer source control allows the single developer to keep a running backup. At any point the developer can refer back to any version of their source code or create branches of the code. This allows the developer to try different scenarios when working on their solution.

There are several free source control platforms a developer can setup and use easily. Git (https://github.com/) allows a developer to setup a project in the cloud and allow the public to view and build of their project code. As a developer if you want to use Git and keep your code private you can choose to pay a monthly fee for this additional functionality. One other advantage of using Git is the social crowd sourcing that a project can take on using the GIT social sharing tools

Visual SVN allow a developer to setup a subversion repository on an internal server. This keeps your code private but prevents the developer from using the social tools of Git. VisualSVN can be downloaded from https://www.visualsvn.com/server/download/.

http://en.wikipedia.org/wiki/Revision_control

Reinventing the wheel

Whenever we add functionality to code we need to go through the entire software development process from requirements gathering to testing. This effort if often spent in vain as common functionality may have already been developed.

When we develop code our first evaluation must be “Do we need to write this or is it already available?” Our first place to look for existing implementations is in the framework provided by our chosen language. E.g. For C# / VB.Net the Core Framework contains a wealth of capabilities.

If the capabilities don’t exist in the Framework we can start to look in third party libraries. Often third party libraries are available that contain what we need for free or very liberal software licenses.

The easiest way to obtain these third party libraries is through a package management service. In .Net the package management service is an extension for Visual Studio called Nuget. Nuget contains a library of thousands of assemblies that can be added to your code to provide additional functionality. Nuget also automatically downloads the libraries and references them in your projects.

Package Managers

- Ruby – Gems
- Java – Maven
- Node.js – NPM
- Microsoft .Net – Nuget

http://en.wikipedia.org/wiki/Package_manager

Developer Tests

All developer tests their code to some extent. Sometimes this is just ensuring that the latest version of the code compiles and runs, other times this may be using test harnesses to exercise only specific sections of your code.

One way to test your code is by implementing structured tests in code. These tests provide the ability to ensure the correctness of your code thought the software development lifecycle.

When written correctly structured tests can be ran on each build of you code to ensure that no regression has occurred.

As you develop tests for your code you gain the ability to modify your existing code with the confidence that you are not adversely effecting existing functionality.

The most common testing framework for .Net is MSTest. This test framework provides us the ability to execute tests on the logic of our code from within Visual Studio.

http://en.wikipedia.org/wiki/Test_automation

Automation

Mechanisms exists for automating many aspects of the software development lifecycle. When we automate a process we reduce the risk of introducing human error and although the initial setup of automation may be an initial hit to productivity the long term benefits of automation can be significant.

One way of automating the software development process is by using a “build server”, a system that can compile your code and create the required build artifacts (e.g. installers).

Some of the more popular build servers for .Net are:

- Team Foundation Server
  - http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx
- Hudson
  - http://hudson-ci.org/
- CruiseControl.Net
  - http://www.cruisecontrolnet.org/
- TeamCity
  - https://www.jetbrains.com/teamcity/

These systems can be free and relatively easy to setup and maintain for even small teams or individuals.

http://en.wikipedia.org/wiki/Build_automation

http://en.wikipedia.org/wiki/Continuous_integration

Help

When writing code remember, you are not alone. The internet provides a plethora of information that can both assist in your development and often outright solve your development problems.

Search Engines (Google, Bing, Yahoo, etc.)

The first step in researching a development problem is to use your internet search engine of choice. When looking for a resolution to an issue you are experience specific terms often return the best results.

Stack Overflow

The gold standard for crowd sourced programming information repositories. If Stack Overflow doesn't have the information you're after ask the question in a concise and well written manner and the community may provide the answer.

http://stackoverflow.com/

ADN (Autodesk Developer Network)

When developing with Autodesk products an ADN membership is invaluable. The ADN team exists to support developers working with Autodesk technologies and are always ready to help.

http://usa.autodesk.com/adsk/servlet/index?id=472012&siteID=123112

Autodesk Consulting

If your development needs are at a larger scale and you need someone to take over or refine the development of an application that is "running" your business an Autodesk Consulting engagement may be the right option for you.

http://www.autodesk.com/services-support/consulting/overview