SD125580 - What are all these acronyms!? An introduction to writing complex Revit addins

Colin Stark

Owner, Developer – StarkBIM Inc.

About the Speaker

- Colin Stark is the owner of StarkBIM
  - Based in Toronto, Canada
  - Provides Revit addins and custom Revit development services
- Previously worked at H.H. Angus & Associates
  - Mid-sized MEP engineering firm based in Toronto
  - Developed suite of in-house Revit tools

Learning Objectives

- Structure an application in multiple layers
- Run unit tests on an application
- Apply these techniques to create a complex Revit addin
- Understand the technologies and libraries needed to make a complex application

Main Focuses

- TDD (Test-Driven Development)
- DI & IoC (Dependency Injection and Inversion of Control)
- Use of these techniques in developing Revit applications
- Helpful tools & libraries for development

The Sample Application

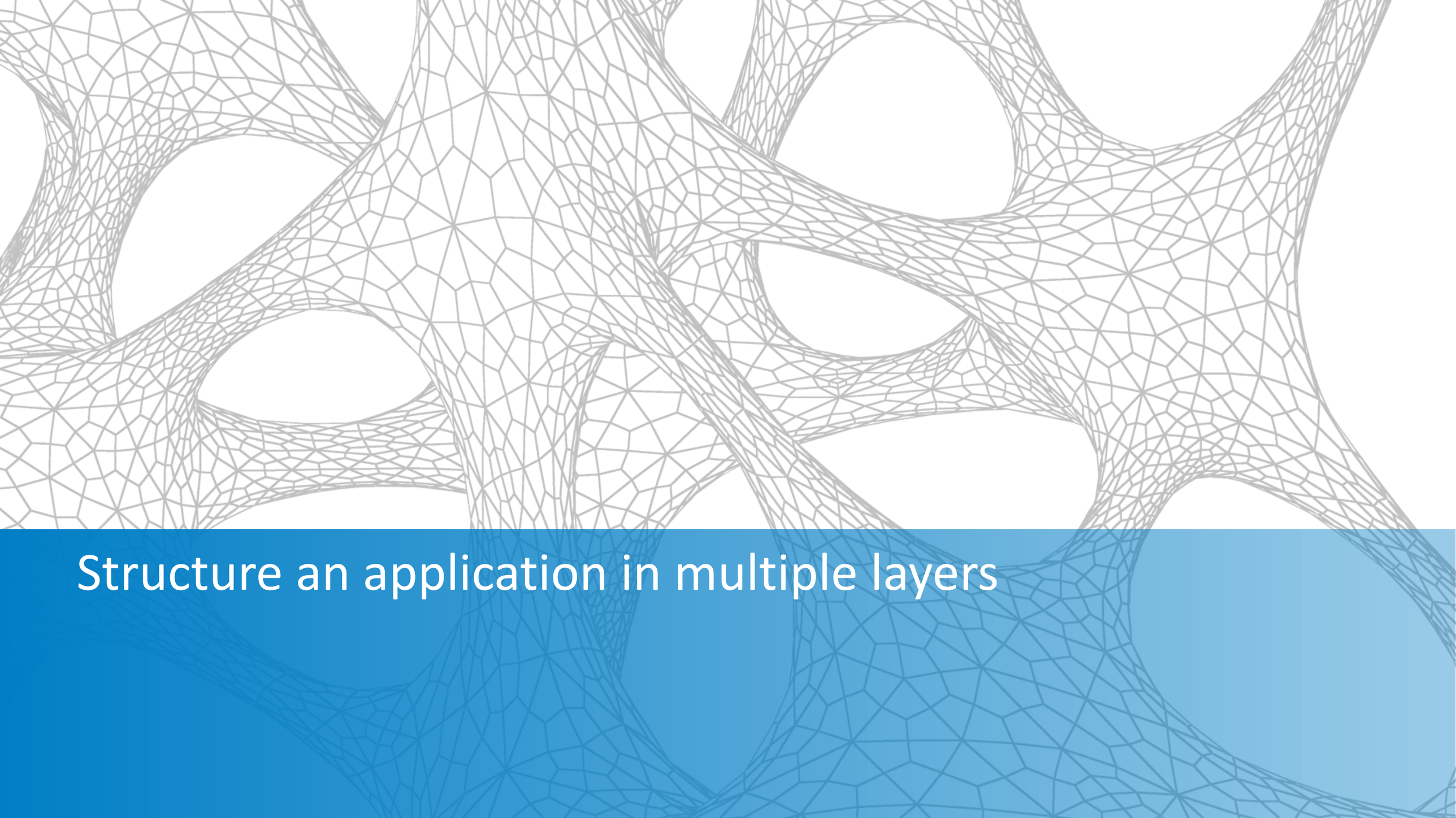
The Sample Application

- The application is available on GitHub at https://github.com/StarkBIM/SampleRevitAddin
- Topics covered will be explored through examination of the project code
- MIT licensed – use as basis for your own projects, including commercial
- Contributions and suggestions are welcome!

First Step – Identify Requirements

- Important to identify program requirements before beginning

- This application will
  - From open, active Revit document, get all sheets
  - Copy name, number and revision info to our own class named Sheet
  - Save file to CSV at user-specified location



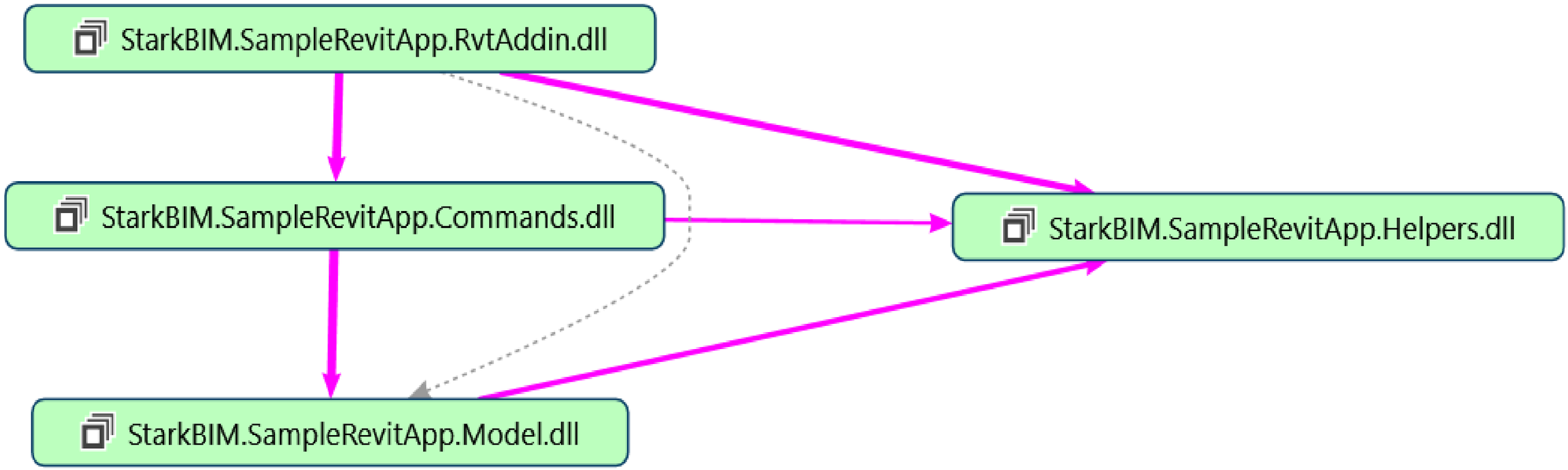
Structure an application in multiple layers

Structure an application in multiple layers

- What is a layer?
  - Logical grouping of functionality or component of an application

- What layers required for this application?
  - Data Model
  - Command
  - Addin
  - Helpers

Solution Structure



Support for multiple Revit versions

- The sample application is set up to support Revit 2015-2018
  - Only tested with Revit 2017
- The RvtAddin and Commands layer reference Revit DLLs
  - Ideally, just the RvtAddin layer would need to
    - Writing an entire layer to abstract the Revit API is very time consuming
- 2 project configurations per version – Debug, Release
- 1 conditional compilation symbol per release – RVT####
- Conditional references in csproj for Revit API DLLs

Demo – Project Structure

Test Driven Development (TDD)

Test Driven Development (TDD)

- A process in which code is written so that it satisfies existing unit tests
- A unit test is a piece of code that runs other code to assert its behavior

1. Define interface which describes expected behavior
2. Create a stub class implementing the interface methods
3. Write failing unit test that assert's the code's expected behavior
4. Write just enough code to make the test pass
5. Repeat steps 3 & 4 until class is complete
6. Refactor & optimize, ensuring all tests still pass

Benefits of TDD

- Primarily, ensures that code functions, even after it changes
  - Refactor & optimize with no fear
- Tests serve as documentation of what the class should do
  - Can always look at tests to see expected behavior
- Generally, tests are run automatically. Immediate feedback if bug introduced
  - Bugs are not all discovered after code is complete enough to run
  - Note: Not always possible with Revit
- Encourages the good software design practices

SOLID Design Principles

- 5 important rules for object-oriented programming design
  - Single Responsibility Principle
  - Open/Closed Principle
  - Liskov Substitution Principle
  - Interface Segregation Principle
  - Dependency Inversion Principle

A few more design principles

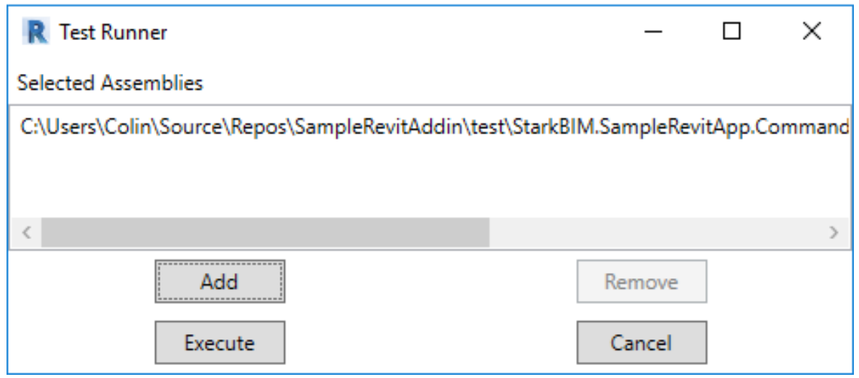
- DRY
  - Don’t Repeat Yourself
- KISS
  - Keep it simple, stupid
- Loose Coupling
  - No tight dependencies between classes
- AAA
  - Arrange-Act-Assert
    - For unit testing

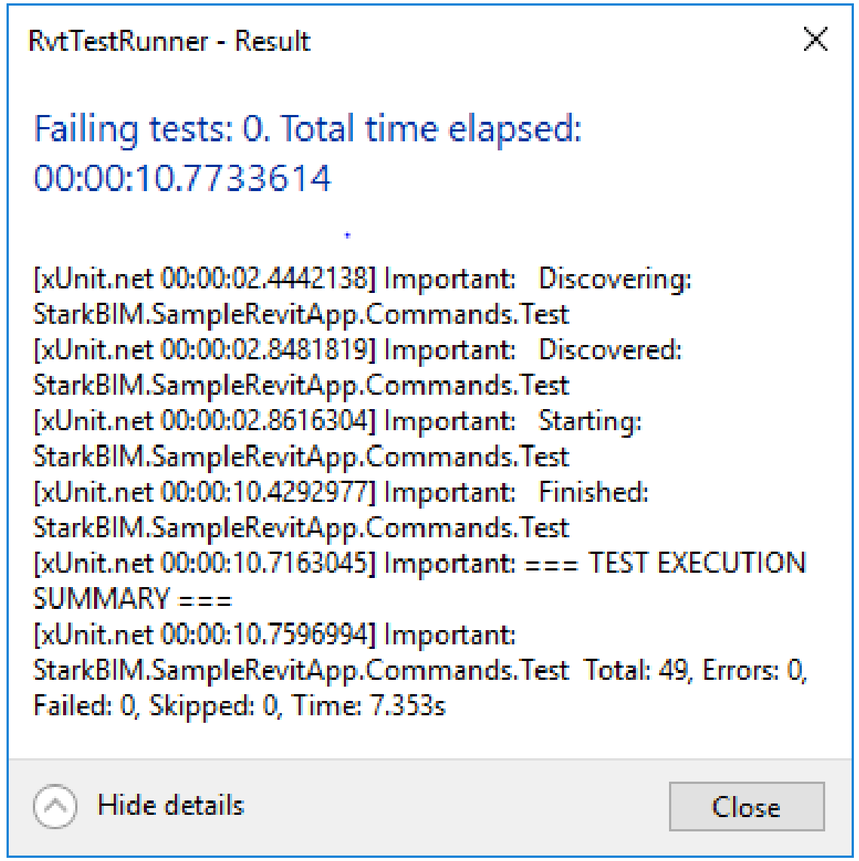
Tools required for TDD

- A testing framework
  - xUnit.net chosen for this project
- A test runner
  - RvtTestRunner, new open source project
    - Available at https://github.com/StarkBIM/RvtTestRunner
- A mocking framework
  - Allows for classes to be tested in isolation
    - Moq (free and open source) used for this project (mostly)

RvtTestRunner

- Revit API classes can only be used inside of Revit
- All common test runners operate in their own process
  - Or, inside of IDE processes
- Hence, new runner required
- Fork of uXnit.net's console runner
- Pre-alpha, limited functionality





Mocking Revit Classes

- Most free mocking frameworks only mock interfaces or virtual members
  - No good for Revit classes!
- Telerik JustMock is used to create mocks of Revit objects
  - Non-free software. Still have occasional issues with it
- Free library Pose looks promising, but could not get it to work
  - Brand new software, will continue testing as it is developed
- Prig is another free alternative that could work, but was not tested
- Microsoft Fakes can only operate inside of Visual Studio – no good
- Typemock is another non-free option, not tested

TDD – Example

- Class from sample application used to demonstrate process
  - RvtClassMapper: copies data from Revit classes to our own classes

- Uses AutoMapper to map the data, configured with a profile
  - Special AutoMapper configuration for Revit
- Unit tests use a setup class to load Revit API DLLs
  - xUnit.AssemblyFixture library on NuGet

Demo – TDD

Create a complex Revit addin

DI & IoC

- Dependency Injection & Inversion of Control
- DI framework provides IoC container
  - Autofac used in this project
- DI framework manages initialization of classes
- Dependencies passed to class as constructor arguments
- Only dependent on interfaces, not implementation
- Configured using modules

IRvtCommand

- DI framework initializes class
  - But Revit initializes IExternalCommand
    - Cannot use DI on IExternalCommand implementations
- Create our own interface: IRvtCommand
  - 3 Members:
    - Name (get-only)
    - Display Name (get-only)
    - Run(ExternalCommandData externalCommandData)

IRvtCommandProperties<TCommand>

- Tied to a command by its generic type argument
- Separates the properties required for creating menu button
  - No need to initialize instance of command to get properties
  - No need to hardcode properties for menu buttons
- Command properties passed to command using DI
  - Name & DisplayName initialized this way
    - Needed during execution for logging, etc.
- Helper methods exist for creating menu items from CommandProperties

GenericCommand<TCommand>

- Piece that ties everything together
- Accepts a class implementing IRvtCommand as type argument
- Implements IExternalCommand
- Has TransactionMode.Manual attribute
- All menu buttons point to GenericCommand
- Uses a static property to access IoC container
  - No way around this, since class is initialized by Revit
- Initializes and runs command, returns result to Revit

Demo – GenericCommand

Demo – Menu Helpers

Demo – IRvtCommand & IRvtCommandProperties

Demo – SampleCommand

Make a change to SampleCommand

- Currently, the command does not notify the user when it completes
- No confidence in successful completion
- Make change using TDD:
  - Add IDialogService, with ShowDialog(string title, string message)
  - Create test verifying that ShowDialog is called, check for failure
  - Add ShowDialog call to command, IDialogService to constructor
  - Implement DialogService by calling TaskDialog.Show
  - Register DialogService with Autofac

Demo – SampleCommand + DialogService

WPF & MVVM

- WPF: Windows Presentation Foundation. Successor to WinForms
- MVVM: Model-View-ViewModel
  - Architectural pattern to separate UI code from business logic
  - WPF supports MVVM through DataBinding
- Normal WPF applications have App.xaml for sharing resources
  - Not possible in Revit addin
- RvtWindow class solves this issue
  - Class inherited from WPF window, loads ResourceDictionary automatically

Demo – RvtWindow & MVVM

Understand the technologies and libraries needed to make a complex application

Technologies already covered

- DI/IoC (Autofac)
- TDD/Unit Testing (RvtTestRunner, xUnit, Moq, JustMock)
- AutoMapper
- WPF/MVVM
- ReactiveUI
- ReSharper
- Also used: Ookii Dialogs for save file control
- Also used: CsvHelper for writing to CSV file

Version Control

- Every project should use a version control system
  - Git is really the only viable option for new projects
- Can be used in conjunction with a CI system (Continuous Integration)
  - Part of a build server which can automatically (among others)
    - Build project
    - Run tests
    - Package & release new version if all tests pass

Static Code Analysis

- Static code analysis allows code to be analyzed without executing it
- Various packages allow checking both style and content of code
  - ReSharper has hundreds of checks, supplemented by its annotations
    - NotNull/CanBeNull, MustUseReturnValue, ContractAnnotation
      - All cause warnings for incorrect use
  - StyleCop analyzers check code for formatting
  - FxCop analyzers check code for content
  - xUnit analyzers check that tests are written correctly

Demo – Static Code Analysis

NuGet

- NuGet is a package management system
- It contains libraries that can be included in projects
- Thousands of libraries available
- Check before writing your own!
- Other commonly required packages not used in sample
  - Data Validation (e.g. FluentValidation)
  - User Controls (e.g. Ookii Dialogs, Extended WPF Toolkit)
  - Configuration (e.g. config.net)

What's Next?

- Code Signing
  - Revit addins should be signed to avoid annoying warnings
- Eliminate dependence on JustMock
  - Continue to explore the use of Pose, Prig
  - Alternately, add another layer that abstracts Revit API
- Dynamic command loading in GenericCommand
  - Probably using MEF (Managed Extensibility Framework)

What’s Next?

- Continued development of RvtTestRunner
  - Code coverage
  - Continuous testing
  - Reduce copied xUnit code to a minimum
  - Set up build server to automatically sign and deploy
- Community involvement!
  - Stronger Revit developer community benefits all

Q & A

AUTODESK®

Make anything.