322279

Automate Your Revit Add-In Testing with Unit Testing

Patrick Fernbach
KLH Engineers

Corey Smith
KLH Engineers

Learning Objectives

- Explore end-to-end tests, integration tests, compatibility tests, and unit tests to create better Revit add-ins
- Learn how to create a unit test project in Microsoft Visual Studio
- Learn how to run a unit test on Revit
- Learn how to apply testing philosophies to your team/company to create better products and increase development speed

Description

Do you ever update a Revit add-in and find that the user sees bugs that shouldn't have been there? Do you have a difficult time getting developers and users to test tools consistently? There are a few types of testing that can help, including end-to-end tests, integration tests, compatibility tests and unit tests. Whether you're a new or experienced programmer, you should know these testing concepts to create testable and maintainable code.

Do you feel like testing takes too long? Creating automated unit tests is a fast way to test code, including new and old features. Unit testing will not only create higher-quality products, but it will also make adding new features faster.

So, you don't currently focus on testing? This session will go through testing concepts and teach you how to create a unit test project and a simple Revit unit test. In addition, it will share the tricks that KLH Engineers, PSC (KLH) uses to push testing to its development team and employees to create better products.

Speaker(s)

Patrick Fernbach specializes in HVAC and plumbing system design at KLH and serves on the software development team. He translates the MEP engineers' needs to software engineers' in order to develop process improvements. He also assists in the creation of Revit add-ins and leads quality assurance and testing efforts to ensure the custom tools are of high quality. In addition, Patrick leads a team that focuses on training employees across three offices and five business units on built-in and custom Revit and AutoCAD tools. The team's goal is to standardize the training documentation, software story creation and feedback channels throughout the firm. Through this collaboration and training, the firm has experienced increased

employee engagement to do more with less. Patrick holds a Bachelor of Science in mechanical engineering from the University of Cincinnati.

Corey Smith is a lead mechanical designer at KLH with over 10 years of experience designing commercial, retail and hospitality buildings. As an expert in CAD and Revit, Corey leads a team of in-house software programmers that develop custom tools and workflows that enhance construction document production. Corey is a frequent presenter at the Cincinnati BIM User Group, which keeps Revit users up to date on the latest BIM technologies. He holds a Bachelor of Science in industrial technology with a focus in computer aided drafting from Morehead State University.

Introduction

There is nothing worse than releasing a new software/tool update and the user getting multiple errors while trying to use it. Errors while using tools can cause the user to lose work, not be able to start tasks, and can destroy the relationship between the user and the development team. It also makes it difficult to sell the tools. Without testing, live demos are unreliable. Nothing is worse than when the user first launches the tool and they get an error, or a FATAL ERROR. Software developers have a responsibility to properly and consistently test tools prior to rollout. Software developers can deploy a few different types of tests to structure the testing environment. Teaching these testing techniques to new users can increase testing, therefore increasing the quality of the tool. The testing types are end-to-end testing, integration tests, compatibility tests and unit tests.

End-to-End Testing

The purpose of end-to-end (E2E) testing is to simulate the user experience from start to finish. E2E testing is meant to be user experience focused. It is intended to validate data inputs from the user and outputs from each application within the workflow. Many times, different software teams may create different applications that are used in a single workflow. E2E testing brings those applications together to ensure that each step in the workflow transfers the proper objects to make a seamless flow of information. This session will cover how to structure E2E testing and when to perform the tests.

Randy Deutsch has coined the term “Superusers,” liaisons between business needs and technology solutions. KLH embraces its superusers as the individuals that carry out the E2E testing. These superusers are people who understand the process and the technology, validating how the tool looks, feels and operates. Superusers’ knowledge of the process cannot easily be transmitted to software engineers; thus, they tend to give more feedback related to overall workflow of the tool(s).

Integration Testing

Integration testing involves taking the individual software modules and testing them as a group. It determines issues between the individual modules when they interactive together. KLH utilizes a structured merging system for integration testing. This merging happens between software teams and the quality assurance (QA) team.

Currently, KLH utilizes three branches in DevOps: Development, Testing and Production. The development branch is where all the new features get created by the software engineers. At the end of each sprint cycle, the development branch is merged to the testing branch. The testing branch is automatically distributed to select team members. At the end of the next sprint cycle, the testing branch is merged to the production branch. The entire company (minus software engineers and QA team) automatically get the production branch. Bugs are typically addressed in production and/or testing branches and merged up to development branch as required.

At KLH, there is a Revit development team and a software team developing its enterprise resource planning (ERP) services. The Revit team often leverages the ERP team’s methods and classes to coordinate internal budgeting and modeling workflows. When the ERP team

makes updates, it is critical for the Revit team to perform integration testing prior to rollout to ensure that all tools are properly working in all three branches.

Compatibility Testing

Compatibility testing is valuable, in the sense that the tools must work in all versions or Revit, at various connection speeds, different hardware, etc. Tools must be tested forward and backward to ensure that bugs are properly dealt with.

KLH has four regional offices, all of which have different connections speeds to the servers than our main office. Software engineers must perform tests that throttle the connection speed to simulate the regional offices' environment. If the load time exceeds acceptable thresholds, the software engineers explore optimization solutions. Optimizing can include caching data and reworking the methods.

KLH is a MEP engineering firm and the version of Revit is often dictated by the architectural counterpart. To stay flexible and effective for all our clients, KLH maintains the latest four versions of Revit (2017, 2018, 2019, & 2020) depending on API capabilities. KLH software engineers start by developing in 2020. When the feature is completed, it is the responsibility of the software engineers and the QA team to test in all versions of Revit to confirm functionality.

Unit Testing

A unit test is intended to test the smallest possible part of any software. An individual component is selected, and unit tests are written to verify that, when given certain inputs, the outputs received are expected. Unit tests can also be used to check that errors are properly being handled. KLH writes all unit tests in C#.

What to Use

Below are the tools that KLH uses to setup Revit automated unit testing.

- NUnit - KLH started out using Microsoft's unit testing libraries but switched to NUnit to overcome complications with those built-in libraries.
  - https://nunit.org/
- Revit Test Framework (console) - The Revit Test Framework (RTF) allows software engineers to conduct remote unit testing on Revit. RTF creates a journal file for running Revit, specifying the model to run it in and the specific tests to run.
  - https://github.com/DynamoDS/RevitTestFramework
- Batch file containing instructions for the Revit Test Framework
- Visual Studio – This is the platform that KLH uses to write all our add-ins, so we leverage this for all of our unit test writing as well.
- Test project setup in Visual Studio – This project contains the ribbon and corresponding ribbon add-in file.
- Test model in Revit – The QA team setup a Revit model in which to run automated unit testing. This model only has some standard KLH families, all settings and naming match the standard. There is no modeling, schedules, details, etc. in this

model. If unit tests related to those components are required, separate models are to be created and the model locations will be updated in the RTF.

How to Set it Up

1. Add NUnit NuGet package to Visual Studio
2. Download RTF from Github and follow the posted instructions
    - Be sure to include RTF in the solution
3. Wrote a batch file to utilize the RTF
    - The batch file copies over the add-in file and the Revit test model into the current directory
    - Then it launches the RTF, referencing the testing dll file, and targeting a Revit exe file to use
    - After the testing is ran, the batch file deletes the copied add-in and Revit file, as well as all log files added to the parent folder
4. Wrote tests using the Revit API and NUnit
    - Declare a constant string in a class to reference the test model in Revit:
        - `public const string TestModel = @"KLH2019TestModel.rvt";`
    - Unit tests references this test model in a RTF attribute
        - `[TestModel(Variables.TestModel)]`
            - TestModel: RTF attribute
            - Variables: static class
            - TestModel: the constant string
        - This is needed to use the Revit API for that unit test
    - Leveraged general unit testing syntax and structure
        - NUnit general attributes
        - Arrange, Act, Assert structure is leveraged
        - Example of a geometry unit test:

```
[Test]
[TestModel(Variables.TestModel)]
0 references | Jacob Reiter, 3 days ago | 2 authors, 3 changes | 2 reviews
public void TestOnLine()
{
    // arrange
    _point = new XYZ(0, 0, 0.5);
    bool result;

    // act
    result = IsPointOnUnBoundLine(_point, _points, _precision);

    // assert

    Assert.IsTrue(result);
}
```

Manual Unit Testing

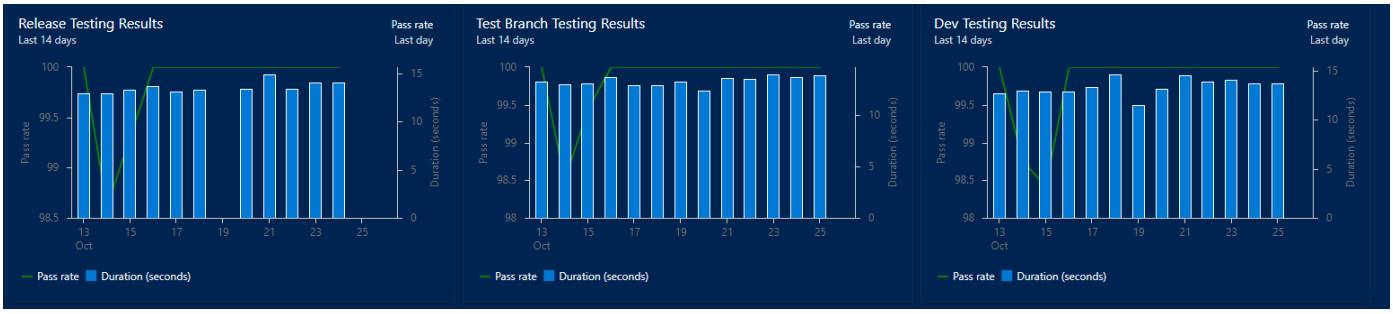
RTF allows units to manually write unit tests in the command line interface. The QA team doesn't leverage this during testing, but it can be helpful when the software engineer is writing a function and wants to confirm that it returns an expected value. Below are the setups to leverage RTF for manual.

1. Build the solution (including ribbon and testing) to create dll files
2. Open command line interface (command prompt, windows powershell, …)
3. Navigate to the directory you want the results log file in
4. Run the batch file in the command line interface
    a. This will run the console version of the RTF
    b. Revit test model will open (allows use of Reit API)
    c. You will be able to see results in command line
    d. Log file will be added to current directory with test results

Automated Unit Testing

The QA team leverages RTF for automated unit testing. Automated unit testing not only tests the function directly after creation, but it also lives on with the function. If updates are made to a function, that unit tests will verify that those updates did not break current functionality automatically. KLH has automated builds of the solution in a pipeline every time a software engineer checks in code. Below are the steps for automated unit testing, the devops engineer will need to be involved.

1. Build a project build pipeline
2. Add the batch file to the pipeline
3. By leveraging automated builds, it will tell the developer if tests failed real-time
    a. See below for a snip of our build pipeline dashboard in DevOps. This is for all three branches.



Apply Testing Philosophies

KLH's core business is MEP consulting services. A software department has been added to our list of departments, so testing philosophies not only need to be applied to our software department, but also incorporated into our daily work to ensure that feedback is properly logged and communicated between the user and the software department. A QA team was created to manage all testing and communication lines were opened with the end user to facilitate

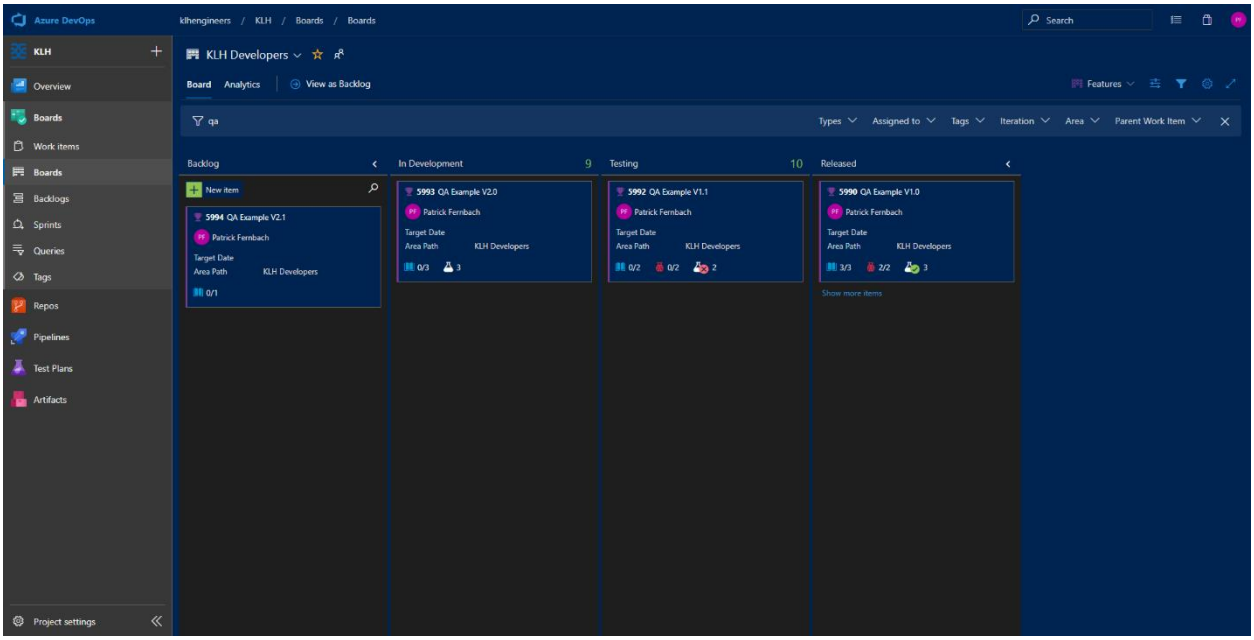
feedback within our company. Some avenues include: email, Microsoft Teams, Yammer (KLH has an KLH Software Forum Group), Software Rollout pages, and Software agent meetings.

One Team

First, decide on one team to manage and perform all E2E tests (test plans and test cases). KLH has created a quality assurance (QA) team that meets and discusses the tests regularly. By having a single team, E2E test writing and implementation can have a structured process and clear expectations. The QA team meets bi-weekly (aligned with KLH's sprint cycle) and assigns features to a QA engineer to write test plans for the feature when it goes into development. The QA team can also discuss errors that are being reported by the users and plan on how to prevent those bugs from getting to the users the next time.

One Location

Secondly, decide on one location in which the written tests will live. One location eliminates confusion of where to start in writing a test and what needs a test written. KLH has leveraged Microsoft Visual Studio's test plans and test cases structure. The tests living in the same location as the new features guarantees that each feature has a test plan written from the user's perspective. When the feature is moved to testing, the QA engineer can act on the test plan and immediately log bugs into the system for the software engineer to fix prior to the official launch of the tool. Having one location not only streamlines the QA process, but also facilitates communication between the QA team and the software engineers creating the applications.



One Location - Azure Devops Board with user stories, test plans, and bugs.

One Culture

Third, changing the culture of a company is an incredibly difficult task, but is imperative in the success of creating tools. KLH has transformed the culture of the company in such a way, that testing tools and providing feedback happens in a daily basis. Revit users are encouraged to write software stories, test developing tools and provide feedback on tools in production. Once feedback has been given to the development team, communication back to the user alerting them that their issue is being addressed has created a culture of trust between the end user and the development team.

Testing is critical to a software development team that wants to produce high quality tools with quick adoption. Now that unit tests can be written for the Revit API, testing has become more efficient and consistent. As the number of unit tests grows, the amount of time the developers must spend verifying previous functionality decreases. Thus, the amount of time dedicated to working on new features and performing more in-depth end-to-end testing increases.

In many development teams, the role of the QA team is not a glorious, respected role; however, it is where relationships between the software team and the end user are made and maintained. A strong testing team, process, and culture is the key to an innovative and sustainable software development team.