A Developer Always Pays His Tech Debts

•••

Wesley Lomax & Jonathan Robbins

Going from legacy to a Helix compliant solution

Why should I make work for myself?

Raise the quality limit

Code decay

Short term pains for long term gains

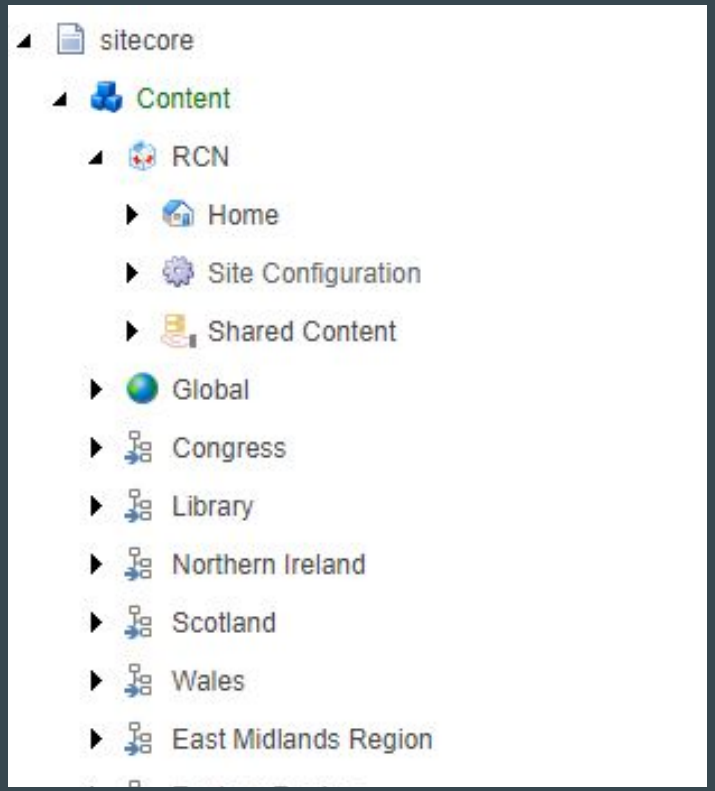
Preempt serious rework

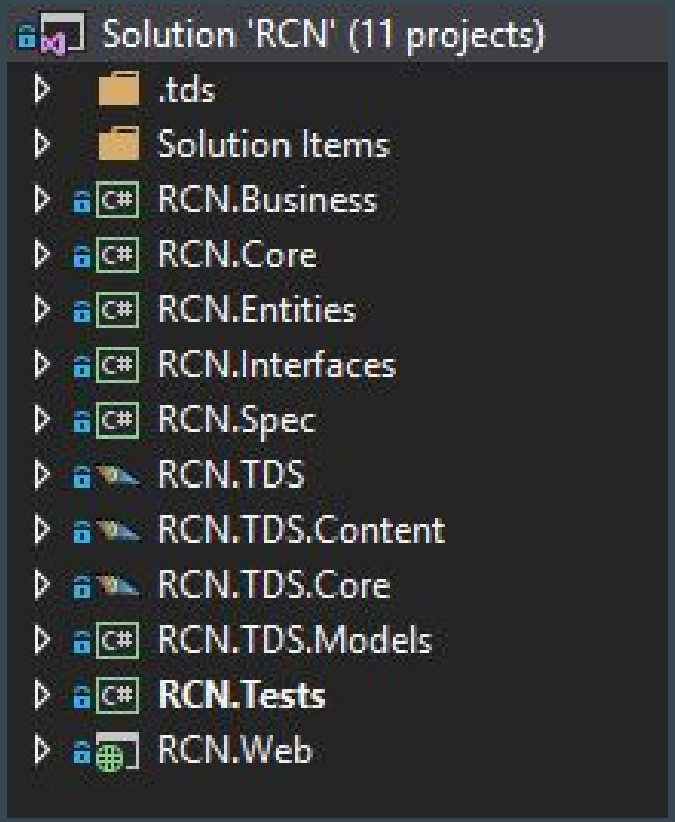
Align with the direction of the Industry

Improve as a developer

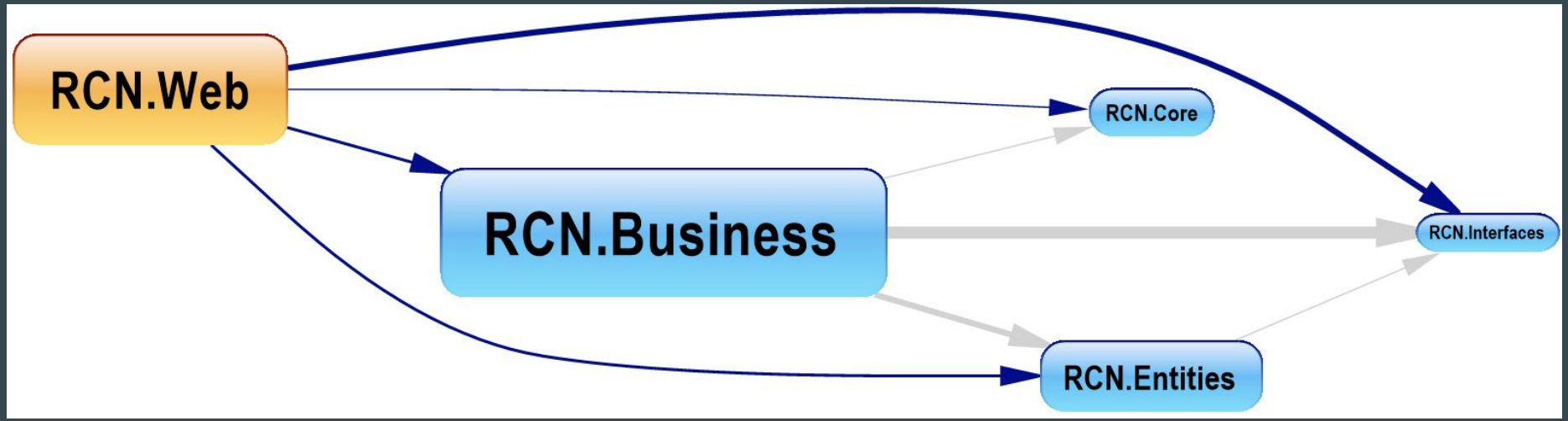
Undeveloping is fun!

What it all looked like





How the solution was structured



Figuring out what's right for us

Membership Organisation and Trade Union

Enterprise solution

Team composition

Varied audiences

Delivering a progressive product

Where we wanted to be

Harden the foundations

Remove limitations of N Tier

Enable future phases

Sort daily headaches

Use modern solutions to problems

But also...

MAKE DEVELOPMENT GREAT AGAIN

Before we got started

Buy in

Robust branching strategy

Unit Test and Regression Test plans

A few Resharper licenses

Bravery!

What we considered

What is right for us

SOLID principles

Modular Design

Helix

Development standards

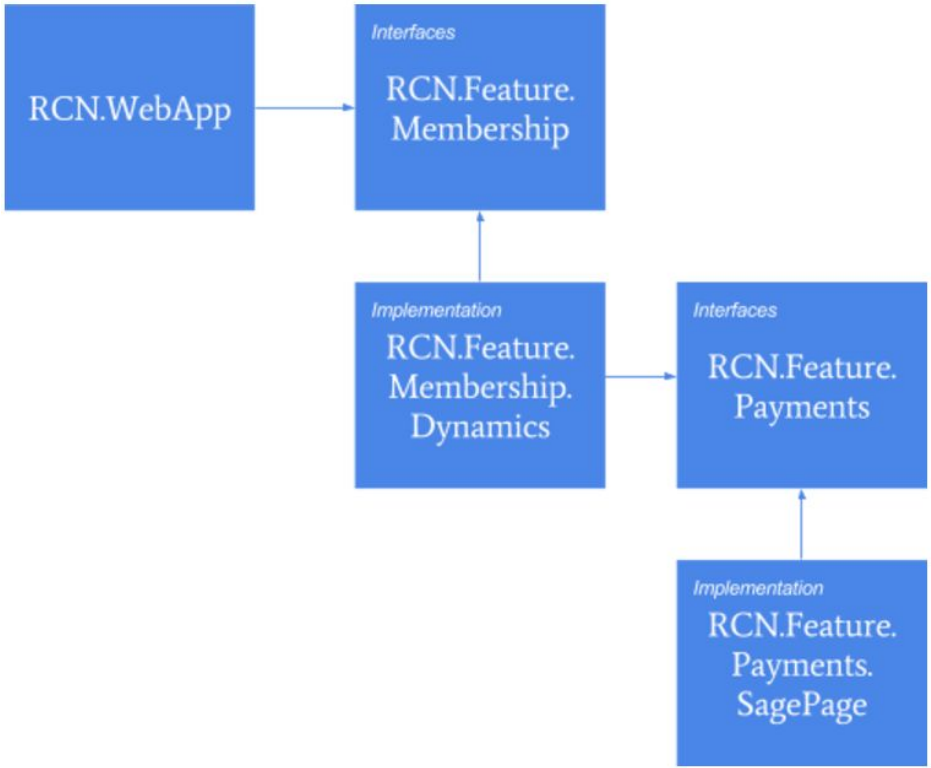
Content strategy

Design Patterns

How we went about it

1. Chose technologies, principles and design patterns
2. Rewrote development guidelines
3. Completed a solution review
4. Audited the information architecture
5. Created a transformation backlog
6. Agreed an implementation approach

Stairway Pattern



How we began implementing the changes

Big Bang approach simply not possible

Phased approach

Features at a time

Boy Scout Rule

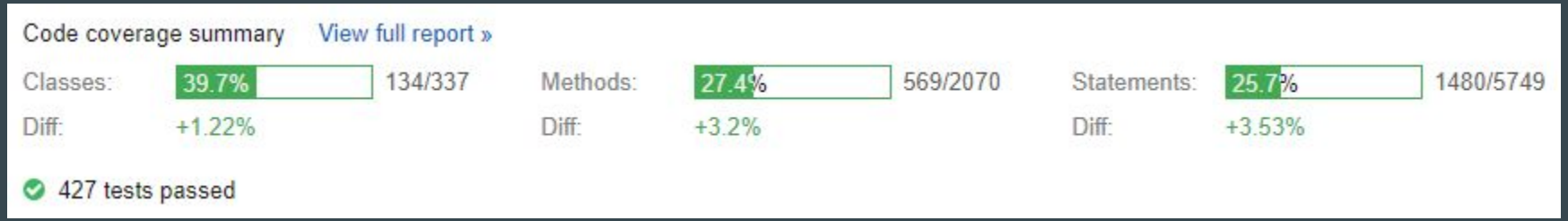
How we tested the commits

No Test or QA team!!

CI server builds every commit to every branch

Unit Tests are run

Code Coverage reports are generated



77

Days of technical debt

14

Days of technical debt

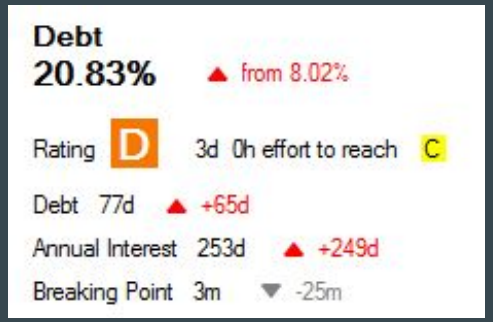
Amount of tech debt we've reduced

Technical debt rating - D

Technical Debt at 20.83%

77 days worth of debt

Breaking Point 3 months

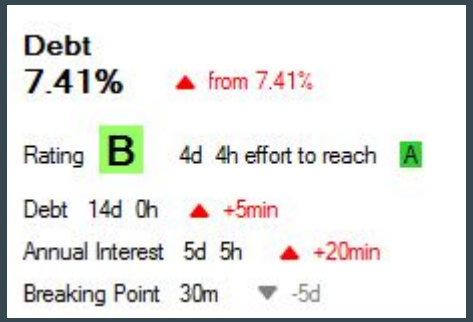


Technical debt rating - B Rating

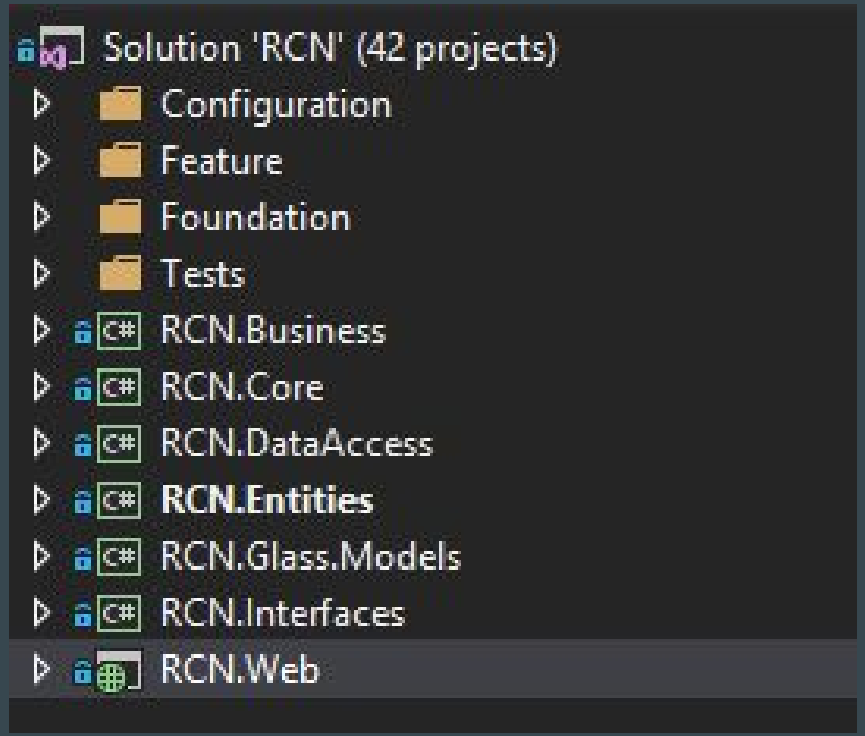
Technical Debt at 7.41%

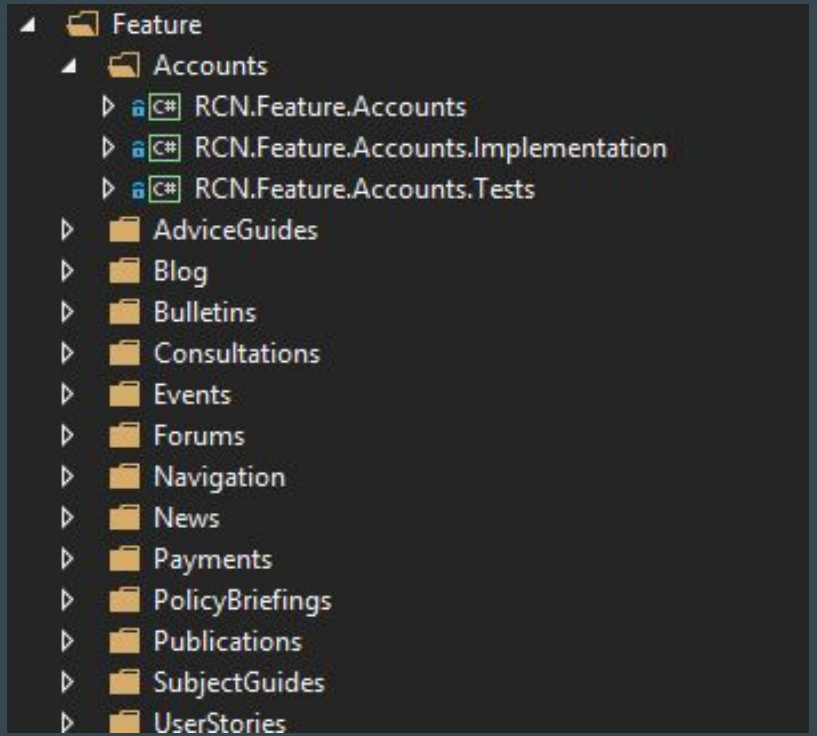
14 days of debt

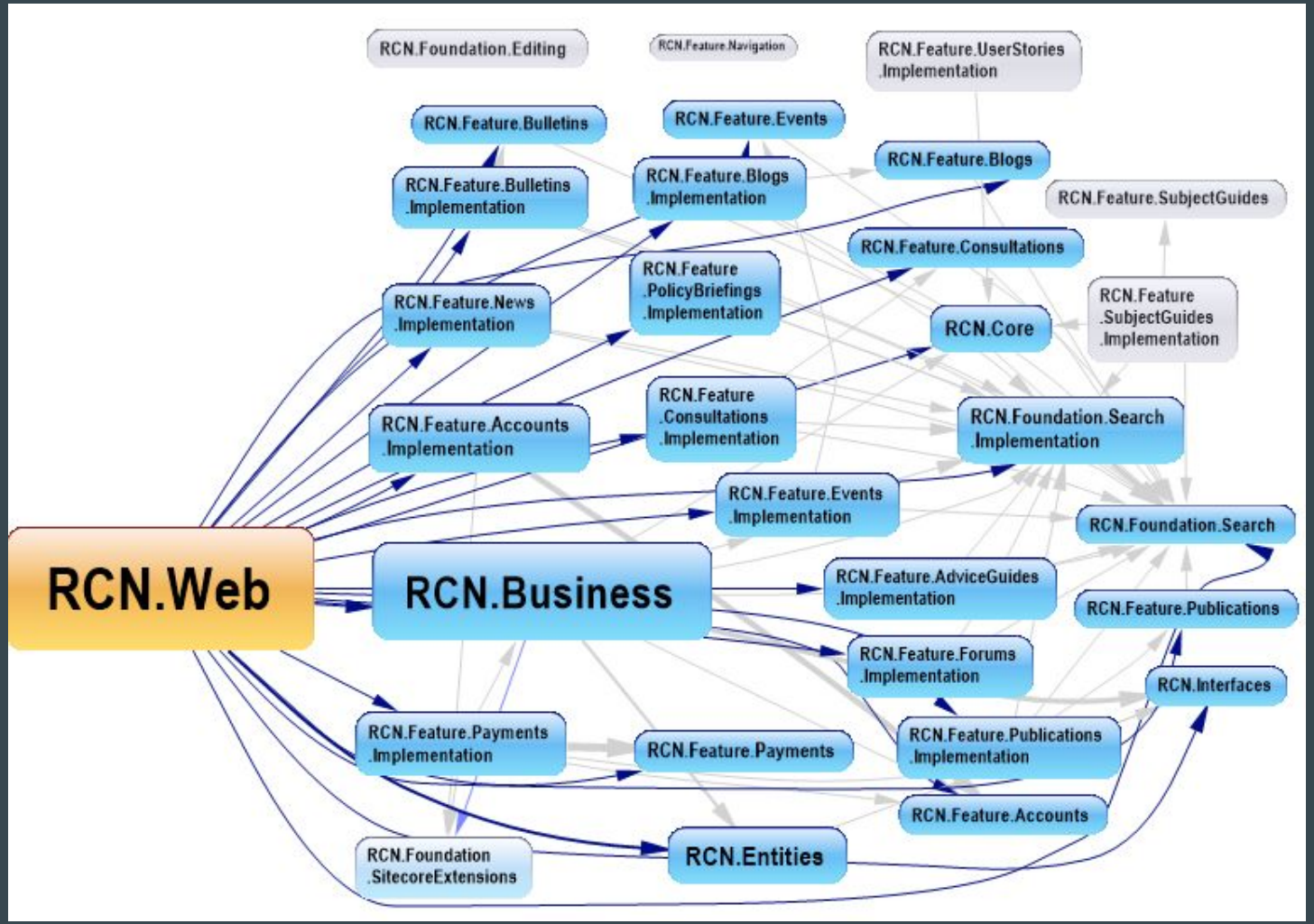
Breaking Point 30 months



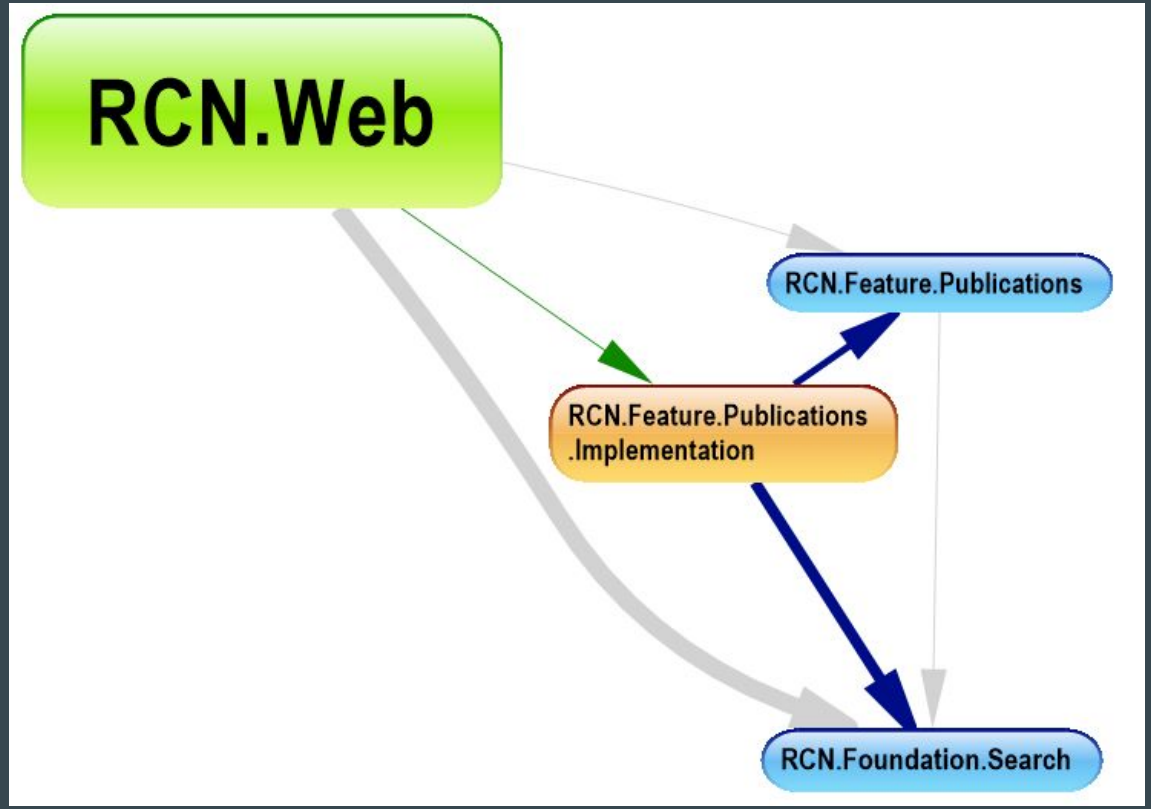
So what it all looks like now






RCN.Foundation.Editing
RCN.Feature.Navigation
RCN.Feature.UserStories
.Implementation
RCN.Feature.Bulletins
RCN.Feature.Events
RCN.Feature.Blogs
RCN.Feature.Bulletins
.Implementation
RCN.Feature.Blogs
.Implementation
RCN.Feature.SubjectGuides
RCN.Feature.Consultations
RCN.Feature
.PolicyBriefings
.Implementation
RCN.Feature.News
.Implementation
RCN.Core
RCN.Feature
.SubjectGuides
.Implementation
RCN.Feature.Accounts
.Implementation
RCN.Feature
.Consultations
.Implementation
RCN.Foundation.Search
.Implementation
RCN.Feature.Events
.Implementation
RCN.Foundation.Search
RCN.Web
RCN.Business
RCN.Feature.AdviceGuides
.Implementation
RCN.Feature.Publications
RCN.Feature.Forums
.Implementation
RCN.Interfaces
RCN.Feature.Payments
.Implementation
RCN.Feature.Payments
RCN.Feature.Publications
.Implementation
RCN.Feature.Accounts
RCN.Foundation
.SitecoreExtensions
RCN.Entities

Let’s zoom into one of those



Our take aways from doing this

Know where you're heading first

Don't go blindly following Helix

Be careful of pulling code weeds

Talk in words clients can understand

Sitecore Powershell Extensions saves (personal)lives

You might wanna check in those deployment packages...

Your IoC not being a separate project may spoil your pretty dependency graph

TL;DL - How do I sort out my legacy solution?

Get the thumbs up

Figure out what is right for you

Decide how you want the implementation to look in 12 months

Find approaches and principles that can help you get there

Review the solution against all this to build a backlog

Agree how to implement the changes with minimum risk

Go break the site