Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS | San Diego

Introducing Automated Testing into Legacy Code

Mickey Gousset
DevOps Architect
Microsoft

~~David V. Corbin~~
~~President / Chief Architect~~
~~Dynamic Concepts Development Corp~~

Level: Intermediate

Code Again for the First Time!

Visual Studio 25 YEARS OF CODING INNOVATION

About the Speaker

David V. Corbin

Microsoft MVP [ALM]
Microsoft ALM Ranger

Over 40 years professional developer experience

President / Chief Architect:

Dynamic Concepts Development Corp.

"Helping teams be better at writing better software"
Established 1984
Based in New Smyrna Beach, Florida

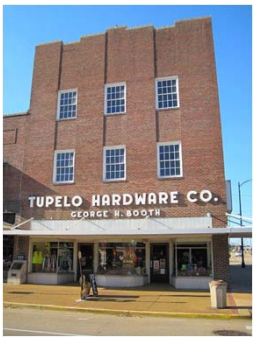
david.corbin@dynconcepts.com

10/10/2018 3

Mickey Gousset

- DevOps Architect
  - Microsoft Global DevOps Customer Advisory Team
- Formerly a 13-year Visual Studio ALM MVP
- Blogger, Author, Dad
- Current Vice: Hearthstone
- @mickey_gousset
- mickey.gousset@microsoft.com
- mickeygousset.com

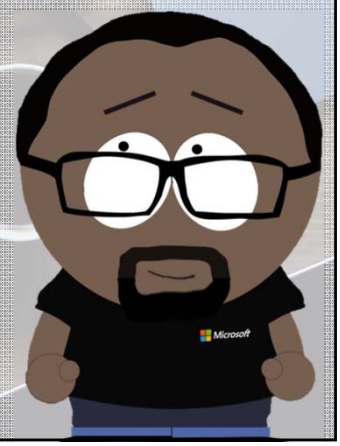
What is Tupelo, MS world-famous for?



DevOps is the union of people, process, and products to enable continuous delivery of value to our end users.

- Donovan Brown

http://bit.ly/WhatIs-DevOps



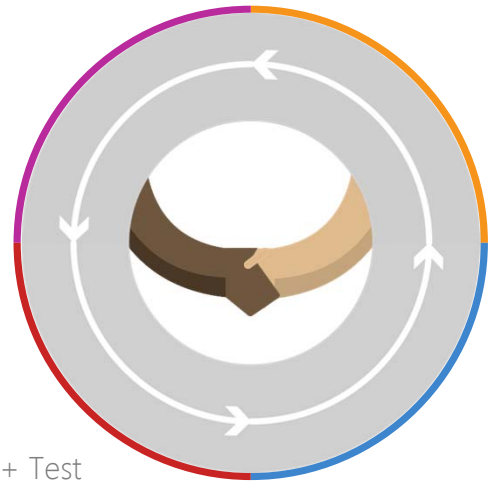
Process

1 Plan

4 Monitor + Learn

Developmen
t

Production



2 Develop + Test

3 Release

What is Legacy Code?

The Dream of Green Field Development

What is a Test?

- Definitive Testing
  - Do this; Check That
- Exploratory Testing
  - Spend some time with this and let us know your thoughts

The Starting Point…

- System / Application already exists and is somewhat "mature"
- Testing is largely limited to Manual Procedures at the System Level

We have all been here…

Typical System Level Testing

A Single Monolithic Block

So How can we Improve?

Common Inhibitors

- High Cost of Writing Automated Tests
- Low Value on Testing Existing Code
- System Level Tests are Good Enough

These are often more Perception than Fact!

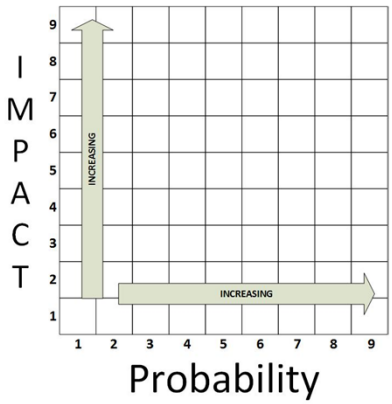
To Move Ahead we need to Mitigate These!

Our Secret Ingredients

- **We know the code** *(basically)* **works…**
  - our primary focus is on detecting "breakage"
- **We have existing manual tests…**
  - We can use this as a baseline for automation
- **System Tests can be Decomposed…**
  - Earlier Testing, Better Targeting, Faster Execution

Based on this, we can move forward

Focus Effort by Ranking



IMPACT	1	2	3	4	5	6	7	8	9
9	145	151	155	165	170	185	195	205	215
8	125	131	135	145	150	165	175	185	195
7	85	91	95	105	110	115	125	135	145
6	55	61	65	75	80	85	95	105	115
5	34	39	44	54	59	64	74	84	94
4	21	26	31	41	46	51	61	71	81
3	13	18	23	33	38	43	53	63	73
2	8	13	18	28	33	38	48	58	68
1	5	10	15	25	30	35	45	55	65

Probability

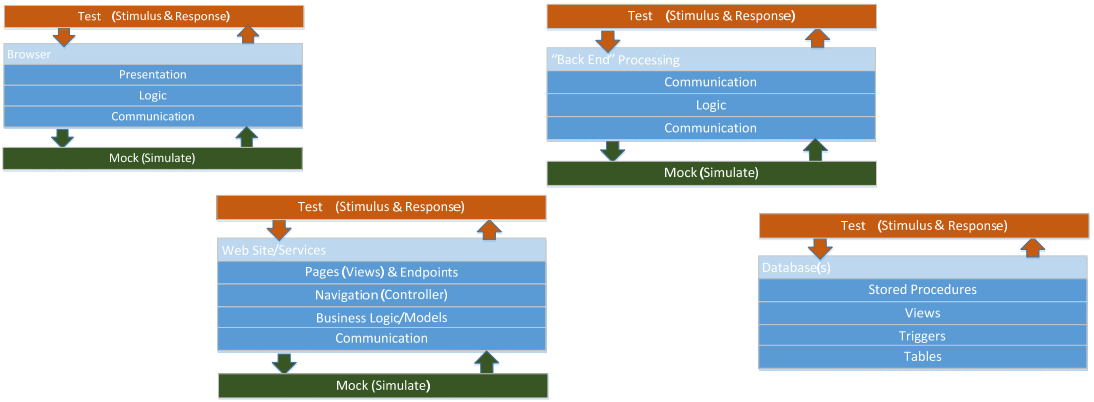
We now nave a means of Prioritizing

Focus Effort by Activity

- **Story/PBI Drivers**
  - **Brand New Code**
  - **Edits to Existing Code**
- **Bug/Defect Drivers**
  - **First step is a failing test**

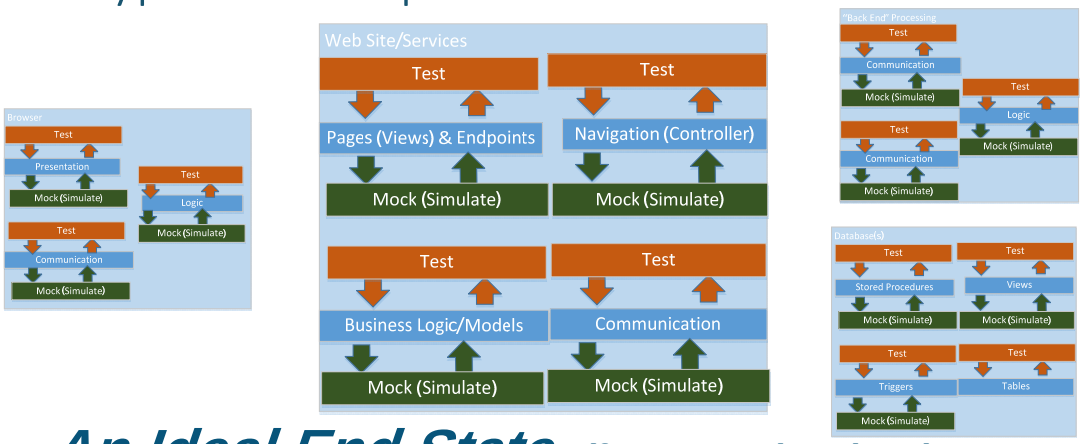
We now have Specific Areas of Value

High Level Decomposition



Better Isolation and Targeting

Hyper-Decomposition



An Ideal End State ***(for many situations)***

We have the **Concepts!**

BUT….

How do we **Apply** them???

It's time to look at some Code!!!

…Lets Go!!!!

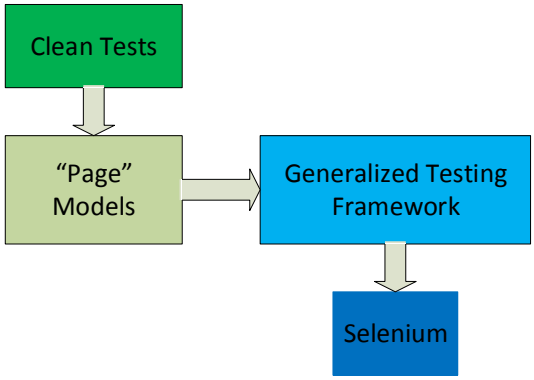
Coded UI / Selenium

- Use Selenium for All Browser based
  - Use Appnium for Mobile and UPW
  - Use CodedUI for WPF and others
- Follow "Page Model" Pattern
- SOLID Design Principles for your Tests

SOLID Principles Of Design

- Single Responsibility
  - A class should have a single responsibility
- Open/closed
  - Software entities should be open for extension, but closed for modification". It should be easily extendable without modifying the class itself.
- Liskov Substitution
  - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- Interface segregation
  - Many client-specific interfaces are better than one general-purpose interface. A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
- Dependency inversion
  - One should depend on abstractions, not concretions. The high level module must not depend on the low level module, but they should depend on abstractions.

Page Model Pattern Overview



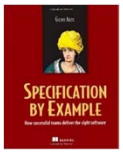
More On Page Model Pattern

- Wrap some HTML in an app-specific API.
  - Allows element manipulation without having to dive directly into HTML
  - getArtist() vs findElementsWithClass("artist")
- Encapsulates mechanics required to find and manipulate data.
  - Tests use methods of the page object to interact with UI on the page
- Benefits
  - Keeps tests and elements locators separate, making code cleaner
  - POM is independent of automation tests. Use it for different purposes with different tests
  - Any UI changes can be easily implemented, updated, and maintained

BA/QA Are not Programmers

- Make the Definition of Tests Accessible to a Wide Audience
- Specification By Example

https://www.amazon.com/Specification-Example-Successful-Deliver-Software/dp/1617290084



Gherkin with SpecFlow

Answers.feature

```
Feature: Ordering answers

Scenario: The answer with the highest vote gets to the top
  Given there is a question "What's your favorite colour?" with the answers
    | Answer         | Vote |
    | Red            | 1    |
    | Cucumber green | 1    |
  When you upvote answer "Cucumber green"
  Then the answer "Cucumber green" should be on top
```

Direct from Text to Test!

Common Pitfalls

- Dead End Risks
  - — Verify continued viability early
- Grandiose Expectations
  - — It is a long Journey

Many small steps on a Long Road

Our problems: September 2014

Tests took too long

Over 22 hours for nightly run

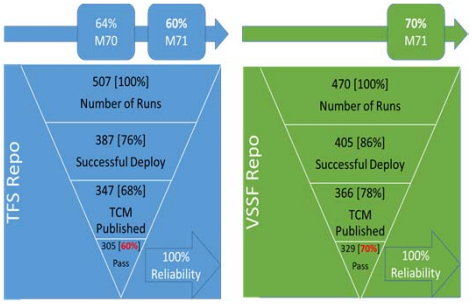
2 days for the full run

Tests failed frequently

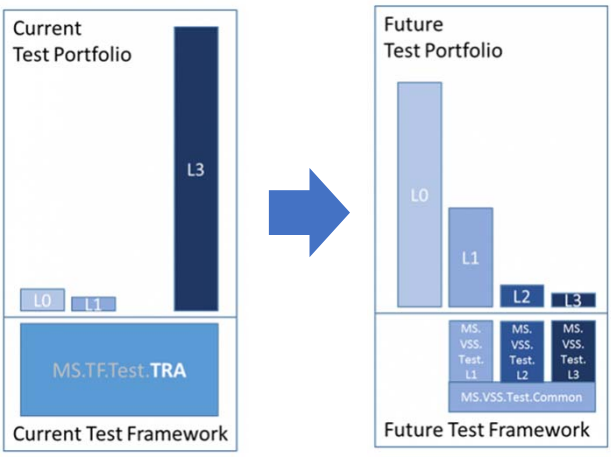
Only ~60% of P0 runs passed 100%;
Each NAR suite had many failures

Quality signal unreliable in Master

Test failure analysis was too costly



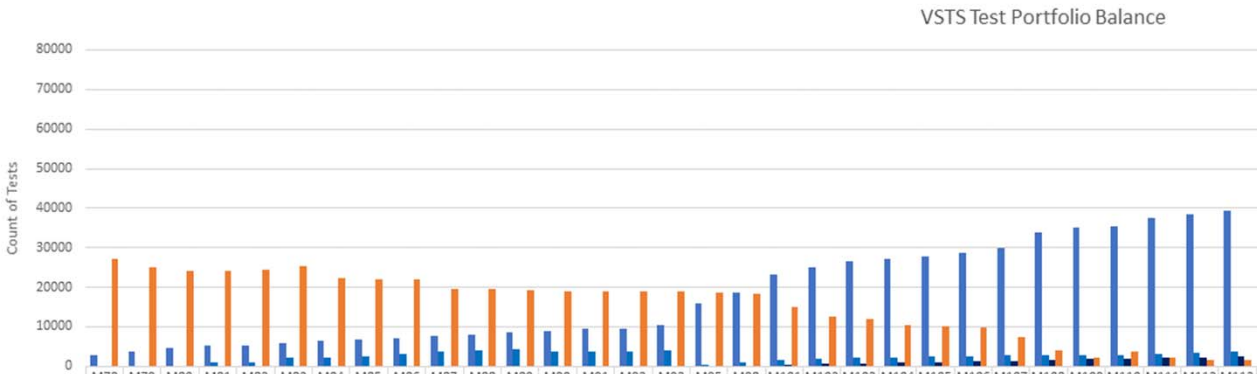
Published VSTS Quality Vision : Feb '15



Principles

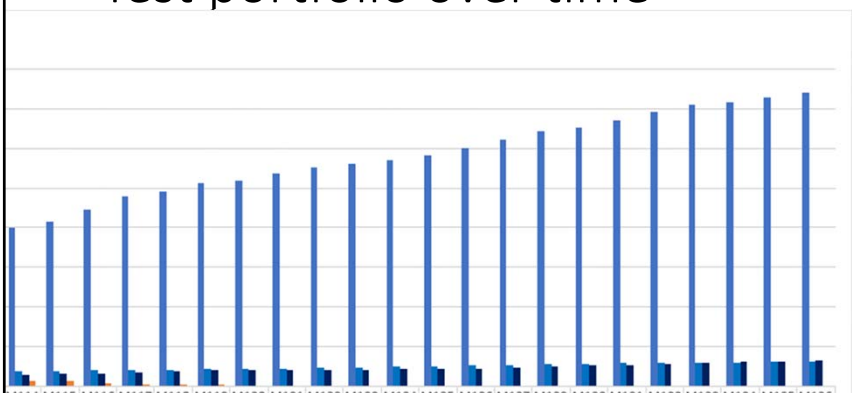
- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive

Test portfolio over time



	M78	M79	M80	M81	M82	M83	M84	M85	M86	M87	M88	M89	M90	M91	M92	M93	M95	M98	M101	M102	M103	M104	M105	M106	M107	M108	M109	M110	M111	M112	M113
L0 Tests	2723	3744	4582	5297	5381	5831	6613	6684	7232	7668	7983	8486	8908	9385	9385	10286	15968	18598	23272	24945	26594	27109	27807	28709	29787	34020	34983	35405	37404	38467	39412
L1 Tests				888	958	2130	2241	2525	3095	3753	4046	4215	3723	3786	3786	4166	479	1153	1465	1774	2177	2289	2438	2597	2735	2813	2935	2966	3127	3528	3592
L2 Tests																			430	590	652	980	1102	1200	1410	1693	1858	1918	2068	2190	2585
TRA Tests	27054	24941	24135	24097	24381	25212	22198	21981	21908	19577	19529	19124	19098	19041	19041	18937	18749	18252	14983	12449	11959	10554	10070	9686	7500	4155	2199	3886	2254	1593	1519

Test portfolio over time



M114	M115	M116	M117	M118	M119	M120	M121	M122	M123	M124	M125	M126	M127	M129	M130	M131	M132	M133	M134	M135	M136
40000	41452	44430	47909	49007	51113	51775	53664	55043	56077	56924	58365	60112	62262	64294	65315	67189	69244	70893	71721	72925	74084
3750	3849	3928	4023	4187	4310	4353	4470	4524	4675	4834	4955	5181	5365	5692	5698	5770	5828	5931	5988	6109	6187
2883	3021	3228	3469	3719	3917	3941	3965	3980	4179	4346	4416	4492	4728	4967	5143	5271	5593	5973	6211	6327	6477
1386	1386	755	181	136	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TYPE	M78	M136	DELTA
L0	2723	74084	+ 71,361
L1		6187	+ 6,187
L2		6477	+ 6,477
TRA	27054	0	- 27,054

Continuous focus on test execution time

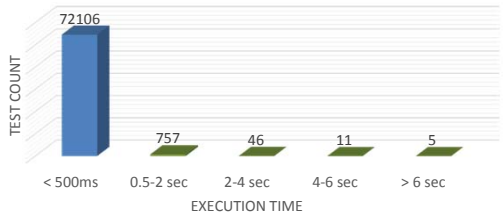
Level 0

- **Execution: 6:00** mins
- **Total Tests: ~72925**
- **Number of Assemblies: 2104**

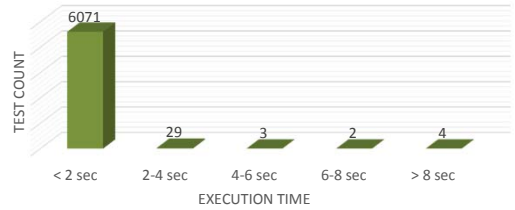
Level 1

- Execution: 3:30 mins
- Total Tests: ~6109
- Number of Assemblies: 41

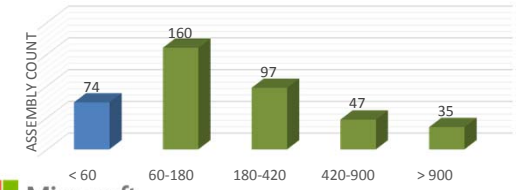
L0 tests by execution time



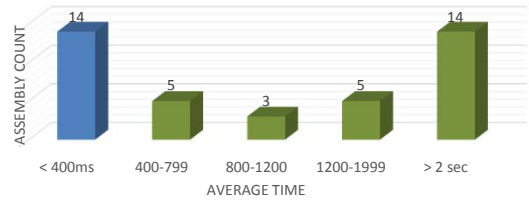
L1 tests by execution time



L0 Assemblies' Average Time chart

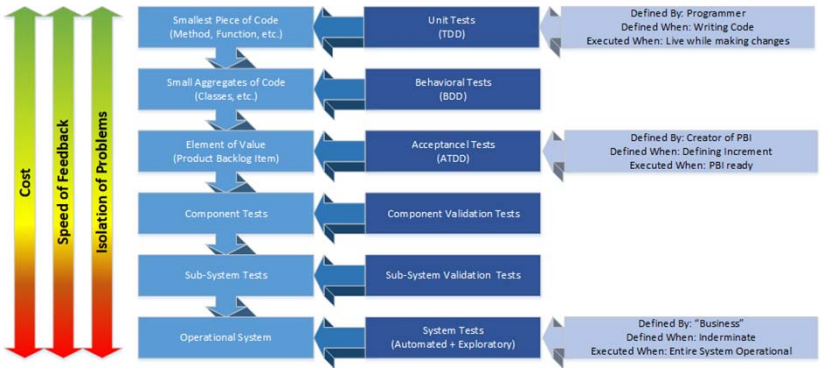


L1 Assemblies' Average Time chart



Microsoft

A Spectrum of Test Types



Contact Information

David V. Corbin

E-Mail: david.corbin@dynconcepts.com

Web-Site: http://www.dynconcepts.com

LinkedIn: linkedin.com/in/davidvcorbin