Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Microsoft HQ

★ CAMPAIGN FOR CODE 2016 ★

Stop the Waste!
Get Out of (Technical) Debt!

Richard Hundhausen
Consultant,
Accentient, Inc.

About Me

- From Boise, Idaho, USA
- President of Accentient
- Microsoft Regional Director
- Microsoft MVP (Visual Studio ALM)
- Professional Scrum Developer
- Professional Scrum Trainer
- Author of books and courses
- richard@accentient.com
- @rhundhausen

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Session Backlog

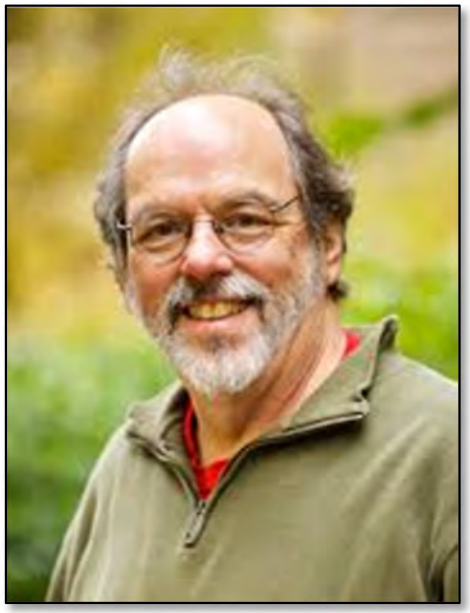
- What is Technical Debt … and what isn't
- Why is it Bad … and why you might want it
- What Causes It
- How to Identify It
- How to Remove It

What Is It?

What Is Technical Debt?

"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly ..."

- Ward Cunningham, OOPSLA 1992



What's Uncle Bob Say?

"A mess is not a debt. Messy code, produced by people who are ignorant of good design practices, shouldn't be a debt."

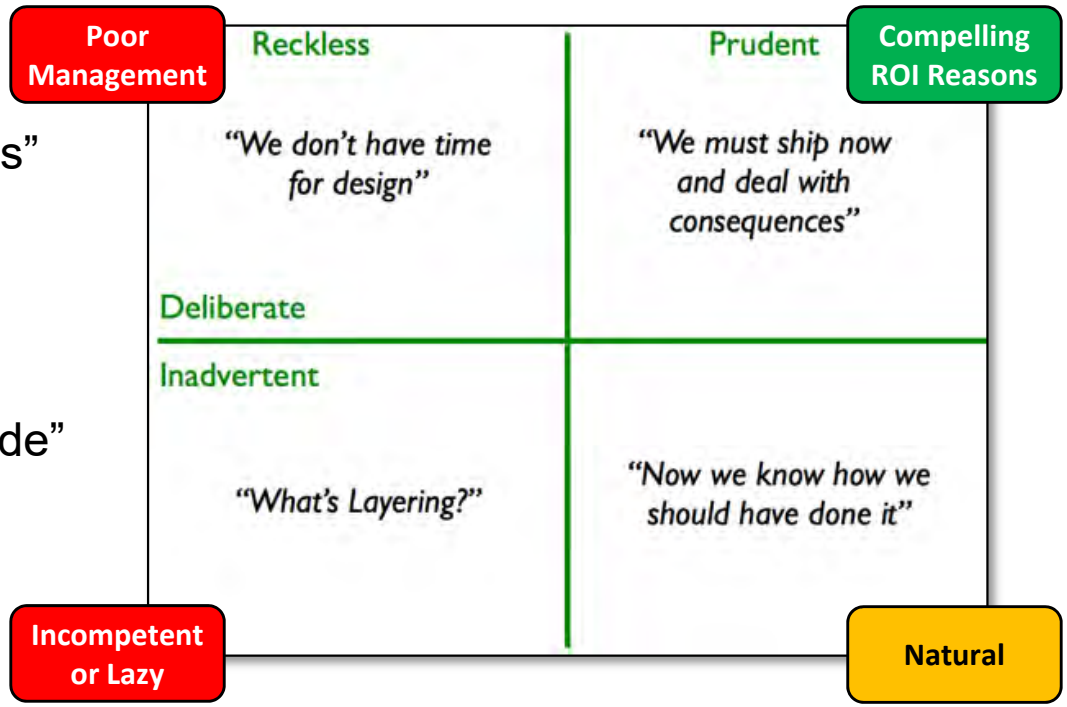
"Technical Debt should be reserved for cases when people have made a considered decision to adopt a design strategy that isn't sustainable in the longer term, but yields a short term benefit, such as making a release."

And Martin Fowler?

“There is a difference between prudent and reckless debt, as well a difference between deliberate and inadvertent debt.”

The Technical Debt Quadrant

- Reckless + Deliberate
  - "Go faster. We have more projects"
- Prudent + Deliberate
  - First to market, regulatory, etc.
- Reckless + Inadvertent
  - "Oops, we changed the wrong code"
- Prudent + Inadvertent
  - Teams should naturally improve their code after gaining experience and knowledge



What Do Others Say?

"Any code which a developer fears to change. That fear is legitimate, and one of the best estimates of where the risk lies."

– Arlo Belshee, XP Guru & Coach, Microsoft

"It is the cumulative total of less-than-perfect design and implementation in your project."

– James Shore, Agile/XP Consultant and Coach

"Technical debt is everything that makes your code harder to change."

– Tom Poppendieck, co-creator Lean Software Development

Got It.
So What Causes It?

Does This Sound Familiar?

- **Manager**: When will the new workflow be available?
- **Developer**: Uh, I *hope* tomorrow, probably by the end of the day.
- **Manager**: No, we need it today. Can't you find a creative way to do it?
- **Developer**: Let me think…
- **Manager**: We have five clients that really need this today and if they don't get it, then they will probably not sign the contract with us.
- **Developer**: But the …
- **Manager**: Please understand the business value of this. Can't you just copy/paste a similar workflow, tweak it, and we'll "fix it" later?
- **Developer**: Sure.
- **Manager**: Great. So we should be able to deploy it this afternoon?
- **Developer**: Ack!

What would you have told the manager?

Remember: If Scope and Time are fixed.

Quality isn't

Which Is Technical Debt?

A. A bug reported by a customer a few months after the product was released.
B. A bug discovered by the team during regression but was determined to not be “release blocking”.
C. A bug that occurs on a new version of a browser that was released after the product was released

Other Examples

- Deciding to forgo writing automated tests, as they normally would, for a tricky section of code
- Deciding to build on a soon-to-be-deprecated framework rather than investing in purchasing, upgrading, and learning a newer version
- Deciding to hard-code connection strings and IP addresses, rather than use a config file
- Having hundreds of customer-specific branches on the same code base

Bad Practices Yield Technical Debt

- Big Design Up Front (BDUF)
- Little/no refactoring
- Lack of automated tests
- Infrequent integration
- Lack of code review / pair programming

Broken Window Phenomena

Hacks, shortcuts, and workarounds can infect the team, and the codebase

Remind me again, why is Technical Debt is so Bad?

As Technical Debit Increases …

- The number of defects increases
- Development and support costs increase
- Product atrophy increases
- Fear and loathing increase

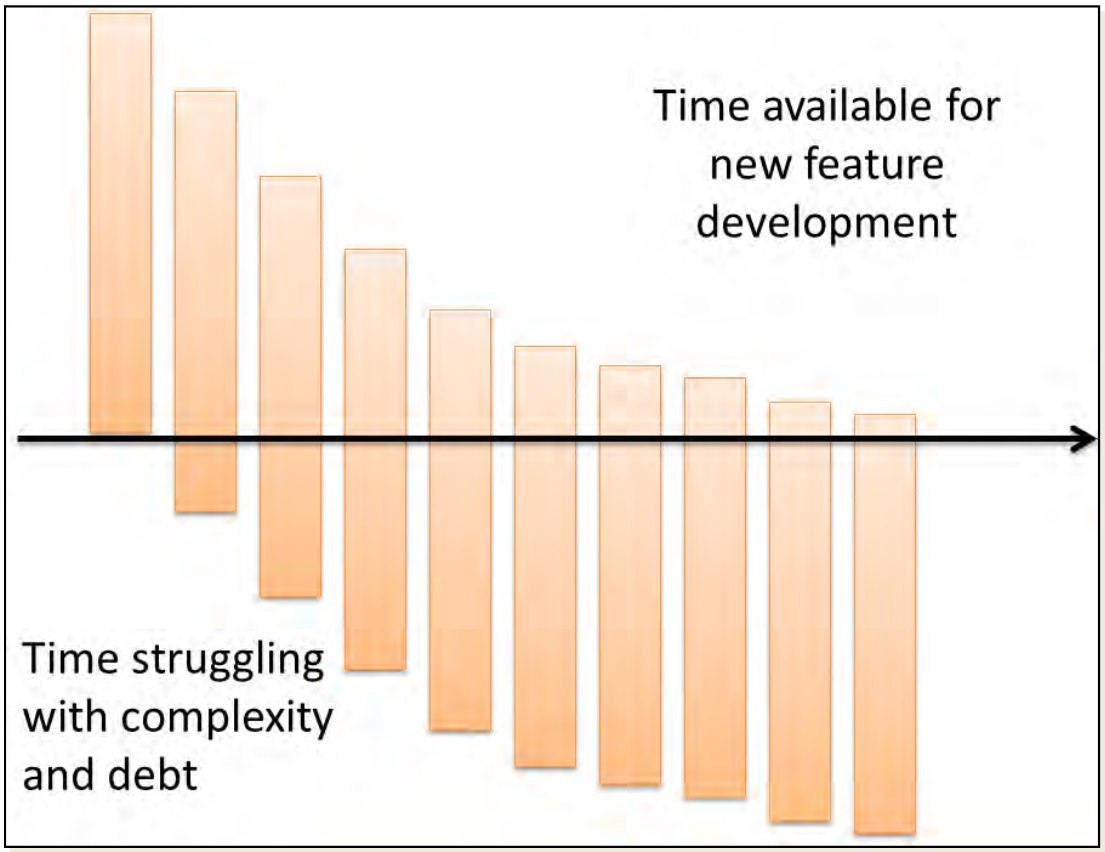
And These Things Drop …

- Product performance
- Team performance
- Stakeholder satisfaction

“Left unchecked, technical debt will ensure that the only work that gets done is unplanned work.”

- Gene Kim, co-author
The Phoenix Project

This Could Be Your Future



That said, sometimes Technical Debt can be leveraged.

Like Using a Credit Card

Sometimes a team makes decisions to ship a less-than-optimal product in order to obtain earlier feedback or a market advantage

But Be Cautious

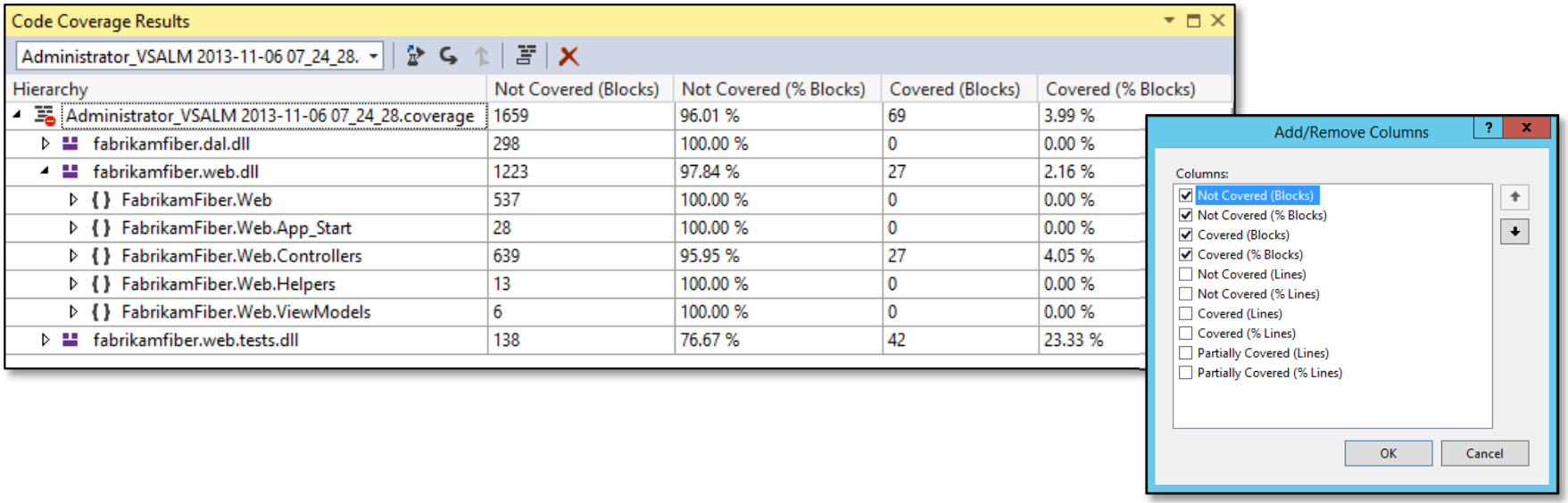
- Make sure you know …
  - The interest rate
  - Your payment plan
  - The cost vs. value (Return on Debt – ROD?)

What about legacy code?

How do I find its Technical Debt?

Code Coverage

- Code Coverage shows # and % of blocks covered and not covered by your testing

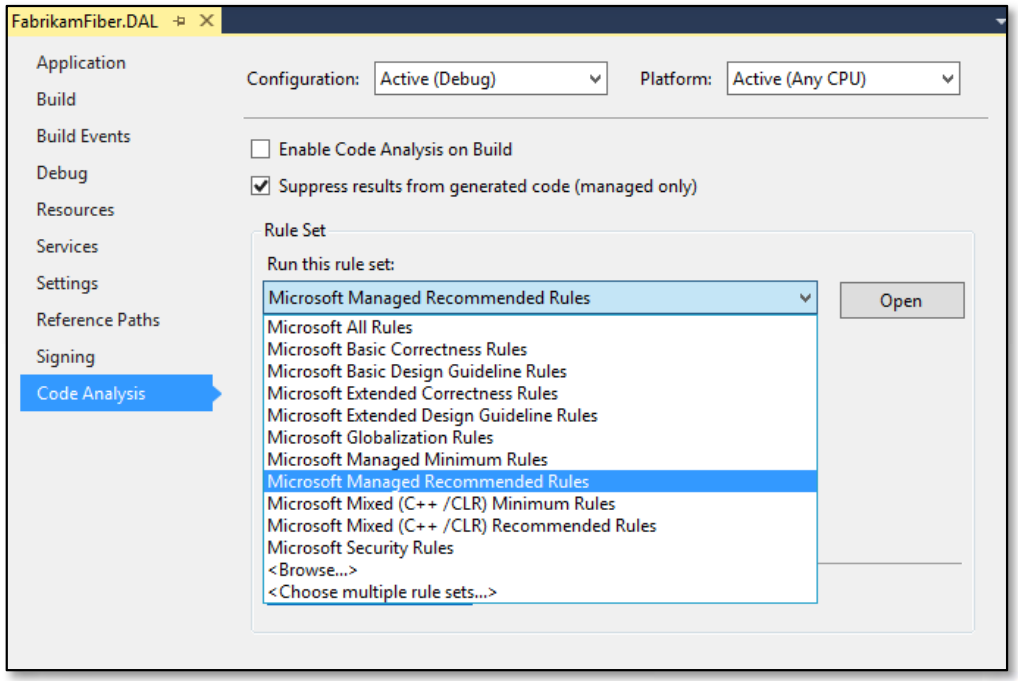


Static Code Analysis

- Code Analysis provides information about “bad” coding practices, such as violations of Microsoft’s .NET framework design guidelines
- Code analysis is available for .NET (managed) and C and C++ (unmanaged/native)
- You can run code analysis on your projects …
  - Manually or Automatically from within Visual Studio
  - As part of Team Foundation Build

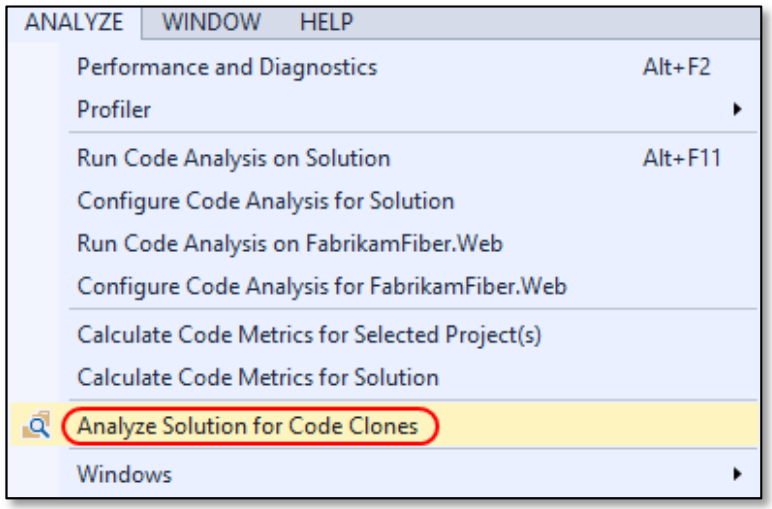
Code Analysis Rule Sets

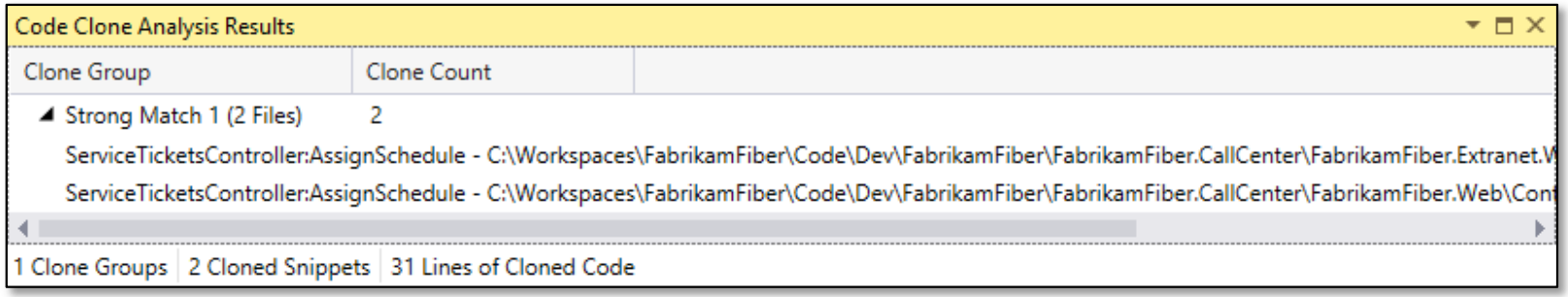
- A rule set is a logical grouping of code analysis rules that identifies targeted issues and specific conditions
- There are over 200 rules!



Code Clone Analysis

- Will find all clones in a solution or clones of a specific fragment
  - Will find fragments that differ in the names of variables and parameters, and in which some statements have been rearranged

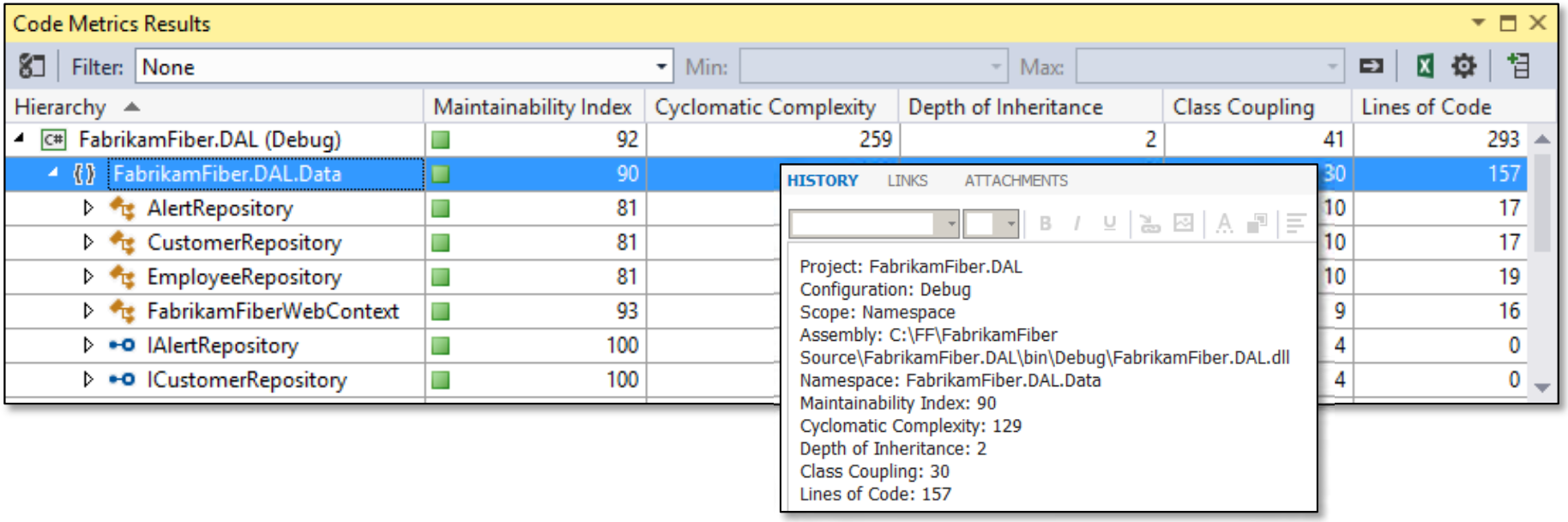




Code Metrics

- Code Metrics is a set of measures that provide insight into the complexity and maintainability of code
- Code Metrics calculates:
  - Maintainability index
  - Cyclomatic complexity
  - Depth of inheritance
  - Class coupling
  - Lines of code

Code Metrics Results



SonarQube

- A platform and tools to continuously analyze and measure code quality
  - Open source
  - Pay for support
  - Pay for some plugins
  - Active community: support, plugins, books

Supports 20+ Languages

- ABAP*
- Android
- C/C++*
- C#
- COBOL*
- CSS
- Erlang
- Flex / ActionScript
- Groovy
- Java
- JavaScript
- Objective-C*
- PHP
- PL/I*
- PL/SQL*
- Python
- RPG*
- VB.NET*
- Visual Basic 6*
- Web
- XML

As You Find it, Make it Visual

In the backlog

Title	Business Value	Effort	Tags
Customer can see upcoming appointments	8	5	
Export vendors to PDF	13	8	
Customer can opt-in/opt-out of paper billing	5	5	
Service rep can view service ticket details from the dashboard	5	5	
Update MVC framework		5	Technical Debt
Technician can report busy/late on Windows Phone	8	5	
Technician can look for closest hardware store from Windows Phone	13	2	
customer branches back into main		13	Technical Debt
onnection strings in config.file		3	Technical Debt
the nearest Fabrikam Fiber location	8	5	

On the story map

How Do I Get Out of Debt?

A better question: How do you eat an elephant?

Have a Definition of “Done”

- Closest thing to a “silver bullet”
- Write it down
- Review and discuss it regularly

Sample Definition of “Done”

- ✓ Code has been reviewed
- ✓ No static analysis errors
- ✓ No complexity analysis violations
- ✓ New code written using TDD
- ✓ All tests pass
- ✓ All acceptance criteria met
- ✓ Product Owner accepts the work

There is no Done-Done

There is only “Done”.

No partial credit (points) for partially-done work. Partial credit usually means Technical Debt.

Keep the Pressure Off

Work at a comfortable, sustainable pace.

Be positive. Manage debt informally.

Have a Cross-Functional Team

Make sure the team has all of the skillsets required to deliver the “Done” work exist on the team.

Collaborate on Design

Prefer in-person whiteboard sessions on design topics to leverage the collective experience of the team.

Design For Testability

Testability == Maintainability

Practice Test-Driven Development

Have an Agile Architecture

Simple and emergent, delivering in vertical (thin) slices, and refactoring to patterns as needed.

Mitigate technical risks early.

Automate Tests

Lack of high-quality automated test leads to a breeding ground for technical debt.

Unit | Acceptance | Regression

Continuous Integration

Check-in/commit/merge your code often, building, and running automated tests.

Prefer not to work in branches.

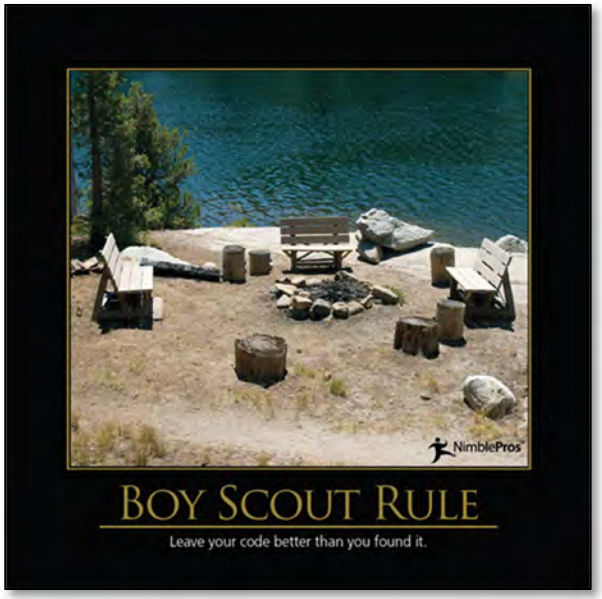
Finish One Thing First

Design the work so that multiple people can “swarm”. Complete the first thing and then move on to the next thing.

Working as a team means working as a team.

The Boy Scout Rule

- Leave your code better than you found it.
  - Prior to every commit
  - Small refactorings in legacy code
  - Write additional tests
  - Update an outdated comment
  - Improve a variable name



Done();

(thank you)

richard@accentient.com | @rhundhausen