Technical Excellence Doesn't Just Happen – Igniting a Craftsmanship Culture

By Allison Pollard and Mike Rieser
This is us!

INTRODUCTIONS
Allison Pollard

- Agile Process Coach and Consultant
- Firm Believer in Continuous Improvement
- DFW Scrum User Group leader and Dallas Agile Leadership Network board member
- Glasses wearer
Mike Rieser (ree-sir)

- Agile Coach and Consultant
- Technical Practices Coach
- “Programmer shirt” wearer
“In an agile project, technical excellence is measured by both capacity to deliver customer value today and create an adaptable product for tomorrow.”

– Jim Highsmith

TECHNICAL EXCELLENCE
Who’s problem is it?

How many programmers does it take to change a light bulb?

https://www.flickr.com/photos/belobaba/6058142799/
Who’s problem is it?

Who should be concerned with the long-term viability of the codebase?

https://www.flickr.com/photos/f-oxymoron/5005673112/
The 9th Agile Principle

Continuous attention to technical excellence and good design enhances agility.

https://www.flickr.com/photos/ikoka/1684313711/
Agile: build the right thing and build it right.

- *Building it right* means it has a simple design, and is maintainable and flexible.
- Successful software needs to be able to evolve to meet expanding needs.
GROUP ACTIVITY
Discuss

• In small groups:
  • Come up with at least 1 anecdote of an extreme in technical excellence. Either “too much” or “not enough” that you can share in 30-60 seconds.

• We’ll try to hear from 1 or 2 of the groups afterward.
What will spark your culture change?

KINDLING CRAFTSMANSHIP
Flaccid Scrum – Martin Fowler

• They want to use an agile process, and pick Scrum
• They adopt Scrum practices, and maybe even the principles
• After a while progress is slow because the code base is a mess

Five Years of Flaccid Scrum

My advice still stands - ensure you take technical practices seriously when introducing Scrum (or indeed any agile approach). – Martin Fowler: 29 Jan 2014
## From Flaccid to Awesome!

<table>
<thead>
<tr>
<th>Internal Quality (Developer Facing)</th>
<th>External Quality (Customer Facing)</th>
<th>Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>High</td>
<td>You’re awesome!</td>
</tr>
<tr>
<td><strong>High</strong></td>
<td>Low</td>
<td>Get Customer-focused!</td>
</tr>
<tr>
<td>Low</td>
<td><strong>High</strong></td>
<td>It won’t Last. Refactor now!</td>
</tr>
<tr>
<td><strong>Low</strong></td>
<td>Low</td>
<td>Refactor versus Rewrite?!</td>
</tr>
</tbody>
</table>
Agile Requires Quality

Waterfall

Quality over Time

Agile

Quality over Time
Process Dependencies

Waterfall
  ↓
Upfront Design
  ↓
Upfront Requirements

Agile
  ↓
Evolutionary Design
  ↓
Refactoring
  ↓
Automated Developer Tests

Test First
Test After
Software Costs

\[
\text{Cost}_{\text{Software}} = \text{Cost}_{\text{Development}} + \text{Cost}_{\text{Maintenance}} + \text{Cost}_{\text{Operations}} \quad (1)
\]

\[
\text{Cost}_{\text{Development}} \ll \text{Cost}_{\text{Maintenance}} \quad (2)
\]

\[
\text{Cost}_{\text{Maintenance}} = \text{Cost}_{\text{Understand}} + \text{Cost}_{\text{Modify}} + \text{Cost}_{\text{Test}} \quad (3)
\]
What is Technical Debt?

Reckless

“We don’t have time for design”

Prudent

“We must ship now and deal with consequences”

Deliberate

Inadvertent

“What’s Layering?”

“Now we know how we should have done it”
How Does Technical Debt Occur?
How Does Technical Debt Occur?

- **Ignorance** – an incomplete understanding of how the system works
- **Complex Requirements** – producing overly complicated implementations, unreduced complexity
- **Over Engineering** – developers building more sophisticated code than is necessary (Speculative Development). [Gold Plating]
- **Undone Work** – deciding “writing automated tests slows us down”
- **Good code / wrong place**
- **Cutting Corners** – “quick and dirty”
Faster is Slower

We’ll “fix it later.”
The Business Made Me Do It!

What a Tech Lead says to his Product Owner:

“To do it right will take 2 Iterations. If we do it this other way, we can do it in 2 days, but the developers will be forever burdened with fragile code and it will impact our delivery on every project in the future.”

What the Product Owner hears:

“You can have it in 2 days!”

Moral:

Be sure you can live with the options you present.
Enhances. Really?

Compare

“Continuous attention to technical excellence and good design enhances agility.”
– 9th Principle, Agile Manifesto

with

“Speed is a long-term side-effect of producing code with the highest possible internal quality.”
– Agile Testing by Lisa Crispin, Janet Gregory

perhaps

Continuous attention to technical excellence and good design sustains agility.
And now for something completely different.

WHAT WE TRIED
Free Book

- *Working Effectively With Legacy Code*
- Provides real examples of “getting out of” a legacy codebase, organized by “reason it can’t be done”...
- **Legacy Code** is code written without tests. Establishes a common language across the technical team for the daily problems they’ll face

Outcome
- Some read it (especially those in the book club)
- Some did not
- Was nice to be able to say, “go read Chapter 15” in the book
Book Club

• Engage people in discussion about the contents of the book, create community ‘buzz’ about the concepts in the book.
• No real disagreements have been encountered!

Outcome
• Book club started very successfully, we finished Legacy Code and Clean Code
• Observation: Millennials do not read books
• Third Book – folks lost interest. Will do a reboot
Code Ninjas

• Creating a community where continuing education is the goal... This is not intended to be a one-man show... any and all are welcome to bring interesting topics.
• Low ceremony!
• PowerPoint okay, Code preferred.

Outcome
• Seen as an “output” device, a way to get a message out.
• Not good as an “input” device for determining direction.
TDD Training

- Introduces the practice of Test Driven Development
- Refreshers welcome
- ‘Ignites the spark’ around Test Driven Development and why we need it

Outcomes
- Shoot for “Test-Infected,” coach the rest
- 18,000 Unit Tests with 100% pass rate every build
- Tasks for “Unit Tests” have disappeared
Maturity Model for TDD

**SHU**
*(Follow)*
- Still learning, follows but may not know why.
- New code comes with new tests.
- Changed code comes with tests around the changes.

**HA**
*(Break Free)*
- Internalized practice, but still needs a coach.
- Understands pros and cons.
- All code is written test-first.

**RI**
*(Transcend)*
- Able to go beyond what has been taught
- Creates a unit testing framework because they needed one and it didn’t exist.
Code Clinics

• A non-judgmental place where developers can bring their coding issues to look for feedback or work collaboratively on solutions, watch and help others, get mentoring if needed, identify how to apply concepts from the book or get some code under test.

Outcome
• Sometimes repurposed into “Code Review”
• A few dedicated “Lurkers”
• As delivery pressure increases, attendance decreases
• Found “classroom exercises” this way

https://www.flickr.com/photos/lloyds-screenies/2696378369/
Transferring Code Ownership

Devs:

This is horrible! They ought to do something about this!

Coach:

Management wants you to have a nice code base to work in, but they aren’t in the code every day.

This is where you work every day.

If you want a nice place to work, you’re going to have to clean it up.

“You” are “they.”

Outcome

“Don’t touch that!” replaced with

“Why didn’t you clean that up?”
“What” is Negotiable, Not “How”

• There is an ‘N’ in INVEST, but that is concerned with scope
• Do not let the Product Owner and the Developers get into a negotiation around whether certain practices will or will not be followed

Outcome
• Have a clear Organizational Definition of Done which specifies standards
• Business not happy, but ...
• Avoid technical-only projects
Tech Lead Interview Process

• Recruiting Tech Leads that are evangelists and mentors for the concepts we care about (TDD, Quality, Refactoring, OO Design)
• Interview includes: a coding problem, a code review problem, a non-technical-solution problem
• Allowed no exceptions for folks hired into a Tech Lead role

Outcome
• “New DNA” rejecting “Old DNA”
• “Supervisor” Tech Leads did not do well
• Hit most on: Senior Devs looking for their next level
Code Reviews

- Originally no definition of “Production Worthy”
- Code is continually ‘assessed’ (every iteration) and feedback given to ensure we continue to steer away from the addition of debt to the codebase
- The more frequent the code reviews, the more likely we are to avoid pitfalls at the end of a project

Outcome

- Reviews with different teams were at different levels
- Smell: sign-off became “required”
- Now community owned
Continuous Isolation vs Integration

- Providing a place for development teams to fast-track characterization tests, safe code refactors / improvements to all teams early, outside of a release cycle
- Refactoring code in this stream gives fellow teams the benefits of refactored code and debt reduction without waiting for a given project to ‘go live’

Outcome
- Before this code cleanup had a reverse incentive due to custodianship
- Every team has used it at least once
- A few stars are huge contributors
Birds of a Feather

- Spontaneously grew organically out of need for decisions and conventions
- Technology related: “How do we use SpringMVC?”
- Not formally created

Outcome
- Email distribution lists were created for each interest
- Encourage and nurture this type of self-organizing solution process

https://www.flickr.com/photos/tambako/8011612529
Practices & Process

- Notice the upper right (highly rated on practices and on process).
- Focus on process and practices and you’ll get delivery!
- Where would focusing only on delivery get you?
WRAP UP!
Invest in your people

CFO: What if we train them, and they leave?
COO: What if we don’t, and they stay?
Create a Safe Environment

“Initiative is punishable.”
– Russian Proverb

• If you want teams to take initiative, you must allow for failure.
• Mistakes are an indicator of trying something new.
• Teams require autonomy to achieve excellence.
Fan the Flame!

• There really is no substitute for a developer that has a spark to make it right.

• When you see the spark, encourage the flame!

Photo by Billy Hathorn, https://commons.wikimedia.org/wiki/File:Kindling_for_starting_a_campfire_IMG_2454.jpg
Perfect (v.)

- Continuous Improvement requires continuous learning – encourage a learning environment.
- Problems are one of the first signs that something needs changed.
- Get to the root cause, and fix it. The symptoms will go away.
Long Term Viability Requires Discipline

“Just put it anywhere” only works for a short time!
THE END
Best Measure of Code Quality

- Hard to game.
Scrum & Extreme Programming (XP)

“People have told me that they see Scrum as the management approach to agile development and XP as the engineering practices that make it effective, both bonded together by complimentary practices and goals.”

– Ken Schwaber