Ron Quartel
Complementing Scrum with Technical Excellence for hyper-productivity
Who is Ron Quartel?

- XP Developer – Code Crafter
- XP Coach
- Agile Consultant / Trainer
- Lean Entrepreneur – Backlog Optimizer™
- Systems Engineer – Mobile Testing & CI
- Founder – FAST Agile™ Scaled Technology
  http://www.fast-agile.com
The fastest software delivery method on the planet today!

FAST Agile™ Scaled Technology
XP = Extreme Programming

When you see slides past this one, Extreme Programming will be referred to only as XP.
How do you get to hyper productivity?

With Scrum
Let’s ask the Doctor – Dr. Jeff Sutherland

“Few implementations of Scrum achieve the hyper productive state for which Scrum was designed (5-10 times normal performance). Those that do all implement variations on XP”

The End

Thank you for coming! (Please be sure to leave good feedback.)
So why don’t companies implement XP?

Stats and Quotes
Agile the easy way (Scrum) has trumped Agile the disciplined way (XP).

More quotes from the Doctor and some stats
Scrum can be rolled out in two days versus XP many months

The Scrum framework for software development was designed to get a team started in two or three days, whereas engineering practices often take many months to implement.

Therefore, it left the question of when (and whether) to implement specific practices up to each team. Scrum co-creators Jeff Sutherland and Ken Schwaber recommend that Scrum teams get started immediately and create a list of impediments and a process improvement plan. As engineering practices are identified as impediments, teams should look to XP practices as a way to improve. The best teams run Scrum supplemented with XP practices.” – Dr Jeff Sutherland

Scrum is easier to adopt without XP

The Scrum Guide (scrumguides.org) doesn't say anything about engineering practices. However, the first Scrum team used everything in XP and more. Radical refactorization occurred daily. I wanted engineering practices specified in Scrum but Ken Schwabber convinced me that Scrum would be easier to adopt and work anywhere if we didn't mandate specific software practices. He was right. – Dr Jeff Sutherland

http://www.reddit.com/r/IAmA/comments/2hw05i/i_am_jeff_sutherland_the_cocreator_of_scrum_ask/ckwi02e - 30th Sep 2014
Scrum’s Lip Service (to XP)

Scrum says you should be doing XP – but in reality it just doesn’t happen.

Scrum & XP: Better Together
15 April 2014

Scrum and Extreme Programming (XP) are two agile processes that work well in tandem.

https://www.scrumalliance.org/community/spotlight/mike-cohn/april-2014/scrum-xp-better-together
2015 State of Agile Report – Version One

STATE OF AGILE
Agile Methods and Practices

AGILE METHODOLOGY USED

- XP: 1%
- DSDM/Atmel: 1%
- Agile Modeling: 1%
- Agile Unified Process (AgileUP): 1%
- Feature-Driven Development (FDD): 2%
- Other: 2%
- I Don't Know: 3%
- Lean Development: 4%
- Iterative Development: 5%
- Kanban: 6%
- Scrum: 5%
- Custom Hybrid (multiple methodologies): 10%
- Scrum/XP Hybrid: 56%
9. What Agile approach is your organization using? (Multiple answers allowed)

Nearly all respondents—95%—report that Scrum is used as their organization's Agile approach. The two other most common are Kanban and Lean, respectively.

AGILE PRACTICES

- Scrum: 95%
- Kanban: 43%
- Lean: 21%
- Extreme Programming (XP): 13%
- Feature-Driven Development: 7%
- Unified Process (e.g., RUP, AUP, OUP): 4%
- Other: 4%
- Team Software Process (TSP): 1%
- Waterfall: 1%
- DSDM: 1%
- Crystal/Crystal Clear: 1%
How do we fix the problem?

And get undisciplined Scrum orgs/teams to want to take up XP?
Sad truth – companies don’t care about improving quality.

If companies really cared about quality, then they would already have implemented XP. The fact is – they don’t care about quality.

Don’t talk about quality. It’s falling on deaf ears.
Talk about speed and/or “Increasing Velocity”

Instead

And the same companies are now all ears. <sigh>

(I just vomited a little bit in my mouth)

This is why “DevOps” and “Continuous Delivery” are the industries darling terms right now...
The Phoenix Project, Hero developers and DevOps

Why didn’t the Brent character ever pair?!?

Why didn’t they create an XP team?
• Collective code ownership
• Pair programming
• Coding standards
• Small releases
• TDD & testing culture
• Continuous Integration
Technical Excellence

Whatever you want to call it – do it!

- Agile Engineering Practices
- Extreme Programming
- Agile Technical Practices
- Code Craftsmanship
- Technical Excellence
- Modern Development Practices
Continuous attention to technical excellence and good design enhances agility.
Recap of Problem

• Scrum chose to go easier to adopt route and leave XP Practices
• Scrum recommends XP practices are worked on the same way as impediments and only when identified
• Dr Sutherland has seen very few hyper-performing Scrum teams
• The only hyper-performing teams Dr Sutherland saw, were doing XP!
• In most companies, speed is more important than quality
How to solve the problem

How do you convince a team and management to take up XP? (When XP is so counter intuitive)
The Scrum Gauntlet of Debt

An exercise/simulation that will illustrate to teams and managers why they need to complement Scrum with XP and what happens if you don’t
Understand room make up

Show of hands :-

• Devs
• Testers
• Managers
• Scrum Masters
• Agile Consultants
• Other
Running the Gauntlet - Requirements

• Volunteers (7-12 volunteers)
• Two walls and two easels or 4 easels
• Chairs (observers can give up theirs while watching if need be)
• Sticky notes (small square size is better but any will work)
• Flipchart (Sticky is best)
• Blue tape
• Something with a countdown timer (smartphone apps work)
• Three surfaces for flipchart sheet. E.g. walls, whiteboard, easel
• Marker pens to write on flipcharts (and whiteboard)
Prep – Create the Sprinting dev room

Clear a gauntlet area in the room and layout like this:

This flipchart sheet is optional. You only need when even numbers and more than 7 volunteers
Create Backlog and Done Boards (Flipcharts)
Tip

If using a flipchart, use the blue tape to hold the bottom corners of the flipchart down
Volunteers and Roles

Ask for volunteers with warning that it is a physical demonstration and there will be running. Do not volunteer unless comfortable with this.

Divide the volunteers into testers and developers using one of these formations depending on the number of volunteers:

- 10 devs and 1 or 2 testers
- 8 devs and 1 or 2 testers
- 6 devs and 1 or 2 testers

Divide devs into two even groups
Prep participants and Timer Role

• Identify another timer volunteer (who has a smart device with countdown timer app)

• Explain to participants that this is a simulation. The goal is not to “win” and game the system. The goal is to mimic what real life looks like

• Instruct participants to act like a dysfunctional team and not communicate with each other (otherwise they start gaming the system)
Placing of volunteers in the gauntlet

Divide devs in two groups and place volunteers like so:

```
    = Flipchart (or whiteboard)
```

```
    Tester
    |
    |      |
    |      |
    Devs  Devs
```

Optional Tester (if even numbers)

fast-agile.com
Rules of the game/simulation

• An iteration lasts 30 seconds

• Developers run to opposite wall, pick a story token (sticky note) from TODO and run back to their starting wall, placing the token in the TO TEST column

• A Dev can only work on one story at a time

• Testers wait by their Done board until they see stories ready to TEST and then run to (either) board, grab up to 3 stories (but no more than 3) and return them to their done board

• Everyone must run (That is why they call it Sprinting!)

• No collusion or working out a system. You are mimicking a dysfunctional team (Be sure to have explained this or else they game the system)

• BE CAREFUL!!!(Re-iterate and ask volunteers again if comfortable)
BE CAREFUL!!!

• Health and safety
• Don’t have anyone pregnant, wearing high heels or with physical disabilities that may cause risk to themselves or others in the volunteers
• Make sure volunteers are comfortable with exercise otherwise switch out
• Re-iterate to volunteers Safety First!
• Be careful!!!
Any Questions? Then Go!
Notes for facilitator – Sprint 0

• While the team are sprinting, you play the role of a manager and shout at the team “FASTER”, “I Need more features” etc.
• Run the first sprint without keeping score (it’s a test run)
• (the reason to not keep score is that because the velocity of the first one is usually lower because they are working it out and it gives you a chance to find out if anyone misunderstood anything)
• At the end of the sprint tell the team to reset the boards
• Ask them if they are warmed up and a little puffed? “That is why they call it a sprint!”
Management Abuses to shout at team

- Faster!
- I need more features!
- What is wrong with this team?
- Why are they so slow?
- Why are your estimates all wrong?
- More velocity!
- Development monkeys!
- Screw code practices – I need Features!
- Stop that pair programming nonsense!
- My last team was better than you guys
- Don’t make me fire you
- I should out source the lot of you
- Etc...
Sprint 1 & introduction of technical debt

- Keep shouting abuse

- This sprint, record the velocity

- Now, place chairs in the gauntlet/sprinting area with the explanation “These chairs represent technical debt. Every story you completed in the last iteration, you left behind technical debt. This is what your codebase starts to look like. Can anyone relate? Do your teams have technical debt” (I’ve yet to have a no answer)

- Be even more careful!!!! No leaping or moving chairs.

- Testers will move one task from TO TEST to TO DO on each run (to represent an increase in bug count due to technical debt)

- Go! Start sprint 2 and be sure to keep the abuse
Sprint 2 Management Abuse

• I don’t want to hear about technical debt excuses – I want features damn it!
• When can we release?
• No refactoring or cleaning up – Features!
• Run, run, run!
• Lazy coders!
• I’m paying you too much!
• There’s going to be overtime!
Sprint 3 & introduction of refactoring

• Put more chairs in the gauntlet to make it near impossible for the team
• Explain to the room that this is what a day looks like for a developer in a legacy system
• This is what their life is like – get confirmation from the room e.g. “can anyone relate?”
• Be sure to point out the team are looking haggard and tired at this point
• Ask them if they think they can keep this pace up indefinitely and what they think of the upcoming sprint!

• Now introduce refactoring and sustainable pace
• Developers are now required to walk and not run (agile principle of sustainable pace)
• Developers are required to clean up after themselves and move one chair out of the way on their return leg. One chair per story

• Keep shouting abuse
Sprint 3 Management Abuse

• What the hell is going on?
• Who said you guys could slow down?
• Stop this refactoring nonsense!
• Run damn it, run!
• I didn’t hire extreme programmers – I hired scrum developers
• I want speed, not quality
• Why are you pair programming?
• There goes our velocity!
Post Exercise discussions

The physical portion is now complete. Ask participants to be seated.

Your recording of velocity should look something like this:

1. Sprint #1 – 13
2. Sprint #2 – 10
3. Sprint #3 – 8

Bring this sheet of flipchart to the front of the room. You now begin some discussions...
Management - Scrum or XP?

• Which sprint felt sustainable? (Scrum or XP)
• Which sprint does management most like? (Scrum or XP)

Working software is the primary measure of progress. Agile processes promote sustainable development.

The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.
How does Scrum and XP differ on a long term project?
Cost of project & ownership – Scrum vs XP

• Which project finished first? (and was cheaper)
• Do you now need a re-write to replace a legacy system? (what is that cost)
• What is the cost of adding new features? (the velocity myth)
• What does it cost to find/fix bugs in production? (clean code vs hairball)
• Do you have domain specialized developers that can hold you hostage? (What is their salary vs junior devs)
Instant Gratification

Versus Delayed Gratification
Stanford Marshmallow Experiment

Stanford marshmallow experiment

Delayed gratification
Scrum’s blind spot

Or Achilles Heel
On a long term project :-

Iterative development without refactoring, will create technical debt and ultimately a legacy system

Code Entropy

Law of Entropy: A system will always move towards the most disordered state
Code Entropy
12,000 Lines of code
Refactor or die

A slow painful death – crushed under technical debt until you are a legacy system

fast-agile.com
The debt analogy applied to code

• Debt accrues interest and compounds
• The longer you leave it – the more it costs to pay it back down

Disciplined financial and development practices

• Pay your debt as you go
• Keep a line of credit that you can manage and use it when you need it (sometimes debt makes sense)
Kitchen Analogy

Poorly factored kitchen with tech debt

Refactored kitchen with no tech debt
Do you think there are bugs in there?
Another Analogy – Camp Ground

Code refactoring is polite & good manners. Like leaving a campground cleaner than when you arrived.

@agileAgitator
Scrum’s blind spot

On a long term project –

Scrum (without XP), will create technical debt and ultimately a legacy system

Scrum without XP = Flaccid Scrum

FlaccidScrum

Martin Fowler
29 January 2009

There’s a mess I’ve heard about with quite a few projects recently. It works out like this:

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

What’s happened is that they haven’t paid enough attention to the internal quality of their software. If you make that mistake you’ll soon find your productivity dragged down because it’s much harder to add new features than you’d like. You’ve taken on a crippling TechnicalDebt and your scrum has gone weak at the knees. (And if you’ve been in a real scrum, you’ll know that’s a Bad Thing.)

http://martinfowler.com/bliki/FlaccidScrum.html
Refactoring is not an island
There are other supporting disciplines that you need
Refactoring
Merciless

Refactoring
Merciless
Refactoring

Agile Testing
Collective Code Ownership

Code Collisions

Merciless Refactoring

TDD

Pair Programming

Agile Testing

Unit Tests
Collective Code Ownership

Code Collisions

Check in little and often

Coding Standards

Merciless Refactoring

TDD

Pair Programming

Unit Tests

Agile Testing
Collective Code Ownership

Code Collisions

Check in little and often

Simple Design

Coding Standards

Refactoring

Merciless

TDD

Pair Programming

Continuous Integration

Unit Tests
Give me an Etch A Sketch and 20 seconds and I’ll show you why you need to compliment scrum with XP
When and how do you refactor?

Refactor Mercilessly
Refactoring Guidelines

• Don’t create refactoring stories (5% exception). It should just happen.
• Stories deliver business value – they shouldn’t be “technical” (mostly).
• If you see it – fix it.
• Leave the campground cleaner than when you arrived.
• Be sensible – don’t refactor everything. The goal isn’t perfection, it’s sustainability.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Slow down to go fast

When am I going to have time to do all of that and get the code written?
Race/Rally Car Analogy

Race car driving is not about driving fast. It’s about driving disciplined.

Speed is an outcome of your discipline and practice.
Be a Code Crafter. Read, Study, Learn & Grow

fast-agile.com
Handle Q&A from Gauntlet Exercise

This is the closing section to the Gauntlet simulation. It’s roughly 15 minutes to this point.
Some typical questions that come up

• How does refactoring get prioritized against feature work?
• Isn’t it up to the PO if we can spend time refactoring?
• Do we create refactoring stories?
• When do you do refactoring?
• How much refactoring should we do? When do you stop?
• We have way too much debt – where do we start?
• Is this only for green field projects?
Some examples of refactoring

• Four hour task with four hours of refactoring

• Two hour task with sixteen hours of refactoring

• The refactoring backlog and refactoring stories (less than 5%) of the backlog
How to start

• Build slack into your sprint. Reduce velocity by 20% and start! (Find out what your sustainable velocity is)
• Read a book (my favourite is The Art of Agile by James Shore)
• Get an XP coach. Your journey of 1000 steps has begun. With a coach, your journey will take around 6 months (another reason why Scrum is favoured over XP) and you will be a SHU XP team.
Too much debt – already legacy?
Where to download this exercise

Soon...

TastyCupcakes.org
Fuel for Invention and Learning
The End

Thank you for coming! (Please be sure to leave good feedback.)
But wait – there’s more
Bonus Material
Bonus material Agenda

• Quality is not negotiable
• How to build an XP Team
• What refactoring looks like and the why of refactoring
• Code Crafting (non gender biased version of Code Craftsmanship term)
• Architecting for changing requirements
Quality is not negotiable

In Extreme Programming...
The XP Tradeoff Pyramid
How to build an XP Team

An approach that I have taken
The why of refactoring

It’s not only for keeping technical debt down – it’s how you keep your codebase agile
How do you architect a system when you know it will change?

If you have a truly agile project (i.e. changing requirements and not just an iterative waterfall project) – then you will need to have an agile codebase i.e. refactoring and emergent design
Done?

Abstract:
“Few implementations of Scrum achieve the hyper productive state for which Scrum was designed (5-10 times normal performance). Those that do all implement variations on XP (eXtreme Programming)” - Jeff Sutherland

Want to go fast? Really fast? Like 5x-10x times faster? Find out how Scrum + XP delivers on this.

The session will start with a lively and interactive exercise which will make it obvious why you need to complement Scrum with eXtreme Programming (XP) and what happens if you don’t. The exercise is easily reproducible and you will be able to facilitate it in your own environment to help sell your engineering teams and management on the advantages of taking on these practices.

Learning Outcomes:
what XP is
• how XP integrates with Scrum
• the what, how and when of refactoring
• how technical debt is created and how to stop it in its tracks and remove it
• how to build an XP team
• how and why Scrum plus XP achieves hyper-productivity
• why Scrum without XP can fail you

fast-agile.com
Where to find me

Twitter - @agileAgitator

Blog – http://agileagitator.com

Email – info@fast-agile.com

Web – http://fast-agile.com

LinkedIn - https://www.linkedin.com/in/rquartel

Facebook - https://www.facebook.com/fastagile