Strong Engineering LLC

Agile Development using Scrum

Duane Strong

Strong Engineering LLC

duanes@strongenging.com

We had a problem in 1968

- NATO conference of 1968 'Software Crisis’
- Massive bugs
- Late
- Failure to deliver working software
- Unhappy stake holders
- Most features unused
- Obsolete on arrival
- Only going to get much, much worse

Roots of the problem

- Agrarian to industrial revolutions
  - Similar models, social stratifications
  - Workers largely unskilled and uneducated
  - Creativity belongs to those who control the means of production
  - Little respect for those doing the work

Frederic Taylor 'Taylorism’

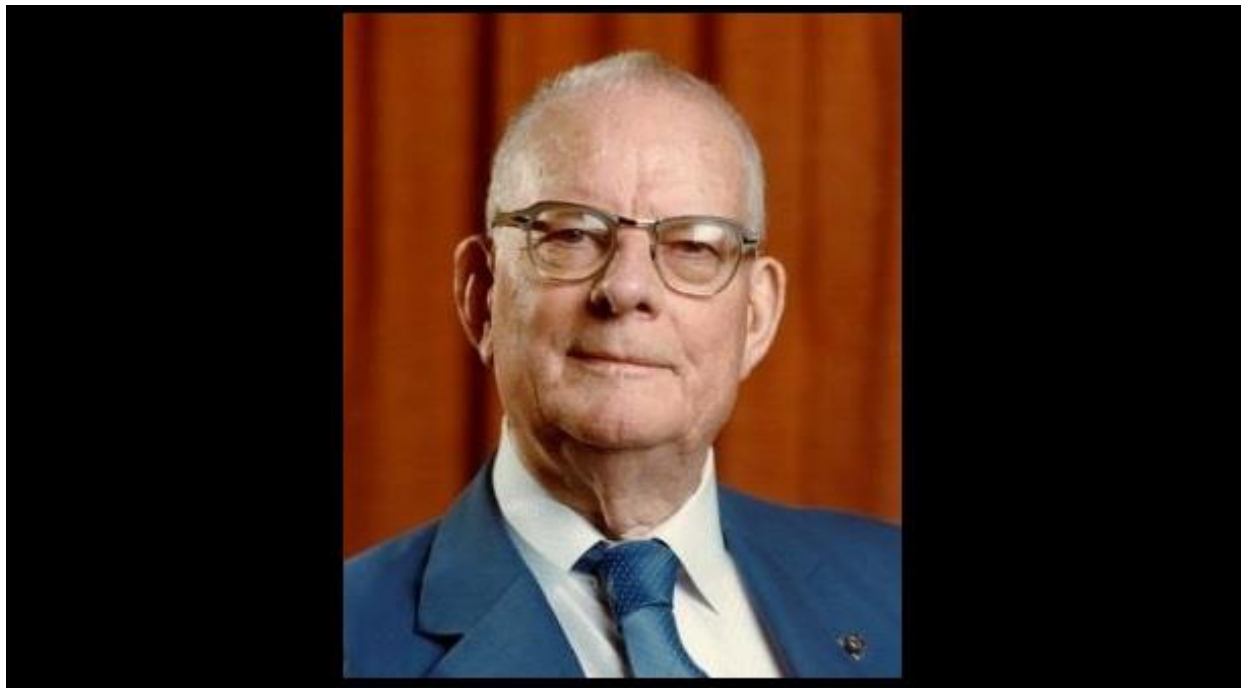
- Managers think, workers do
- Managers closely supervise workers
- Standardized tasks
- Quality control by batch inspection and rejection, carrot and stick
- Up front planning, specification
- Large 'ceremony' moving from stage to stage, waterfall model
- But it did massively increase production

Industrial paradigm fails for software development

- Increasing up front planning, scheduling, design made no improvement
- Value creation vs. value extraction
- Complex emergent specification and design
- Impossible to plan entirely before hand
- Impossible to specify completely up front
- Rapid change
- Workers know more than the managers, highly educated and skilled
- “Decision latency”

Roots of the solution

- William Edwards Deming, Japan WWII reconstruction
- Plan Do Check Act
- Kaizen
- The Toyota Way
- lean manufacturing
- Kanban



A new paradigm shift - agile

- It's not a progression, it's a revolution
- It can't be phased in
- Fundamental thinking, behavior of people and organizations has to change
- Many fail to adapt, organizations and individuals
- Results can not be measured in the same ways
- Cargo cults
- Superficial – Mechanical – Competent – Enlightened

'Agile manifesto' 2001

- Value individuals and interactions over processes and tools
- Value working software over comprehensive documentation
- Value customer collaboration over contract negotiation
- Value responding to change over following a plan

It is NOT that the thing on the right has NO value, we just value the thing on left more

Early snake oil

- Ironically heavy in prescription. "Follow these steps and silver bullet"
- Badly integrated into existing power structures
- Seen as latest fad
- More money more problems
- False starts, but now converged
- Still, “licensed scrum master”

Enter SCRUM

- Hirotaka Takeuchi and Ikujiro Nonaka 1986 paper "The New New Product Development Game"
- Jeff Sutherland and Ken Schwaber 1995 conference presentation
- A framework not a methodology, it has to be applied as each organization sees fit
- A simple few page description, but like chess easy to learn hard to master
- Requires discipline to respect the rules of the game, the players, accountability

The Fundamentals

- Embrace change
- Self organize
- Business value focus
- Continuous improvement
- Non interruption
- Customer centric
- Radical transparency

The Values

- Commitment
  - To team
  - To quality
  - To learn and improve
- Focus
- Openness
- Respect
- Courage

The Players

- Small team
- No one player is 'in charge' of the others. Each must collaborate with the others
- Self organizing

1. The Product Owner
2. The Development Team
3. The Scrum Master

But they COLABORATE! No hierarchy. No silos. No us vs. them. One team.

The Product Owner

- Is a one person role that injects the business perspective into the process
- Represents other stakeholders to the rest of the team
- Maintains an overall product vision and roadmap
- Maintains the product backlog (issues) continually and in order
- Decides release dates and content

The Development Team

- Self organizes – how much and how to do it
- Develops code, tests, PCB, mechanicals, etc. – cross functional
- Estimates the backlog efforts
- Sets the amount of backlog items to do per sprint

The Scrum Master

- Is a one person role that facilitates the collaborations between the product owner and development team members.
- Master of no one, master of scrum skills
- Coaches and mentors the players in the game of Scrum
- Shields the team from interference
- Removes obstacles or impediments
- Provides retrospectives
- Guides continuous improvement

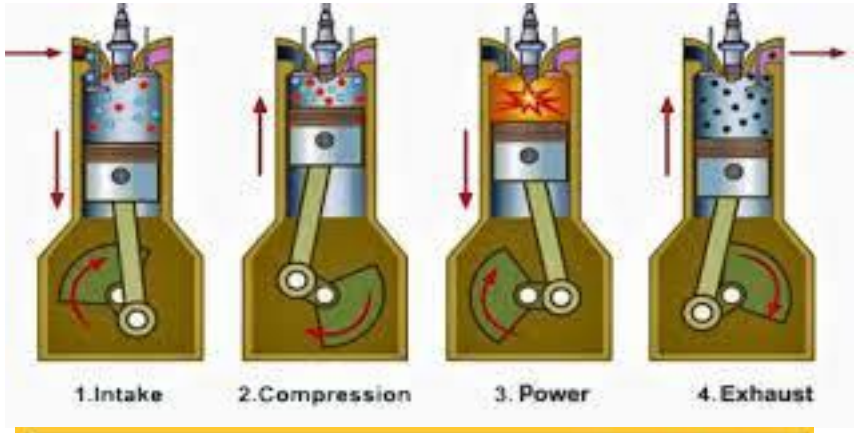
Sprints

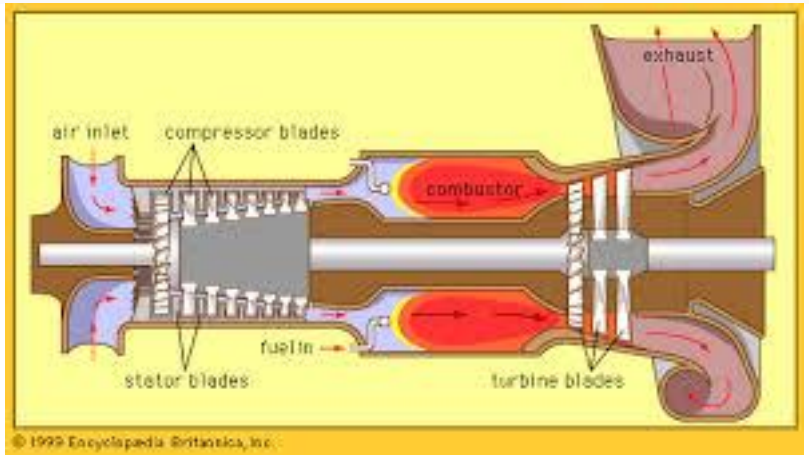
- Badly named. A rugby thing? Continuous never ending cycles of work kept at a maintainable “all day” pace. It’s a value attempt.
- Consistent time boxed iterations of backlog items
- Delivers a working version of software each time
- Best effort to complete, non interruptible but cancellable
- Comprised of
  1. The sprint itself
  2. sprint planning
  3. daily scrum meetings
  4. sprint review
  5. sprint retrospective

Why Sprints?

You don’t have to have sprints to be agile but…

- Scrum vs. Kanban, piston engine vs. turbine engine
- Chaos of change vs. sticking to a plan
- Continuous delivery
- Allows reflection and correction
- Provides a sense of velocity





Sprint Planning

- Relies on backlog refinement to make it less onerous
- Before the sprint starts, all team members
- Get backlog items into a sprint backlog
- Get sufficient definition on items
- Get sufficient estimations
- Get items in priority
- Decide what can get done in the sprint

Daily Scrum Meeting

- SHORT 15 minutes. Stand up. Every day. Maybe
- Three questions
  - What did you do?
  - What are you doing?
  - Any impediments?
- Walk the board
  - Issues from right to left
- But make it whatever works for you

Sprint Review

- Share what is DONE – demos, videos, papers
- Get feedback on what the team has done from outside the team
- Get closure on the sprint, celebrate accomplishments
- Determine actual team ‘velocity’, how much a sprint can hold

Sprint Retrospective

The continuous improvement part

Scrum Master runs it

Does it have to be at the end of every sprint?

Free food works great

- How has team worked together?
- What did you like?
- What was lacking?
- What failed?
- What did you learn?
- What do you long for going forward?

The Artifacts

1. Product backlog
2. Sprint backlog
3. Sprint deliverable

Part 2

“From the trenches” or “What I wish someone told me when I started”

Overall – don’t be pedantic. There is no existing recipe. Take whatever people tell you (even me) in and find your own path.

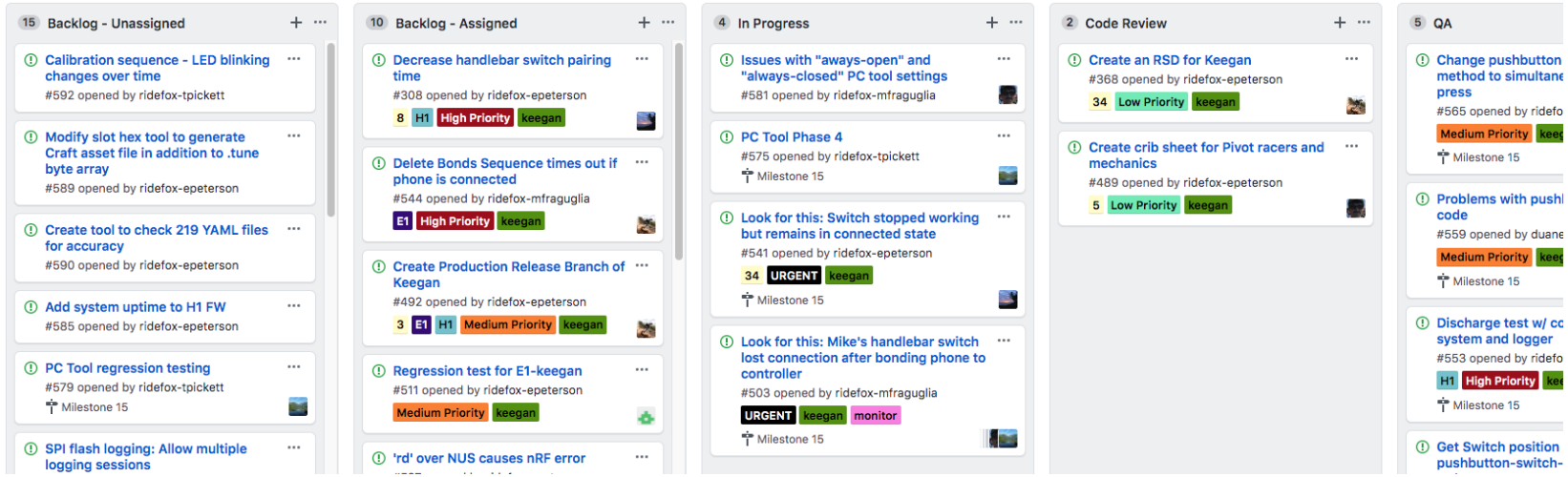
However, don’t ignore well trodden paths. They are usually well trodden for a reason.

Backlog Refinement

- Continuous not time boxed
- Issues as ‘user stories’
  - As a <role> I want <feature> so I can <rationale>
- Issues as behaviors
  - Given <context> when <event> then <result>
- Issues as bugs
- Anyone can put anything into the backlog. I will get triaged by the product owner
- Error on the side of putting things in the backlog vs. dropping the ball
- Ok if not fully fleshed out
- Break down large stories into issues that can be sprint-ified

Visualizing Progress

- Transparency - information radiators
- Burn down or others
- Kanban (Project) board
- No status reports!



Velocity

How many story points on average can the team do in a sprint?

- What's a story point?
  - Is it hours?
  - Is it days?
  - Is it arbitrary?
- Non linear due to cone of uncertainty
- People are bad at absolute estimates, but good at relative ones

Planning Poker

I have an idea how hard this story is.
You have an idea how hard this story is.

Without influencing each other ahead of time, put your cards on the table.

If we don’t have the same estimate, one of us knows something the other does not. Figure that out.

Failure

Failure is part of emergent design. Failure on an attempt to add value without recovering some knowledge about that failure is waste.

We don’t strive to fail but…

- We should accept it as part of the value creation exercise
- Make failures quick to discover
- Keep failure cheap
- Figure out how to extract knowledge from failure

This is the purpose of the sprint retrospective

Kanban/Scrumban

Scrum is not the only way to be agile. Many teams have decided that Scrum requires too much overhead.

- Kanban focuses on flow of issues rather than fixed sprints
- #NoEstimates – size issues to be the unit of work, estimate by count of issues.
- May be a better fit with Continuous Integration and Continuous Deployment
- Can be applied to “what you do now” as opposed to a sea change