SOFTWARE CRAFTSMANSHIP

gone opn

talks@goneopen.com

Summary

Software craftsmanship is a movement about getting better at software development particularly through better coding skills. This talk will look at some key discussions over the last ten years with a particular focus on Sennett's ideas from *The Craftsman* and ask: what does it mean to become a craftsman or craftswoman? how do we get better? I also look at why as craftspeople we might be troubled and when we may need to be vigilant! I will try outline how this is relevant to practices like continuous integration and test-driven development.

gone opn

Software Craftspeople

- Who is a craftsperson?
- What's been said over the last decade?

Who is a craftsperson?

- Manual skills to produce things
- To do things for their own sake
- A level of freedom to experiment
- Reward of competence
- Within the context of a community to meet standards

Richard Sennett: The Craftsman

gone opn

Software Craftsmanship: over the last decade or so

- The Craft of Testing (1998)
- Pragmatic Programmer: From Journeyman to Master (2000)
- Software Craftsmanship: The new imperative (2001)
- Clean Code: A Handbook for the Agile Software Craftsman (2008)
- Software Craftmanship Manifesto (2009)
- Cutter Consortium Special Edition (2010)

Software Craftsmanship

- Against "good enough software" design
- Software is becoming more, not less, labour-intensive
- Away from manufacturing metaphors
- Attention to people and their skills
- Against specialisation
- Experience-heavy teams
- Focus ship, cleanup & evolve
- Need Apprentices, Journeyman structure
- Need Master structure too

Pete McBreen, Software Craftmanship

Software Craftsmanship Manifesto

Not only working software,

but also **well-crafted software**

Not only responding to change,

but also **steadily adding value**

Not only individuals and interactions,

but also **a community of professionals**

Not only customer collaboration,

but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable

Software Craftsmanship: contested practices

- TDD (particularly test-first)
- Continuous Integration
- Pair Programming

Bob Martin, podcast

gone opn

Why are we troubled?

Motivation

Developing Skills

conflicting measures
of quality

gone opn

Motivation

- Projects depress coders
- Depressing organisational practices

Motivation

"

Developments in high technology reflect an ancient model for craftsmanship, the reality on the ground is that people who aspire to being good craftsmen are depressed ignored, or misunderstood by social institutions. These ills are complicated because few institutions set out to produce unhappy workers. People seek refuge in inwardness when material engagement proves empty; mental anticipation is privileged above concrete encounter; standards of quality in work separate design from execution

(Sennett, p.145)

Project modes depress coders

- Project strategies are in place before the delivery team is in place
- People who work with clients are higher up on the food chain
- Analysis driven projects miss opportunities and create hierarchies (money is spent on documents and “architects” are high status)
- More complexity and scale equates to more people rather than more experience

Also see McBreen

gone opn

Depressing organisational practices

- Sink or swim approaches to personal development
- No clear career paths: practically, technical jobs have ceilings within organisational structure compared with management
- Shoulder tapping that creates individual competitiveness
- Are you rewarded for doing a good job for its own sake? My experience is rarely

Also see McBreen

gone opn

Developing skills

- How repetition is organised
- How TDD learning is organised
- Making good use of technology
- Conflicting measures of Quality

How repetition is organised

- Wary of ideas that skills are innate
- They are practiced and repeated
- Skill development is based on *how repetition is organised*
- *Problem in project-based work is* developing skills outside the immediate task that would in fact help inside it – *TDD is good example of this (of many)*

How TDD learning is organised

- TDD is difficult to learn on the job
- TDD is difficult to learn not on the job
- Teams need it baked into their process: *done*
- Novices need structure to follow and scaffolding to practice *eg test automation pyramid*
- Types of tests need to match complexity or importance of architecture: *eg classical TDD in some parts, mocking for others, story tests for system wide*

Making good use of technology

- Problem of vendor solutions – often too generalised – *eg drag-until-you-drop development or visual deployment strategies*
- Leads to a problem of overdetermination *which is the goal when trying to commoditise development*
- Represses difficulty – *test-first is hard*
- Creates unrealistic totalised pictures – *eg ideals of high code coverage metrics*
- But the difficult or incomplete are positive events for learning or improvisation

gone opn

Conflicting measures of Quality

- Correctness and functionality
- Quality with CI & Deployment

Conflicting measures of quality: correctness and functionality

"

What do we mean by good-quality work? One is how something should be done, the other is getting it to work. This is a difference between correctness and functionality. Ideally, there should be no conflict. In the real world, there is. Often we subscribe to a standard of correctness that is rarely if ever reached. We might alternatively work according to the standard of what is possible, just good enough – but this can be a recipe for frustration. The desire to do good work is seldom satisfied by just getting by.

(Sennett)

Quality - Getting it right and getting it done: CI & deployment

- Being able to deploy a system quickly the first time is very different from being able to deploy a system quickly (and stably) every time
- Standards for continuous deployment are often contentious in projects
- **Tacit:** embedded within the scripts that move code through environments and integrate
- **Explicit:** breaking tests and integration points serve as critique
- Measures of standards must be observable from the outside: build server reports serve to create corrective actions

gone opn

Craft

Acquiring and Developing Skills: Thinking TDD through Richard Sennett

- General
- Specific

General techniques of a good craftsperson

- ✓ works on the whole
- ✓ is able to delay
- ✓ skilled in ambiguity

gone opn

Some specific techniques

hands

Expressive *Instructions*

Arousing Tools

minimum force

Richard Sennett, The Craftsman

Hands: rhythm of concentration

Red, green, refactor, checkin, get notification

Hands: rhythm of concentration

- The hand informs the work of the mind: advanced hand technique might inform technical skills
- Tempo and anticipation: you should not be overly conscious of your hands as you loose the insight of anticipation
- The whole: working in small cycles stressing the beat
- Delay and ambiguity: develop the skill of anticipation (through change tolerant code) because we don't need a full understanding to begin work (knowledge is always partial)

Expressive instructions: show don't tell

How do we know software works?
How do we know how its works?

- Dead Denotation:
  static documentation, difficult to understand test suites
- Scene Narrative/Sympathetic instruction:
  user stories (bdd, stdd, atdd ...)
- Metaphor:
  technical debt, test smells, bargain hunting

Arousing Tools

"

Getting better at using tools come to us, in part, when the tools challenge us, and this challenge often occurs just because the tools are not fit-for-purpose. They may not be good enough, or it's hard to figure out how to use them. The challenge becomes greater when we are obliged to use these tools to repair or undo mistakes. In both creation and repair, the challenge can be met by adapting the form of a tool, or improvising with it as it is, using it in ways it was not meant for. However we come to use it, the very incompleteness of the tools has taught us something.

(Sennett, p.194)

Arousing Tools: general purpose tools

- xUnit as a simple, all purpose tool rather than fit-for-purpose
- Help us make *and* leaves space for repair: ***bdd, stdd help solve different problems***
- Making and repairing are the same process
- It is a tool that serves as curiosity's instrument
- It allows for engaging with the code and system as a whole and in parts (quickly, precisely and repeatably): ***test automation pyramid is an example of making distinction between types of tests – unit (no dependencies), integration (one dependency) and system/acceptance tests (interactions between dependencies)***

Arousing Tools: structures people's imagination

10 years on and xUnit has spawned innovation: classical, mocking, BDD, story-tests, syntax helpers, machine tests, text-test, FIT

- **Reformatting**: explore the problems in different ways depending on experience and depending on the nature of the software – letting the tool be changed – for example nUnit/MSTest is extendable or chainable by StoryQ. Gallio can run MbUnit, xUnit, nUnit, MSTest tests. Any framework works with mocking frameworks.
- **Adjacency**: bringing together coding and testing to see that test first is as much about design as verification
- **Surprise**: positive role in stimulating imagination – letting more people into the process – FIT expects the client to write the tests, BDD suspects that more than just the original developer is reading the code
- (Not Defying) **Gravity**: expanding competence in what can be done and who counts as the delivery team!

Minimum Force: managing delay to create ambiguity

- **Resistance in code is found & made:** code and test smells
- **TDD focuses on friction:** technical debt, change tolerant code
- **Delay is writing test code:** test first development
- **Ambiguity is about not knowing and forces us to think about economy:** refactoring to patterns, micro-refactoring, domain limitations
- **Minimum force is patient refactoring:** as opposed to hacking and using the debugger. Adding features is easy when the code is clean – keeping it clean enough to add is the problem. That's one reason to refactor features into the code rather than endlessly bolting them on

What should we be vigilant against?

- Splits between problem solving and finding
- Things that separate head and hand
- Forgetting that acts of repair are proving grounds
- Thinking that ambiguity can be removed
- Forgetting that people are the most important part of software

based on Richard Sennett, The Craftsman

gone opn

Craftsmanship: Getting Better

- Quality driven
- Improving
- Deliberate practice
- Necessary practice

Quality-driven craftsmanship

- ✓ understands the importance of a sketch – a working procedure for preventing premature closure
- ✓ places positive value on contingency and constraint – problems are sites of opportunity
- ✓ avoid pursuing a problem relentlessly because perfection becomes self contained
- ✓ avoid perfection that degrades into self-conscious demonstration
- ✓ learns when it is time to stop – further work is likely to degrade

Richard Sennett, The Craftsman

gone opn

How do we improve?

- people can and do improve because ability is learned: *we need better career structure*
- but not necessarily linearly as you often you take detours: *we time to practice and innovate*
- you need to take instruction: *often by watching and observing - and taking in the whole*
- you need to use imagination: *and encountering resistance, ambiguity and improvising*
- but we need tools that propose rather than command: *standardisation tends to work against us*

Deliberate Practice

- Coding Kata (and also kata casts)
- Dojo
- Open Source projects
- Personal projects
- Learn new languages
- Pairing
- Discussion lists

Necessary Practice

- We need to take responsibility for our learning
- Find time within/without our job
- Being highly skilled in toolsets does not equal being a craftsperson: *the craftsperson can evaluate when to and not apply techniques and a combination of toolsets*
- It does take time but that alone is not enough
- Where are the masters you look up to and learn from?
- Who provides you with feedback?
- Are you taking time to instruct others?

References

Primary References

- The Craftsman, *Richard Sennett*
- Software Craftsmanship, *Pete McBreen*
- Clean Code, *Bob Martin*
- Software Craftsmanship Manifesto - *manifesto.softwarecraftsmanship.org*

Podcasts:

- *dotnetrocks – Bob Martin, Cory Haines*
- *Richard Sennett in England with Royal Society*

Slide share references

Search: http://www.slideshare.net/fsearch/slideshow?q=software+craftsmanship

- *http://www.slideshare.net/goeran/software-craftsmanship-ntnu*
- *http://www.slideshare.net/urilavi/software-craftsmanship-1-meeting*
- *http://www.slideshare.net/CoryFoy/growing-and-fostering-software-craftsmanship*
- *http://www.slideshare.net/henrydjacob/craftsmanship-the-meaning-of-life*
- *http://www.slideshare.net/giordano/better-software-developers-3989933*