Software Craftsmanship

Carl Erickson, PhD
Atomic Object LLC

These notes are a condensation of the ideas found in Pete McBreen’s excellent book, *Software Craftsmanship: the new imperative*
Software Craftsmanship

• The guild system worked for centuries
  – an effective means of propagating arcane knowledge
• Acknowledges that software is complicated
  – and getting more so
• Puts people back in the picture
  – instead of standardizing them away
• Recognizes that learning requires practice
Implications of Craftsmanship

• Pride of work and visibility
  – signing your work

• Taking responsibility
  – not hiding behind a license agreement

• Relationship to users
  – direct, not through intermediaries

• Reaching a mass market
  – advantage of software over other crafts
Delivery Dates & Quality

• Software projects often miss deadlines
  – but less often miss estimates
• Unrealistic dates put craftsmen at risk
  – reputation is what they live by
• Shipping with bugs
  – comes from the “good enough” camp
• Select developers on quality, not cost
  – Deming’s point about total cost, not project cost
Sign Your Work

- Craftsman live by their reputation
  - customers will choose the best
  - developers will not work with bad customers
- Reputation requires visibility
  - open source as prime model
- Accountability for mistakes
  - learning requires feedback
We’re Not All Equal

• Software engineering is about managing large numbers of average developers
• Small teams of very good developers can produce amazing applications
• The best developer is 10x better than an average developer
  – so pay them 10x as much and hire 1/10 of them
The Need for Iteration

• Iterative, incremental delivery is the only way
  – paying 10x average means high expectations
  – not sacrificing quality may mean schedule risk
• Small teams, faster delivery mean less risk
  – than large teams with long delivery cycles
• Customers need to engage
  – or risk wasting very expensive developer time
Do-it-Right vs Debugging

• Which release is better?
  – minimal features, rock-solid, functional
  – feature rich, unstable, buggy
• Debugging your way to stability
  – the “good enough” approach
• Booch’s observation on complex working systems
• Aligning interests
  – customers want great software
  – craftsmen want to build great software
Software Craftsmanship

• General skills versus specialization
  – complete job, start to finish, into maintenance
• A talented craftsman can know the system
  – avoid the inefficiencies of specialization
• Pride of work means all aspects
• Craftsmanship requires mastery
  – more than skills and knowledge
  – attitude and commitment
  – learning new things, readily admitting mistakes
  – pass along knowledge
Becoming a Craftsman

• Schools aren’t good, since they mostly ignore
  – situated learning
  – legitimate peripheral participation
• Apprentice to a master
  – learn by doing real work
  – and from other apprentices
  – start simple, move up in task complexity as you learn
• Seek others who share passion, enthusiasm, and pride
Characteristics of Mastery

• Becoming productive quickly in a new tech
• Years of experience delivering and maintaining
  – to gain insight into what makes a system last
• Not interested in unstable technologies
  – cult of the new works against long-lived apps
  – time to learn tools and tech must be paid off
• Willing to pass the craft along
  – 1 craftsman x 2 journeymen x 2 apprentices
The Apprentice

• More about learning than teaching
  – load on the craftsman’s productivity
  – self-reliant, coachable, enthusiastic

• Feedback from others
  – crucial to learning
  – reviewing work of the craftsman

• Ask good questions, bring new technologies
  – so apprentices contribute and teach as well
The Journeyman

• When you’ve learned enough as apprentice
  – about technologies, practices, and craftsmanship
• But still have things to learn from craftsmen
  – and experience to gain
• Journeymen perform the bulk of work
  – working together, or with a craftsman
  – coaching and guiding apprentices
What Can Software Engineering Teach Craftsmanship?

• Size and complexity matter
  – reducing team size reduces communication and coordination burdens
  – modular decomposition helps with complexity

• Programming in the large is different
  – talented craftsmen and expressive language shifts the boundary of what is “large”
Learning from SE, cont.

• Structure is important
  – OO was created to tackle complexity and size
  – Ditto decomposition and design patterns

• Change is inevitable
  – so embrace and handle it well (iterative process, incremental delivery)

• Change before delivery is risky and expensive
  – so deliver more frequently
Learning from SE, cont.

• Communication within the team is crucial
  – so don’t have teams spread across multiple sites
• Communication with customers is crucial
  – so co-locate them with developers
• Craftsmanship is personal
  – get people together
  – let them build trust
Learning from SE, cont.

• Documentation is always wrong
  – so don’t waste time producing it
  – produce documentation that will stay current (test suites)
  – keep the code readable

• Incremental development manages risk
  – tackle unknowns quickly
  – deliver early and often
Learning from SE, cont.

• Accurate estimates are expensive
  – Initial estimates based on high level functional descriptions can be wrong by a factor of 4x
  – With main requirements identified, reduced to factor of 2x
  – With all requirements identified, reduced to factor of 1.5x
  – With detail design done, reduced to 1.25x

• Believe the initial estimates
  – Or alter the scope
Conclusion

Software Craftsmanship offers an alternative model to Software Engineering.

• Respects the importance of the individual
• Addresses the software crisis in a systemic fashion
• Is a better match for most projects
• Has a successful historical precedent