The Keys to Agile Software Development

Jon Kern
Agile Something-or-Other
TechnicalDebt.WetPaint.com
JonKern@comcast.net

INTRODUCTION

- Doing software since 80s, O-O since ~late 80s
- Aerospace engineer (5 yrs jet engine R&D, 10 years DoD consulting, flight simulation, centrifuge, fighter agility, etc.)
- Formed Lightship Inc in ’95, clients like IBM, Ingersol...
- First published agile method in ’97
- Co-author Java Design w/ Peter Coad
- Led OO/Java workshops, mentored hundreds
- Joined Peter to form TogetherSoft Sep ’99
- Formed the Coad-Certified Mentor group; mentored on OO, FDD
- Led Together Product Development teams in Russia
- Co-author Agile Manifesto, 2001
- After growing to $56M and 400+, sold TogetherSoft to Borland in late 2002
- Recruited by OptimalJ (MDA) Team 2003, left late 2006
- Consulting since 2006, mentoring teams on architecture, development process, and agile methods to build business solutions
What’s on Tap

**Agenda**
- How *do* we develop software anyway?
- Present 3 simple keys to software development
- See how an agile process is a necessary -- but not sufficient -- component of successful teams
- Real world examples

**Goals**
- Learn simple fundamentals that apply to every software project
- Find out how “It’s the Business, Stupid” applies to your team
- See how missing one of the 3 keys is damaging to optimal success
- Learn how to gauge competing strategies and solutions to strike a balance to find the “Sweet Spot” for the business

How Is Software Built?

**People,**
**Process,** and
**Tools!**

**In about that order…**
- Skilled teams will trump poor/non-existent process
- Skilled teams will create process to be more effective
- Skilled people will create and use tools in an effective manner
- Process and Tools cannot, in and of themselves, make up for lack of skill and wisdom (though process can help provide checks/balances)
- Gray matter is a pre-requisite
- This is knowledge work! (not grunt work)
Is there one critical architectural principle for software development?

- 3-tier
- Peer-to-peer
- Design Patterns
- Object-oriented
- Aspect-oriented
- SOA
- Composite
- Modularity
- Distributed
- Subroutines/procedures
- MVC
- Layered
- Peer-to-peer
- Design Patterns
- Object-oriented
- Aspect-oriented
- SOA
- Composite
- Modularity
- Distributed
- Subroutines/procedures
- MVC
- Layered

Separation of Concerns

- An architectural best practice, the “father” of many patterns
- Keep business logic in one place/layer
- Architect with the future in mind, by capturing the Domain of the business
- Keep technology at arm’s length and “in its place”
- Isolate change to the smallest place(s)
What Happens without SoC?

- It takes 80% of the effort just to “keep the lights on”
  - App seems “brittle” – hard to change without causing breakage (low modularity/cohesion, high coupling)
  - It is hard to add new features
- No one can really point to your business model
- Hard to train new developers on the application because of:
  - redundant and/or repeated code, dead code
  - lousy documentation
  - confusing semantics
  - convoluted workflow
  - inconsistent architecture/code
- Performance can suffer over time (or immediately)
Only You Can Prevent Technical Debt

http://TechnicalDebt.wetpaint.com

How do we reliably turn Requirements & Design into the desired application?

- Code snippets
- Frameworks
- Pair Programming
- Use senior developers
- TDD
- Agile Process
- Outsource
- RUP
- Lots of UML
- Heavy-weight Process
- Model-Driven Architecture
- Domain-Specific Lang
- Kowboy Koding
Consistent Construction - Application Arch.

- **Application architecture**
  - Application architecture is what the customer wants and needs
  - It is the set of features, often separated into major subject areas (or packages), what the client pays for
  - Independent of the technical solution
  - This is where you need to make rapid and visible progress
  - This is where you will want your developers to be creative
  - There are frequently common design elements that can be presented to the user across the application -- for aesthetics and efficiency
  - Consistent handling of domain elements to promote ease of use (versus demanding recall or use of help/tech support)
  - Consistent use of actions across the app
Consistent Construction - Technical Arch.

- **Software architecture**
  - Technical architecture is the “necessary evil” -- and often it can make or break the entire project/product line
  - This represents the “how” an application gets delivered to the end user, driven largely by “-ilities”
  - If we could simply specify application features and have them automatically built within the correct technical architecture and all of the coding guidelines, that would be great
  - Technical architectures are often designed at the outset of a new application development project (Thin Slice/Spike) that has a demanding aspect to be met
  - However, some architectures can be reused on subsequent projects (to save costs and increase consistency across products/projects)
  - Technical architecture changes more slowly than requirements and should generally be completely refactored across the app if/when necessary

Consistent Construction - Critical Details

- **Software Nuts and Bolts**
  - Matching the technical details to your needs is non-trivial
  - Technical aspects -- such as using modern OO languages -- can have crucial impact on your success
    - Greater numbers of qualified developers available
    - More sophisticated tools exist
    - Greater availability of information
  - Coding standards, naming conventions, APIs are all part of translating requirements into working code
  - As you build out the features, you are weaving the requirements through the architecture, using good coding practices and guidelines

- **Consistency is a quality unto itself!**
Consistent Construction - Putting it Together

- Application Arch.
- Technical Arch.
- SOA, Distr., Web App
- Variable
- Consistent
- Construction Guidelines
- Patterns, Conventions
- Working App!

What Happens without Consistent Arch.?

- **Consistency is a key quality**
  - Whether we are talking about code conventions, UI, class/table names, major architectural pieces, or even sample data

- **The nightmare:**
  - You can't spot any patterns to follow
  - Development is always an adventure
  - Maintenance is more challenging
  - Refactoring is hard to accomplish
  - You can experience inconsistent behavior
  - Developers looking in the wrong places
  - Manually and painstakingly fix the errors across all inconsistent bits
How do we get the team coding an application such that it meets the business needs?

- MIL-STD-498
- Waterfall
- XP
- Evolutionary Prototyping
- Crystal
- FDD
- TDD
- Lean Process
- DSDM
- BDD
- CMMI
- RUP
- SCRUM
- IEEE 12207
- Six Sigma
- Agile Process
- ICONIX

Software Requires Balance

- **Planning**
  - Need to have budget and schedule estimates
  - Guesstimates needed, but we don’t have enough to go on
  - We want variable scope -- but with fixed team & schedule
  - Planning should be ongoing, but Gantts are hard to keep updated
  - Dev team might do well with simple issue tracker, but “Corporate” needs more

- **Requirements/Analysis**
  - Just start coding with minimal requirements, the application will emerge
  - Too much requirements (details) and the analysis can be wasteful
  - Some projects are more flexible than others
  - Work with BAs, users, proxies
  - Gather all information upfront, users off limits during development
Software Requires Balance… continued

- **Design**
  - Architectures are emergent for the App UI and the software
  - App design effort needs to match the criticality of User Experience
  - Tech design effort depends on severity of non-functional drivers

- **Coding**
  - Code quality, consistency, working results, craftsmanship
  - Easy access to experts for covering the nuances of requirements
  - Development tools, processes, automation big part of success factors

Software Requires Balance… continued

- **Testing**
  - Build quality and tests in from the start
  - Conduct inspection tests, add quality as needed based on QA
  - Type & depth of testing varies -- How much is needed?
  - Testing tools, processes, automation big part of success factors
  - Frequent delivery & on-going feedback from users

- **“Peopleware”**
  - Software is significantly impacted by people issues
  - Staff ranges from Superstars to fresh graduates
  - Company culture, team culture -- might clash or complement
  - Product visionary, responsible “owners,” active users
Software Development is a Major Challenge

- We have a confluence of business requirements, processes, and cultures that all “meet” in a software product
- We can practice Separation of Concerns
- We can design and specify a good Application design
- We can specify a solid technical architecture
- But how do we put it all together since
  - A prescriptive process may be too rigid and ineffective
  - Minimal process is too risky
  - We need to maintain checks and balances
- What can help us achieve a balance?

Agile Development

- Agile is easy to comprehend and practice, but hard to master
- A constant state of mind that never rests, not a fixed set of steps!
- Always communicate as much as possible
- Shrink waterfall cycle into smallest iteration size
- Ensure working application and feature progress
- Constantly reflect and improve Software & Process
- Deliver business value Frequently
- Running Software: Necessary – but not sufficient!
- Reduce time between cause & effect
- A conundrum?
Agile Manifesto

- Individuals and Interactions
- Working software
- Customer collaboration
- Responding to change
- Processes and Tools
- Comprehensive documentation
- Contract negotiation
- Following a plan

While we value the things on the right, we value those on the left more

http://agilemanifesto.org

Putting 3 Keys Together!

- Separation of Concerns
- User Interface
- Business/Domain
- Application Arch.
- Technical Arch.
- Construction Guidelines
- Working App.
Questions

Thank You!

- Contact me at JonKern@comcast.net
- Community Site: http://TechnicalDebt.wetpaint.com