

E-Framework 2007

agile development

Scott Wilson, CETIS

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

<http://www.cetis.ac.uk/members/scott/foaf.rdf>



This work is licensed under a Attribution-NonCommercial-ShareAlike 2.0 licence

CETIS

Or...

“The main thing is making sure the main thing stays the main thing”

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

What is agile development?

- Principles
- Values
- Patterns

The Agile Manifesto

Individuals and interactions over
processes and tools

Working software over comprehensive
documentation

Customer collaboration over contract
negotiation

Responding to change over following
a plan

Methodology

- Agile isn't the same as sloppy!
- XP: Tailor practices to the project

Understand your deliverable

- Known unknowns... who are the users? What is the type of innovation? What is the process?
- Depending on the answers, you may want to change methodology

UML Is Not a Religion

- Understand how you use diagrams
 - Fowler: UML modes
 - Reflection & communication
 - Inspiration or model
- Choose diagrams that help. Don't draw diagrams that no-one will use.

Use Cases Are Not Always The Answer

- Use cases are best used where there are *known* users engaged in *known* processes
- In many other cases, you are engaging in product design - consider starting with a marketing plan and some wireframes instead
- No-one added a tag cloud because a use case said so

Patterns vs Requirements

- Sometimes patterns give a better answer than your users
- Use someone else's design brain
- E.g. don't invent a new audio selection interface - do what every other system does (copy iTunes)
- Nielsen's rule: 99% of the time, users use something *other* than your system

Patterns and the EF

- SUMs, domain models, process models etc. can be used a bit like analysis patterns

Design matters

- Its easier and cheaper to make applications that run in Powerpoint when you need to try out design ideas
- The GUI and UE should be central, not peripheral
- Sometimes its easier to start with the GUI/UE and work backwards to the logic and services

Design cheats to improve user satisfaction and feedback

- Support principal of least effort - user shouldn't have to use your system just because "its good for them"
- Flatter the user, it makes them feel good and say nice things about the software :-)
- Attractive things *work better*. Eye candy isn't just surface fluff

Release early and often...how?

- Look hard at the implementation plan
- Deliver a sequence of fully working modules with restricted functionality (you can actually test these with *real people*)
- 100% working with 70% of the functionality is *far* better than 70% working with 100% of the functionality.
- Remember Gall's Law

Gall's law

From Wikipedia, the free encyclopedia

Gall's Law is a [rule of thumb](#) for system design from [John Gall](#)'s book [Systemantics: How Systems Really Work and How They Fail](#):

A [complex system](#) that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system.

This law is essentially an argument in favour of underspecification: it can be used to explain the success of systems like the [world wide web](#) and [blogosphere](#), which grew from simple to complex systems incrementally, and the failure of systems like [CORBA](#), which began with complex specifications. Gall's Law has strong affinities to the practice of [agile software development](#).

Prototypes

- Its OK to throw away prototypes
- As noted earlier, powerpoint prototypes are extremely cheap and disposable
- Wireframes can work really well

Frameworks: a two-edged sword

- Frameworks enable you to move much faster by reusing code that solves common problems
- But they can come with hidden costs, including bloat and learning curve
- Sometimes its quicker to just write the part you need
- E.g. Castor vs JAXB vs XmlBeans

Convention

- Conventions enable reuse of technique which may lead to code reuse
- Choose popular combinations of tools to leverage other people's work
- Stick with the conventions of the tools, even in mixed implementations (i.e. don't program in ruby as if it were java)

Process

- Meet f2f at least weekly, preferably daily (or skype)
- Everyone should stay 'close to the prototype'
- No-one should be allowed to 'hoard' their code or deliverables
- Agile development ideally requires a small team with high levels of contact and commitment (no need to 'design by contract' within the team)

Tarpits

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

- Tarpits are inescapable morasses of unsolvable problems
- Avoid early by design
- You can best dissolve rather than solve tarpit problems
- Examples include security and configuration, ORM, XML

Some things I've found out

- Axis and J2EE are configuration hell
- REST can be implemented using POJS far more quickly than using any frameworks
- Ruby on Rails makes very quick very nice web apps and REST APIs
- Its easier to start from the human UI before doing the machine API
- No-one likes reusing components with weird dependencies

Web 2.0 patterns

- Do one thing well
- Work across devices
- Encourage participation
- Make public data public
- Narcissistic features

Web 2.0 principles

- **Simplicity** over Completeness
- **Long tail** over Mass Audience
- **Share** over Protect
- **Advertise** over Subscribe
- **Syndication** over Stickiness
- **Early Availability** over Correctness
- **Select by Crowd** over Editor
- **Honest voice** over Corporate Speak
- **Participation** over Publishing
- **Community** over Product

Cheers!

- scott.bradley.wilson@gmail.com
- <http://www.cetis.ac.uk/members/scott>
- <http://www.cetis.ac.uk/members/ple>
- <http://www.tencompetence.org>
- <http://www.xcri.org>
- <http://del.icio.us/tag/ple>