Designing Open Software 2.0

Colin Clark
Inclusive Software Architect,
Adaptive Technology Resource Centre
Who is this guy?

- I'm a software developer at heart
- Trained in fine art and cultural studies
- 10+ years at the ATRC
- Core contributor to Sakai
- Lead developer and interim project manager for the Fluid Project
  - http://fluidproject.org
Who are you?

- Programmers?
- IT managers or business analysts?
- Designers?
- Librarians?
Topics for Today

• Focus on *how* we work with technology
  – Development process
  – Software development models
  – Planning and supporting software

• Open source software culture
  – Strengths and weaknesses
  – Decision making

• An “on the ground” perspective
The Whys
Establishing Thoughts

• Software is terrible!
• Good intentions aren’t enough
• Our work should be driven by shared values
• Collaboration is hard, but it pays off
Software Failure

• Bottom line: most software projects fail
• Standish Group’s CHAOS Report:
  – 18% cancelled outright
  – 53% “challenged:” over budget, over time
  – 31% successful
• What about usability?
Envisioning Better Software

- **Inclusive**: open and adaptive
- **Usable**: software that respects people
- **Reliable**: tested, pliable, crafted
- **Natural**: physical, spatial, environmental
Defining Values

- Focus on people
- Emphasize communication
- Satisfy user goals, not just feature lists
- Open, interoperable, and free
Software Development Values

• Quality and reliability
• Simplicity
• Incremental and improvable
• Honesty and reflection
• Shared responsibility
The Hows
Software Methodologies

- Ad-hoc
- Classical Waterfall
- Agile and Extreme Programming
- Goal-directed Design
Ad-hoc

- Probably the most common methodology among uncoordinated developers
- Unplanned development: constantly solving the same process problems over again
- Often very quick at first, but never scales over time or complexity
Waterfall

- The prototypical stereotypical methodology
- Sequential: step-by-step phases
- Lengthy design and specification stages
- Assumes requirements are clear, fixed, and absolute
- Fundamentally unresponsive to change
Agile Development

• A nearly meaningless buzzword, but...

• Take a number of proven software practices:
  – Test
  – Integrate
  – Put code into production
  – Code reviews
  – Planning

• Focus on the things that work, and do them all the time
Agile Challenges

• Developer culture
• Organizational culture
• Interfacing with designers and testers
• Staying on track while responding to user feedback and change
Interaction Design

- Define a new role for designers
- Emphasis on ethnography and research
- Users and designers decide requirements
- User goals drive development, not technology
- Design should permeate the development process
Design is More Than Skin Deep

Consistency: skin, terminology, widget choice, page layout, color, font, etc…

Navigation, functionality, structure...

Goals, context, work practice, activity…
Challenges for Interaction Design

- Developer culture (again!)
- Research takes time
- Interfacing with agile development
- Tools and techniques
Inversion of Control

• No, not the Spring framework…
  – Design and use drives development, not coding
  – Think about abundance, not scarcity
  – Build trust and respect into the process
  – Attention to detail becomes really important
What about in open source?

• Questions to think about:
  – Is there a distinct open source development methodology?
  – Can we successfully use an existing methodology?
  – What are the strengths of the open source approach?
  – How does it compare to traditional development?
Open Source

“Every good work of software starts by scratching a developer’s personal itch.” (Eric S. Raymond)

• Whose itch? Who gets to help scratch?
Open Source

“Every good work of software starts by scratching a developer’s personal itch.” (Eric S. Raymond)
Strengths of Open Source

• Collaboration is the heart of OSS
• Enlightened self interested
  – Contribute your skills, reap the benefits of everyone's contribution
• Open is a substantial business value:
  – Control over your investment
  – Code review: potential for better code
  – Spreads the risk
Commitment to Interoperability

• Production vs. Consumption of data
• Ability to adapt, re-purpose, and reuse:
  – Accessibility
  – Longevity
  – Innovation
• Vendor and format lock-in is a terminal plague in our industry
Some Interesting Rhetoric

• “A Spectre is haunting multinational capitalism--the spectre of free information. All the powers of 'globalism' have entered into an unholy alliance to exorcize this spectre: Microsoft and Disney, the World Trade Organization, the United States Congress and the European Commission.”

– The dotCommunist Manifesto
Community Source Software

• Collaboration among larger institutions
• Clear governance strategies
• Specifically defined roles
• Funding!
• Less about individual commitment and more about organizational collaboration
Benefits of Community Source

- Supports academic values
- Emphasizes software as an essential tool, not commodity
- Adaptable to unique institutional needs
- Communities of common interest
A Weakness of Open Source

• A fundamental problem:
• Technology-driven, not user-driven
  – Lots of coders, few usability people
  – Emphasis on technical merits, not usefulness
  – Tends to separate and modularize; misses cross-cutting concerns
Protecting Precarious Values

• What's a precarious value?
  – The stuff we forget to do
  – The stuff we do at the end
  – The stuff we don't have time for
  – The stuff we don't understand
Protecting Precarious Values

- Usability
- Accessibility
- Quality Assurance
- Security
- i18n
- Documentation & help
- Others?
- The intersection of software with people
Why are they precarious?

1. Poor planning
2. Team composition
3. Ignorance
The Whos
Skills for Successful Software

• An ability to understand user needs, observe users in context, perform usability studies, and drive requirements
• A firm understanding of UI conventions and patterns
• A sense of effective visual design, layout, and style
• Knowledge of how to make software accessible for people with disabilities
• The ability to build user interface designs in HTML & CSS
• A strong understanding of the Web’s architecture
• An understanding of presentation frameworks in Java
• The ability to write secure and fast service APIs
• An understanding of database design and relational theory
A Stupid Bet

I bet $500 that we can’t find anyone in this room who can do all of this work simultaneously!
The Nature of the Team

• Successful software is built by people with different skills who are willing to work together
• Spectrum of skills, experience, language
• Identify skill gaps and train or hire
• Open source software is all about collaboration: use it to our advantage
Open Source Decision Making
Choosing Open Source

- Free software isn't free. So what?
- Investment in people vs. licenses
- Who supports it?
- Who's to blame?
- Decision making factors
  - In house development resources
  - Training resources
  - Usability and best fit for the job
Releasing Open Source

• Releasing open source software is:
  – Way harder than you think
  – Not a dumping ground
  – A huge time commitment
  – Thoroughly satisfying
Making the Decision

• Things to think about:
  – Long-term vision
  – Who cares?
  – Infrastructure
  – Licensing
  – Community model
  – Project coordination
Summary

• Know your software values
  – Build them in from the beginning
  – Emphasize people over technology
  – Interoperability is critical

• How you do it matters

• Invest in strong, interdisciplinary teams

• Open is hard, but will pay off if you understand the culture
Some References

• InfoQ Article on Software Failure:
  http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS

"Producing Open Source Software, Karl Fogel
"About Face 2.0, Cooper and Reimann
"Extreme Programming Explained, Kent Beck
"DotCommunist Manifesto, Eben Moglen

• Fluid Project
  designing software that works for everyone
  http://fluidproject.org