# AGILITY XL

**Robert J. Schaaf**

rjschaaf @ ieee.org

June 18, 2007

# How large is Extra Large?

- Projects with more than 200-1000 software engineers
- Spread over multiple development locations
- 1000's of users,  and multiple categories of users
- Alt.: 1000's of systems, and multiple categories of systems

# What are agile methods?

What purpose characterizes all agile methods?

- **Many definitions, no clear consensus**
  - **Ex.: Methods with values and principles from the Agile Manifesto**
  - **Ex.: To improve collaboration between customer and developer**

# What are agile methods?

Let's start with the Agile Manifesto…

- **Four value statements as in "We value A over B"**
  - **Do A but not B?**
  - **Practical meaning of a value statement is undetermined**
- **Twelve statements of principle**
  - **8 are truisms, ex.: "Build projects around motivated individuals"**
  - **1 is a claim, "Agile methods promote sustainable development"** with an overriding dependency on the quality of staff
  - **1 is a contested claim, "Best design from self-organizing teams"**
  - **2 of 12 point to generally-accepted, agile characteristics:**
    ***Accommodation of late changes in customer needs**
    ***Frequent iterations of code**

# What problems do code iterations solve?

What about “Do It Right The First Time”?

- **DIRTFT is great *if* all stakeholder needs are known early on**
- **However, four inconvenient situations:**
  1. **Customer changes his mind about his needs**
  2. **Customer discovers breadth and depth of his needs**
  3. **Developer’s trouble in eliciting the customer’s needs**
  4. **Emergent software properties, affecting customer needs**

- **Iterations are a way**
  - **–To elicit customer needs**
  - **–To absorb changes**

# Perspectives of iterations

- **Customer perspective**
  - **You will not get all of your needs satisfied in the first iteration, or in the early iterations, but there is a steady progression to full satisfaction**
  - **You can adjust your expectations to what's feasible**
- **Developer perspective**
  - **You may be unable to change this iteration, but you can respond to a new/changed/more specific customer need in a next iteration**
  - **You can adjust your resources for optimum effect**

# Definition

An agile method is a method

for the development and acquisition of software –

the method features a fast-paced stream of operable software iterations, and

the customer's immediate feedback regarding his real use of each iteration.

The feedback represents and causes an evolving understanding of

the customer’s possibly changing needs.

Through subsequent iterations,

the evolving understanding finds its way back into the software.

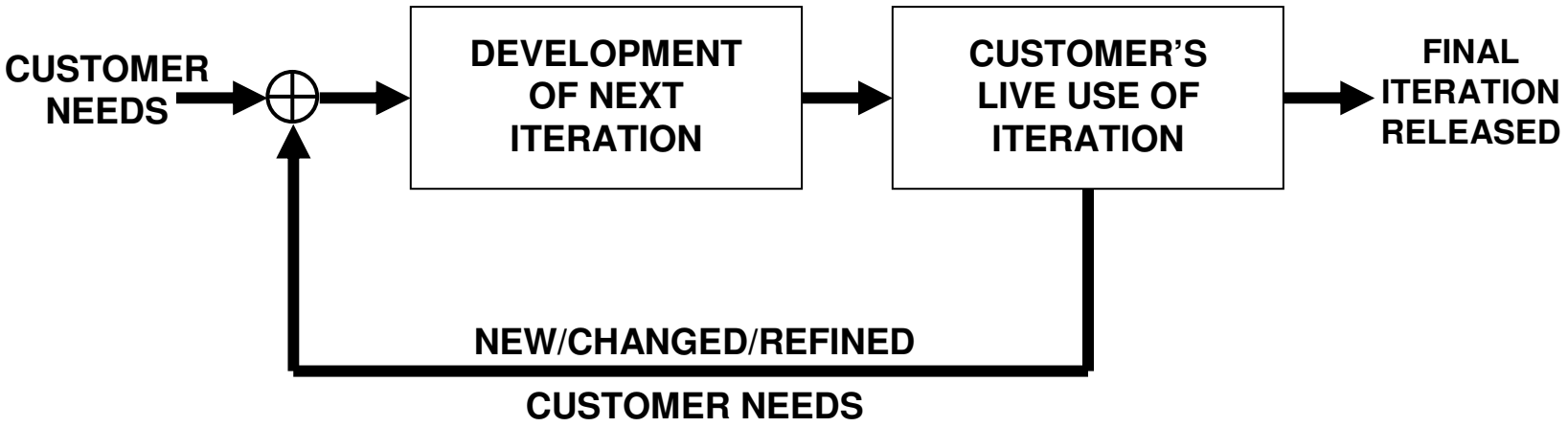
- **The definition defines a class, not one particular method**
- **Covers XP-like methods, but certainly is not limited to them**

An agile method is a method
for the **development and acquisition** of software –
the method features a **fast-paced** stream of **operable** software iterations, and
the customer's immediate feedback regarding his **real use** of each iteration.
[…]

- **Definition addresses developer *and* customer**
  - **Development, acquisition, operation and usage**
  - **Understanding by developer *and* customer**
- **Intervals between iterations from 1 day to a few weeks**
  - **Loss of dynamic if interval is > 8-12 weeks**
  - **Fixed or variable intervals, dependent on method**
- **Real use by real users, *not* testing**
  - **Degree of reality influences the success of the method's application**

# Feedback loop

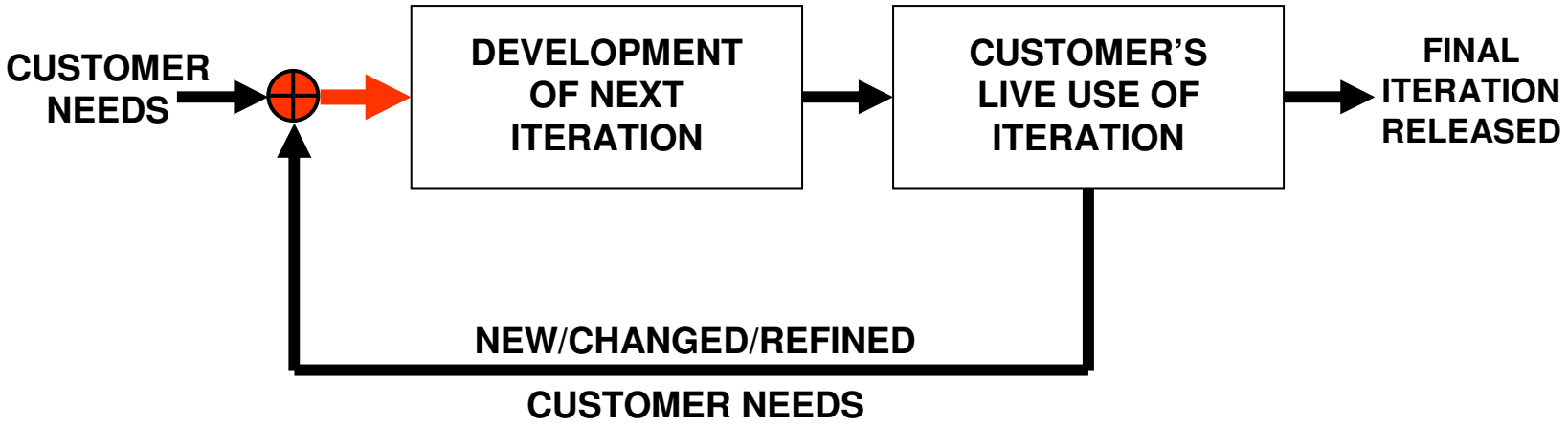
[…] the evolving understanding finds **its way back into the software**.




**Use of the software to develop more software**

# Opportunism

- **Agile methods are opportunistic methods**
  - **However, the opportunism doesn't have to be spread around**




**Opportunism does NOT have to affect the developer's process framework or his quality management system**

# Are smaller teams/projects the "home ground" of agile methods?

- **YES if you only consider the XP-class of agile methods**
  - **See a prominent paper [1] for this view**
  - **Another article [2] reports an average team size < 15, ideal ~10**
  - **Examples of method features that may constrain team size:**
  - -Daily all-hands meetings
  - -Collocation of customer and developer
  - -Process framework?
  - -Hooks into quality management system?
- **NO if you consider:**
  - **Empirical evidence from large projects**
  - **The need for agility in large projects**
  - **Large software projects through large sets of small teams?**

# Empirical evidence from large projects

- **Feedback, change and iteration -although not predominant- are as old as software engineering [3] – even iterations with short intervals**
- **Evidence from personal experience**

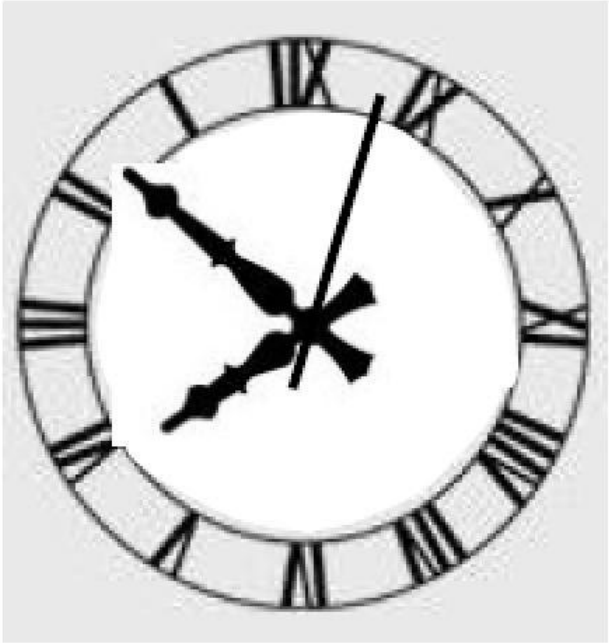
# From personal experience

- Project to develop a family of switch/router products
- Core version, metro version, gateway version, etc.
- ~1000 software developers + ~1000 hardware engineers
- Initial releases with several millions of source statements
- ~100 customers, close relationships with a handful
- Distributed development:
  - Core product in country A (250 software developers)
  - Metro product in B (250)
  - Software platform in C (100+)
  - Software tools in D (<100)
- Family based on a novel, distributed control architecture
  - Every product with 10's-100's of mini-servers
  - Much room for emergent behavior, some of it nasty

# After long start-up problems…

- The core version sub-project fell into weekly iterations
  - For immediate use in the network > feedback to next iterations
- Enabling factor: Core machine easy to insert and take-out
  - Much more difficult to do this with a metro machine
- Pattern established:
  - Weekly iterations, sweeping up whatever is ready and complete
  - Continuous regression testing of daily builds, by the developer
- Aber es lauft!  But it works!

# Aber es lauft



# Results

- **In ~1 year, the core product iterated to commercial status**
  - **For mass production and delivery to the country A customer**
- **The metro sub-project unable to put iterations in live use**
  - **Took more time and effort to reach commercial status**
- **Once the metro product had enough functionality…**
  - **Metro projects for later countries used iterations + feedback**

# Large projects, probably more than smaller ones, need feedback and iteration

| | |
|---|---|
| • Customer changes his mind | The larger the software, the more there is to change one's mind about |
| • Customer discovers breadth and depth | The larger the software, the more breadth and the more depth |
| • Elicitation issue | The larger the software, the more complexity, the more there is to elicit |
| • Emergent properties | Large software usually means exponentially larger amount of emergent behavior |

- **Emergent properties: Problem – and attraction – of large systems**
- **Real use of real software is about the only way to discover emergent properties**

**Natural terrain for agile methods**

# Large software projects based on numerous small teams?

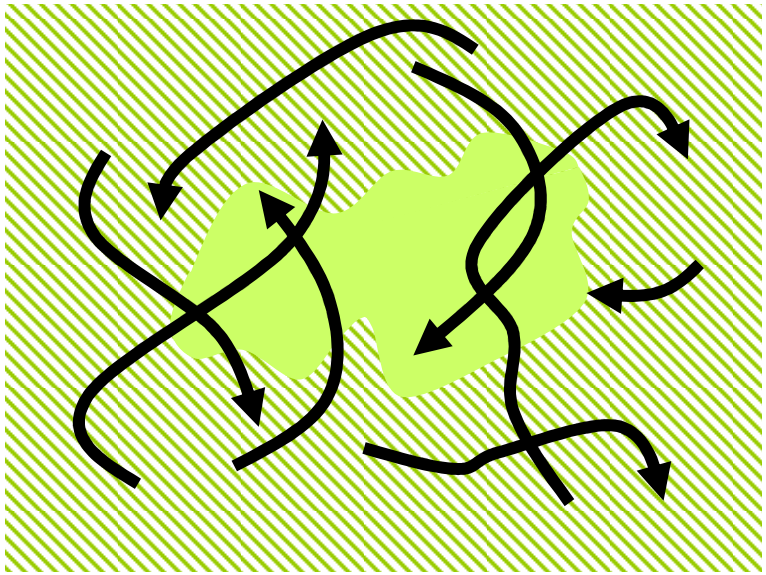
- **Literature suggests self-directed/self-organizing teams**
  - For instance: M. Lindvall et al., "Agile Software Development in Large Organizations," IEEE Computer, December 2004
- **Evidence of successful self-directed /self-organizing singletons**
  - For instance: Architecture team, User Documentation team
  - I.e.: Where authority over change can be localized
- **No evidence of 20+ sd/so teams for bulk of software development**
  - Main issue would be: Change that spans multiple sd/so teams

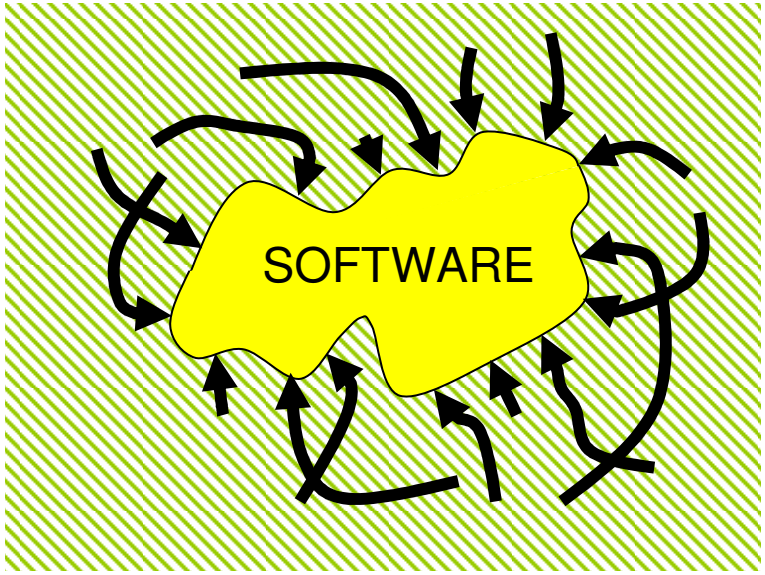
**Approach unlikely to scale up for bulk of software development**

# Mistakes in using agile methods

- **No customer-development agreement on mode of operation**
  - Dependency on the specifics of the particular agile method
  - For the class as a whole, **prior agreement** on at least:
    - -Length of the interval between iterations
    - -Degree of possible incompatibilities between iterations
    - -Quality of the iterations, including user documentation
    - -Scope of the customer's live use of the iterations
    - -Time required by customer to put iterations in use, provide feedback
    - -Communication channels, incl.feedback channel, user documentation
    - -Handling of software defects
- **Feedback of software requirements, not customer needs**

## Customer needs vs. Software requirements






Stated by customer
Irrespective of software boundary
Stated directly, as needs
Or indirectly, as feedback on use

Derived by developer from needs
Define the software, the solution
Input to design, testing, etc.

**“Tell me your problem, don’t give me a solution”**

# More mistakes to avoid

- **The agile developer transferring activities to the customer, e.g.:**
  - Customer deriving software requirements from customer needs
  - Customer prioritizing software requirements instead of needs
  - Customer deciding the features of individual iterations
  - Customer sharing in risk management of software development
  - Customer participating in verification whether software meets requirements

**Developer to deliver software that meets needs**
**Customer to use the software and provide feedback**

- **No user documentation with each iteration of the code**
  - User documentation is a reference point for developer's testing
  - User documentation allows customer to plan his live use of the iteration
  - Helps customer decide whether apparent features are intended by developer

# More mistakes to avoid

- **Treating tools as a matter of lesser importance**
  - 1st place: Change & Configuration Management tool
  - Support for **granularity** of check-in, e.g.:
    Code is unit-tested
    Code is component-tested
    Code is subsystem-tested
    Code is system-tested

- **Combining iterative development with one-shot contracting**
  - Iterative contracting, 6-12 months between contract iterations
  - **Incremental awards** of moneys/resources to the developer
  - Both parties may terminate contract with the next contract iteration

# More mistakes to avoid

- **The customer's use of iterations takes on the nature of testing**
  - No sharp dividing line between live-use and testing
  - Customer may be driven to testing by low quality of the iterations
  - Degree of agile success depends on:
    1. Quality of the iterations, limited in intended features as they are;
    2. The realistic nature of the customer's use of the iterations

- **Trying agile methods on projects not amenable to these methods**

# Contra-indications

- **Customer unable/unwilling to put iterations in live operation**
  - **Good reasons: Safety, security, conversion time/effort**
- **Related: Developer unable to provide compatible iterations**
  - **Architecture may minimize incompatibility from iteration to iteration**
- **Contributor: Complex software interfaces**
  - **Interfaces with users, other systems, data**

**Agile methods are not for every situation**

# What alternative if iterations of working software can't be used?

- **Reliance on working software → reliance on models**
  - **Model-driven software development [4]**
- **Models play same role as iterations, they trigger feedback**

| Advantages of working software | Advantages of models, modeling |
|---|---|
| Nothing beats real use | No possibly-risky use |
| Shows emergent properties | Allows for high level of abstraction |
| Straight path to end product | Analytical strength |
| No dependence on modeling skills | Less cost initially (but an overhead!) |

Even better: Agile methods + Modeling

# How does an agile method get started?

**As said before, an agile method uses the feedback from software use to develop more software**

- **Besides, early on the architecture needs to take shape**
  - **To maximize the compatibility from iteration to iteration**
- **When there's not yet software for the necessary feedback**

**Model-driven development/architecture [5] can serve as a bootstrap for agile methods**

# Conclusions

1. The class of agile methods is characterized by feedback, change and iteration
2. Large projects are the natural terrain for agile methods, excepting those individual methods with features that constrain them to relatively small teams
3. No evidence found that small-scale methods can be extended to the bulk of a large sw project
4. Agile methods, while powerful, are not for every situation, also not for every large project
5. Agile methods may leave presently-used process frameworks and quality management systems intact
6. Agile methods are not an occasion to fudge the line between the customer and the developer
7. In combination with agile methods, explore modeling, carefully

# References and Acronyms


**[1] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," IEEE Computer, June 2003**

**[2] K. Richards, "Early mainstream: Agile develops in the enterprise," Application Development Trends, July 2006**

**[3] C. Larman and V.R. Basili, "Iterative and Incremental Development: A Brief History," IEEE Computer, June 2003**

**[4] "Model-Driven Software Development," IBM Systems Journal, Volume 45, Number 3, 2006**

**[5] A. Brown, "An Introduction to Model Driven Architecture," IBM, February 2004, http://www-128.ibm.com/developerworks/rational/library/3100.html**


| | |
|---|---|
| AM | Agile Manifesto |
| DIRTFT | Do It Right The First Time |
| XP | Extreme Programming |