

Scaling Agile Processes

Agile Software Development in the Large

Agility Days 2002

Jutta Eckstein
Nicolai Josuttis

„Large“

- **Large in...**
 - Scope
 - Time
 - People
 - Money
 - Risks
- **We focus on „Large Teams“**
- **Large is relative**
 - 1, 2, 10, 100, 2000 People

Characteristics of „large“

- **Processes do not „scale linearly“**
 - New kinds of problems occur
- **For example: communication**
 - With 2 people you have to communicate
 - With 10 people you can't discuss all aspects in one group
 - With 100 people you can't have all people in one room
 - With 1000 people you can't know all people

Communication

- **Face-to-face communication is always preferred**
- **But**
 - large teams have limitations
 - feedback might be a problem

Therefore:

- **Communication channels must be a subject of change**
 - Use different communication channels and switch them over time
 - Locate the project members as close together as possible
 - Change the location
- **Establish a (virtual) communication team**

Communication Media

- **E-Mail and Meetings**
- **Plenary assembly and info meetings**
- **Parties and Food**
- **Wiki-Web**
- **„Floating desks“**

Transparency

- **Communication and Transparency for**
 - developers
 - QA
 - controlling / revision
 - customers
- **Practices and Values:**
 - Shared ownership
 - Shared knowledge
 - Shared skill
 - No head monopolies
 - Honesty

Simplicity

- **KISS**
 - „I built a lot of large systems,
but I never built a complex system“
[Kerth, Meszaros, Doble, OOPSLA2000]
- **Because simple systems**
 - are better maintainable
 - don't require smart programmers
- „Simplicity comes from **conceptual integrity**.“
[David Parnas, XP2002]
- **A system architect is the main step towards conceptual integrity**

Architecture

Architecture

- **is always subject of change**
 - Progress means change
 - Don't try to finalize the architecture *before* growing the team
 - Domain teams will formulate the requirements
 - The development level of the retrospectives enable to speak with one voice
- **is influenced by team size**
 - Avoid any kind of bottleneck (technical, structural, organizational)

But:

- You might have to define the architecture as finalized

Refactoring

- **Missing skills**
 - They don't know how, why, when and where to refactor
 - However: unit tests as safety net
- **Code size and compile time matters**
 - Example:
 - Change an interface specification in a fundamental class
- **Refactoring with global impact has to be done outside business hours**
 - Weekend
 - Holidays

Development Cycles

- **The larger the team, the shorter the development cycles**
 - Stay always at the lower end of suggested development cycles:
 - 1 to 4 week iterations
 - 1 to 3 month releases
- **Fixed time-box**
- **At the end:**
 - result (success or reasons for failure)
 - retrospective
- **Consider the first cycle as an investment**

Retrospectives

- **Retrospectives cover 2 levels:**
 - Development level: Evolution of the software
 - Meta level: Inspection of the process
- **Attendance**
 - Team leads
 - Floating
- **Format**
 - Less open discussion, more team work
 - Facilitation techniques
 - Change Format
- **Follow-up on other channels**
 - Also non-attendees can participate
 - E.g. Wiki

Integration

- **Single Integration Point**
 - Beware: This is a planned bottleneck
- **Continuous integration is a problem**
 - time limitations for integrating one team after the other
- **Fixed integration dates (also nightly builds) are a problem**
 - multiple integration makes error detection more difficult

Therefore:

- **Integration/Build Team**
 - provides scripts for simplifying building and integrating the system
 - maintains progress
 - measures project status
 - collects changes

Outsourcing

- **Make sure External Teams...**
 - integrate their own with the whole project's development cycle
 - accept and embrace change
 - in requirements or in architecture
 - communicate open and honestly about the status of their development
- **This will be supported by:**
 - A change in the contract
 - Instead of passing them the requirements, you might pass them the tests
 - Invite them to the retrospectives
 - Ask them to sit permanently or occasionally on-site

Customer Involvement

Defined single customer is rare, more typical are:

- **Large invisible customer base**
 - typical for standard software
- **Community of customers**
 - often not homogenous, but competitive
 - no accepted representative

Therefore:

- „**Customer On-site Office**“
- **Customer surrogate**

“[...] designing for a single customer is the most effective way to satisfy a broad audience.”

[Alan Cooper in The Inmates Are Running the Asylum]

Organization Structure

- **Large companies are often departmental structured**
 - Organization assumes linear development
 - Established cross-cut teams
 - MaP, QA, revision, controlling
 - Tend to control rather than to support
 - Are rarely accepted / respected by the project members
- **Therefore:**
 - Integrate them early-on
 - They should consider themselves as service-providers

Certification

„In general, certification leads to paralysis. Certification only records the status quo. Innovations don't evolve by sticking to something. Innovations evolve by breaking rules.“

[Monika Bobzien, management consultant and TQM ISO 9000 auditor]

But:

- **A certified process also allows change**
- **The process could be adapted to serve the project**
 - projects are never alike
- **CMM doesn't contradict an agile process**
 - management vs. development methodology

Lessons Learned

- **Communication is the key**
 - Continuous feedback
- **Stability means death**
- **Common sense**
 - Use your brain (and feel/smell the problems)
- **Break rules**
- **Continuous Integration is the biggest problem**

Many Thanks!



Nicolai M. Josuttis
agile@josuttis.com
www.josuttis.com



Jutta Eckstein
agile@jeckstein.com
www.jeckstein.com