
Best Practices for Geographically Distributed Teams or what enterprises can learn from OpenSource

October, 15th 2008

Rainer Heinold

Agenda:

- Why do teams fail?
- What can enterprises learn from OpenSource communities?
 - Infrastructure
 - Process
 - Interoperation
- What else to consider?

Why do teams fail?

- Unsuitable infrastructure and tools
 - Can't communicate
 - Hard to exchange data
- Wrong process
 - Works fine for a co-located team
- Requirements are not communicated well
 - Used the wrong version
 - Language issues
 - Results did not get properly tested
- ...

Why do teams REALLY fail?

- Most of the reasons on the previous slide are perceived or pretended issues
 - „We can't ...“ often is „We don't want ...“
- If you group the real reasons they fall in one of the following categories:
 - Technical
 - Political
 - Psychological
 - Cultural
 - Budgetary
- Technical reasons make less than 5%, the majority of failures are caused by political, cultural and psychological issues (fear is #1!)
 - The fear to become replaceable and transparent

What can enterprises learn from open source communities?

- Successful OpenSource communities share common patterns:
 - Openness of information
 - Centralized and easy to access infrastructure
 - Shared responsibility
 - „Meritocracy of peers“
 - Simple but enforced processes and rules
 - High visibility and support through the infrastructure
- Communication is one of the first areas that should be changed
 - Change point-to-point exchange immediately to centralized mailing lists
 - Archives make it easier for new team members to get up to speed
 - Traceability is a nice side effect
 - Link Subversion commits to a mailing list

- It should be easy for team members to get access all relevant information
 - Group data around classification; this allows you to get rid of path based authorization
- Single vs. Multiple repositories
 - Shared components should be in a separate repository (responsible team has read/write, all others have read permissions)
 - Multiple repositories seem to be more resilient against change
 - Single repositories have often a complex path-based authorization scheme
 - If teams follow the OpenSource model, a single repository allows an easier traceability
- Openness goes well beyond code
 - For example, weekly snapshot builds, mailing lists, documentation ...

- Golden Rule-of-thumb

LESS IS MORE

- Don't replicate based on an assumption
 - Every replica increases the Total Cost of Ownership (TCO)
 - Administration (server and network is getting more complicated)
 - If you need to work offline for a certain period of time, check if SVK can be an option
- Seriously evaluate if servers need to be hidden (accessible just by parts of the team)
 - Read-only access doesn't hurt and fosters reusability
- Replication seriously hampers in most cases openness

- Re-use „As Is“ is a myth (at least in 99.99% of all cases)
 - Changes have to be possible
- So what about patches?!?
 - Good idea, but often the biggest waste of resources
 - A patch does not provide new features, it just allows the re-use
 - It often leads to a fork – 2 very similar objects are maintained by 2 teams
- Establish a clear gateway like Open Source projects
 - The contributor can offer the patch to the core team responsible for the piece of software
 - If the patch is accepted, the responsibility for maintaining the change is transferred to the core team
 - However, a new release is always done by the core team; it can contain contributed new features

Branching strategies

- Stable trunk
- Unstable trunk
- Agile

- Projects should allow 2 reporting lines
 - The official hierarchy is based on contract and job description
 - The unofficial hierarchy is based on contribution and knowledge in the project
 - The unofficial hierarchy is earned, not granted
 - It exists anyway in most projects (all developers have a certain area of expertise)
 - Task should be distributed based on the unofficial hierarchy
 - Use a simple model (like rookie, contributor, committer)
 - This helps to deal with social or political issues – it does not solve them!

Simple and enforced working models

- Open Source projects use simple process flows, but these can not be circumvented
 - For example, only committers can commit to certain branches
 - Culture of merciness – errors are allowed, if you learn from them
- Limit the lock base working model to certain types of objects
 - Binary data that is hard to merge
 - Documents
- The checkout/edit/merge paradigm causes less merge conflicts than you might perceive

Centralized vs. Distributed

- Subversion has been designed for a centralized approach
 - Even in a highly distributed team a single repository should have the latest status; it allows everybody to get access to all changes
- Utilities like SVK allow a disconnected and distributed usage model
 - This is not meant to be the default usage model
 - Work a well defined scope for a limited period of time
- A pure distributed approach can be implemented with Subversion and Add-ons
 - Based on our experience it works only
 - Smaller teams
 - Independent work packages
 - Clear scope (for example in a pure Agile process model)

What else should be considered?

- Review the SLA and contract with your ISP
 - The internet becomes a crucial part of your infrastructure
- Replicate only if necessary
 - Don't assume or make judgements based on previous experiences
 - Subversion has been designed to work in a highly distributed context
 - Create a resilient infrastructure
- Keep potential changes in mind
 - Rule of thumb – if a change can be expected within the next 6 months, you can plan with it.

Thank you!

Questions?!