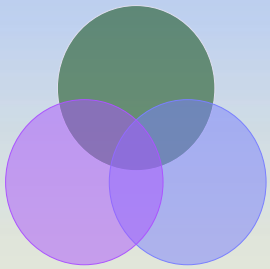
Agenda

- SCM in Context
- Agile v “Traditional” SCM
- Codeline-Related Patterns

The Context

- SCM is Part of the Puzzle:
  - Architecture
  - Software Configuration Management
  - Culture/Organization



The Goal: Working software that delivers value.

Traditional View of SCM

- Configuration Identification
- Configuration Control
- Status Accounting
- Audit & Review
- Build Management
- Process Management, etc

Effective SCM

- Who?
- What?
- When?
- Where?
- Why?
- How?

Think about the entire value chain.

What is *Agile SCM?*

- *Individuals and Interactions* over Processes and Tools
  - SCM Tools should support the way that you work, not the other way around
- *Working Software* over Comprehensive Documentation
  - SCM can automate development policies & processes: Executable Knowledge over Documented Knowledge
- *Customer Collaboration* over Contract Negotiation
  - SCM should facilitate communication among stakeholders and help manage expectations
- *Responding to Change* over Following a Plan
  - SCM is about facilitating change, not preventing it

What Agile SCM is Not

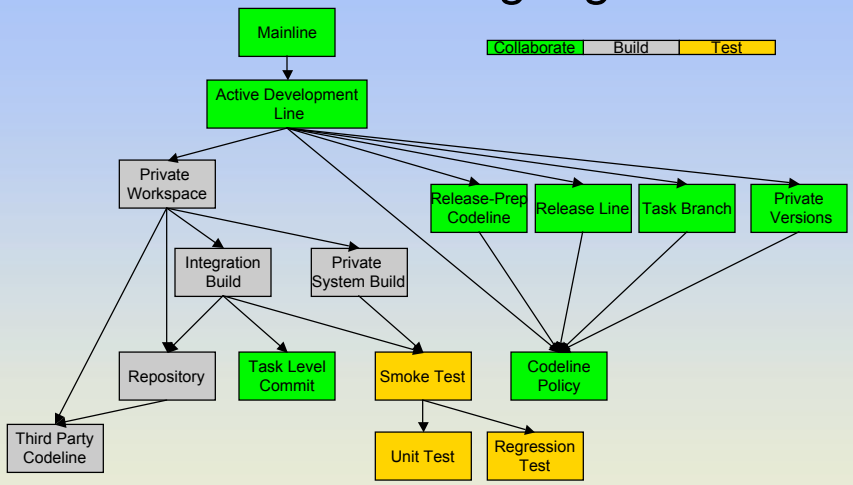
- Lack of process
- Chaos
- Lack of control

Agile SCM is about having an Effective SCM process that helps get work done.

Creating an Agile SCM Environment

- Decide on a goal
- Choose an appropriate Codeline Structure and set up the related policy
- Create a process to set up workspaces
  - Private
  - Integration
- Build & Deploy is an Iteration 0 Story
- Integrate frequently at all levels
  - Developer Workspace
  - Integration Build
- Deploy frequently
- Test

The SCM Pattern Language



Active Development Line

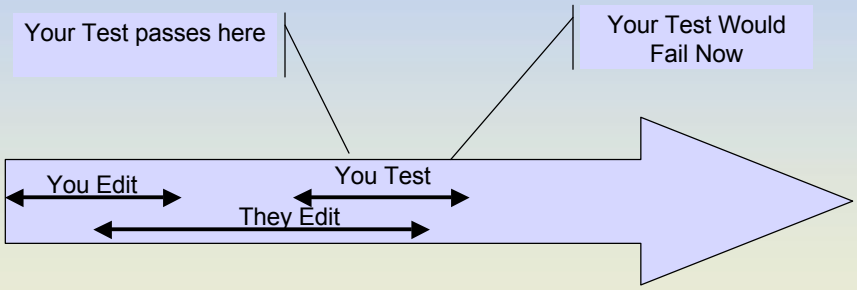
- You are developing on a *Mainline*.
- **How do you keep a rapidly evolving codeline stable enough to be useful (but not impede progress)?**

Active Development Line (Forces & Tradeoffs)

- A Mainline is a synchronization point.
- More frequent check-ins are good.
- A bad check-in affects everyone.
- If testing takes too long: Fewer check-ins:
  - Human Nature
  - Time
- Fewer check-ins slow a project's pulse.

Phase Shift

- Long running tests increase the likelihood of phase shift.

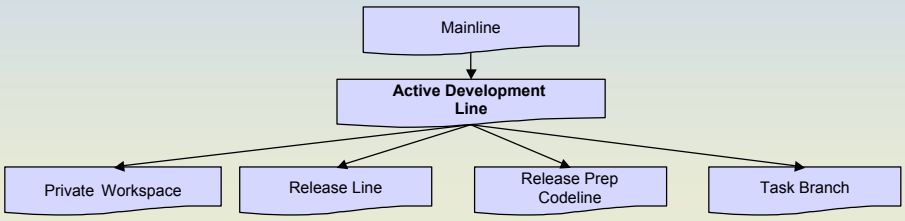


Active Development Line (Solution)

- Use an *Active Development Line*.
- Have check-in policies suitable for a “good enough” codeline.

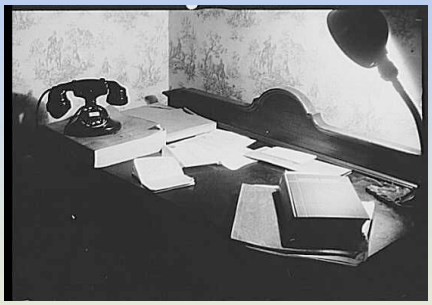
Active Development Line (Unresolved)

- Doing development: *Private Workspace*
- Keeping the codeline stable: *Smoke Test*
- Managing maintenance versions: *Release Line*
- Dealing with potentially tricky changes: *Task Branch*
- Avoiding code freeze: *Release Prep Codeline*



Private Workspace

- You want to support an *Active Development Line.*
- **How do you keep current with a dynamic codeline and also make progress without being distracted by your environment changing from beneath you?**



Private Workspace (Forces & Tradeoffs)

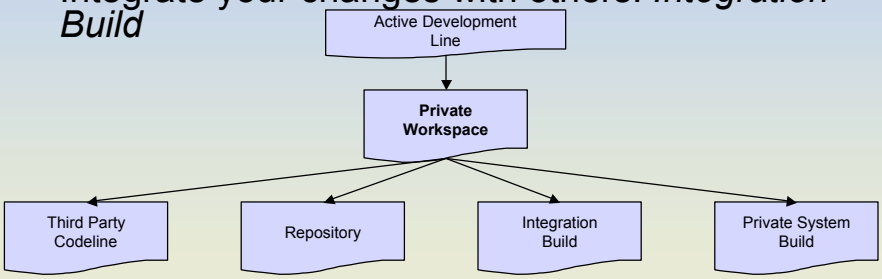
- Frequent integration avoids working with old code.
- People work in discrete steps: Integration can never be “continuous.”
- Sometimes you need different code.
- Too much isolation makes life difficult for all.

Private Workspace (Solution)

- Create a *Private Workspace* that contains everything you need to build a working system. You control when you get updates.
- Before integrating your changes:
  - Update
  - Build
  - Test

Private Workspace (Unresolved)

- Populate the workspace: *Repository*
- Manage external code: *Third Party Codeline*
- Build and test your code: *Private System Build*
- Integrate your changes with others: *Integration Build*



Release Line

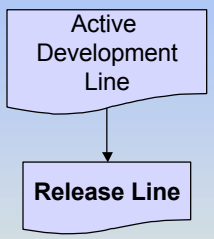
- You want to maintain an *Active Development Line*.
- **How do you do maintenance on a released version without interfering with current work?**

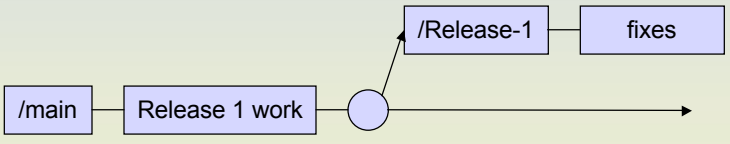
Release Line (Forces & Tradeoffs)

- A codeline for a released version needs a *Codeline Policy* that enforces stability.
- Day-to-day development will move too slowly if you are trying to always be ready to ship.

Release Line (Solution)

- Split maintenance/release activity from the *Active Development Line* and into a *Release Line*.
- Allow the line to progress on its own for fixes.





Task Branch

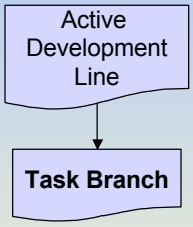
- Some tasks have intermediate steps that would disrupt an *Active Development Line*.
- **How can your team make multiple, long-term, overlapping changes to a codeline without compromising its integrity?**

Task Branch (Forces & Tradeoffs)

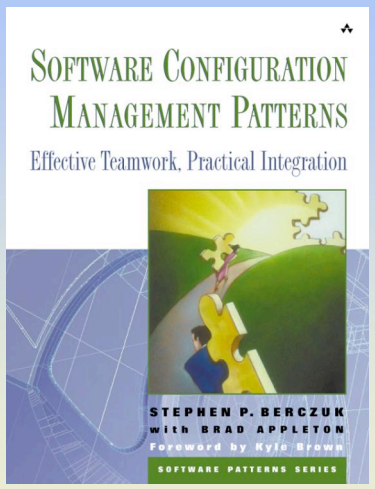
- Version Management is a communication mechanism.
- Sometimes only part of a team is working on a task.
- Some changes have many steps.
- Branching has overhead.

Task Branch (Solution)

- Create a *Task Branch* off of the *Mainline* for each activity that has significant changes for a codeline.
- Integrate this codeline back into the *Mainline* when done.
- Be sure to integrate changes from the *Mainline* into this codeline as you go.



Learn More



- Pub Nov 2002 By Addison-Wesley Professional.
- ISBN: 0201741172
- Sites:
  - www.scmpatterns.com
  - www.berczuk.com
  - www.cmcrossroads.com