SoftwareEntwicklung ▸ Beratung ▸ Schulung

Project Organization vs. Build- and Configuration Management

Web Site:
www.soebes.com
Blog:
blog.soebes.com
Email:
info@soebes.com

Dipl.Ing.(FH) Karl Heinz Marbaise

Agenda

1. Initialization
2. Specification
3. Implementation
4. Project Evolving
5. Organization
6. Building Software
7. Dependencies
8. Build Systems
9. Feature Implementation
10. Conclusion

1. Initialization

- Efforts, requirements and resources will be planned
  - Implementation efforts
  - Test efforts
  - etc.

1. Initialization

- Human resources
  - How many developers?
  - How many testers?
  - How many Q&A?
  - How many people for operations?
  - etc.

1. Initialization

- Hardware resources
  - How many computers for developers?
  - How many computers for testers?
  - How many Q&A computers?
  - How many computers for operations?
  - etc.

2. Specification

–Many documents will be produced

- Architecture Documents
- Design Documents
- Test Plans
- etc.

3. Implementation

–The implementation phase starts with a „small“ number of developers.

–The „small“ number of developers needs something to build the software, cause they want to do some tests.

•They start with their IDE for that purpose.

4. Project Evolves

–Over the time more and more developers join the team.

–Now some kind of organization on the source code level must be introduced.

5. Organization

–The requirement by the developers is to work as independent as possible without any impediments from others.

•The idea of „Modules“ will be born.

•The source tree will be organized based on that.

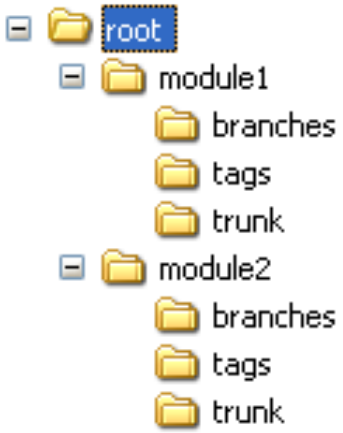
5. Organization

–Requirements will be organized based on the modules

•Every Module-Team will implement the given requirements

5. Organization

The following Subversion structure will be the result:

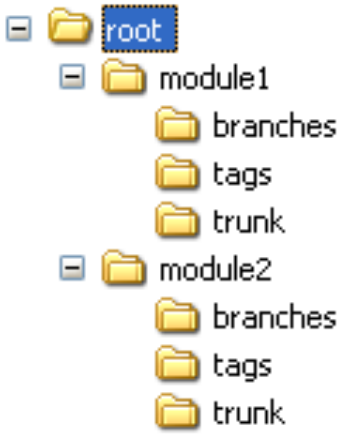


6. Building Software

How to build a defined state of the software based on the given structure?

–Build a particular tag of the modules?

–Which tags should be used?



6. Building Software

–What you need is a solution to describe a state of your software system.

6. Building Software

–The following describes a state of your application:

•Module1

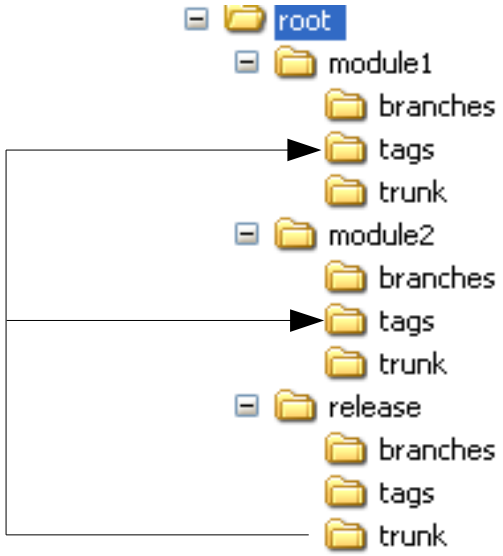
–Release 1.2.3

•Module2

–Release 1.1.6

6. Building Software Solution I

Introduce some description of what a system is defined by a supplemental file, let us call it „release.xml“ and store it into release/trunk



6. Building Software Solution I

Pro's

–No change of the current structure needed.

–Simply to integrate modules which results in simply changing the contents of the release.xml file.

6. Building Software Solution I

Con's

–No commit on system level possible only per module.

–No branching on system level possible only per module.

–No merging on system level possible only per module.

–release.xml file is hand maintained.

6. Building Software Solution I

Con's

–Checkout of the whole system only possible by using supplemental tools (may be self implemented).

–An integration is not really „integrated“, cause no merge has been done.

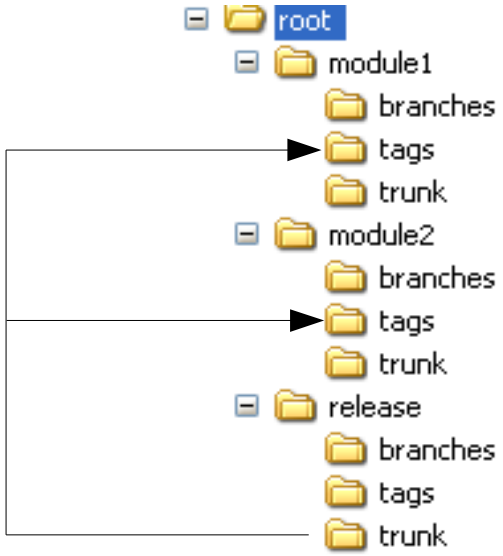
6. Building Software Solution I

Con's

–No support of existing tools for such approach.

6. Building Software Solution II

Introduce some description of what a system is defined of via svn:externals:



6. Building Software Solution II

Pro's

–No change of the current structure needed.

–Checkout of the whole project simple.

–Tagging can be done via svn commands.

6. Building Software Solution II

Con's

–No commit on system level possible only per module (limitations of svn:externals).

–No branching/merging on system level possible only per module.

–svn:externals are hand maintained.

6. Building Software Solution II

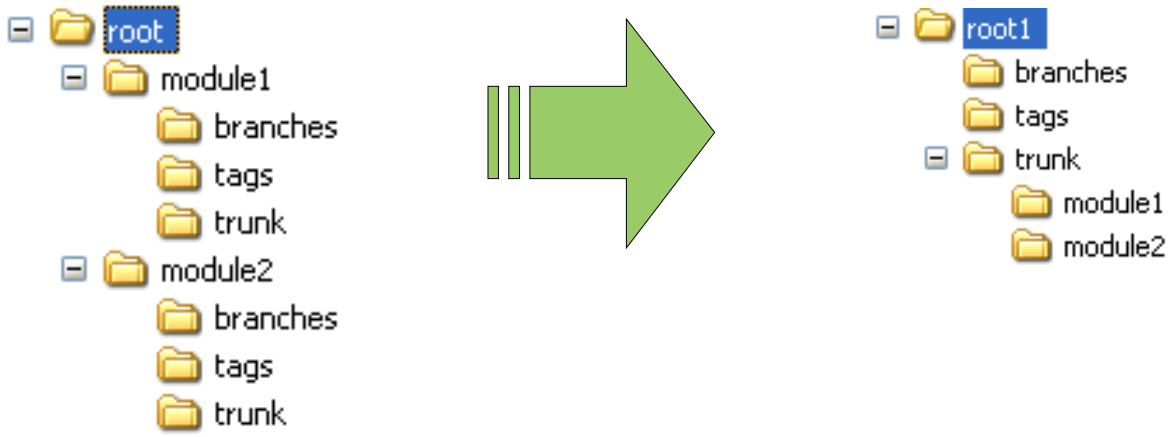
Con's

–No comparison between releases on system level possible only per module.

–No use of svn log --use-merge-history possible

6. Building Software Solution III

Change the structure of your project according to “best practice” which is recommended in Subversion:



6. Building Software Solution III

Con's

–Changing of the structure is needed.

–You have to define a branching strategy:

•Integration lines, Release lines etc.

•Branching on system and module base etc.

6. Building Software Solution III

Pro's

–Checking out of the whole project is simple.

–Tagging/Branching/Merging can be done via svn commands.

–svn log --use-merge-history can be used.

7. Dependencies

What about the dependencies between the modules?

–Module1 depends on Module2?

7. Dependencies

Solution I, II and III:

–Couldn't handle this, because dependencies between the modules are not handled by the „release.xml“ file nor by svn:externals property.

•Note: A dependency could be a pre-build, provided or runtime dependency.

7. Dependencies

Solution I, II and III:

–Result:

•You have to introduce a new file in the modules like „dependency.xml“ which describes the pre-build dependencies for each module.

8. Build Systems

What kind of build system do you need?

–Maven, Gradle, Ant (+Ivy),

–CMake, SCons, Make...

–Self made ?

8. Build Systems

Maven 2/3

–Build Life cycle

–Dependency Management on module level

–Deployment, Repositories, Versioning system

–Release cycle

–Site generation / Reporting

8. Build Systems

Gradle

–Build Life cycle

–Dependency Management on module level (Maven)

–Deployment, Repositories, Versioning system (?)

–Release cycle (?)

8. Build Systems

Ant (+Ivy)

–Dependency Management on module level

•Maven like

–Only target driven

8. Build Systems

CMake, SCons(?), Make...

–No dependency management on module level.

–No deployment

–SCons some kind of repository

9. Feature Implementation

How to implement features in solution I and II?

–In fact not possible only on module level but not on system level.

–The integration is done later via an integration build.

9. Feature Implementation

How to implement features in solution I and II?

–Branching only on module level possible.

–If you have many features in different modules you have a „Big Bang“-Integration or „puzzle-integration“

9. Feature Implementation

How to implement features in solution I and II?

–“Continuous Integration” (CI) NOT possible.

•Only on module level but NOT on system level.

9. Feature Implementation

How to implement features in solution III ?

–Simply create a feature branch to implement it. Later merge into a release/integration line.

–Integration can be done simply by using a merge and a following build.

10. Conclusion
Solution I, II

–Module based development not feature oriented.

–Not possible to merge on a application level only on module level.

–No view in VCS on the whole system.

10. Conclusion
Solution III

–Feature oriented development simply possible by using branching strategy.

–Tagging/Branching/Merging on application level via SVN.

–A complete view in VCS on the whole system.

On-line Sources I

–[1] Branching strategies

•Subversion Conference 2008

–[2] Maven

•Linux Tag Berlin 2009

–[3] Continous Integration

•Hudson

Questions?

gearconf2010@soebes.com

Thank you for your attention.