CoSpace
Configuration and Change Management Report

1 Introduction

Configuration in a system is a set of documented technical descriptions. For example, in a car building project, a configuration would be stating that it has four wheels underneath. In our case, in a software system, it has the same notion. To maintain both product development and deployment in a project, configurations should be consistent and synchronous across the project. For example, while a tester should be aware of which version to use for a particular package for making his/her tests, a developer should follow the result of the test before making any improvements, and also for the deployment all the configurations should be consistent. Right in that part, configuration and change management come into play.

Configuration management is the process of controlling and tracking the changes in an evolving product. It tells the team how configurations should be performed. It consists of different steps like version control and auditing which we mentioned in the following sections. While configuration management considers how the changes should be applied and tracked, change management deals with changes itself. In which criteria changes should be taken into account, in which states a new change should be applied, and so on so forth.

In the following sections, we first define the main purposes with the management plan, then main specifications and key considerations respectively.

2 Purpose

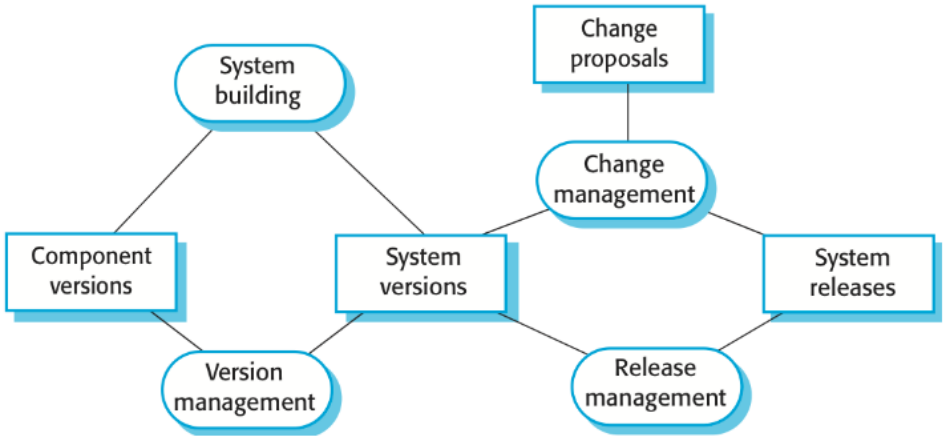
In our project, we are encountering different types of reasons for making changes. Most frequent ones are happening while our software testers catch an error in the system, or when a design revision makes up one's mind. On the other hand, when a new feature is implemented it should be joined with the entire project without affecting other progress. Since these changes happen regularly in our project all the developers should obey some configuration routines. For example in the current state of the project we have encountered almost 150 new changes in the system, this is strong proof of the rate of evolvement.

While a developer applies a change in the database, other developers who work on the server-side should be aware of these changes and take action if needed. To do so each change should be well documented which contains a brief description of the change and the author. Also to prevent redundant shifts, two developers working on the same task, each developer should be assigned specific tasks and they should be documented as well.

In that sense with this document, our main purposes are defining the configuration management routines and rules for the maintainable project development cycle. It also plays an important role in guiding developers who make daily changes in different parts of the project. With such a management plan our other purposes are documenting the changes, version controlling, easy deployment, and risk management when needed, etc.

3 Configuration and Change Management Specifications

Configuration management plans consist of four activities, following subsections first briefly define those activities and then describes how we considered them in our project.



Version Management

Version management is for handling multiple systems and component versions. In our system, we divided the versions based on features. When a new feature is headed to the task queue, the assigned developer opens a new branch named with his/her name and version/feature name and starts to work on that version. Developers also can open new branches from a sub-branch for dividing versions into two or more if that is needed. After ends up with the implementation he/she should make tests on his/her own branch. Finally, after some additional routines that we'll mention in the following sections, it is time for merging the branch with the main development branch. All the developers should merge their branches with the development branch so that each tested version can be combined under a maintainable branch. These were for component versions; for system versions, we are considering the product schedule. Milestones like making a demo to the stakeholders or MVP are determined by our system version decisions. We are keeping our latest system version under the master branch. Since all the component versions merging with each other under the development branch, when a new system version is releasing we are simply merging the development with the release branch.

System Building

This activity considers creating a complete system by combining multiple components. In our system, both server-side and client-side systems are kept separate from each other. For the compilation processes of these systems, we are actively documenting the required steps. When releasing a new system version the corresponding building documents should be updated if it is needed for keeping up-to-date building instructions.

Change Management

Change management is for ensuring the system evolution is a managed process. As mentioned in the project plan document, we are following an iterative development process, Rational Unified Process. In that sense within an iterative process of development, test, and change; project, and configuration managers are regularly taking feedbacks from different stakeholders. Customers might decide a revision, testers might detect an error or software analyst detects an out of scope implementation.

Some factors that we consider for deciding change in our system are:

- The consequences of not making the change
- The benefits of the change
- The costs of making the change
- The product release cycle

Release Management

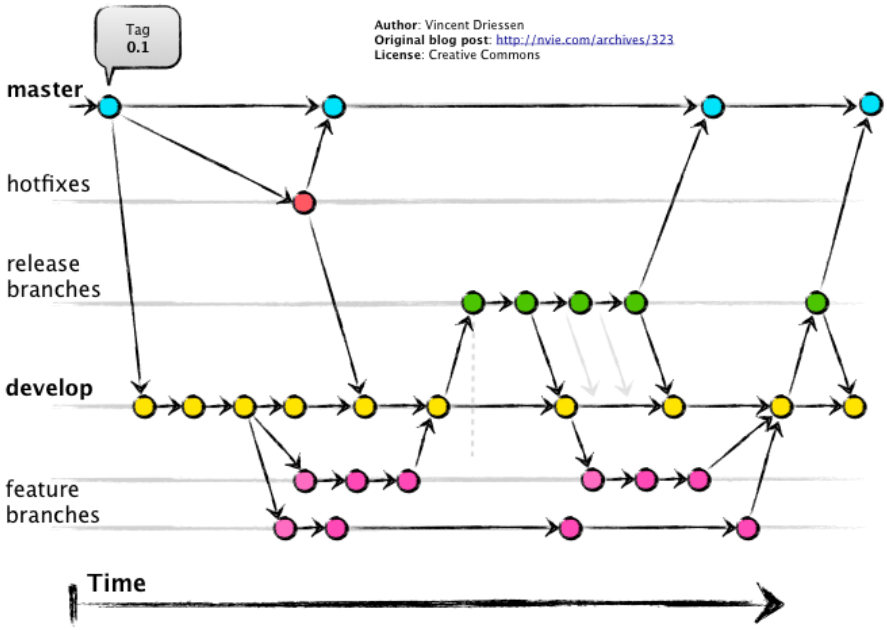
Preparing software for external release and keeping track of the system versions that have been released for customer use. When planning a release we are considering few key factors:

- Demands from the customers.
- Technical quality of the system
- Platform changes

When there is a new release in the schedule there are a couple of things that have to be done. Corresponded code-bases for the release version should be organized with documented building instructions. Moreover, the physical devices which will be used while in the production should be organized appropriately. After the release is done this activity does not come to an end. The release should be tracked so that in the case of any problem it should be reported for fixing until the next release.

4 Key Considerations

As the project daily changes and evolves by the developers, it is hard to keep track of those configuration management activities. At that point configuration management tools come into play. We are using GitHub, a web-based version-control and collaboration platform, for that purpose. While GitHub provides easier version control, we also used Git flow for conforming standardized configuration routines. This flow is as follows:



Create a Branch: A reason for opening a branch may differ, features, bug fixes of releases, but the main convention is the same. Developer opens a branch named his/her name and one or two-word description so that without getting into details each developer can understand necessary information by looking at the name.

Add Commits: Once the branch is opened, developers start adding commits to their corresponding branch. Each commit should be attached with a one-sentence long description while adding to the branch.

Open a Pull Request: Once the commits come to an end and the tests are made, the developer should open a pull request. While sending the pull request he/she also assigns a reviewer for reviewing the changes that are implemented in that branch.

Merge: After the changes are verified by the reviewer and the branch passes the tests now it is finally time for merging the branches.

Milestones and tasks: For keeping track of the tasks, we are using GitHub issues with priority tags and short descriptions.

5 References

Sommerville, Software Engineering, 10 ed., Chapter 25