**Configuration Management**

**Slide 3:**

Software systems always change during development and use. Bugs are discovered and have to be fixed. System requirements change and you have to implement these changes in a new version of the system. New versions of hardware and system platforms become available and you have to adapt your systems to work with them. Competitors introduce new features in their system that you have to match. As changes are made to the software, a new version of a system is created. Most systems, therefore, can be thought of as a set of versions, each of which has to be maintained and managed.

If you do not have effective CM procedures in place, you may waste effort modifying the wrong version of a system, deliver the wrong version of a system to customers, or forget where the software source code for a particular version of the system or component is stored.

CM is useful for individual projects as it is easy for the developer to forget what changes have been made. It is also essential for team projects where developers are working at the same time on a software system. CM helps developers not interfere with each other’s work.

**Slide 5:**

CM policies and processes define how to record and process proposed system changes, how to decide what system components to change, how to manage different versions of the system and its components, and how to distribute change to customers. Configuration management tools are used to keep track of change proposals, store versions of system components, build systems from these components, and track the releases of system versions to customers.

**Slide 6:**

Different companies talk about the same concepts using different terms. There are historical reasons for this.

**Slide 8:**

Change is a fact of life for large software systems. To ensure that the changes are applied to the system in a controlled way, you need a set of tool-supported, change management processes.

**Slide 9:**

CR – change request;

The change management process is initiated when a ‘customer’ completes and submits a change request describing the change required to the system. This could be a bug report, where the symptoms of the bug are described, or a request for additional functionality to be added to the system.

**Slide 19:**

To support version management, you should always use version management tools (also known as version control tools or source code control systems). These tools identify, store, and control access to different versions of components.

**Slide 21:**

Delta – a list of differences; list of changed lines used to re-create the most recent component version.

Since the most recent version of a component will be used, most systems store that version in full. The deltas then define how to re-create earlier system versions.

**Slide 22:**

Most software development is a team activity, so situations often arise where different team members work on the same component at the same time.

**Slide 28:**

Building is a complex process, which is potentially error-prone, as there may be three different system platforms involved.

**Slide 31:**

Build tools available and a build system may provide some or all of the above features. The build script is a definition of the system to be built. It includes information about components and their dependencies, and the versions of tools used to compile and link the system.

**Slide 32:**

Agile methods recommend that very frequent system builds should be carried out with automated testing (sometimes called smoke tests) to discover software problems. Frequent builds may be part of continuous integration process.

**Slide 35:**

Agile methods recommend that very frequent system builds should be carried out with automated testing (sometimes called smoke tests) to discover software problems. Frequent builds may be part of continuous integration process.

**Slide 37:**

The argument for continuous integration is that it allows problems caused by the interactions between different developers to be discovered and repaired as soon as possible.