Plugin packs and concrete versioning

Context

Currently, plugin versions, if unspecified, resolve to the "latest" from the repository, which (depending on the repositories included) can include snapshots. It also means that builds change change without the POM changing.

In addition to the ones the users specify, several plugins with no version attached are implied by the lifecycle, making it quite cumbersome to specify all the plugins that a build might use in the POM to ensure that the versions are locked down over time.

While this situation is problematic, the alternative of requiring every single plugin version be provided is too cumbersome and verbose a task in many situations - particularly in the case of the implied plugins.

It should also be noted that anything in the Maven installation itself (such as the settings, or in the past the plugin registry) is not an adequate solution - the POM must be the definitive reference for how to build the software and changing Maven installations should have no effect.

Out of Scope

The original discussion also touched on the following, which are related but separate issues:

- locking down versions at release time where they were not specified (a more general problem, as it includes not only RELEASE/LATEST, but ranges too).
- separation of "declaration" from "instantiation" for a POM.

Solution

Required Plugin Versions

After implementation, Maven will require the version in plugin definitions that are bound to the lifecycle in the POM from modelVersion 4.1.0+. Plugins will not be resolved to the latest version (except for the CLI exception listed below).

However, in 4.0.0 modelVersions, Maven will continue to allow the RELEASE as the version for backwards compatibility.

Super POM

The Super POM should declare a default version of the clean plugin as a special case. This will not be the case for the site plugin, or standard packaging plugins.

Plugin Packs

The additional requirement to declare all versions is not an inconvenience when the plugin is already declared - just one additional line.

However, for the implied plugins (jar, compile, resources, etc) - this would involve an undesirable amount of boilerplate. To better facilitate this, we should add the concept of plugin packs.

After the releases of one or more included plugins, a pack release can be made that includes the plugins and their versions. This is achieved through the deployment of a POM project that declares a number of plugins - this has the advantage of having the same syntax, as well as easy cut-and-paste if necessary. For example, the Java plugin pack:

```
<project>
  <groupId>org.apache.maven.plugins.packs</groupId>
  <artifactId>maven-java-plugins</artifactId>
  <version>1.0</version>
  <packaging>pom</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.1</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.2</version>
      </plugin>
      ...
    </plugins>
  </build>
  <reporting>
    <plugins>
      ...
    </plugins>
  </reporting>
</project>
```

To incorporate this into a project:

```
<project>
  ...
  <packaging>jar</packaging>

  <pluginPacks>
    <pluginPack>
      <groupId>org.apache.maven.plugins.packs</groupId>
      <artifactId>maven-java-plugins</artifactId>
      <version>1.0</version>
    </pluginPack>
  </pluginPacks>
  ...
</project>
```

Only the build and reporting plugins sections of the POM will be incorporated into the project - not the plugin management or other sections such as configuration. In addition, if multiple plugin packs are declared, they should be applied sequentially, overriding any prior.

Note: this facility could later be addressed by mixins in the POM.

Plugin Management

The plugin management section, when declaring a version, must always win over all plugin packs.

A version will not be required in the plugin management section (which can still be used to apply configuration to an already-defined plugin, including those called from the CLI).

CLI

When running a goal from the command line, it should seek the plugin from those defined in the POM (including their versions). If the plugin is not defined in the project at all (or in the case that there is no project, such as with archetype:create), at present the current rules will be retained. It should be a recommended best practice to declare any plugins you use from the command line in the POM that are not once-off goals.

A future enhancement may be to be able to (and if so, require) plugin versions to be declared in the Maven settings files.

Optional: Decouple Packaging Lifecycle Definition

The definition of lifecycles and packaging types currently occurs in the maven-core's components.xml, with others defined in plugins via extensions. With the possible exception of the POM packaging, these can all be moved into the plugins that define the main package goal.

Given this, the user will be required to declare the plugin that contains the packaging (including it's version). Like the earlier requirement, this must not apply to POM's with the previous model version.

Note that a plugin may still contain one or more packaging.

The lifecycle still does not declare plugin versions - this is to be defined in the plugin packs.

Optional: Additional Tooling

It would be beneficial to write a plugin that can look at the current versions in a project, and suggest possible updates available in the repository (and apply them in an interactive fashion). This may suit the maven-pom-plugin.

Votes

+1	
0	
-1	jason (in its current form)