# MODEL DRIVEN SOFTWARE DEVELOPMENT

## LECTURE : 11

# VERSIONS

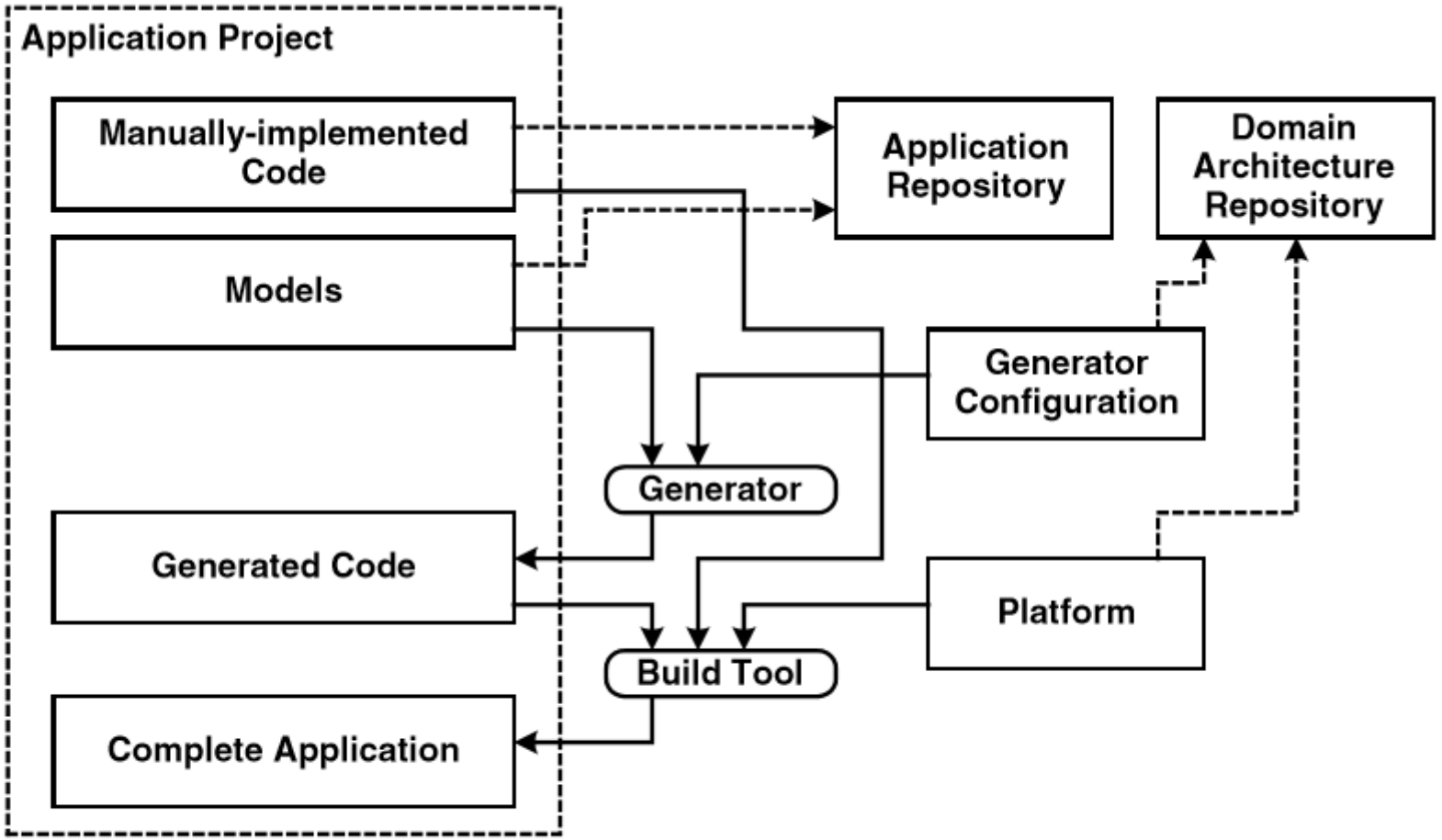
- Resources are versioned in order to capture a snapshot of the current state of the resources at one specific point in time
- Resources are versioned by tagging them with a version label
- When a resource is versioned, it means that a non-modifiable copy of it can be retrieved from the repository
- Versioning a project saves the line up of all resource versions in the project

# VERSIONS -MDSD

- In an MDSD project, the following aspects must be managed or versioned:
  - The (generic) generation tools. Currently, these tools are often still in development themselves – because of this, it makes sense to have them under version control.
  - The generator configuration, the generative part of the domain architecture. This includes the DSL/profile definition, metamodel, templates, and transformations.
  - The non-generative part of the domain architecture, the MDSD platform.
  - The application itself: models, specifications, and (in most cases) manually-developed code
- The generated code ideally is not versioned, because it is reproduce-able from the model at any given time
- Of course, this idea can be applied sensibly only if the manually-created and the generated code are separated structurally in the file system
  - This is one reason why we value this separation

# STRUCTURE OF APPLICATION PROJECTS -MDSD

- The models of the application, as well as the manually-created code, are located in the application repository.
- The generator creates the generated code, including configuration files and so on, supported by the generator configuration, located in the domain architecture repository.
- The application is next generated with the help of the build script (in most cases this is also generated). This step uses the manually-created code of the application and the platform, the latter taken from the domain architecture repository.




Projects, artifacts, and repositories

# VERSION MANAGEMENT IN MIXED FILES

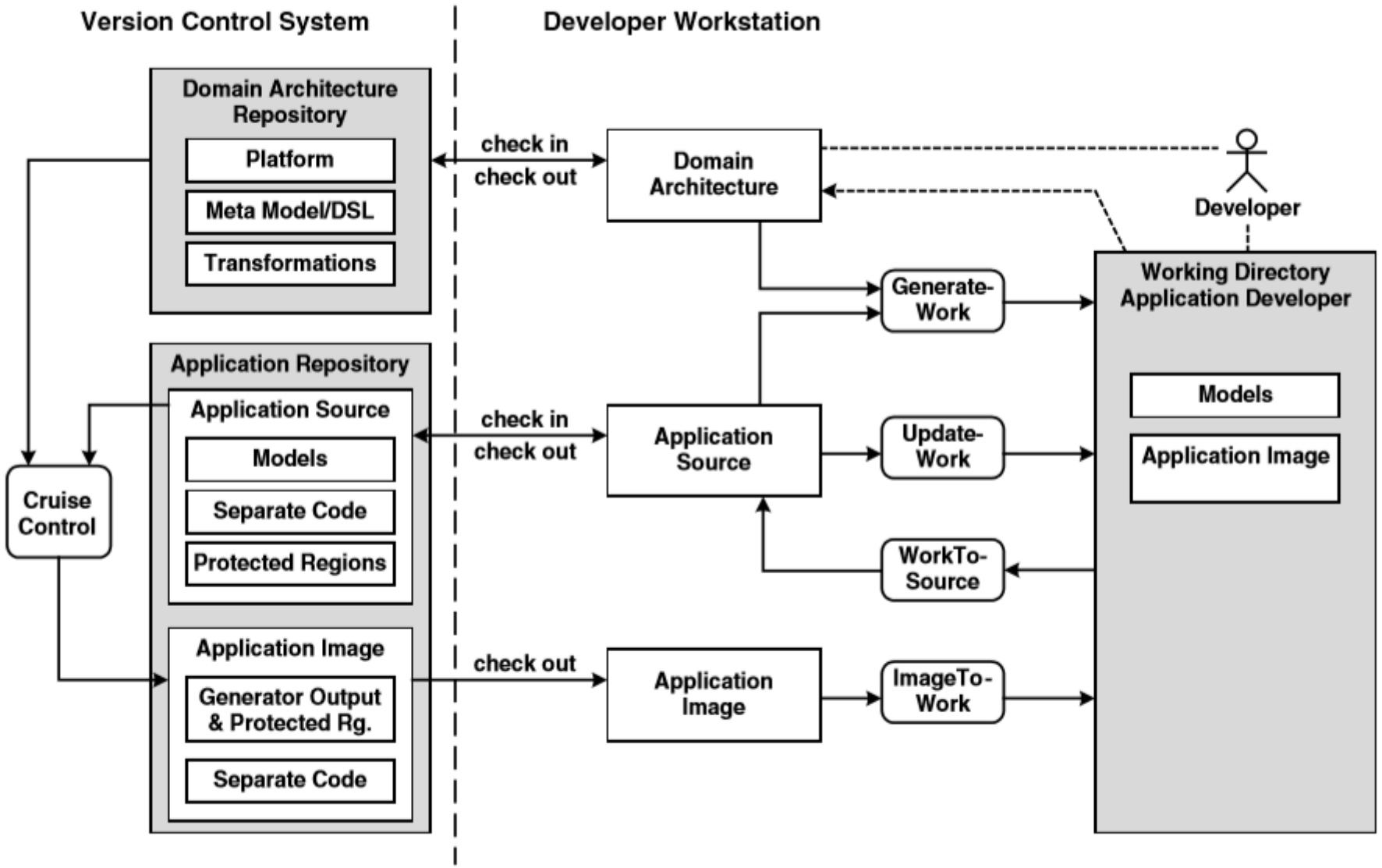
- Protected regions that can be defined in the generator templates.
- As a consequence, markings are created in the generated code that are used by the generator during iterative regeneration, to find and preserve the manually-created code contained in it.
- These markings are hidden syntactically from the compiler or interpreter of the target language by labeling them as comments.
- Obviously the use of such protected regions leads to files in which generated and non-generated code is mixed.

# VERSION MANAGEMENT IN MIXED FILES

- The problem here is that these files can usually only be versioned as a whole.
- This results in redundant code being checked in, because the generated code (without the contents of the protected regions) is, after all, not source – the source would be the (partial) model from which the code was generated.
- These redundancies can lead to inconsistencies during development in a team.
- The inconsistencies will become increasingly problematical as the team grows larger
- This step uses the manually-created code of the application and the platform, the latter taken from the domain architecture repository.

# VERSION MANAGEMENT IN MIXED FILES

- For example developer A changes something in the model or the architecture while developer B is programming domain logic contained in a file whose generated portion is affected by A's changes.
- This situation can cause a problem or conflict when checking in either A or B, because the data is no longer up-to-date.
- The reason for this is a redundancy in the repository.
- The objective must be to avoid this redundancy and for example to manage the contents of the protected regions in isolation
- On the other hand, an isolated protected region is usually of little help to the application developer, because they need the context of the generated code as guidance and to execute the compile-run cycle.


Version Control System
Developer Workstation
Domain Architecture Repository
Platform
Meta Model/DSL
Transformations
check in
check out
Domain Architecture
Developer
Generate-Work
Working Directory Application Developer
Application Repository
Application Source
Models
Separate Code
Protected Regions
Cruise Control
check in
check out
Application Source
Update-Work
Models
Application Image
WorkTo-Source
Application Image
Generator Output & Protected Rg.
Separate Code
check out
Application Image
ImageTo-Work

# VERSION MANAGEMENT IN MIXED FILES

- The real (that is, non-generated) application sources are managed in the application repository
- Manually-created, application-specific source code that is not organized in protected regions is labeled separate code here
- In addition, one can automatically create an application image, that contains the generated as well as the manually-created code
- The actual application development takes place in a separate work directory.
- The synchronization between this work directory and the (local) directories that are also controlled during versioning takes place for example via scripts and Ant tasks

# SCRIPTS OF SYNCHROZATION

**GenerateWork**

- This script applies the domain architecture on the application model and integrates the checked-out handwritten code from separate files and protected regions with generated code through a generator run.
- In other words, it produces a complete source code image of the application in the developer's working directory from scratch

**UpdateWork**

- Only the handwritten code is updated

**WorkToSource**

- Changes to the working directory are reduced to changes of real sources (generated code is not source)
- and made available to the version management system, so that check-in can take place in a way that is compatible with the structures in the version control system

**ImageToWork**

- The result of this script is not very different from GenerateWork
- It refreshes the complete source code image in the working directory from the version control system