

Software Management in a Model-Drive Software Factory

The Experiences of a Release Manager

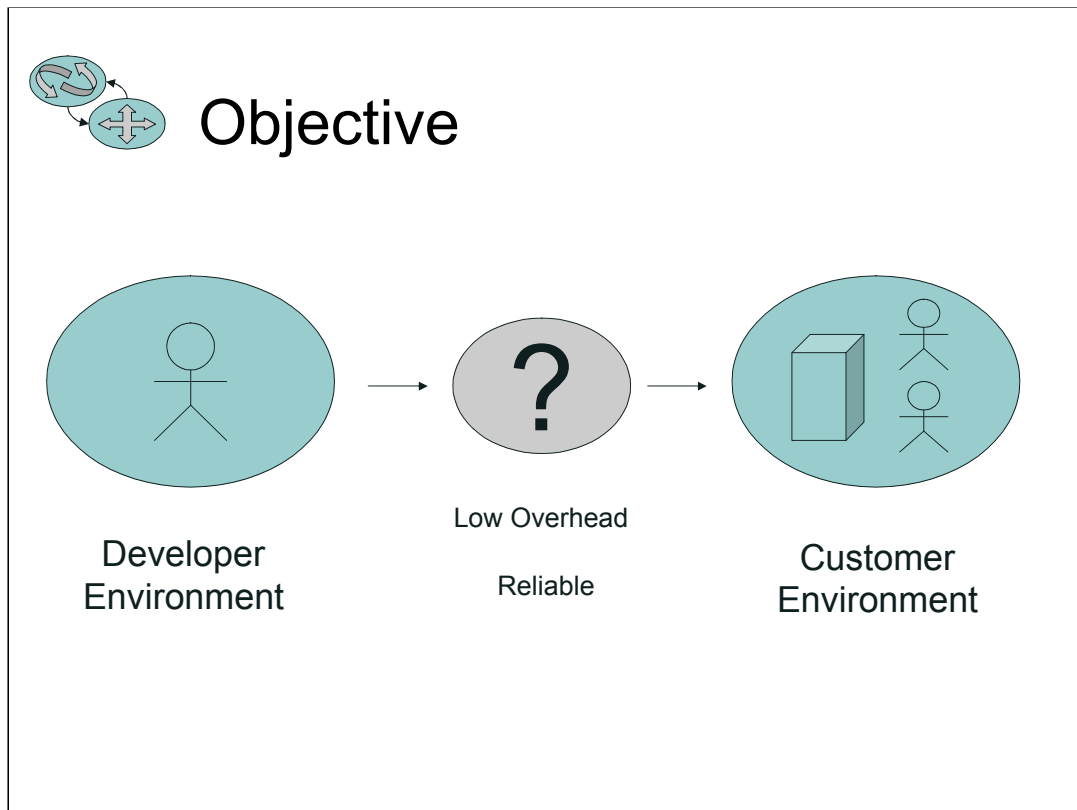
In order to support its business plans, Bay Technologies has created a software factory around a Model Driven Architect (MDA) approach to generate our custom workflow products.

The factory itself is the combination of the MDA technology tightly coupled with automated software management processes. The automated processes are based in part on the ideas of Agile Development and in part from our company's experience in performing SOE projects.

There appears to be substantial literature describing the philosophy and approach to MDA but little about the process environment needed to support it. The benefits of MDA are focused primarily on the creation of source code while the software management process is in effect untouched by MDA.

Our experience in building the factory shows that real business benefits are possible from reducing the delivery overhead by linking the MDA and software management process. MDA allows not just software patterns to be codified, but also the software management patterns. The model not only describes the code, but how to build, version, source control, install, and upgrade the resulting system.

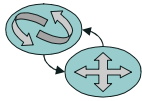
This experience paper is intends to cover how taking a holistic approach to the software factory improved delivery capability and quality in areas that have not been considered by MDA.



The focus of this discussion is on the Software Management process within the MDA environment.

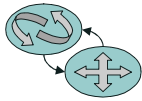
Much has been said about using Model Driven Development (MDD) to develop software and all of the benefits that accrue to the analyst and developer. But all this neglects the practicalities of managing the software. The overhead of software development is managing it, collecting source, compiling, packaging, releasing, installing, upgrading, and supporting.

The core concern of Release Management is getting a system from the development environment into the customer environment.



Principles

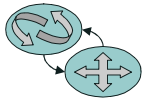
- 10 Minute Principle
 - 10 minutes on an activity repeated 1000 times is 20 working days
- 10 Minute Corollary
 - Trying to shave 10 minutes by bypassing a process will loose you 2 days



Principles

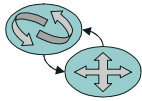
Networking *should* be boring.
If you're having fun, something is
going horribly wrong.

This applies to software development!



Our Environment

- Known, stable problem domain
- Single core application architecture
- Mature models and transformations
- Multiple products
- Customer specific variations of product



The Journey

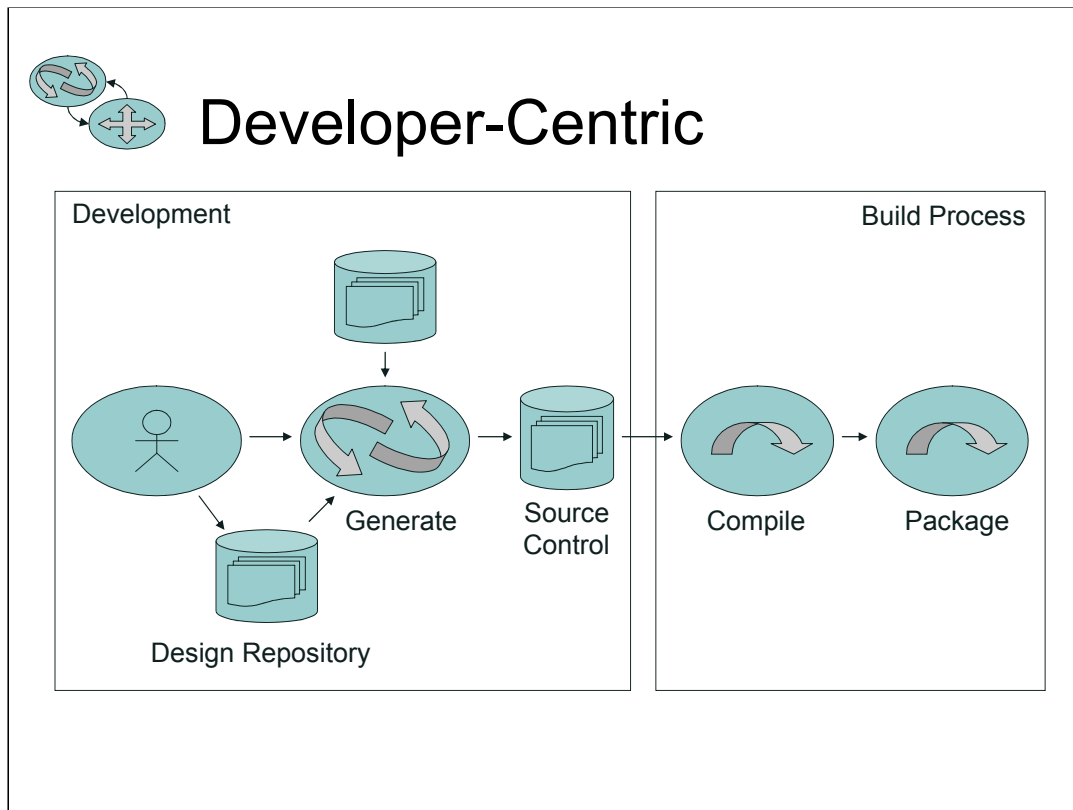
Developer Centric

- Standard IDE Tools
- Point tools to deploy
- Developer compiles
- Developer collection of release manifest
- Manual install and update.

Factory Centric

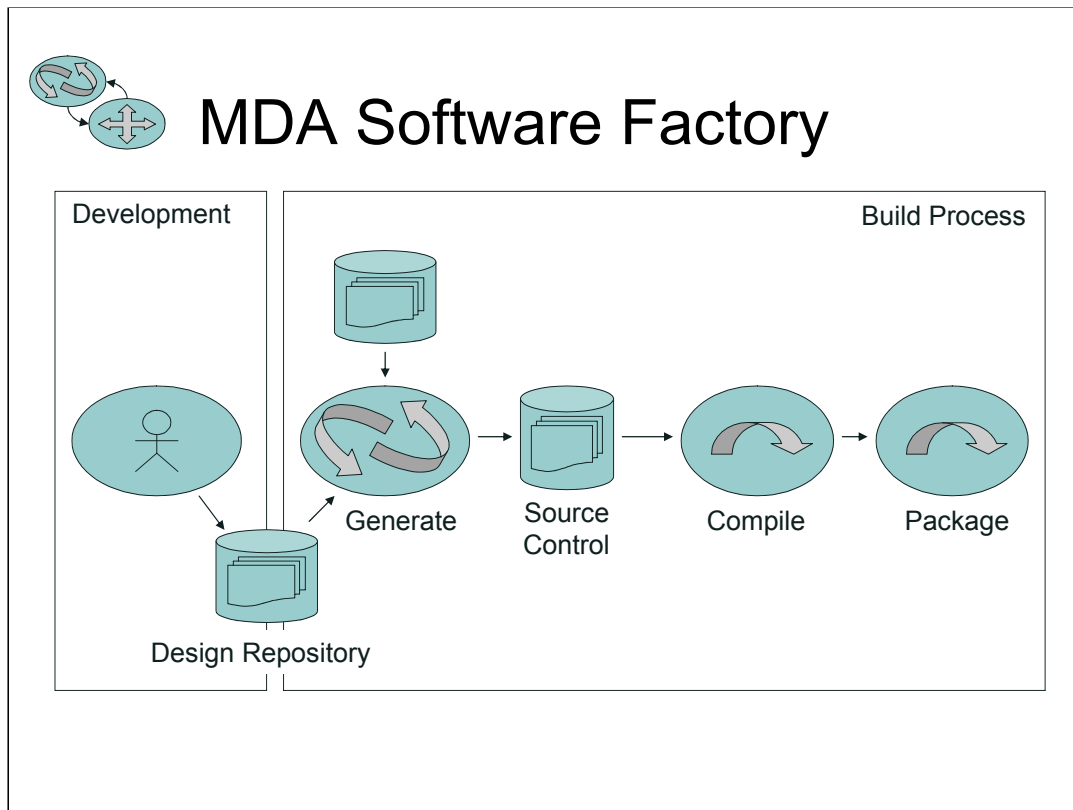
- Model Driven Architecture
- Build server and build process
- Automated release
- PHD Install and Upgrade.

The evolution from a Developer – Centric model to a Factory centric model

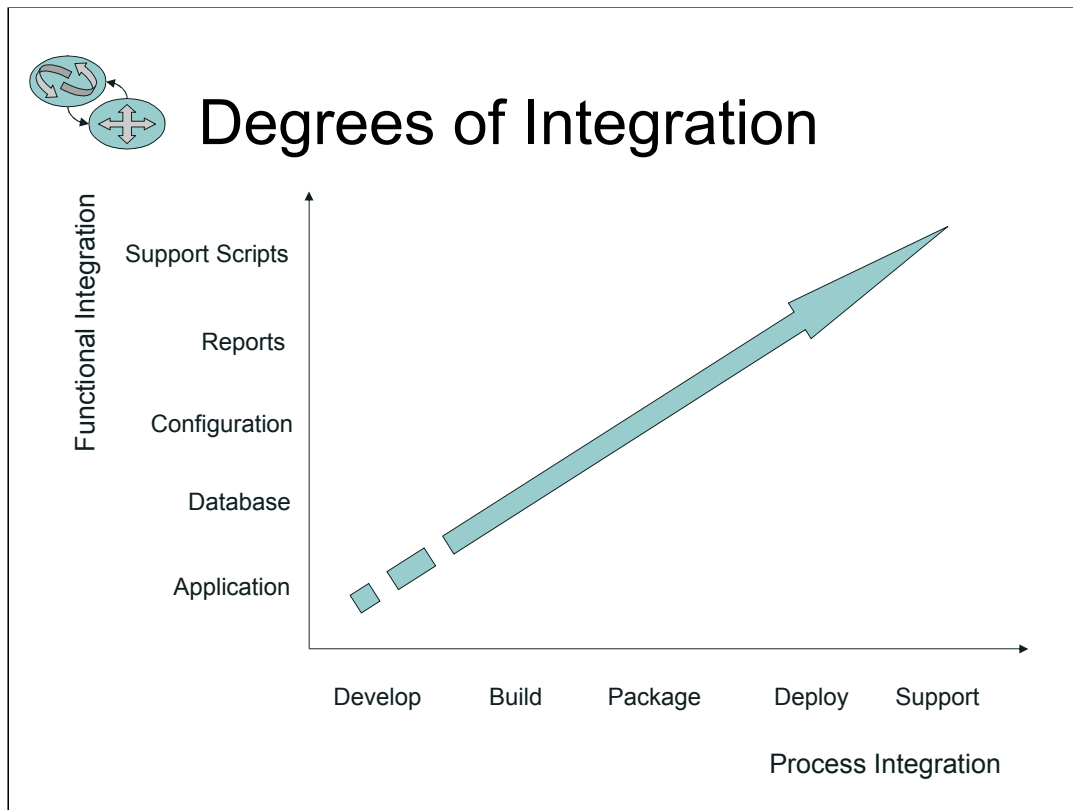


In a developer centric process the developer is in full control of the artefacts that make up the source of the application. This assumption is core to most development methodologies be they agile or the more traditional waterfall process. Indeed even Agile Model Driven Development is essentially developer centric.

The term “developer centric” is used to denote the level at which the developer exercises control over the software management.



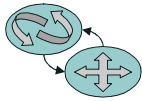
The factory centric process moves primary control over the source artefacts to the factory build process. Developers still work with the MDA tools to define the model details. Developers can even generate, compile and run software locally - > but none of these artefacts save the model details are input to the build process via the Design Repository. The artefacts that come under source control are generated and managed by the factory.



Integration is the alignment of design detail between areas of concern. For instance in software terms it is the alignment of the various software interfaces between objects. Or the alignment of Data Access classes and database stored procedures.

But in a wider functional sense it is the alignment of design details between all parts of the system. Integration may also be along a process dimension such as the creation of compiler commands or release manifests from the application model definition.

The high value quadrant is greatest degree of both functional and process integration.



Database Example

Database Model

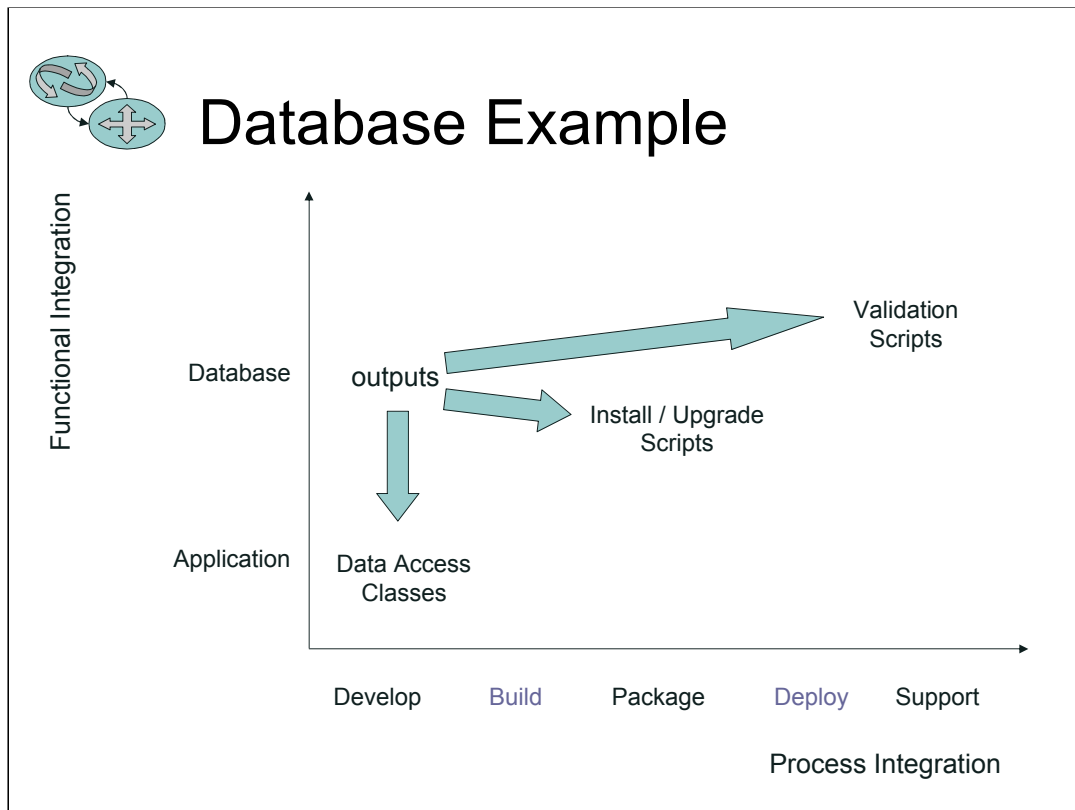
- o Data access classes feed workflow model
- o SQL scripts to install package
- o Update process controlled by build version
- o Validation scripts used for support

Initially the database model created the data access classes to be used by the workflow application and the SQL Scripts to create the database objects. The installation package relied on a single concatenated installation SQL script.

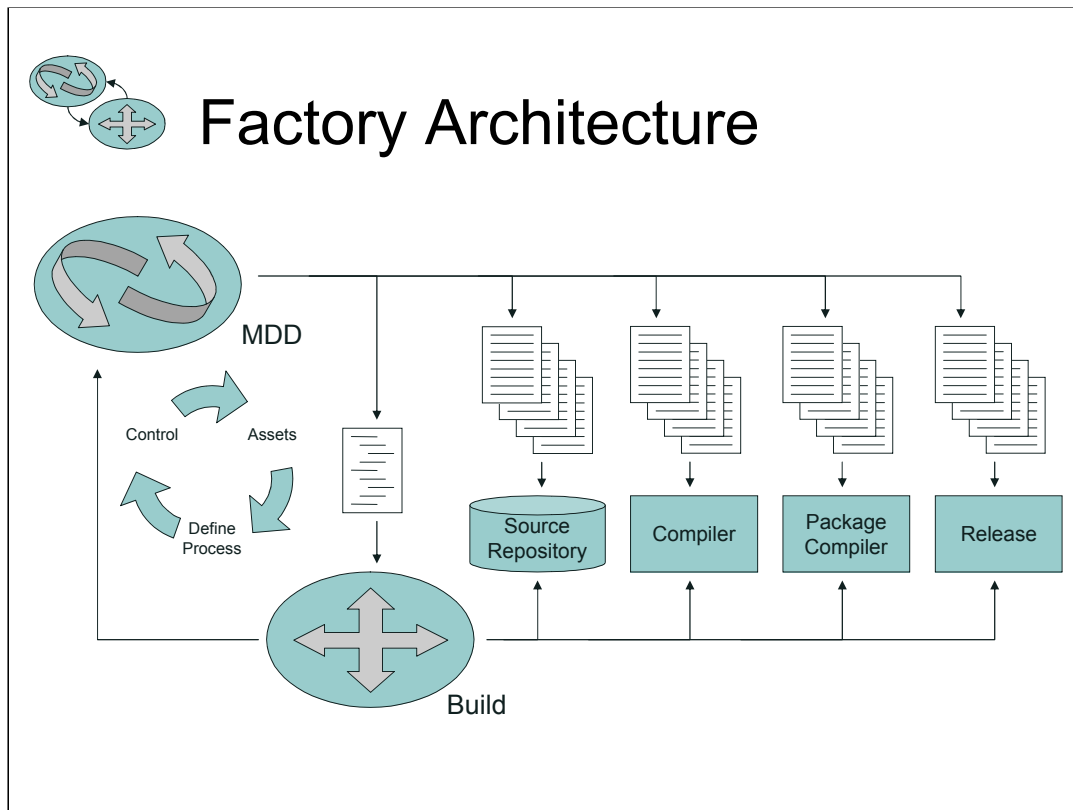
However these scripts did not support upgrading an existing database. After the initial release further updates were assembled and managed by hand. Any consistency created by the model was lost.

A database install or upgrade process was designed to be part of the install package. This process relied on the build to write version information into a schema update SQL Script. This allowed ensured that only the correct updates were applied during the installation. The database model was refactored to provide the SQL Scripts required by the installer.

To ensure that the inconsistencies in the target database were detected and corrected the database model also produces a schema validation script which will report on any schema variations in the target database from the definition.

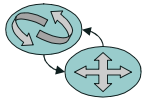


This demonstrates the process integration in the development, build and packaging processes, and with the validation scripts through to support.

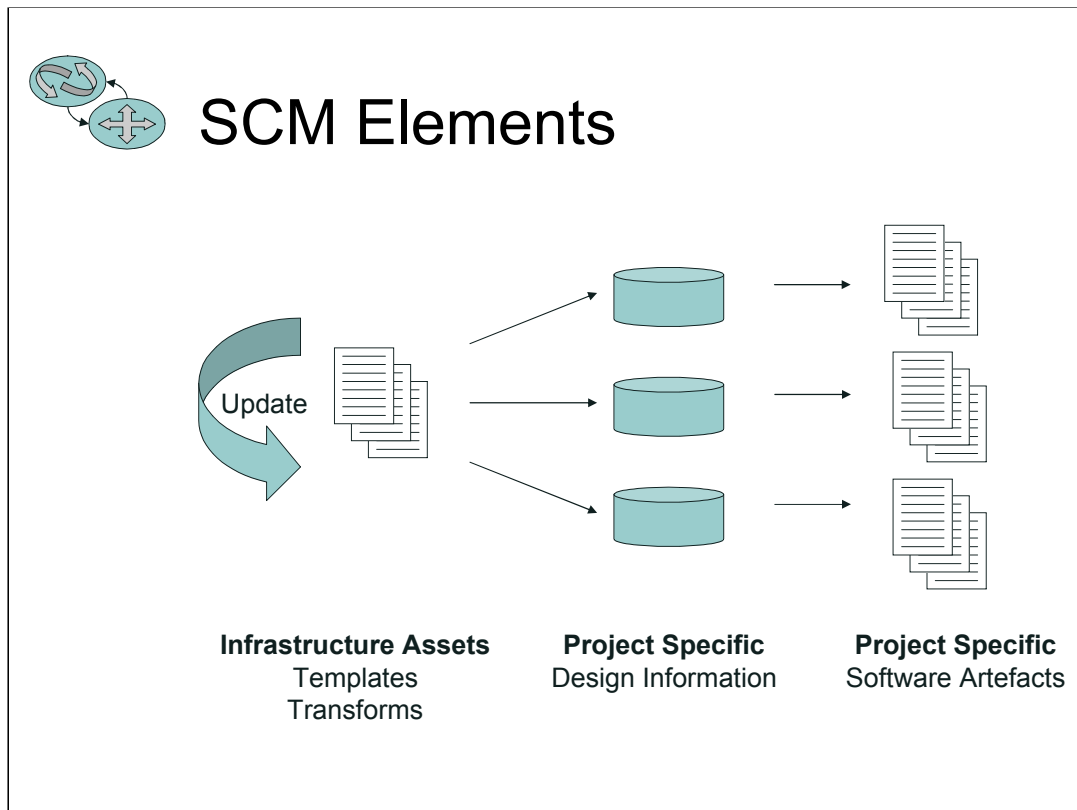


The MDD Tool is used to create assets that are to be consumed, by developers, by the build process, by other models, by the compiler, by the packager. The Build process coordinates and drives the different tools in the chain. The core build process itself relies on scripts from the MDA Tool to drive the different stages of the build.

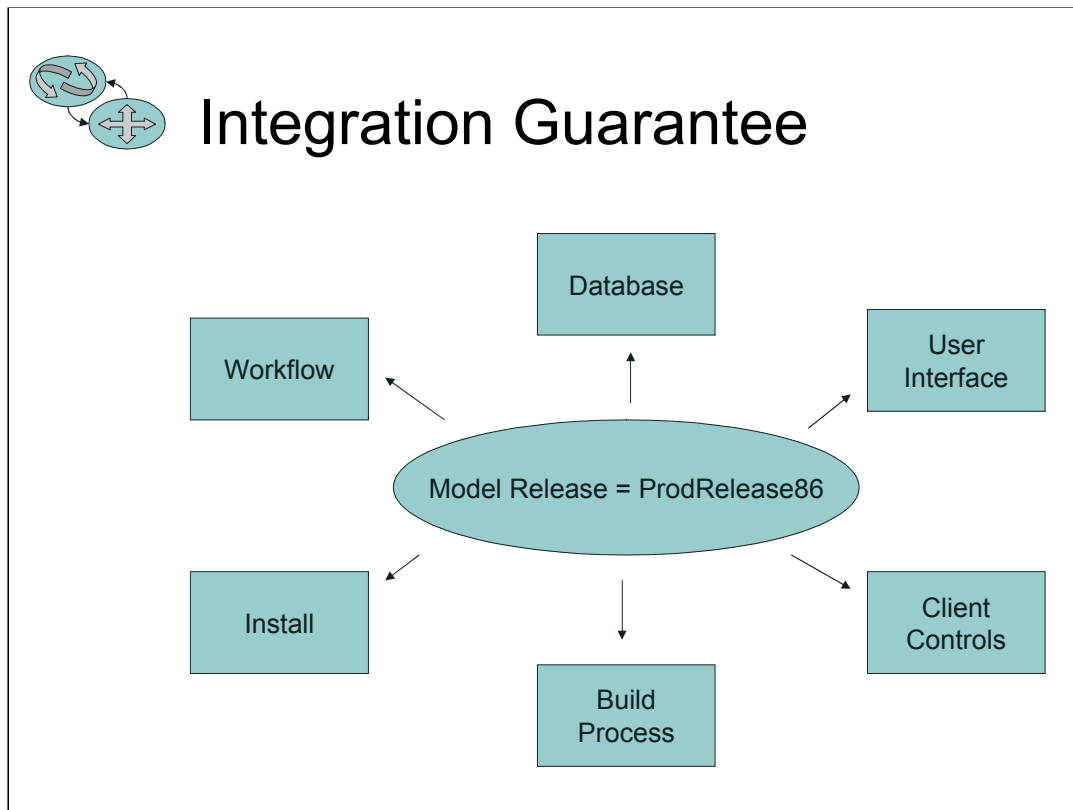
By generating artefacts for the various tools as well as controlling scripts for these tools a level of the build control is embedded into the model MDA process. It is this feedback process that gives us the ability to apply consistent process across our entire product suite.



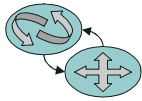
Software Configuration Management



The data associated with the modelling tool can be divided into two categories – a set of abstract components, templates and transformations that we call the *Infrastructure* and the data associated with the application under development. The infrastructure represents those artefacts created by the Model Architect. Changes to the infrastructure are propagated to all products.



We apply a development – release cycle over this model infrastructure, because stability in the infrastructure is fundamental to stability in the product development. A critical attribute for each release of the model Infrastructure is the *Integration Guarantee*: all artefacts from this release will integrate in both functionality and process. Hence the install package will correctly deploy the database scripts and reports, among other things.



Configuration Options

Build Server - Microsoft Internet Explorer provided by Co-operative Solutions

Address: http://hamburga/build/buildconfig.asp

*** MINCOR4 ***

Build Site Project

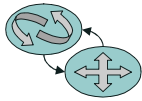
Config Settings

Model Version	ABSv4-PRDRelease85A	ABSv4-PRDRelease85A
Current Release ID	1.20	1.20
Current Build Number	1332	
Build Mode	skip	skip
Egen Version	Egen_2_5_1_00	Egen_2_5_1_00
Egen Database	EGEN2	EGEN2
WIX Version		Wix-2.0.4415.0
Email notification		

Save Settings

Done Local intranet

For developers the area of concern in source management becomes a high level view of which release of the infrastructure is required for the release of the product under development.

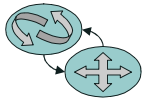


Example - Installation

- Hand crafted
- Changes applied to each product
- Changes need to be in step with model version of product
- Modelled
- Changes applied once to model release
- Automatically in step with model version of product

The installation package was one of the last components to be modelled. Prior to this each install definition was kept in step with the requirements of the application individually. Many of the changes needed to be kept in step with the revision of the infrastructure version. Projects varied as to which version of infrastructure they were on, depending on their release requirements.

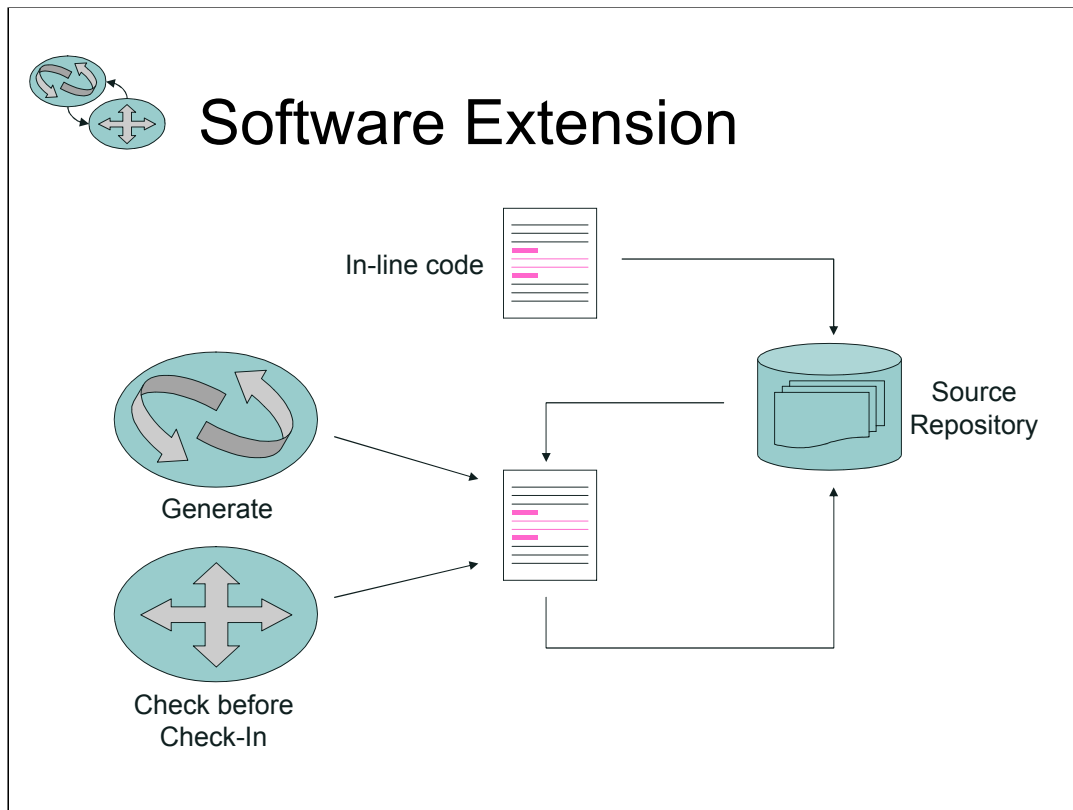
By modelling the installation these requirements and changes can be codified in the model templates. As projects are updated to a level of infrastructure they inherit all of these changes automatically.



Software Extension

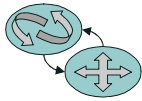
- In-line code fragments
- Overwrite files
- Side by side files
- Binary content (images)

In the practical implementation developers do have some direct influence over source code. The Model tool employed allows for fragments of in-line code to be added generated files. The build itself can override generated code on a file by file basis. Also additional classes and components can be added to the application to extend the applications beyond the boundary of the model. These activities however represent only a small proportion of the application, although they often represent a significant proportion of the software management overhead.



Where code is introduced into the application there are a number of built in checks to provide safe guards against common problems that we have experienced, such as the generation process removing In-line code incorrectly.

Developers only need to become involved in manual source control where there are non-modelled artefacts to be managed for the release. And due to the scope of the modelled application these become a low percentage of the total application files.



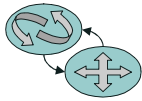
Comparison

	<i>Agile</i>	<i>Model-Driven</i>
People	Highest priority	Not explicit
Process	Medium priority	Not explicit
Technology	Lowest priority	Central
Model	Secondary	Primary
Software	Primary	Secondary

Wegner, H., *Agility in Model Driven Software Development? Implications for Organisation, Process and Architecture.*

This comparison is drawn between Agile processes and Model-Driven development is drawn from Wegner[1]. Although there is a very different focus on the importance of the model neither of these methodologies addresses the software management issue. That is that source artefacts must be managed, stored and that real software product must be compiled, packaged and released to the customer in a way that they can use.

[1] Wegner, H., *Agility in Model Driven Software Development? Implications for Organisation, Process and Architecture.*

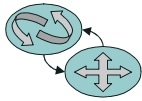


Comparison - Factory

	<i>Model-Driven</i>	<i>Factory</i>
People	Not explicit	Enrichment
Process	Not explicit	Automate
Technology	Centre	Limit set
Model	Primary	Secondary
Software	Secondary	Secondary
Product	Not explicit	Primary

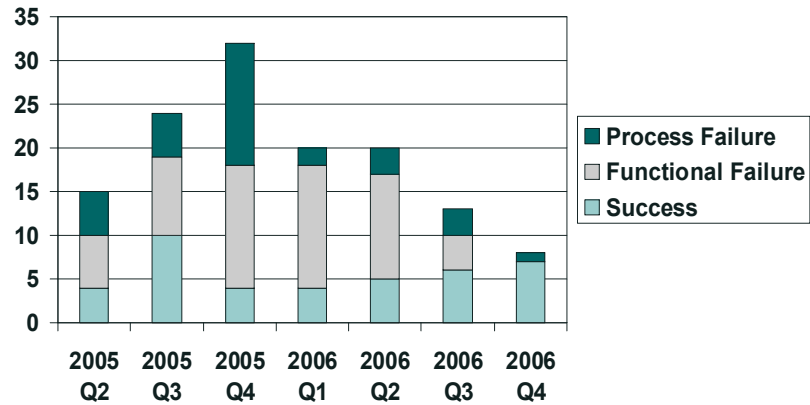
In regards the source code creation process our approach is a Model-Driven approach, the output of the factory is an installable system, not the source representation of an installable system. Hence we do need to consider the generated and manipulated software as a key asset equal to the model. But more important still is the application domain which is rele

We also focus on people in that we need to have them focused on high value tasks such as requirements, and application building, not on the mechanics of managing source or building installation packages.

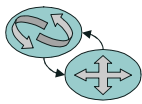


Pay Off - Quality

Number of Releases per Quarter

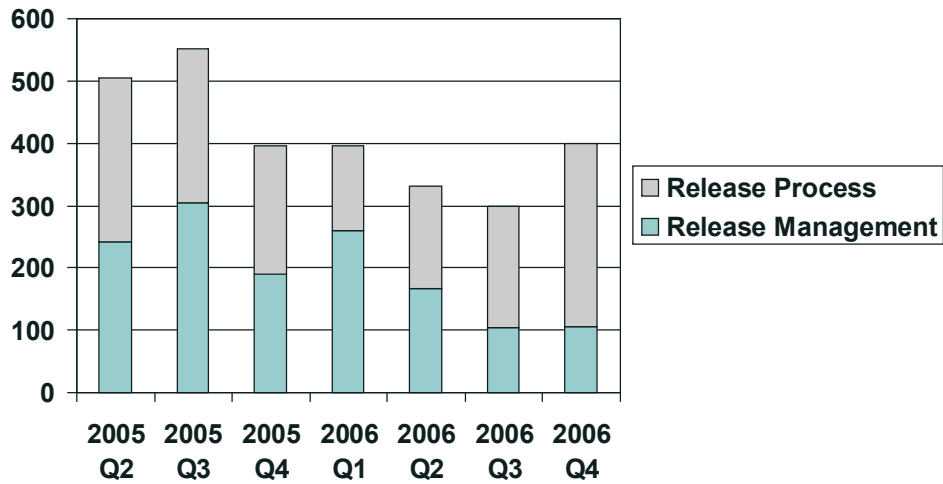


This graph shows all production releases made to customers from the deployment of our first production modelled system in Q2 2005. Successful releases were ones that were accepted and stable in production. Those recorded as functional or process failures required us to re-release or patch the release to achieve acceptance, even if that was a single failure.



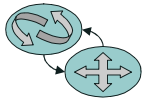
Pay Off - Overheads

Doing Releases v Developing Process



This represents the total hours effort towards either performing releases themselves (Release Management) or developing processes to support and improve the release management (Release Process).

There will always be some manual release components, to the extent of communication with customers, release planning and final inspection of releases.



Pay Off – Enrichment

- How much learning in another release?
- Move focus from files to enhancing
 - Enhancing developer tools
 - Enhancing customer experience
 - Enhancing support capability
- Move focus to fascinating process problems