Staying safe and sound thanks to MDSD

Andreas Kaltenbach / Karsten Thoms
Agenda

• Most Famous Backwards Compatibility Example
• Terminology
• Where Are Lifecycle Aspects Concealed - Four Examples
  - API Incompatibility
  - Schema Derailment
  - Language Changes
  - Continuity of Operations
• Rules of Thumb
Who are we?

Andreas Kaltenbach
InterComponentWare AG
- Software Developer -
- Trainer -

Karsten Thoms
itemis AG
- Software Architect -
- Consultant -
Most Famous Backwards Compatibility Example

JDK 1.0

```java
java.lang.Cloneable

java.lang.Object

#clone() : java.lang.Object
+equals( obj : java.lang.Object ) : boolean
#finalize() : void
+getClass() : java.lang.Class
+hashCode() : int
+notify() : void
+notifyAll() : void
+toString() : String
```
Most Famous Backwards Compatibility Example

JDK 1.0

java.lang.Cloneable

java.lang.Object

@clone(): java.lang.Object
+equals( obj : java.lang.Object ) : boolean
#finalize() : void
+getClass() : java.lang.Class
+hashCode() : int
+notify() : void
+notifyAll() : void
+toString() : String

JDK >1.0

java.lang.Cloneable

java.lang.Object

+clone(): java.lang.Cloneable
+equals( obj : java.lang.Object ) : boolean
#finalize() : void
+getClass() : java.lang.Class
+hashCode() : int
+notify() : void
+notifyAll() : void
+toString() : String
Most Famous Backwards Compatibility Example

JDK 1.0

```java
java.lang.Cloneable
java.lang.Object

#clone(): java.lang.Object()  
+equals( obj : java.lang.Object ) : boolean
#finalize() : void
+getClass(): java.lang.Class
+hashCode(): int
+notify(): void
+notifyAll(): void
+toString(): String
```

JDK >1.0

```java
java.lang.Cloneable
java.lang.Object

#clone(): java.lang.Object()  
+equals( obj : java.lang.Object ) : boolean
#finalize() : void
+getClass(): java.lang.Class
+hashCode(): int
+notify(): void
+notifyAll(): void
+toString(): String
```
Terminology
Backwards Compatibility

- A system is **backwards compatible** if
  - it is complies with interfaces from earlier generation(s) of the system
  - it can process data from earlier generation(s) of the system
  - service consumers are unaffected by version change
Terminology
Update & Upgrade

• Update
  - Action to improve existing software product
  - Feature set is not extended, but improved
    - e.g. closing a security exploit
  - Interfaces and data representation stay intact

• Upgrade
  - new version or addition of an already existing software product
  - provides a different feature set
  - may lead to backwards incompatibility
Terminology
Update & Upgrade

• Update
  - Action to improve existing software product
  - Feature set is not extended, but improved
    - e.g. closing a security exploit
  - Interfaces and data representation stay intact

• Upgrade
  - new version or addition of an already existing software product
  - provides a different feature set
  - may lead to backwards incompatibility
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive
Backwards Compatibility is expensive

\[ N = \frac{x(x-1)}{2} \]
Backwards Compatibility is expensive

\[ N = \frac{x(x-1)}{2} \]
Backwards Compatibility is expensive

\[
N = \frac{x(x-1)}{2}
\]

\[
N = x-1
\]
API Incompatibility
Example: Web Service Interface
API Incompatibility
Example: Web Service Interface

Client A

System

v 1.0

v 2.0
API Incompatibility
Example: Web Service Interface
API Incompatibility
Example: Web Service Interface
API Incompatibility
Example: Web Service Interface
API Incompatibility
Backwards Compatibility Layer

Client A

Client B

Backwards Compatibility Layer

System

v 1.0

v 2.0

v1.0-v2.0
API Incompatibility
MDSD applied

Model (v1.0)

Model (v2.0)

Transformation Code
API Incompatibility
MDSD applied

Model (v1.0) → Diff Model (v1.0-v2.0) → Model (v2.0)

Transformation Code
API Incompatibility
MDSD applied
API Incompatibility
MDSD applied

Model (v1.0) → Diff Model (v1.0-v2.0) → Transformation Code → Change Report

Model (v2.0)
Schema Derailment
Example: Schema Evolution

```
v 1.0

Address
- streetName : String
- streetNumber : String
- zipCode : String
- city : String
- country : String
```
Schema Derailment
Example: Schema Evolution

```
v 1.0
- streetName : String
- streetNumber : String
- zipCode : String
- city : String
- country : String
```

```
v 2.0
- streetName : String
- streetNumber : String
- zipCode : String
- city : String
- country : String
- region : String
```
Schema Derailment
MDSD applied

Model (v1.0)

Model (v2.0)
Schema Derailment
MDSD applied
Schema Derailment
MDSD applied

Model (v1.0) → Diff Model (v1.0-v2.0) → Change Report → Model (v2.0)
Schema Derailment
MDSD applied

Model (v1.0) -> Diff Model (v1.0-v2.0) -> Change Report

Model (v2.0) -> Diff Model (v1.0-v2.0) -> SQL Upgrade Scripts
Schema Derailment
MDSD applied
Language Changes
Language Changes

![Diagram of Models, Metamodel, and Generator]
Language Changes
Language Changes

Diagram showing the relationship between models, metamodels, and generators in version 1 and version 2.
Language Changes

version 2

version 1

Models

Metamodel

Generator

Models

Metamodel

Generator
Language Changes
Language Changes
Language Changes: Anti Corruption Layer
Language Changes: Anti Corruption Layer
Language Changes: Anti Corruption Layer
Language Changes: Anti Corruption Layer

version 2

Models → Metamodel → M2M PMM2 -> IMM2 → Metamodel → Generator

version 1

Models → Metamodel → M2M PMM1 -> IMM1 → Metamodel → Generator
Language Changes: Anti Corruption Layer
Language Changes: Anti Corruption Layer

Diagram showing the flow from Models to Metamodel to M2M (PMM2 -> IMM2) to Metamodel to Generator, with version 2 on the upper right and version 1 on the lower right.
Language Changes: Anti Corruption Layer

version 2

version 1

Models → Metamodel

M2M

PMM2 -> IMM2

Metamodel

M2M

PMM1 -> IMM2

Model Migration Script (manual)

Model Migration Script (generated)

DiffModel

PMM1-PMM2

Models

Metamodel

Model Migration Script (generated)

Model Migration Script (manual)

DiffModel

PMM1-PMM2

version 1

version 2

Models → Metamodel

M2M

PMM1 -> IMM1

Metamodel

M2M

PMM1 -> IMM2

Metamodel

Generator
Continuity of Operations
Example: How customers affect product evolution
Continuity of Operations
Example: How customers affect product evolution
Continuity of Operations
Example: How customers affect product evolution
Continuity of Operations
Example: How customers affect product evolution

Customer 1

Customer n

v 1.0

v 2.0

NG
Continuity of Operations
Example: How customers affect product evolution
Continuity of Operations
Example: How customers affect product evolution
Continuity of Operations
Migration Shock Absorber: Versioned Templates

MDSD Template Program Flow

Model

v 1.0
Continuity of Operations
Migration Shock Absorber: Versioned Templates

MDSD Template Program Flow

Model → v 1.0 → v 2.0 → Model
Continuity of Operations
Migration Shock Absorber: Versioned Templates
Continuity of Operations
Migration Shock Absorber: Versioned Templates
Continuity of Operations
Migration Shock Absorber: Versioned Templates

MDSD Template Program Flow

Model

= Version Decision Point

v 1.0

v 2.0
Continuity of Operations
Migration Shock Absorber: Versioned Templates

MDSD Template Program Flow

Model

= Version Decision Point
Continuity of Operations
Migration Shock Absorber: Versioned Templates

Benefits

• Product could evolve, improve
• Backwards compatibility no more a limiting factor
• No All-Or-Nothing-Option, enables smooth migration
• Easy deprecation of legacy functionality
Rules of Thumb

• Lifecycle aspects are sleeping giants and exist everywhere
• Incremental migration keeps complexity manageable
• Model deltas are as well models again
• Once deployed, it’s legacy
• Backwards compatibility is user-friendly but expensive
• Anti Corruption Layer as best practice
  - provides stable public interface
  - allows for isolated internal refactorings
Thank you for your attention