
I will talk about strategies around migration. Olivier will talk about some of the approaches and tools that he used with his clients.

Surely with the new Eclipse 4 Application Platform, the Eclipse 3.x APIs should be considered deprecated, and new development should use the E4AP?

# No.

This answer may surprise many in this room.

“Much unhappiness has come into
the world because of bewilderment
and things left unsaid.”

Fyodor Dostoyevsky

There’s some confusion out there about what Eclipse 4 means for both new and existing applications.
In this talk, we’ll say the things left unsaid, and dispel the bewilderment.

## Questions

| | |
|---|---|
| Is Eclipse 3.x dead? | YES and NO: Depends what's meant by "Eclipse 3.x"? |
| Should new development always use Eclipse 4? | YES, maybe NO: Depends what's meant by "Eclipse 4"? |
| Should companies rewrite existing Eclipse 3.x applications on Eclipse 4? | NO, maybe YES: The compat layer makes all 3.x apps into Eclipse 4 apps |
| Is it possible to migrate my application? What are the prerequisites? How should I go about it? | Almost certainly. |
| Why did we bother with Eclipse 4? | Good question! |

We are frequently asked questions like the following

## Overview

- What IS Eclipse 4?
- Should new applications target Eclipse 3.x or Eclipse 4?
- 3 Approaches to porting your application or plugin to Eclipse 4
- Other considerations
- Q&A

What are we migrating from and what are we migrating to?

A good opportunity to reconsider how your app is doing things

# Eclipse 3 is Dead! Long Live Eclipse 3!

“Eclipse” can mean: Distributions & Implementations

1. The Eclipse Platform 3.x Runtime: the old binary distributions (Indigo and prior) [DEAD]
2. The Eclipse Platform 4.x Runtime: the new binary distribution shipped since Juno (2012)

Includes the unfortunately named “Compatibility Layer” that implements the Eclipse Workbench APIs

3. The Eclipse 3 Application Platform (aka RCP): IWorkbench, IWorkbenchWindow, IViewPart, IEditorPart, …
4. The Eclipse 4 Application Platform (aka Eclipse 4 or E4): stripped-down workbench model (EMF-based Modelled Workbench, DI, etc.)

API

Big part of confusion arises in confusing names.

APIs vs Implementations

Look & Feels

The implementation may be gone, but the API lives on
Huge amount of code written against the Eclipse 3 Application Platform that will never be rewritten.

# Naming Proposal

Let’s be clear in our naming:

- E3, E3.x, E3AP, Workbench, RCP: Eclipse Platform Workbench APIs (IWorkbench & friends)
- E4, Eclipse 4, E4AP: Eclipse 4 Application Platform APIs

And a side-note:

- e4: an incubator project

## A Short History of Eclipse: How Eclipse 4 came to be

The “e4” effort

- Wide-ranging discussions over all Platform components
- Most changes folded back into respective components
- But Platform/UI changes brought forward massive restructuring

“e4” originally began in 2007 to examine plug-in developer pains from 3.x and see how things could be done better. If we could start over again, what would we do differently?

The Eclipse Platform 3.x is an amazing toolset for building applications, but its UI abstractions, centred around views and editors, are overly constraining for some types of apps. The E4AP is the result of relaxing those constraints

## Gap in Layers of Abstraction

Eclipse 3.x Workbench: Editors, Views, Contributions (quite opinionated)

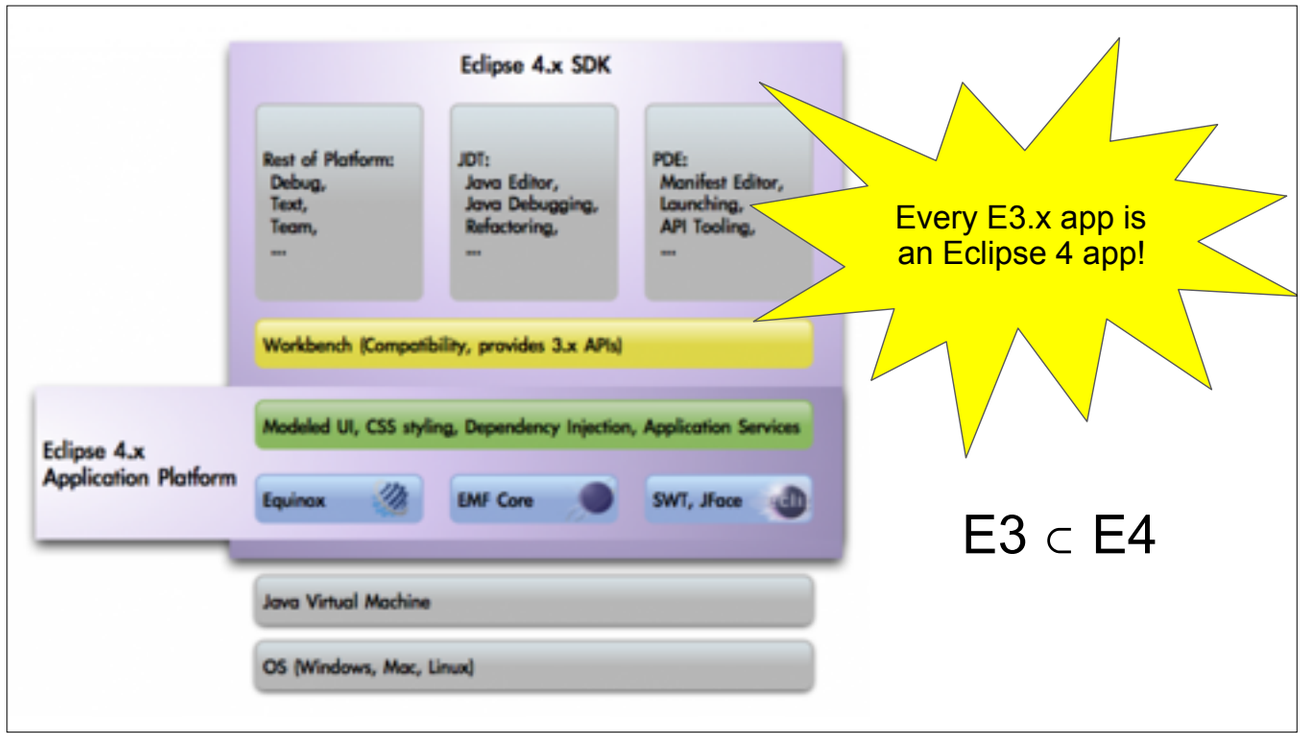
GAP

JFace: Window, Dialog; Viewers, DialogPage

SWT: Shell, Composite

Eclipse 3.x Workbench is pretty opinionated.
Some projects ended up writing their own UI frameworks on top of JFace

So we filled in the gap. E4 provided the mechanisms for apps to escape from the strictures of Eclipse 3.x Workbench




Key takeaway: Every app built on the 3.x Workbench APIs are Eclipse 4 applications.
They can take advantage of the Modelled Workbench, Dependency Injection, contexts, and more.

So what is the relationship between E3 and E4?  I think best thought in terms of separation of policy from mechanism:
3.x specifies a UI Policy
E4 provides the mechanism

## E4 Promotes Separation of Policy & Mechanism

- Mechanism: the parts of a system and possible actions to effect change
- Policy: decisions on what changes can be done to a system

Ideal: A mechanism should be able to support multiple policy specifications

When mixed, system is brittle: hard to change, hard to test

Eclipse RCP: E4 is the mechanism; the Compat Layer implements the Workbench policy

Policy vs Mechanism comes from Operating Systems research world, where separating the policy from mechanism was found to produce more modular systems

Policy: what can be done (what gives rise to being ‘opinionated’)
Mechanism: how it can be accomplished

Classical Examples:

- Payroll: Policy: who gets paid what amount, and when; Mechanism: how that money is transferred
- Memory Management: Policy: FIFO, LRU; Mechanism: managing page lists, shuttling pages to storage

E4

- Mechanism: parts, stacks; Policy: when can a part be moved between a stack?

# The Eclipse RCP UI Policy

An open and extensible workbench

- Takes contributions from anywhere
- Plugins can contribute to many places via extension points (e.g., perspectives, new views, new editors, trim, commands, handlers, bindings)
- Strong notion of editors and views
- Users can dramatically change layout
- Users can close views (stacks are collapsed)
- Implemented in SWT

Openness has consequences:

- Can’t control contributions
- Can’t control where those contributions show
- Can’t constrain your users
- Can’t implement on other UI technologies

E4 provides the mechanism

Key takeaway: if you build on top of E4, you need to design your own UI policy.
This is not a trivial undertaking.

## Migration Questions

How far do you want / need to go? What's your stopping rule?

How much do you rely on principles and elements embedded in RCP UI Policy?

- Properties, Help, wizards

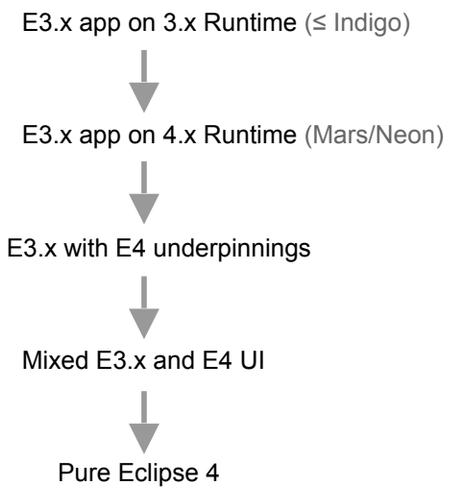
What future avenues are you enabling — or cutting off?

What business value does migration deliver?

We've talked about Eclipse 3 v Eclipse 4, and how every Eclipse 3 application is an Eclipse 4 application.

So if you have an app based on the Eclipse 3.x Runtime, and you want to move to Eclipse 4.x, what do you do?  How far do you go?

## Migration Paths



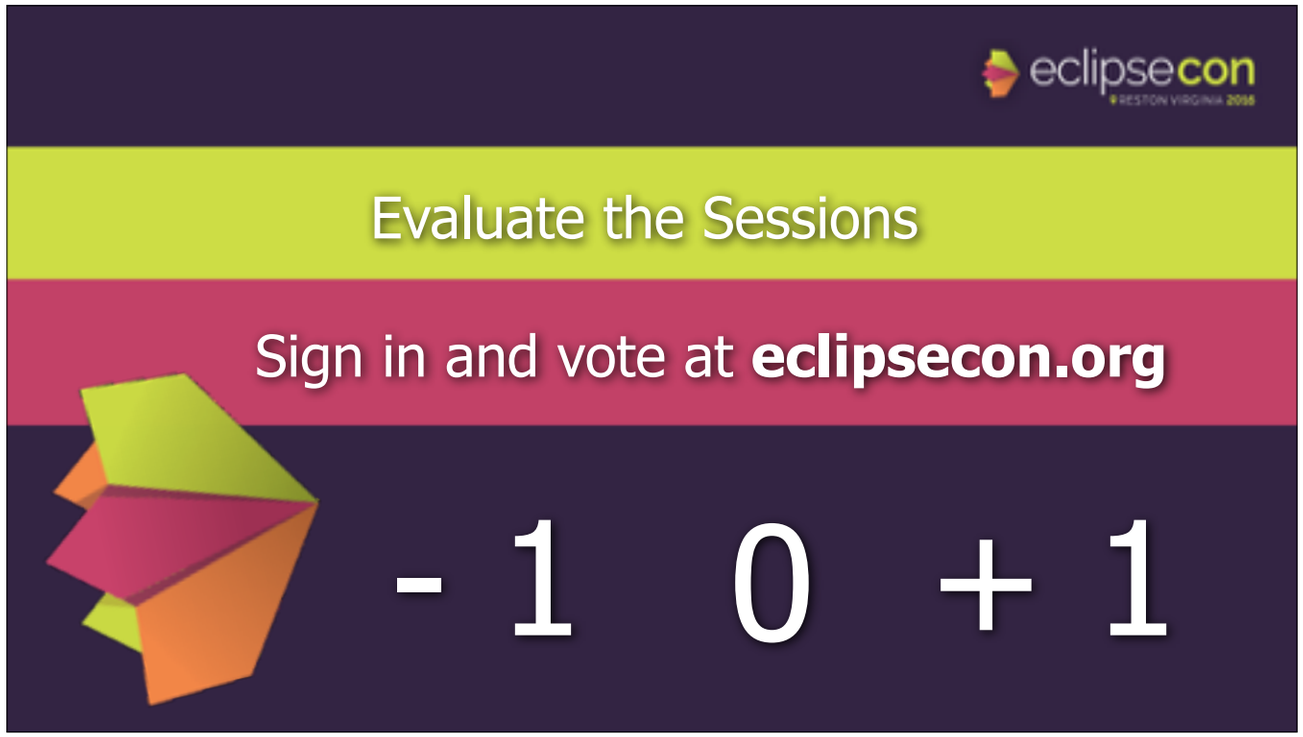

Straight-forward porting effort
Beware Presentation changes and use of internals

Use CSS, take advantage of contexts & DI

Rewriting/refactoring UI views as Eclipse 4 components

Substantial rewrite

You could just move to the Eclipse 4.x Runtime and call it quits.  That entails fixing up access to internals


eclipsecon
RESTON VIRGINIA 2016
Evaluate the Sessions
Sign in and vote at **eclipsecon.org**
-1 0 +1