Aus <http://www.heise.de/developer/artikel/Eclipse-4-die-naechste-Generation-der-freien-IDE-962171.html>  
  
stoßen früher oder später an einen Punkt, an dem sie nicht sicher sind, welchen Weg sie gehen sollen. Es gibt einfach zu viele Mittel, bestimmte Ziele zu erreichen.  
  
  
Durch den exzessiven Einsatz von OSGi-Bordmitteln soll sich bei Eclipse 4 eine bessere Modularisierung des Eclipse-Kerns erreichen lassen. Die Plattform versucht sich mehr auf ihre eigentlichen Aufgaben zu konzentrieren und erweiterte Funktionen in andere Komponenten (Bundles) auszulagern  
  
  
Konzepte wie **Singletons** haben die Entwickler komplett aus dem Eclipse-Umfeld verbannt und durch eine neue Strategie ersetzt – den "**Context**". Er vermittelt zwischen den Verbrauchern und Anbietern unterschiedlicher Services beziehungsweise kann die Services auffinden, auch ohne direkte Abhängigkeiten.  
  
  
Um die Aufgabe zu lösen und die Abhängigkeiten als Verbraucher solcher Services so gering wie möglich zu halten, wurde das **Dependency Injection** (DI) Pattern zum Teil der e4-Kernarchitektur. Es erlaubt, dass harte Abhängigkeiten auf bestimmte Teilsysteme nicht nötig sind, sondern sich alle Metadaten und Services über den Context injizieren lassen  
  
zieht man als Grundlage den Java Specification Request (JSR) 330 – "[Dependency Injection for Java](http://jcp.org/en/jsr/detail?id=330)" – heran, den seit kurzem offiziellen Java-Standard für DI.   
  
Beim Thema **Eventhandling** hat sich bei e4 im Gegensatz zu seinem Vorgänger einiges verändert. Bei Eclipse 3.x war es Usus, eine schier unüberschaubare Anzahl an Eventhandlern zu registrieren. Mit e4 wurde ein neuer Event-Bus eingeführt, der über das typische Publish/Subscribe-Muster die Events an interessierte Handler verteilt.  
  
  
Ein anderes großes Manko der aktuellen Eclipse-3.x-Workbench ist die **Komplexität**, die auf einen Anwendungsentwickler zukommt, der eine einheitliche Oberfläche erstellen will: Menüs oder Toolbars lassen sich deklarativ anmelden, das Layout einer Perspektive ist zu erstellen, die dazugehörigen Views und Editoren müssen angemeldet werden, ohne über etwaige Interaktionsmittel nachzudenken. Bisher kann das auf vielen unterschiedlichen Wegen geschehen, es ist jedoch nicht möglich, für einfache Zwecke das an einer zentralen Stelle zu tu  
  
  
Ein weiteres Manko, das mit der Historie von Eclipse gewachsen ist, ist beispielsweise die **strikte Trennung von Views und Editoren**. Die Trennung war zu Anfang vielleicht recht sinnvoll, die Grenzen sind in den letzten Jahren jedoch verwischt.  
  
Auch die Wiederverwendung einer UI-Komponente (beispielsweise einer View) in einem Dialog war teilweise nicht sauber realisierbar. Viele Probleme lassen sich mit einem einfachen, jedoch ausdrucksstarken Modell abbilden. Z  
  
  
Mit der **CSS** Engine ist es ohne größere Umwege realisierbar, entweder bestimmte Gruppen oder einzelne Widgets nach eigenem Bedarf zu dekorieren. Das ermöglicht es, Entwicklern von RCP-Anwendungen mehr Kontrolle über die Darstellung der Widgets zu geben.   
  
Nicht nur das Einbetten von anderen Komponenten wird erleichtert, auch die Sprachbarriere soll in Eclipse 4 weitaus geringer ausfallen  
  
  
  
  
Service programming models generally define three distinct participants: service providers,service consumers, and a broker or container that manages binding of service providers toconsumers. Basic implementations of this programming model, such as the OSGi