Fallbeispiel Eclipse

Seminar
Technologie- und
Innovationsmanagement

Joachim Fornallaz
Benno Luthiger
Stefan Rüttimann

Zürich, Januar 2004

Inhaltsverzeichnis

Abstrakt

Dieses Fallbeispiel beschreibt die offene Werkzeugplattform Eclipse.

Eclipse ist ein von IBM initiiertes und gesponsertes Open Source Projekt. Das Projekt gehört dem Konsortium Eclipse.org. Dieses wird von einem Boards of Directors geführt. Die konkreten Projektarbeiten werden durch das Project Management Committee geleitet.

Eclipse besticht durch seine extrem modulare Architektur. Die Applikation ist mit Java-Technologie implementiert.

Zur Erklärung der Open Source Strategie, die IBM im Falle von Eclipse gewählt hat, bieten sich die Strategie des „Sponsoring von Technologie-Standard“ (Garud u. a. 2002), die „Innovation Happens Elsewhere“-Strategie (Gabriel u. Goldman 2002) sowie das Geschäftsmodell „Komplementärprodukt“ (Hecker 1999, Raymond 1999) als Erklärungsmuster an.

1. Die Eclipse Tools Plattform

Eclipse ist eine offene Plattform für integrierte Software-Entwicklungswerkzeuge. Eigentümerin des Projekts ist das unter dem Namen Eclipse.org firmierende Konsortium. Dieses Konsortium organisiert die Weiterentwicklung von Eclipse, koordiniert die Beiträge aus der Entwickler-Gemeinschaft und stellt die Hardware für den Download der Software zur Verfügung.

1.1. Zweck

Eclipse ist eine offene Tools-Plattform, d.h. Eclipse bildet eine Basis-Architektur, auf welcher beliebige Plug-ins eingefügt werden können, um die vom Anbieter gewünschte Funktionalität bereitzustellen.

Die leichte Erweiterbarkeit der Eclipse-Plattform macht es mit dem Einfügen von geeigneten Plug-ins möglich, beliebige Software-Entwicklungsprozesse komplett innerhalb der Eclipse-Plattform durchzuführen. Eine typische Web-Anwendung besteht nicht nur aus dem eigentlichen Programm-Code, geschrieben in einer Programmiersprache wie beispielsweise Java, sondern auch aus anderen Ressourcen wie z.B. HTML- und XML-Dateien, Bildern, Klängen und anderen Medien, Datenbankabfragen und weiterem mehr. Mit einem zweckmässig erweiterten Eclipse können all diese Ressource mit dem gleichen Programm gepflegt und entwickelt werden. Ein Aufstarten von externen Programmen ist nicht mehr notwendig.

Die Eclipse Plattform ist in Java realisiert, was eine Plattform-Unabhängigkeit erleichtert. Zu Beginn war das Projekt für verschiedene Windows-Versionen und Linux verfügbar, heutzutage kann es auch auf Macintosh- und diversen UNIX-Systemen eingesetzt werden.

1.1.1. Architektur

Für den Erfolg einer Software-Applikation ist nicht nur wichtig, über welche Funktionalität sie verfügt und wie fehlerfrei sie läuft, sondern auch, wie schnell sie an geänderte technologisches Umfeld und auf sich ändernde Benutzerwünsche angepasst werden kann. Bei klassischen monolithischen Applikationen setzt jede Änderung der Funktionalität einen Eingriff in den Code der Applikation voraus. Fordern die Benutzer einer solchen Applikation neue oder erweiterte Funktionalität, müssen sie den nächsten Release der Software abwarten. Eine solche Vorgehensweise ist aufwendig und langsam.

Mit der Plug-in-Architektur steht eine Software-Bauart zur Verfügung, durch die eine Software schneller und dynamischer auf unterschiedliche oder sich ändernde Benutzerwünsche angepasst werden kann. Plug-ins sind Software-Komponenten, die in eine bestehende Applikation eingepflanzt werden können, ohne dass der Software-Code angepasst und die Applikation danach neu kompiliert werden muss. Voraussetzung ist natürlich, dass die Plug-ins die Schnittstellen korrekt implementieren, welche die Applikation zur Verfügung stellt, um Plug-ins als solche zu erkennen und zu verwalten.

Eclipse hat dieses Software-Muster auf die Spitze getrieben. Bei Eclipse ist „alles“ ein Plug-in. Jegliche Funktionalität wird in Form von Plug-ins angeboten. Der eigentliche Kern von Eclipse ist äusserst schlank. Die Plattformlaufzeitumgebung von Eclipse besteht praktisch nur aus der Funktionalität, um Erweiterungspunkte anzubieten, aktuell eingefügte Plug-ins zu verwalten und bei Bedarf zu aktivieren. Ist ein Plattform-Kern wirklich minimal, so kann er nicht beliebig viele Erweiterungspunkte anbieten. Der Entscheid für eine minimale Plattform-Architektur macht es notwendig, dass die Plug-ins nicht nur die Plattform erweitern können, sondern selbst wieder Erweiterungspunkte anbieten können, welche von

weiteren Plug-ins genutzt werden können. Auf diese Weise können Plug-ins die Basis von Plug-ins bilden, die ihrerseits wieder das Fundament für weitere Plug-ins darstellen. Auf diese Weise stellt Eclipse eine hochdynamische Applikation dar. Mit Eclipse können vollkommen unerwartete Anwendungen gebildet werden. Dies macht die Charakterisierung von Eclipse als „Apache für Entwicklerwerkzeuge“ (Lentzsch 2002, S. 45) und „Emacs des 21 Jahrhunderts“ (Martin Fowler in Gamma 2003, S. 5) absolut zutreffend.

1.1.2. Projektgliederung

Das reine „Eclipse Project“ gliedert sich in die drei Teilkomponenten *Plattformlaufzeitumgebung*, *Java Development Tools* (JDT) und *Plugin Development Environment* (PDE). Die Plattformlaufzeitumgebung bildet den Kern des Systems. Dieser bildet das Fundament und verwaltet die Plug-ins, die der Plattform die eigentliche Funktionalität spenden. Ohne solche Plug-ins wäre Eclipse nutzlos.

Die *Java Development Tools* bestehen aus einer Sammlung von Plug-ins, die Eclipse zu einer mächtigen Java Entwicklungsumgebung machen. Neben einem ausgefeilten Java-Editor wird die Programmierarbeit durch Debugging- und Test-Werkzeugen, durch ein Hilfesystem und ein System zum Versionen- und Konfigurations-Management unterstützt. Eine besondere Stärke der JDT sind die Code-Restrukturierungsfunktionen (Refactoring). Damit lassen sich grössere „Umbauarbeiten“ in Java Projekten elegant und schnell vornehmen. Auf herkömmliche Art würden solche Aktionen um ein Mehrfaches an Aufwand kosten.

Das *Plugin Development Environment* (PDE) ist ein zentraler Bestandteil des Projekts, besonders im Hinblick auf die Vision, die Eclipse zu Grunde liegt. Das PDE bildet die Grundlage für das Erstellen von Zusatzmodulen. Es stellt unter anderem Funktionen und Assistenten zur Verfügung, um neue Plug-ins zu erstellen, zu entwickeln und zu testen.

Zum offiziellen Eclipse Projekt zählen kleinere Sub-Projekte. Dazu gehören beispielsweise die C++ Development Tools, mit denen sich mit Eclipse neben Java-Projekten auch Software entwickeln lässt, die mit der Programmiersprache C++ realisiert wird. Kürzlich sind zwei neue Unterprojekte hinzugefügt worden. Eines davon heisst „Eclipse Visual Editor“ und erlaubt das Erzeugen von Benutzeroberflächen auf einfacher Weise per Maus.

Unter www.eclipse-plugins.info befindet sich eine komplette Liste mit ausführlichen Beschreibungen zu allen angebotenen Plug-ins. Zurzeit existieren schon über 400 Eclipse-Erweiterungen von Drittherstellern. Im Gegensatz zum Eclipse Projekt selber sind nicht alle Module unter einer Open Source Lizenz freigegeben, sondern werden von kommerziellen Anbietern vertrieben.

1.2. Initiatoren

Der strategische Initiant von Eclipse ist IBM, während Object Technology International (OTI) die notwendige Technologie, d.h. die Eclipse-Architektur und den Code beisteuerte. OTI ist seit Beginn der 90er Jahre eine hochkarätige Code-Schmiede. So wurden beispielsweise ENVY/Smalltalk, VisualAge for Java und VisualAge Micro Edition von OTI geschrieben.

Seit 1996 ist OTI eine Tochtergesellschaft von IBM. Auf Grund der Erfahrungen, die sich OTI mit VisualAge for Java und VisualAge Micro Edition[1] erarbeitet hat, wurde OTI 1999 damit beauftragt, die Code-Basis von Eclipse herzustellen.

Der eigentliche und offizielle Start von Eclipse erfolgte im November 2001, als IBM die Freigabe von Eclipse unter einer Open Source Lizenz ankündigte[2] und die entsprechende Code-Basis der zu diesem Zweck gegründeten Eclipse.org vermachte. Zu diesem Zeitpunkt waren als weitere Firmen Borland, Merant, QNX Software Systems, Rational Software, RedHat, SuSE und TogetherSoft an Eclipse.org beteiligt. Die Geburt dieses Konsortiums wurde von den Beteiligten ausgiebig gefeiert und die Gründungsmitglieder gaben sich im Folgenden optimistisch (siehe Anhang B: Aussagen der Gründungsmitglieder).

1.3. Weitere Beteiligte

Seit der Bildung des ursprünglichen Konsortiums wurde das Teilnehmerfeld von Eclipse.org laufend erweitert. Neben Fujitsu, Hitachi, HP, SAP und Intel stiessen noch zahlreiche weitere Branchenführer hinzu. Mittlerweile umfasst das Konsortium ungefähr 175 Partner, die zusammen mehr als 1200 Software-Entwickler stellen und die Entwicklung von Eclipse.org in 63 verschiedenen Ländern vorantreiben (siehe Anhang A: Eclipse Bord of Directors).

Eclipse.org wird von einem Board of Directors geleitet. Dieses Gremium wird von den Mitgliederfirmen bestellt.

1.4. Projektstand

Die Entwicklung an Eclipse wurde vom Eclipse Project Management Committee (PMC) weitergetrieben. Im Juni 2002 wurde die zweite Version von Eclipse freigegeben. Aktuell ist Version 2.1.2 verfügbar. Im Moment wird an Version 3 gearbeitet. Laut Releaseplan soll diese Version im Juni 2004 fertig gestellt werden.

1.5. Lizenz

Beim Gebrauch von intellektuellem Eigentum wie Software ist die Lizenzierung von zentraler Bedeutung. Lizenzen bestimmen, wie und wenn Drittparteien Software einsetzen können und welche Kosten daraus entstehen. Während proprietäre Software-Produkte ihre eigenen, spezifischen Lizenzen haben, gibt es auf dem Gebiet der Open Source eigentlich drei gebräuchliche, generische Lizenzformen:

- BSD License
- GNU General Public License (GPL)
- GNU Library General Public License (LGPL)

---

[1] VisualAge for Java und VisualAge Micro Edition wurden als IBM-Produkte vertrieben.

[2] „On November 5, 2001, IBM announced its donation of $40 million worth of tools to the Eclipse project." (Brody 2001)

Trotzdem ist zu erwähnen, dass die meisten Open Source-Projekte und -Anwendungen eine Lizenzform wählen, die von einer oder mehreren der oben genannten Formen abgeleitet ist. Open Source-Projekte und -Anwendungen können auch unter mehr als eine Lizenzform gestellt werden.

1.5.1. BSD License

In der BSD Lizenz müssen bei jeglichen Erweiterungen eines Open Source-Produktes die Autoren und Beitragsleister des ursprünglichen Source Codes angegeben werden. Der Source Code kann für eigene Entwicklungen benutzt und abgeändert werden, ohne dass die daraus entstehenden Produkte BSD Lizenz unterstellt oder in anderer Form quelloffen sein müssen.

1.5.2. GNU General Public License (GPL)

Wie bei der BSD Lizenz werden auch bei der GPL Haftungsansprüche gegenüber früheren Beitragsleistern gänzlich ausgeschlossen. Der grosse Unterschied besteht jedoch darin, dass alle Programme, die in irgendeiner Form unter der GPL lizenzierten Code enthalten, wiederum unter der GPL veröffentlicht werden. Diese Einschränkung macht GPL-Software im wesentlichen für eine Verwendung im kommerziellen Umfeld unbrauchbar. Bekannte Produkte / Projekte: MySQL, OpenOffice, GNOME, Emacs, SAMBA u.a.

1.5.3. GNU Library General Public License (LGPL)

Die LGPL entspricht im wesentlichen der GPL, unterscheidet sich aber dadurch, dass die von LGPL geschützten Bibliotheken die Entwicklung von kommerzieller, nicht der GPL unterliegender Software ermöglicht. Somit ist die LGPL eigentlich die Verbindung zwischen freier und kommerzieller Software Entwicklung. Bekannte Produkte bzw. Projekte, die unter der LGPL stehen, sind: KDE, KOffice oder JBoss.

1.5.4. Common Public License (CPL)

Die Eclipse-Plattform steht unter dem Schutz der CPL, welche im Grunde eine abgeleitete Form der GNU Lizenzen ist. Die CPL kombiniert Eigenschaften der GPL und der LGPL und versucht im wesentlichen, die kommerzielle Nutzung von Software zu erleichtern. Beitragleistende können unter Einhaltung gewisser Richtlinien das abgeänderte oder erweiterte Produkt unter ihrer eigenen Lizenz weiter verbreiten. Die kommerziellen Kontributoren können jedoch bei Klagen keine Haftungsansprüche gegenüber früheren Beitragsleistern geltend machen. Wenn nun beispielsweise eine Organisation ein unter der CPL stehendes Programm kommerziell vertreibt, so muss diese Organisation alle früheren Beitragsleister gegen Haftungs- oder Garantie-Ansprüchen von klagenden Drittparteien abschirmen. Sollte ein Gericht die Ansprüche der klagenden Drittpartei als gültig befinden, so haftet der kommerzielle Vertreiber des Produktes, auch für Code-Abschnitte, die von anderen Beitragsleistern erstellt wurden.

Die CPL ist von der Open Source Initiative (OSI) anerkannt. Der genaue Wortlaut der CPL kann unter www.opensource.org/licenses.cpl.php nachgeschlagen werden.

1.6. Erfolg und Verbreitung

Im Herbst 2003 wurden unabhängig voneinander zwei Umfragen gestartet, die beide das Ziel hatten, den Marktanteil der verschiedenen Entwicklungs-Werkzeuge zu ermitteln. Beide Umfragen sind, was ihre wissenschaftliche Aussagekraft betrifft, nicht über alle Zweifel erhaben. Eine bedeutende Verzerrung kommt schon dadurch zustanden, dass es bei beiden Umfragen nur um Java IDEs ging. Aus diesem Grund taucht beispielsweise VisualStudio, die Entwicklungsplattform von Microsofts .NET Technologie, in beiden Umfragen nicht auf. Auch der Umstand, dass die beiden Umfragen nicht allzu viel Übereinstimmung zeigen, was der Anteil und die Rangfolge der einzelnen Entwicklungs-Tools betrifft, stärkt diesen Zweifel. Interessant ist aber, dass in beiden Umfragen Eclipse deutlich an erster Stelle liegt. In der Umfrage von QA Systems[3] liegt Eclipse mit einem Anteil von 41% an der Spitze, gefolgt von Borlands JBuilder mit gerade noch 14% Marktanteil. In der Studie von Jerason[4] beträgt der Anteil von Eclipse auch noch stattliche 31%, diesmal vor IntelliJ mit 19% Marktanteil.

Diese Zahlen lassen die Aussage zu, dass Eclipse als Java-Entwicklungsplattform die Führungsposition eingenommen hat. Interessant wäre es zu bestimmen, welche Position Eclipse als Entwicklungswerkzeug für beliebige Ressourcen, d.h. nicht nur Java Quellcode, belegt. Eine andere interessante Frage ist, wo überall Eclipse als Basis für eine Java-Applikation auf dem Client-Desktop dient. Hier zu genauen Zahlen zu kommen ist schwierig, da es die Eclipse-Lizenz ausdrücklich erlaubt, Eclipse-basierende Applikationen unter einem beliebigen Namen zu vertreiben. Als Indikator für die Marktdurchdringung von Eclipse kann allenfalls die Anzahl von Plug-ins betrachtet werden, die für Eclipse verfügbar sind. Allein die Tatsache, dass es mittleweile mehr als 400 Plug-ins gibt, deutet auf eine grosse Akzeptanz und hohe Verbreitung der Eclipse Plattform hin.

Ein weiterer Hinweis für den Erfolg des Projektes besteht in der Anzahl der Mitgliederfirmen im Eclipse.org Konsortium. Je mehr Unternehmen sich verpflichten, aktiv diese Plattform zu unterstützen, umso mehr steigt die Wahrscheinlichkeit, dass Eclipse zur universellen Entwicklungsplattform wird. Anfang Dezember 2004 zählte das Eclipse-Konsortium bereits über 40 Mitglieder. Neben IBM, RedHat oder Hewlett-Packard zählen auch Ericsson und sogar Intel zu den Teilnehmern des Konsortiums.

[3] Samplegrösse 1409. QA Systems ist ein Mitglied des Eclipse Konsortiums (siehe QA Systems 2003).

[4] Samplegrösse 5372 (siehe Jerason 2003).

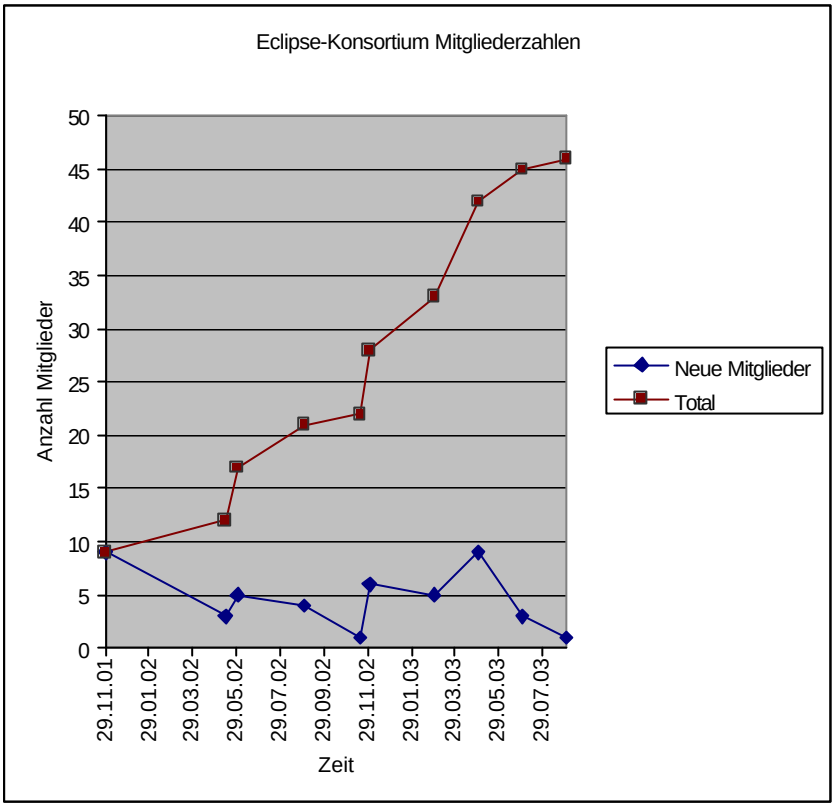

Abbildung 1: Entwicklung der Mitgliederzahl im Eclipse-Konsortium

1.7. Strategisches Umfeld von Eclipse

Die Konkurrenten von Eclipse.org sind nicht zahlreich, dafür aber umso bedeutender in der Computer- und Software-Branche: Sun Microsystems und Microsoft. Ähnlich wie IBM mit Eclipse versucht Sun Microsystems mit ihrem Produkt NetBeans die Vormachtstellung, die nach wie vor von Microsoft und deren Entwicklungsplattform VisualStudio .NET eingenommen wird, zu brechen.

Während die Stärke von Eclipse die Integration verschiedener Hersteller und die so gewonnene Kompatibilität und Grösse ist, setzt Sun im Kampf gegen Microsoft auf 18 neue Module für die NetBeans Umgebung.

Scott Hebner, Marketing Director von IBM, sieht die Situation wie folgt: “I'd argue that the most ominous threat to the Microsoft developer community is Eclipse. We're an open alternative to a closed environment, Visual Studio .NET. Just as Microsoft is often quoted about its concerns about Linux and J2EE, this opens up another flank on them.” (Karpinski 2002)

Das weitere Umfeld von Eclipse/WebSphere Studio bildet IBMs WebSphere Application Server (WAS). Bei letzterem handelt es sich um einen echten J2EE Applikationsserver. Damit steht der WebSphere Application Server in Konkurrenz zu BEA Weblogic und zum Sun Applikationsserver.

2. Theoretische Reflexion

Gemäss Aussagen von IBM (Brody 2001) hat sie die Entwicklung von Eclipse mit mindestens $40 Mio. unterstützt. Aus welchem Grund mobilisiert eine Firma eine solche Summe für ein Produkt, welches andere Firmen und Personen frei beziehen und nutzen können?

Zur Erklärung der Open Source Strategie, die IBM im Falle von Eclipse gewählt hat, bieten sich die Strategie des „Sponsoring von Technologie-Standard“ (Garud u. a. 2002), die „Innovation Happens Elsewhere“-Strategie (Gabriel u. Goldman 2002) sowie das Geschäftsmodell „Komplementärprodukt“ (Hecker 1999, Raymond 1999) als Erklärungsmuster an.

2.1. Eclipse als neuer Technologie-Standard

In ihrem Artikel beschreiben Garud, Jain und Kumaraswamy (2002) am Beispiel von Sun und dessen Technologie-Sponsoring von Java, wie eine Firma versuchen kann, einen sich herausbildenden Standard zu prägen, welche Möglichkeiten sich ihr dadurch bieten und welche Herausforderungen sie in diesem Prozess zu meistern hat. Den theoretischen Rahmen, den sie für ihr Fallbeispiel verwenden, kann sehr gut auch für den Eclipse-Fall eingesetzt werden.

2.1.1. Technologische Standards

Jede Technologie braucht einen ihr entsprechenden institutionellen Raum. Die Regeln des institutionellen Raums bestimmen die Produktion, die Verteilung und den Verbrauch der Artefakte, welche eine Technologie charakterisieren. Technologische Systeme bestehen aus einem Set von Komponenten, die miteinander interagieren und Werte schaffen für die Nutzer der Technologie. Die Leistungsfähigkeit eines Systems ist nicht nur abhängig von der Effizienz seiner Komponenten, sondern auch von der Kompatibilität der einzelnen Komponenten untereinander. Die Kompatibilität der Komponenten wird gewährleistet durch gemeinsame Standards. Aus diesem Grund stellen die technologischen Standards die Schlüsselfacetten eines institutionellen Raums dar. Zusammen mit den Spezifikationen über Form und Funktion der System-Komponenten bilden die Standards, welche die Interaktion dieser Komponenten regeln, die Architektur eines Systems.

Unternehmen, welche die Fähigkeit oder Macht haben, einen Standard zu kontrollieren, können bei der Nutzung einer Technologie eine Schlüsselstellung einnehmen. Der Kampf um Standards ist speziell von Bedeutung bei Technologien, die in Entstehung begriffen sind. In solchen Situationen ist auch der institutionelle Raum in Bewegung und Firmen können durch die Gestaltung der Standards die Eigenschaften der von ihnen geförderten Technologie direkt in die sich entwickelnden institutionellen Strukturen einbauen. Im Erfolgsfall kann dies den Firmen bedeutende kompetitive Vorteile verschaffen. Dieser Sachverhalt gilt um so mehr in Gebieten wie der Informationstechnologie, die durch Netzwerkexternalitäten und positive Skalenerträge charakterisiert sind.

2.1.2. Mobilisierung

Eine Firma, die in eine zukunftsweisende Technologie investiert hat, kann einen reichhaltigen Ertrag realisieren, wenn es ihr gelingt, ihre Technologie zum Standard zu machen. Unter Marktbedingungen kann aber kein Unternehmen alleine ein Standard setzen. Um das den bestehenden Standards inhärente Beharrungsvermögen zu überwinden, muss sie Partner mobilisieren. Die Schwierigkeiten in dieser Phase

kommen von zwei Seiten. Einerseits haben die dominanten Aktoren der bisherigen Technologie, d.h. diejenigen Unternehmen, die einen bestehenden Standard kontrollieren, wenig Interesse an einer neuen Technologie und den damit einhergehenden Veränderungen, die ihren bevorzugten Standard gefährden. Andererseits können auch Firmen, die an der Überwindung bestehender Standards interessiert sind, Vorbehalte gegen den angebotenen neuen Standard haben, wenn ein Einschwenken auf den neuen Standard ihr Entwicklungspotential beschränkt. Diese Problematik ist allen Strukturen gemeinsam, die sowohl Medium wie auch Ergebnis von unternehmerischem Handeln sind. Solche Strukturen ermöglichen und fördern einerseits unternehmerische Tätigkeiten, indem sie den Aktoren die Möglichkeit geben, verschiedene Komponenten eines technologischen Systems verteilt, d.h. unabhängig voneinander zu entwickeln. Gleichzeitig schränken diese Strukturen die Entwicklung auf einen bestimmten Pfad ein.

Um dieses Beharrungsvermögen zu überwinden, muss der Technologie-Sponsor ein grosses Kollektiv für den neuen Standard mobilisieren. Dies ist nicht einfach und der Sponsor braucht viel Überzeugungskraft und grosse soziale Fähigkeiten, da potentielle Rivalen überzeugt werden müssen, sich auf einen noch unsicheren gemeinsamen Standard einzulassen, welcher diese möglicherweise für die Zukunft in einen kompetitiven Nachteil versetzt. Eine geeignete Mobilisierungs-Strategie, mit welcher diese Schwierigkeit überwunden werden kann, ist die Strategie der *offenen Systeme* (Garud u.a. 2002, S. 12). Diese Strategie besteht darin, einen Teil der technologischen Basis des neuen Standards in die öffentliche Domäne zu legen. Damit schafft der Sponsor geeignete Anreize für andere Parteien, am angebotenen Standard zu partizipieren. Diese können durch ihre Teilnahme von den Vorarbeiten des Sponsors profitieren und brauchen nicht selbst in die Entwicklung zu investieren, um einen analogen technischen Stand zu erreichen.

Mit einer Strategie der offenen Systeme wird aus einem privaten Gut ein öffentliches Gut. Gemäss der Theorie öffentlicher Güter (Olson 1965) lädt dies zu Trittbrettfahren ein. Typische Folgen von solchem Verhalten sind die Unterversorgung des öffentlichen Guts oder deren Übernutzung. Wo allerdings auf Grund von Netzwerk-Externalitäten Skalenerträge möglich sind, kann die Produktion eines öffentlichen Guts andere Aktoren anregen, auch zum öffentlichen Gut beizutragen. Jeder Beitrag eines komplementären Produkts zum technologischen Feld vergrössert deren Belastbarkeit (*carryng capacity*, vgl. Hardin 1968) und verstärkt den Impuls der aufstrebenden Technologie.

Zur wirkungsvollen Mobilisierung kann eine Strategie des offenen Systems mit einer Strategie des Erwartungs-Managements flankiert werden. Eine solche Strategie besteht darin, Visionen der zukünftigen Entwicklung und Erwartungen von künftigen Erträgen zu pflegen. Firmen, die sich auf einen aufstrebenden Standard einlassen, haben in der Gegenwart noch keinen Nutzen von dieser Entscheidung. Sie sind darauf angewiesen, dass andere zur Teilnahme am Standard überzeugt werden können. Ein Technologie-Sponsor muss deshalb versuchen, Erwartungen über den zukünftigen Nutzen des Standards zu erzeugen, indem er den Zugang zu zukünftigen Technologien vorankündigt, auch wenn die Produktmärkte dafür noch nicht entstanden sind. Wenn eine solche Strategie des Agenda-Settings erfolgreich ist, können wechselseitig abhängige Firmen dazu überzeugt werden, auf den fahrenden Zug aufzuspringen (Bandwagon-Effekt). Je stärker das Feld der beteiligten Firmen ist, um so grösser wird die Legitimität des Standards. Dies kann in der Folge dazu führen, dass in der Art einer selbsterfüllende Prophezeiung der versprochene Nutzen einer Kooperation tatsächlich realisiert werden kann.

Eine liberale Lizenzierung eines aufstrebenden Standards ist von grosser Bedeutung für den Erfolg in diesem Stadium. Das Lizenzmodell soll es den Teilnehmern erlauben, die Technologie zu modifizieren,

solange die Modifikationen den anderen Beteiligten wieder zur Verfügung gestellt werden. Damit können mehr Innovationen verwirklicht werden, als wenn nur der Technologie-Sponsor Veränderungen anbringen könnte. Auch kann mit einer solchen Flexibilität gewährleistet werden, dass die Teilnehmer die Technologie für ihre eigenen Zwecke interpretieren können, was es ihnen erlaubt, ihr Produktangebot von demjenigen der anderen am neuen Standard kooperierenden Teilnehmer zu differenzieren.

2.1.3. Pflege des neuen Standards

Die Gefahr einer Strategie des offenen Systems besteht darin, dass auf diese Weise auch die Konkurrenten und insbesondere die herausgeforderten Parteien Zugang zur neuen Technologie haben. Dies eröffnet einem Konkurrenten die Möglichkeit, in böswilliger Absicht am Standard teilzuhaben. Eine beliebte Strategie ist z.B. den Standard aufzunehmen mit der Intention, diesen zu „vergiften", d.h. diesen so zu erweitern, dass er in inkompatible Versionen zerfällt. Ein Konkurrent wird auch versuchen, auf dem Feld der Interpretation die Bedeutung des Standards in Frage zu stellen. Ist es eine universale Plattform, wie der Technologie-Sponsor ankündigt, oder ist es bloss eine gelungene Applikation, eine nützliche Entwicklungsumgebung, wie es der Konkurrent darstellt?

Neben diesen politischen Motiven, einen sich entwickelnden Standard zu stören, gibt es auch strategische Gründe für Parteien, vom Standard abzuweichen, obwohl sie ihn formal unterstützen. Dies ist z.B. der Fall, wenn eine Firma sich in ihren Möglichkeiten behindert sieht, auf dem Standard aufbauende Produkte von denen der anderen Anbieter zu differenzieren.

Ob eine Abweichung vom Standard aus politischen oder strategischen Gründen erfolgt ist für den Technologie-Sponsor unerheblich. In jedem Fall ist eine Abweichung problematisch, weil dies dazu führen kann, dass der Standard in inkompatible Versionen zerfällt. Damit schwindet aber der Anreiz von anderen Kooperationspartnern, auf diesem Standard zu entwickeln. Dem Technologie-Sponsor droht somit die Gefahr, die Kontrolle über den Standard zu verlieren. Gleichzeitig verliert er die Glaubwürdigkeit bezüglich seines Versprechens, ein einheitliches Feld um den Standard zu erzeugen und zu gewährleisten. Damit der Standard aufrecht erhalten wird, sind die politischen Fähigkeiten des Technologie-Sponsors gefordert. Der Sponsor kann auf die Gefahren, dass der Standard aufgeweicht wird oder zerfällt, reagieren, indem er spezielle Kontrollmechanismen etabliert. Diese Mechanismen können rechtlicher Art sein, um Gegenmobilisierung zu vereiteln oder technischer Art wie z.B. mit Testprozeduren, um die Kompatibilität von unterschiedlichen Implementierungen des Standards sicherzustellen.

2.1.4. Spannungsfeld des Technologie-Sponsors

Die Ausführungen zeigen, dass der Technologie-Sponsor einem Spannungsfeld ausgesetzt ist. Einerseits muss er mit einem liberalen Lizenzierungswesen für den Standard mobilisieren, andererseits muss er sicherstellen, dass der Standard nicht fragmentiert wird. Ursache dieses Problems ist der Umstand, dass der angestrebte Standard sowohl Medium wie auch Resultat der gemeinsamen Aktionen ist. Aus diesem Grund sind die Kooperationspartner immer auch Rivalen. Das Problem tritt verschärft auf, wenn der Technologie-Sponsor keine neutrale und unabhängige Institution ist, sondern ein Wettbewerbsteilnehmer mit eigenen Interessen. In einer solchen Situation droht der Technologie-Sponsor in eine Legitimitäts-Falle zu geraten. Diese Legitimitäts-Falle besteht darin, dass der Sponsor sowohl die Rolle des Regel-Erzeugers (*rule creation*) wie auch diejenige des Regel-Durchsetzers (*rule enforcement*) einnimmt (Garud u.a. 2002, S. 27). Als Regel-Erzeuger bestimmt der Sponsor, welche Regeln in den Standard integriert

werden und wann sie Geltung erlangen. Als Schiedsrichter über die Regeln muss der Sponsor darüber wachen, dass die beteiligten Parteien sich an die Regeln halten und nicht vom Standard abweichen. Aus nachvollziehbaren Gründen sieht sich der Technologie-Sponsor legitimiert, beide Rollen gleichzeitig auszuüben. Aus der Sicht der anderen am Standard beteiligten Aktoren sieht das aber so aus, dass sie immer einen Schritt hinter dem Sponsor nachhinken. Eine solche Situation mündet zwangsläufig in einen Glaubwürdigkeitsverlust des Technologie-Sponsors.

2.1.5. Eclipse: Applikation oder Standard?

Die Open Source Strategie, die IBM mit Eclipse gefahren ist, scheint Sinn zu machen, wenn man IBM als Technologie-Sponsor versteht, der mit Eclipse einen neuen Standard etablieren will. Die Freigabe des Quellcodes passt perfekt in das Mobilisierungs-Schema, in der Initialphase andere Parteien an der Technologie beteiligen zu lassen, um sie damit ins Boot für den neuen Standard zu nehmen. Zugleich konnte IBM mit der Code-Spende an Eclise.org der Legitimitäs-Falle entgehen, da nun das Board of Directors des Eclipse-Konsortiums für die Pflege des Standards verantwortlich ist. Eclipse.org ist als Meritokratie ausgestaltet, das heisst ein Mitglied erhält umso mehr Rechte, je mehr es leistet. Auf Grund seines grossen Engagements ist IBM weiterhin im Board of Directors vertreten und kann damit über transparente Regeln den Standard mitgestalten. Noch grösser dürfte der Einfluss sein, den IBM dadurch ausübt, dass bis anhin das Eclipse Project Management Committee ausschliesslich von OTI-Mitarbeitern besetzt ist.

Es fragt sich allerdings, ob es sich bei Eclipse wirklich um einen Standard handelt. Eclipse ist in Java implementiert und zweifellos stellt die Java-Technologie einen Standard dar. Dies auf folgendem Grund: Die Existenz von unabhängigen und inkompatiblen Betriebssystemen stellt für Softwarehersteller ein Problem dar. Diese müssen sich entscheiden, für welches Betriebssystem sie ihre Software-Applikationen entwickeln wollen. Wollen sie auch die Benutzer der anderen Plattformen mit ihrer Applikation erreichen, so müssen sie einen grossen Aufwand betreiben, um ihre Software auch auf andere Betriebssysteme zu portieren. Mit Java wird es möglich, dieses Problem zu umgehen. Mit dieser Technologie kann eine Applikation plattformunabhängig („write once, run anywhere“) implementiert werden. Ein Anbieter von Produkten, die in Java implementiert sind, kann mit dem gleichen Softwarecode sowohl (im Prinzip) den Windows-, den Linux-, den Macintosh-Markt und diverse andere abdecken. Offensichtlich funktioniert das nur, wenn die verschiedenen Betriebssysteme, welche die Java-Technologie enthalten, sich exakt an den von Sun festgelegten Standard halten und sich die Anbieter von Software-Produkten auf diesen Umstand verlassen können.

Diesen Sachverhalt nutzt auch Eclipse, wird doch damit gewährleistet, dass diese Plattform auf allen verbreiteten Betriebssystemen läuft. Eclipse bietet allerdings mit seiner extrem schlanken Plug-in-Architektur etwas, was Java nicht enthält und was Eclipse für Anwendungen auf dem Client-Computer attraktiv macht. Der IBM-Manager Handy (in Jernigan 2001, S. 2) beschreibt das wie folgt: „There are vendors who are very interested in supplying just a plug-in for a certain particular specialty they have; or technology for a particular industry. That would not have been possible before, because they could not have sustained the cost of bringing such a plug-in or technology to market as a stand-alone product. So this opens up a whole new area for expansion, for certain companies to provide specific plug-ins.”

Dank Eclipse muss ein Anbieter einer spezielle Funktionalität oder Technologie nicht eine eigene Stand-alone Anwendung herstellen, um sein Produkt auf den Markt zu bringen, sondern kann Eclipse

nehmen und seine Funktionalität als Plug-in implementieren. Damit bleiben ihm bedeutende Investitionskosten erspart. Damit diese Vorgehensweise allerdings funktioniert, muss gewährleistet sein, dass die Art und Weise, wie Eclipse Plug-ins erkennt und aktiviert, den Spezifikationen der Erweiterungspunkten und Erweiterungen entspricht, die das Eclipse-Konsortium festgelegt hat. Diese Spezifikationen stellen aber genau das dar, womit ein Standard definiert wird. Damit kann ohne Zweifel gefolgert werden, dass Eclipse die Bedeutung eines Standards zukommt.

2.2. „Innovation Happens Elsewhere"

Die von Gabriel u. Goldman (2002) beschriebene und am Beispiel von Suns NetBeans-IDE veranschaulichte Strategie „Innovation Happens Elsewhere" (IHE) basiert auf dem Ansatz, dass eine Firma all jene Technologie, die nicht zu ihrer Kernkompetenz zählt, in die öffentliche Domäne legt. Eine Firma, welche gemäss IHE-Strategie handelt, muss als erstes diejenigen Ideen und Technologien identifizieren, welche die Basis ihrer kompetitiven Stärke auf dem Markt bilden. Diese Kernkompetenzen bauen üblicherweise auf einem System von Technologien auf, welche die Firma entweder einkauft oder, mangels Anbieter, selber entwickeln und pflegen muss. Das Ziel der IHE-Strategie ist, selber unterhaltene Technologien ausserhalb der Kernkompetenz in die öffentliche Domäne zu legen, um dadurch Innovationen von Aktoren ausserhalb der Firma in Gang zu setzen. Solche Innovationen können dann zugunsten der Technologien, welche die Kernkompetenz der Firma bilden, ausgewertet werden, um auf diese Weise die Wettbewerbsfähigkeit der Firma zu stärken.

In diesem Verständnis ist eine Open Source Strategie die Anwendung der IHE-Strategie auf den Software-Bereich. In diesem Fall nutzt die IHE-Strategie die Eigenheit aus, dass Software in bestimmten Fällen in der Geschenk-Ökonomie angesiedelt ist und nicht im Bereich der knappen Güter, welche den Mechanismen der klassischen Ökonomie unterliegen. Das Grundprinzip der Geschenk-Ökonomie ist, dass Geschenke einen Rückfluss von Gaben in Richtung des Schenkers stimulieren (Mauss 1966). Im Falle von Open Source bestehe dieser Rückfluss aus Bug-Fixes, Erweiterungen, Ideen etc.

Unter der Voraussetzung, dass es sich bei der freigegebenen Technologie nicht um eine abgeschlossenen Applikation, sondern um eine dynamisch erweiterbare Plattform handelt, kann dies andere Firmen anregen, diese Plattform zu benutzen, um mit eigenen Erweiterungen versehen als eigenständige Produkte zu vermarkten. Dies setzt allerdings voraus, dass sie Open Source Lizenz ein solches Geschäftsmodell erlaubt. Der Nutzen für die IHE-Firma besteht in einem solchen Fall darin, dass solche Entwicklungen zu Innovationen führen können, welche von der IHE-Firma nicht antizipiert wurden. Auf diese Weise kann die IHE-Firma mit der freigegebenen technologischen Basis völlig unerwartete neue Märkte erschliessen.

Ein weiterer Nutzen der IHE-Strategie besteht darin, dass sich um die Open Source Software eine entsprechende Community bildet. Für die IHE-Firma ist die Existenz einer dynamischen Community relevant auf Grund der Kommunikation, die sich dadurch zwischen Firma und Community ergeben. Eine Technologie-Firma kann mit Hilfe von *white papers*, Werbung und anderem PR-Material versuchen, Publizität für ein Produkt zu erzeugen und zu Informationen über dessen Marktpotential zu kommen. Diese Methoden werden vom Publikum aber in erster Linie als Werbung wahrgenommen, wobei der sachliche Kern solcher Botschaften nur mit Vorbehalt anerkannt wird. Dies führt dazu, dass mit solchen Methoden viel Aufwand ein verhältnismässig grosses Rauschen erzeugt wird, ohne dass dies aber zu erkennbarem Nutzen führt. Die Kommunikation der Firma mit der Open Source Community um ein

Projekt, das von der Firma gesponsert wird, stellt für die Firma dagegen eine weit effizientere Möglichkeit dar, zu technologie- und marktrelevante Informationen zu kommen. Voraussetzung ist natürlich, dass diese Kommunikation ernsthaft und vertrauensvoll geführt und dass auf Anliegen und Anregungen von Beitragsleistenden eingegangen wird. Der Vorteil dieser Option ist, dass sie mit bedeutend weniger Aufwand realisiert werden kann.

Werden diese Chancen (Gabenaustausch, Möglichkeit unerwarteter Innovationen und Kommunikation mit der Commuity), die sich aus der IHE-Strategie ergeben, von der IHE-Firma gezielt genutzt, so setzt dies die Firma in die Lage, Innovationen im Umfeld der Kernkompetenz der Firma zu „umarmen". d.h. für die eigene Wettbewerbsfähigkeit nutzbar zu machen.

Die Kernpunkte der IHE-Strategie sind also die Identifikation der Kernkompetenz, d.h. desjenigen Bereichs der Firma, der proprietär bleibt, aus der richtigen Wahl der Lizenz, der Architektur der Technologie und der Kommunikation mit der Entwickler-Community. Damit Innovationen im Umfeld des Projekt-Sponsors möglich sind, müssen die entsprechenden Anreize gesetzt werden. Die Software-Architektur muss es unabhängigen Anbietern erlauben, das Produkt als Basis von Erweiterungen zu benutzen, und gleichzeitig müssen die Lizenzbedingungen so ausgestaltet sein, dass die Vermarktung der Erweiterungen möglich ist. Eine GNU-Lizenz beispielsweise würde einen solchen Anreiz vernichten.

Wie oben ausgeführt erfüllt Eclipse diese Bedingungen in hohem Mass. Die Architektur ist bewusst dahin getrimmt, dass Eclipse als Basis für beliebige Plug-ins fungieren kann. Gleichzeitig postuliert die CPL, welche die Freigabe des Quellcodes regelt, explizit die Vermarktungsmöglichkeiten der auf Eclipse basierenden Produkte. Eclipse erscheint damit IHE-tauglich. Allerdings sind keine Äusserungen von Seiten IBMs bekannt, die dahin weisen, dass IBM gezielt nach Eclipse-induzierten Innovationen Ausschau hält, um solche für proprietäre Produkte und Prozesse zu benutzen.

2.3. Eclipse als Komplementärprodukt

Die Strategie „Give Away the Razor, Sell Razor Blades" (Raymond 1999, S. 12) beschreibt das Geschäftsmodell, welches darin besteht, ein Produkt unter Preis zu vertreiben mit dem Ziel, die Käufer in eine lock-in Situation zu bringen, was in der Folge den Verkauf von Komplementärprodukten zu einem guten Preis gestattet.

Kernpunkt einer solchen Strategie ist die Existenz eines Komplementärprodukts, dessen Verkauf durch die Open Source Strategie angeregt werden kann und dabei für die Firma, die das Open Source Produkt sponsert, einen Ertrag abwirft, welcher die Kosten der Freigabe wieder aufwiegt. Im Falle von IBM sind das mit Eclipse immerhin mindestens 40 Mio $.

Welches ist nun das Komplementärprodukt im Falle von Eclipse? Zur Beantwortung dieser Frage muss das grössere Umfeld untersucht werden, in welchem Web-Applikationen spielen. Das grössere Umfeld stellen die Applikations-Server eines Unternehmens dar. Gemäss Conte (2002) geht IBM bei ihrer Eclipse- und WebSphere-Strategie davon aus, dass dasjenige Software-Unternehmen das IT-Konto einer Firma kontrolliert, welche den Applikations-Server des Unternehmens liefern kann[5]. Entsprechend dieser

[5] „IBM's WAS [WebSphere Application Server] strategy is based on the belief that whichever vendor supplies an enterprise its application server will control the enterprise's IT account" (Conte 2002, S. 1).

Erkenntnis hat IBM im Sommer 2001 mit dem WebSphere Application Server (WAS) ihren ersten wettbewerbsfähigen J2EE Applikations-Server vorgestellt.

Bei der Evaluation der unterschiedlichen Produkte auf dem Markt für Applikations-Server kann die Tatsache, dass ein bestimmter Applikations-Server von einer verbreiteten Entwicklungsumgebung optimal unterstützt wird, der ausschlaggebende Punkt sein. Die enge Verzahnung von Entwicklungsumgebung und Applikations-Server reduziert die Entwicklungszeit und erhöht somit die Produktivität von Entwicklern deutlich. Eine Geschäftsidee kann viel schneller realisiert, getestet und auf dem produktiven System installiert werden, wenn die Entwicklungsumgebung auf die produktive Umgebung abgestimmt ist. Oder wie es Conte (2002, S. 1) ausdrückt: „[D]evelopers love slick tools, and a developer who uses a particular tool suite naturally promotes the tool's target platform".

Gemäss dieser Interpretation ist Eclipse als Basis von IBMs WebSphere Studio Entwicklungsumgebung[6] das Geschenk und der WebSphere Application Server das entsprechende Komplementärprodukt. Software-Entwickler, die mit Eclipse vertraut sind, steigen auf WebSphere Studio um, wenn sie ein mächtigeres Werkzeug benötigen. Müssen diese Entwickler in der Folge die IT-Verantwortlichen beraten bei der Wahl des Applikations-Servers, werden sie die Produktivitäts- und Kosten-Vorteile auf Grund der engen Integration mit einem WebSphere Application Server vorrechnen. Eine Marktführerschaft von Eclipse steigert also die Akzeptanz der WebSphere Studio Entwicklungswerkzeuge, und diese wiederum öffnet die Tür für die Anschaffung des WebSphere Application Servers. Mit jedem Verkauften Applikations-Server sichert sich IBM einen langfristigen Ertragsstrom, womit die Kosten für die Offenlegung des Eclipse-Quellcodes problemlos abgeschrieben werden können.

Wenn mit der Wahl des Applikations-Servers der Kampf um die IT-Infrastruktur eines Unternehmens gewonnen werden kann, dann lohnt sich die Lancierung einen Open Source Produkts, welches die Chancen des eigenen Applikations-Servers entscheidend erhöhen, allemal.

2.4. Schlussfolgerung

In diesem Kapitel haben wir drei Strategien diskutiert, mit denen IBMs Open Source Strategie im Falle von Eclipse erklärt werden kann.

Unserer Meinung nach kann die Strategie von Eclipse als Komplementärprodukt das Handeln von IBM am besten erklären. Dies aus dem Grund, weil der Nutzen dieser Strategie im Erfolgsfall gut abzuschätzen ist. Auch mit Eclipse als Technologiestandard kann das Verhalten von IBM gut verstanden werden, allerdings ist der monetäre Nutzen dieser Strategie, falls sie erfolgreich ist, schwieriger zu bewerten. Auch die Strategie von „Innovation Happens Elsewhere" verhilft zu interessanten Einsichten, wenn sie auf den Eclipse-Fall angewendet wird. Allerdings ist zu bemerken, dass wir keine Hinweise gefunden haben, dass IBM diese Strategie verfolgen würde. Auch ist IBM nicht dafür bekannt, sich auf eine Kernkompetenz zu konzentrieren.

[6] „Basically, Eclipse is the open source version of the WebSphere Studio Workbench" (Handy in Jernigan 2001, S. 2).

3. Anhang A: Eclipse Bord of Directors

Firma	Eintritt	Bemerkung
Borland Software Corp.	November-01	
IBM	November-01	
Merant	November-01	
QNX Software Systems Limited	November-01	
Rational Software	November-01	Ab 06.12.2002 IBM Tochter
Red Hat, Inc.	November-01	
SuSE	November-01	
TogetherSoft	November-01	Jetzt Borland Tochter
WebGain	November-01	Webgain ist heutzutage nicht mehr Mitglied des Eclipse-Konsortiums
Fujitsu Software Corp.	Mai-02	
Serena Software, Inc.	Mai-02	
Sybase, Inc.	Mai-02	
Hitachi, Ltd. Software Division	Juni-02	
Instantiations, Inc.	Juni-02	
MontaVista Software	Juni-02	
Scapa Technologies Limited	Juni-02	
Telelogic	Juni-02	
ETRI	September-02	
Hewlett-Packard	September-02	
MKS Inc.	September-02	
SlickEdit Inc.	September-02	
Oracle	November-02	
Catalyst Systems Corporation	Dezember-02	
Flashline Inc.	Dezember-02	
Parasoft Corporation	Dezember-02	
SAP AG	Dezember-02	
Teamstudio, Inc.	Dezember-02	
TimeSys Corporation	Dezember-02	

Ericsson	März-03
Logic Library, Inc.	März-03
M7	März-03
QA Systems	März-03
SilverMark, Inc.	März-03
Advanced Systems Concepts	Mai-03
CanyonBlue Inc.	Mai-03
Ensemble Systems Inc.	Mai-03
Genuitec	Mai-03
INNOOPRACT Informationssysteme GmbH	Mai-03
Intel Corporation	Mai-03
Micro Focus	Mai-03
Tensilica Inc.	Mai-03
Wasabi Systems, Inc.	Mai-03
Codefarm	Juli-03
Embarcadero Technologies	Juli-03
SAS	Juli-03
Metanology	September-03

(Quelle: www.eclipse.org/org/main.html)

4. Anhang B: Aussagen der Gründungsmitglieder

Simon Thornhill, Borland: “Borland is pleased to be a founding board member of the Eclipse.org consortium, and looks forward to working with other industry leaders to establish open standards.”

Andreas Weiss, Merant: “We're very impressed by the power and flexibility of the Eclipse Platform."

Dan Dodge, QNX Software Systems: “Embedded developers need an extraordinary range of tools, but to be truly productive, they need tools that can work together in a seamless, intuitive fashion. With the Eclipse Platform, it’s now much easier for developers and tool vendors to integrate their rich toolsets into a cohesive whole”

Dave Bernstein, Rational Software: ”We’ve been working closely with IBM to integrate our products with Eclipse to ultimately provide a single, integrated user experience for developers and other practitioners on a software team.”

Michael Tiemann, RedHat: “As the open source community grows, we need open source development tools that meet the needs of more and more developers.”

Jürgen Geck, SuSE: ”The success of Linux and the Open Source computing model has changed the IT landscape. As this change matures, we see new open systems that extend the power of open development. SuSE is proud to be an active part in the evolution of the Eclipse Platform.”

Todd Olsen, TogetherSoft: ”Our mission of 'improving the ways people work together' is certainly embraced in the spirit of the Eclipse Platform.”

(Quelle: www.eclipse.org/org/pr.html)

5. Literatur

Adams, Greg; Marc Erickson (2001), Working the Eclipse Platform, http://www-106.ibm.com/developerworks/library/os-plat/

Adams, Greg and Rawn Shah (2002), A View Under The Hood: How Eclipse Work, http://www.iapplianceweb.com/story/OEG20020304S0072

Brody, Steve (2001), Interview: The Eclipse code donation. IBM's Marc Erickson gives the details on Eclipse, http://www-106.ibm.com/developerworks/java/library/l-erick.html

Conte, Paul (2002), Inside Eclipse and the WebSphere Studio Family, http://www.e-promag.com/eparchive/index.cfm?fuseaction=viewarticle&ContentID=1708

Gabriel, Richard P. and Ron Goldman (2002), Open Source: Beyond the Fairytales, http://opensource.mit.edu/papers/gabrielgoldman.pdf

Gamma, Erich (2003), Eclipse, more than a Java IDE, http://www.wit.at/events/nachlese/eclipse/eclipse_WIT2003.pdf

Garud, Raghu; Sanjay Jain & Arun Kumaraswamy (2002), Institutional Entrepreneurship in the Sponsorship of Common Technological Standards: The Case of SUN Microsystems and Java, http://opensource.mit.edu/papers/sunjavagarud1.pdf

Hardin, Garret (1968), The Tragedy of the Commons. Science, 162:0, S. 1243-48.

Hecker, Frank (1999), Setting Up Shop: The Business of Open-Source Software. IEEE Software, 16:1, S. 45-51 (siehe auch http://www.hecker.org/writings/setting-up-shop.html)

Eclipse.org (2003), Eclipse Project Charter - v0.52, http://www.eclipse.org/eclipse/eclipse-charter.html

Eclipse.org (2003), Who's running this project? http://www.eclipse.org/eclipse/team-leaders.html

Jerason (2003), Jerason's Java IDE survey, http://datadino.com/forum/viewtopic.php?t=4

Jernigan, Rayme (2001), IBM's Scott Handy explains why Eclipse means more Linux applications, http://www-1.ibm.com/linux/news/handy.shtml

Karpinski, Richard (2002), Open Source Tools: A Look At Eclipse, http://www.theopenenterprise.com/story/TOE20021007S0005

Lentzsch, Karsten (2002), Der Plattform etwas beibringen. Javamagazin, 2002:2, S. 16-17.

Lentzsch, Karsten (2002), Licht ins Dunkel. Javamagazin, 2002:2, S. 44-56.

Mauss, Marcel (1950), Essai sur le don, Paris (Presses Universitaires de France).

Olson, Mancur (1965), The Logic of Collective Action. Public Goods and the Theory of Groups, (Cambridge).

OTI (2003), Eclipse Platform Technical Overview, http://www.eclipse.org/whitepapers/eclipse-overview.pdf

QA Systems (2003), Java IDE Market Share Survey, http://www.qa-systems.com/products/qstudioforjava/ide_marketshare.html

Raymond, Eric S. (1999), The Magic Cauldron, http://www.catb.org/~esr/writings/magic-cauldron/

Smith, Tom (2002), IBM Builds An Open-Source Ecosystem, http://www.theopenenterprise.com/story/TOE20021114S0001

Steppan, Bernhard (2003), Sunblocker. iX, 3:12, S. 48-52.

Wartala, Ramon (2003), Sonnenflecken. iX, 3:12, S. 54-59.