5 Epilog

So klein das Programm auch aussehen mag, soviel Arbeit und Energie hat es doch gekostet. Wenn nichts bekannt ist, weder die Sprache(n), noch die Ablauflogik(en) noch die diversen Gepflogenheiten und Konventionen und wenn so gar keine „kleinen und großen“ Tricks zur Verfügung stehen ...

... dann sieht alles reichlich trübe aus.

Doch Ruby ist wirklich eine saubere Sache und dasselbe gilt für Rails – beide nehmen dir viel Aufwand ab und mit ein bisschen Übung gewöhnst du dich sogar daran, zuerst die einfachste Lösung auszuprobieren.

Und dann fängt es an, richtig Spaß zu machen. Gewiss, eines muss klar sein – Einarbeiten in das ganze System „so mal nebenbei“ geht nicht. Mir ging einiges an Zeit verloren, bis ich einsah, dass es sich nicht so ein, zwei stundenweise machen ließ, möglichst noch am Abend, wenn dein Kopf sowieso auf Halbmast flaggt – ich musste mir schon größere Zeitscheiben vom Tag abschneiden, an denen am Stück gearbeitet werden konnte. Und selbst dann dauerte es zwei bis drei Wochen, bis dieses verlorene Gefühl der Ahnungslosigkeit sich langsam legte und der alte Spieltrieb wieder Oberhand gewann. Bei Ruby gelang dies tatsächlich bereits während „Schritt 2“.

Denn wie gesagt:

Ruby ist einfach himmlisch.

Mit Rails zusammen ist es eine leicht erlernbare Grundlage der datenbank-basierenden Web-Programmierung für alle, die gerne programmieren – oder die rasch vorführbare Ergebnisse zeigen müssen.

Und alle, die mehr als nur einfaches statisches HTML/CSS beherrschen, die schon in die modernen Skriptsprache wie Perl, PHP oder Python hineinschnupperten, für die „Regular Expressions“ kein Fremdwort sind und die vor allem die Art und Weise kennen, mit offenen Komponenten und Frameworks umzugehen ...

... für all die wird es längst nicht so viel Zeit und Energie kosten, sich in Ruby und Rails einzuarbeiten.

Es lohnt sich nämlich.

Und zum Abschluss noch ein Wort an die Zauderer, die lieber die Finger lassen wollen von dem „unreifen Rails“:

Nicht alles, was sich schon im nächsten Quartal bezahlt machen kann – überlebt auch die nächsten fünf Jahre.

Was wäre, wenn ein Carl Benz oder ein Werner Siemens nur 90 Tage weit gedacht hätten?

Sicher ist Rails noch jung – doch Ruby gehört seit Jahren zu den verbreitesten Skriptsprachen in Japan, das zwar sehr technikverliebt ist, aber sicher nicht als unprofessionell abgekanzelt werden kann.

Und so nebenbei bemerkt: Ist es nicht merkwürdig, dass Industrien, die Alte Hasen als überholt ansehen und nur das Junge anbeten – genau diese „Jugend“ gerne als Gegenargument benutzen, um etwas **nicht** anzuwenden?

Das geschieht leider gerade in Deutschland – ein paar Leute, die sich auf amerikanischen Foren wohl fühlen, hören etwas von Rails, probieren es kurz aus und werfen es wieder weg.

Weil der Debugger noch nicht akzeptabel ist (ja, das ist er tatsächlich noch nicht) ...

... weil sie die Performanz noch nicht so beherrschen, wie sie sich das vorstellten ...

... weil hier noch ein Fehler entdeckt wird oder da noch einer ...

oder mit klaren Worten ausgedrückt:

Weil es nicht das ist, was sie immer schon gemacht haben ...

... weil Dampfmaschinen eben doch viel mehr können als das neumodische Elektrozeug, Herr Siemens, sehen Sie sich nur die Eisenbahn an!

> Werner Siemens, Wikipedia:
> Siemens glaubte fest an den Siegeszug der elektrischen Energie, der mit der Dynamomaschine möglich erschien. Aber es gab noch zu wenig praktikable Anwendungen, um der neuen Technologie zum Durchbruch zu verhelfen.

Doch für jede Technik gibt es ein Zeitfenster, in dem sie gedeihen kann – die Dampfmaschinen haben ihres hinter sich, dafür treiben Elektromotoren jetzt die Eisenbahnen an.

> „Bringing Ruby to the Enterprise", Brad Banister, 2006
> While Ruby isn't ready to be used in every aspect of the enterprise, it can fit into many areas of an application strategy. By doing so a business can reap the productivity gains from the agility that the Ruby platform offers.

Ruby on Rails hat das Zeitfenster des großen Erfolges noch vor sich – in Amerika beginnt es bereits, in die Software-Häuser einzudringen ...

... **auch** gegen den Widerstand vieler Experten in diesen Firmen. Aber die Eleganz von Ruby und der pragmatische CoC-Ansatz von Rails bieten einfach zuviele gute Chancen für die ständig steigenden Anforderungen an eine Industrie, die erst noch lernen muss, vielseitig verwendbare Einzelteile zu bauen. Unerlässlich dafür sind nämlich Normen („Conventions"), die die Voraussetzung dafür bieten müssen, dass die Einzelteile „anonymisiert" zu größeren Anlagen zusammengefügt werden können.

„Old Economy" lässt grüßen! Früher wurden Schrauben auch für den benötigten Einzelfall konstruiert und sicher gibt es heute auch noch teure Stücke, für die Schrauben individuell gefertigt werden ...

... doch wo wären wir im Maschinenbau, wenn sich Schrauben nicht über DIN-Normen günstig produzieren und anhand ihrer „Schnittstellen", eben dieser DIN-Normen, präzise in Konstruktionen integrieren ließen, ohne dass jeder Einzelfall „schraubentechnisch" geprüft werden müsste?

Genau das ist es, was die Zukunft des Web 2.0 fordert: normierte Einzelteile, die sich perfekt über ihre Schnittstellenbeschreibungen definieren und deshalb vielseitig verwenden lassen.

Aber die großen Firmen wie SAP®, Oracle®, Microsoft® und IBM® werden es vermutlich nicht schaffen, die Anforderungen für Web 2.0 zu liefern, denn um Normen nützlich zu machen, müssen sie generell sein, müssen sie allgemein anerkannter Standard sein. Doch nicht nur Microsoft beweist tagtäglich, dass es keine Standards außer den eigenen akzeptiert.

Für Mashups, die auf dem Internet per Drag & Drop zusammensetzbar werden – was exakt der Traum von Web 2.0 ist – dürfen jedoch keine Grenzen existieren. Grenzen bauen nur Barrieren auf. Wenn jeder Service tausend Adapter benötigt, um sich an die jeweiligen Platzhirsche anbinden zu können, dann werden sich eben auch nur Platzhirsche die Erzeugung von Services leisten können.

Das Problem solcher Services ist aber, dass sie vor allem dem Gesetz der Aktionäre unterliegen, die Kundennutzen oder die Wartbarkeit und Qualität der Produkte und Betreuung höchstens sekundär interessiert. Da ist kaum Platz für produktorientierte Kreativität, kaum Platz für fundamentale Innovationen jenseits des RoI, denn diese brauchen Offenheit, keine Grenzen, weder technische noch bilanzielle. Information ist immer eine Frage der Zeit und damit ist auch Wissen immer eine Frage der Zeit – und neues Wissen braucht deshalb schlicht Muße und Freiraum, eine Spielwiese ohne Beschränkungen und Scheuklappen, denn was aus Neuem werden kann, entscheidet nicht der Plan: Das entscheidet ausschließlich die Zugänglichkeit, der Nutzen und die Zeit.

Das Web 2.0 wird deshalb nur dann eine Chance haben, wenn Open Source überlebt – nur die „Spielwiese“ Open Source hat in den letzten Jahren den Fortschritt vorangetrieben. Der Internet Explorer dürfte dafür das prächtigste Beispiel sein: Erstellt, um Netscape zu bekriegen (der berühmte „Browser War“), verharrte er auf seinem frühgeschichtlichen Entwicklungszustand, bis Firefox auftauchte.

Auch SOA konnte erst in das Bewusstsein der Führungen dringen, als Open Source auf dem Internet die Webservices längst zum Alltag gemacht hatte. Erst dann wurde deutlich, was es für Vorteile bringen kann, wirklich unabhängige Blackboxes zu erzeugen und zu großen Einheiten zusammenzufügen.

Zuvor war die typische Antwort immer gewesen:

> Das haben wir schon immer so gemacht.

Deshalb werden die großen Firmen Ruby oder Rails vielleicht niemals lieben lernen. Denn sie wollen nicht wirklich das Web 2.0, das Kunden rebellisch macht und jedem eine Chance gibt: Alten Hasen, jungen Kreativen, Außenseitern ...

Web 2.0, die Chance für alle, die bescheiden genug für den „Long Tail“ sind, den kleinen Markt von Spezialinteressen, der von Massenproduktion nie bedient werden wird und der sich auch nicht auf Millionärsmessen verlustieren kann: Der Markt des „Long Tails“ taugt oft nicht für hohe Profite, aber er taugt für Innovation, für Vielseitigkeit, für Mashups aus kleinen, billigen und sehr einfachen Grundkomponenten ...

... Mashups, die eine gemeinsame Basis brauchen – eine Basis, so weit verbreitet, dass sich auch Schüler und Studenten, Hobby- und Freizeitprogrammierer darauf tummeln können ...

... um auf diese Art viel Streu zu produzieren ...

... aber eben auch Weizen der feinsten Art.

Das Zeitfenster dafür tut sich gerade auf. Und Ruby, mit seiner informationsnahen Struktur, mit dem tiefen Verständnis für informative Prozesse, das sich in seiner Art, Iteratoren als etwas zu sehen, was nichts in der Architektur der Software verloren hat (wie es die ML-Methode verlangt) – Ruby, das sich selbst nicht so wichtig nimmt, am eigenen Code Änderungen erlaubt und Anbindungen an andere Entwicklungssprachen leicht macht, das Metaprogrammierung und Individualität fördert, dieses Ruby hat genau durch die intuitive Nähe zur Information „evolutionär“ die besten Chancen, sich aus der Masse der Skriptsprachen hervorzuheben. In Japan ist das längst der Fall und bei den neuen Projekten von Apache findet sich Ruby bereits in den vorderen Reihen der aufgezählten Sprachen.

Auch Rails weist mit CoC, DRY (Don't Repeat Yourself) oder seiner Art, business model und Datenbank simpel und überschaubar zu korrelieren, ein intuitives Verständnis für informative Prozesse auf.

Die daraus folgende leichte Erlernbarkeit, der kompakte Code selbst bei einem komplexen Umfeld wie einer browserbasierenden Oberfläche, die Möglichkeit, günstige Middleware wie Apache oder MySQL verwenden zu können, die Begeisterung der Jugend, die Ruby und Rails gerade in den westlichen Ländern vorantreibt ...

... das alles sind die besten Voraussetzungen für eine Basis, die Web 2.0 tatsächlich ermöglichen kann.

Aber wohl nicht im nächsten Quartal.

Und nicht, wenn die Lösung immer nur „von den anderen" erwartet wird. Was Information, die Grundlage jeglicher Informationstechnik, exakt ist, wissen bis heute weder die großen Firmen der Informationsverarbeitung noch die Wissenschaftler von Harvard oder Stanford – und auch Linux® oder Ruby ist nicht „auf ihrem Mist" gewachsen.

Die großen Errungenschaften müssen nicht immer von den „üblichen Verdächtigen" abgeliefert werden. Wer darauf wartet, kann vielleicht bis zum St. Nimmerleinstag warten – und verschenkt sich damit möglicherweise seine eigene Zukunft.

Deshalb ist dieses Buch besonders an die Alten Hasen gerichtet, die um jedes bisschen Zukunft ringen müssen, die täglich gegen die Zeit, gegen die Vorurteile des amerikanischen Way of Business ankämpfen, dass nur die Jugend Flexibilität und Erfindungsreichtum bieten kann – die gezwungen sind, ihre veralteten Arbeitsformen über Bord zu werfen und neue Wege einzuschlagen, weil sie wissen, dass ihre Firmen, die heute vielleicht noch prosperieren mit den proprietären Methoden, morgen genauso am Ende sein können wie das namenlose Heer derjenigen, die die letzten Katastrophenjahre der IT nicht überstanden haben.

Es ist aber auch an diese ihre Firmen gerichtet, die neue Wege brauchen, um ihre Software auf die kommende Zeit auszurichten, damit sie den gewachsenen Anforderungen standhalten kann.

Nur der „Long Tail“ und die Nischenmärkte bieten Platz für die kleinen Software-Häuser und Einzelkämpfer. Die aber brauchen günstige Entwicklungs- und Ablaufumgebungen. All das kann Ruby und Rails bieten und je mehr auf den Zug aufspringen, desto eher werden auch die anfänglichen Mängel schwinden.

Wer aber wartet, bis Amerika es wieder einmal vormacht ...

... der lässt sich dadurch auch die Butter vom Brot nehmen.

Rails ist dort, wo Java vor 10 Jahren war? Technisch und personell? Aber Java lässt sich immer noch nicht leicht erlernen und seine Frameworks überdecken sich tausendfach, sodass Java-Programmierer und Java-Programmierer nicht wirklich dasselbe sind. PHP ist viel weiter verbreitet? Aber auch nur, weil es von Einzelkämpfern wegen seiner Einfachheit und Webtauglichkeit trotz anfänglicher Mängel vorangetrieben wurde. Interfaces gibt es nicht in Ruby? Ja, auch dieser Einwand war zu hören.

Wer von kurzfristigen Projekten lebt, mag Ruby vorläufig noch links liegen lassen – wer aber Jahre investieren muss, um die vorhandene Software zu migrieren, sollte heute damit anfangen.

Denn Ruby und Rails werden sich selbst einen Markt schaffen und damit an Reife gewinnen, dafür sorgen schon die „hungrigen“ amerikanischen Firmen, die ihre Chance genau im Neuen sehen, genau dort sehen, wo die Konkurrenz noch nicht groß ist und sie selbst die Vorreiterrolle übernehmen können.

Das Zeitfenster für Ruby und Rails tut sich gerade auf.

Doch ob Deutschland davon profitieren kann, hängt nicht an Amerika – das hängt nur an deutschen Entwicklern, die jetzt Entscheidungen treffen müssen, die weit über das nächste Quartal hinaus gehen werden.

Denn was wäre, wenn ein Carl Benz oder ein Werner Siemens nur 90 Tage weit gedacht hätten?