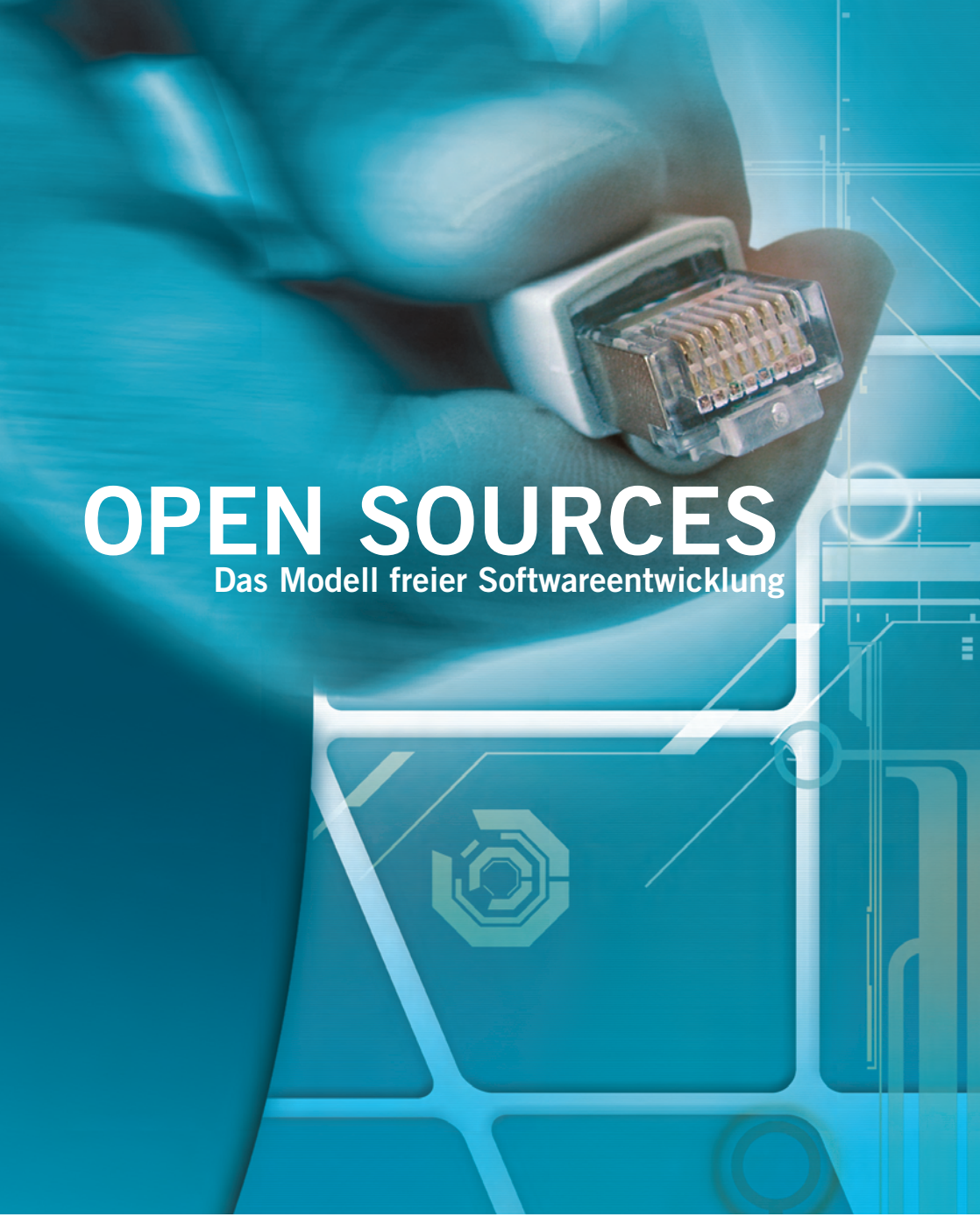


OPEN SOURCES

Das Modell freier Softwareentwicklung

iframe

IT-Solutions and Consulting AG

Inhaltsverzeichnis

Warum kostenlose Software trotzdem gute Software sein kann

Kleine Modellanalyse der Open Source Entwicklung nach dem Anreizprinzip

1. **Annäherung an ein neues Modell**
2. **Selbstorganisation versus Anarchie**
   - 2.1. Anreize bei der «freien» Mühle
   - 2.2. Die «proprietäre» Mühle
3. **Leistung, Motivation und Organisation**
   - 3.1. Bedürfnisse vieler als Integrationsansatz
   - 3.2. Entwicklung als systemimmanenter Faktor
   - 3.3. Intrinsische Motivation der Leistungsersteller
   - 3.4. Hierarchie und Heterarchie als Organisationsprinzip
   - 3.5. Die Umkehrung der ökonomischen Logik
4. **Das Modell der Open Source Entwicklung**
   - 4.1. Lizenzen – Urheberrechte – Wirkungsabsicht
   - 4.2. Die Bedürfnisse vieler als Integrationsansatz
   - 4.3. Entwicklung als systemimmanenter Faktor
   - 4.4. Intrinsische Motivation
   - 4.5. Hierarchie und Heterarchie
5. **Was bringt die Zukunft?**

Glossar

Open Source Software – die wichtigsten Begriffe

Als ich 1998 im Rahmen eines Seminars in Strategie und Organisation an der Universität St. Gallen das Wertschöpfungsmodell von Open Source Software vorstellte, stiess ich auf wenig Verständnis. Die Gesichter von Studenten und Dozenten zeigten alle Facetten menschlichen Widerspruchs, von totaler Konfusion bis hin zu klarer Ablehnung. Es schien unfassbar, dass im Internet wettbewerbsfähige Produkte «einfach so» entstehen können – ohne klare Organisationsstrukturen, ohne Managementmodell, ohne Finanzcontrolling. Als ich dann auch noch erwähnte, dass die Produkte für jeden kostenlos zum Download bereitstehen, zerschellte das Modell an den monetär-ökonomisierten Weltbildern der Anwesenden: So etwas war undenkbar und konnte wissenschaftlich nicht ernstgenommen werden.

In der Zwischenzeit ist Open Source Software in der deutschen Bundesverwaltung angekommen, und die EU-Kommission denkt laut darüber nach, Software nur noch nach diesem Modell entwickeln zu lassen. IBM hat eine halbe Milliarde in die Entwicklung investiert, und sogar Microsoft nimmt Open Source Software heute ernst. Open Source Software ist mein täglich Brot. Die iframe IT-Solutions and Consulting AG verdient ihr Geld mit der Weiterentwicklung, Anpassung und Implementation von Produkten, die das Ergebnis eines neuen und faszinierenden Wertschöpfungsmodells sind. Die folgende Broschüre stellt Ihnen dieses Modell genauer vor. Die Lektüre lohnt sich – Sie werden überrascht sein.

Michael Jäger,
Geschäftsführender Partner
iframe IT-Solutions and Consulting AG, Widnau

Warum kostenlose Software trotzdem gute Software sein kann

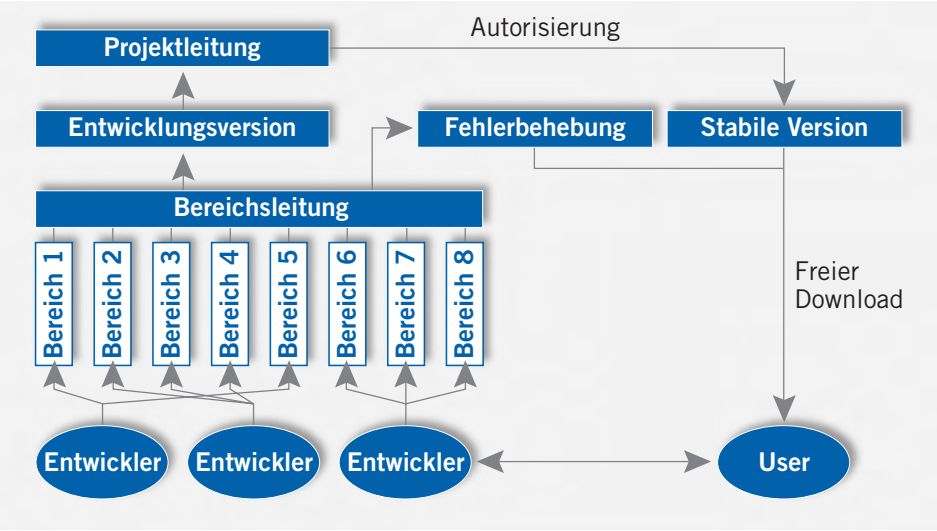
Grosse Softwarekonzerne verzeichneten im letzten Jahr trotz weltweiter Konjunkturknicks und des 11. Septembers satte Milliardengewinne. Firmen sind zunehmend abhängig von Standardsoftware geworden und brauchen – Wirtschaftsentwicklung hin oder her – neue Versionen, Upgrades und Lizenzen. Microsofts neues Lizenzmodell «Software Assurance» verursacht Unternehmen nun auch noch bis zu 200% Mehrkosten. In einer weltweiten Studie (Sunbelt Software Inc, März 2002) gaben 38% der befragten Unternehmen an, sich nun nach anderen Lösungen umzusehen. Widerstand regt sich, doch scheinen Alternativen spärlich gestreut.

Open Source Software ist eine vollwertige Alternative und eine erfolgreiche dazu. Sie deckt nahezu den vollen Leistungsbereich vom Server über die Datenbank bis zum Desktop ab. Bereits 44% der deutschen Unternehmen setzen Open Source Software ein – nicht nur weil sie kostenlos ist.

Was ist Open Source Software?

Zu Beginn der neunziger Jahre stand Linus Torvalds, ein finnischer Student, vor einem Problem. Er brauchte ein einfaches, aber stabiles Betriebssystem, das möglichst günstig, am besten kostenlos sein sollte. Also entwickelte er es kurzerhand selbst. Zufrieden mit seiner Arbeit, stellte er das Ergebnis ins Internet; nicht nur das Programm, sondern auch jede einzelne Code-Zeile, den sogenannten Quell- oder Source-Code, den es zur Funktion benötigt. Jeder sollte das Betriebssystem mitsamt dem Bauplan kostenlos herunterladen, benutzen, weitergeben, verändern und entwickeln können – eine Revolution in der Softwareentwicklung. Und die Leute kamen scharenweise.

Über kurz oder lang begannen Benutzer Vorschläge zu machen, hinterliessen im Internet ihre eigenen, verbesserten und veränderten Versionen des Systems. Um diesem Wildwuchs die Stirn bieten zu können, beschloss Linus Torvalds, jeweils eine von ihm autorisierte Version anzubieten. Wer hierzu eine Entwicklung beisteuern wollte, musste auf seinen Segen hoffen. Von nun an arbeiteten Programmierer unter Leitung eines Verantwortlichen nach klaren Regeln an einer Entwicklungsversion. Nur durch die Projektleitung freigegebene Versionen wurden den Benutzern zum Einsatz empfohlen und regelmässig durch Fehlerbehebungsprogramme ergänzt.

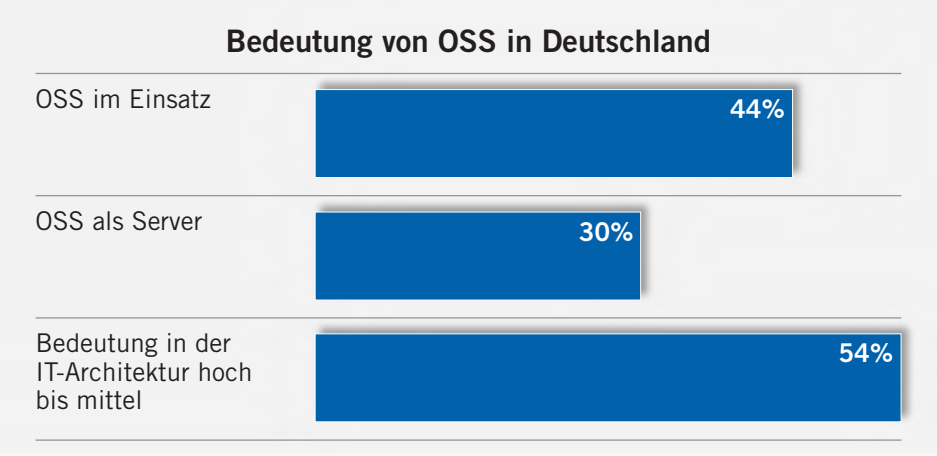


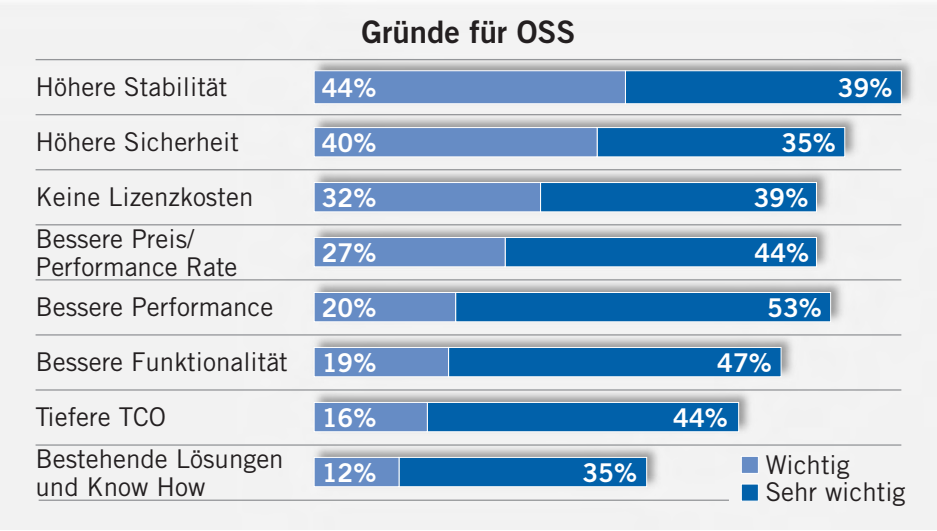
Entwicklungsmodell von Open Source Software

Nicht ganz so einfach, aber ungefähr so entstand Linux, die erfolgreichste Open Source Software. Seither werden regelmässig neue, verbesserte Versionen veröffentlicht, Zusatzsoftware wird entwickelt und Server um Server erobert. Es ist eine funktionierende Variante zur klassischen Softwareentwicklung entstanden – und mit ihr ganze Industriezweige. Das Modell fand bald auch in anderen Bereichen Verwendung. Mail-, Text- und Bildverarbeitungsprogramme mussten entwickelt werden, die User brauchten eine graphische Benutzeroberfläche, und das Internet forderte immer weitere, noch komplexere Anwendungen. Heute findet sich für fast jede «Standardsoftware» ein kostenloser Ersatz. Open Source Software bietet ausgezeichnete Qualität, ohne Bindung an eine Firma, ohne Lizenz-Kosten und ohne Abnahmeverpflichtung. Die Entwicklung und Verbreitung schreitet so rasant voran, dass sich immer mehr Freunde, aber auch Feinde finden. Trotz der überzeugenden Marktleistung und der kostenlosen Verfügbarkeit gilt Open Source Software hierzulande gerade bei Führungskräften noch als exotisch und unausgereift. Höchste Zeit, sich vertieft mit dem Thema auseinanderzusetzen.

Bedeutung von Open Source Software

Im Juli 2002 veröffentlichte die Berlecon Research Group eine Studie über die Bedeutung von Open Source Software in Deutschland, Schweden und England. Ziel des Auftraggebers, der Europäischen Komission, war es, harte Daten bezüglich der Wichtigkeit und der Rolle von Open Source in der Wirtschaft als Grundlage für politische Entscheide zu bekommen. Die Ergebnisse sind überraschend deutlich ausgefallen.





Quelle: Berlecon Research Company, 2002

Bereits 44% aller befragten deutschen Unternehmen, sowohl for- als auch non-profit, setzen Open Source Software ein – grösstenteils als Serverbetriebssystem. Für mehr als die Hälfte ist sie ein wichtiger Bestandteil der IT-Architektur.

Nach den Gründen befragt, gaben rund 80% höhere Stabilität und Sicherheit bei Open Source Produkten an. Die Lizenzkosten und die Total Cost of Ownership sind nur von nachrangiger Bedeutung. Bei den Unternehmen steht die Funktionalität der Software im Vordergrund, die nicht nur System-Administratoren zu überzeugen vermag, sondern zumindest in Deutschland auch für Entscheidungsträger zum relevanten Faktor wird. Hierzulande kämpft Open Source Software hingegen immer noch mit einem Akzeptanzproblem.

Die Probleme des Exoten

Die meisten PC-Benutzer interessiert es herzlich wenig, wieviele Code-Zeilen hinter einem Programm stecken. Schön, wenn die Software funktioniert, oftmals aber ärgerlich, wenn sie es nicht tut. Mit Programmen lässt sich dennoch viel Geld verdienen, vorausgesetzt man besitzt eine Marketing-Begabung vom Schlage eines Bill Gates, ganz egal ob der Code nun gut oder schlecht ist. Für den Anwender bleibt Software eine Black Box. Es ist überraschend irrelevant, wie stabil oder sicher ein Programm ist, solange der Käufer nur vom Nutzen überzeugt werden kann. Eine zentrale Rolle in dieser Nutzeneinschätzung spielt der Preis: Günstig soll es schon sein, aber gratis? Was nichts kostet ist auch nichts wert, lautet gemeinhin die Überzeugung.

Open Source Software ist gratis. Es stehen auch keine Millionenetats zur Verfügung, um Anwender mit hübschen Plakaten und ausgefeilten Kampagnen vom Wert der Produkte zu überzeugen. Mit Linux ist ein zentraler Vertreter der Gattung Open Source

Software in die Schlagzeilen gekommen. Ein bisschen Euphorie zu Beginn, als das Thema noch neu war, immer wieder Meldungen, wenn Open Source Produkte einen weiteren Prozentpunkt an Marktanteilen erobert hatten, dicke Schlagzeilen, als Microsoft die Open Source Entwickler zur quasi-terroristischen Vereinigung erklären liess. Es wurde viel über Open Source geredet und geschrieben, aber selbst sprach sie nie, weil sie weder eine Firma noch eine Organisation ist, sondern purer Code. Es gibt keine Pressestelle, die für Open Source Software mit einer Stimme sprechen könnte.

Software hat wie jedes Produkt ein «Look and Feel». Nicht nur die Verpackung, sondern auch die Benutzeroberfläche lassen den Anwender spüren, wessen Produkt er gerade verwendet. Vom Klang bei der Windows-Anmeldung bis hin zu Adobes typischen Desktop-Symbolen sind die Anwendungen durchgestylt. Das «Look and Feel» der meisten Open Source Programme besitzt die Ästhetik eines DOS-Fensters. Zwar nähern sich die Benutzeroberflächen seit einiger Zeit Stück für Stück dem Aussehen und den Strukturen der verbreiteten kommerziellen Systeme an, doch fehlt eine greifbare und konsequente Design-Linie.

Open Source Software passt so gar nicht in die Welt des IT-Management. Es gibt keine Anbieter, die neue Versionen gewährleisten, keine Firmen, die man bei Mängeln verklagen könnte, keine Aussendienstmitarbeiter, die bei Problemen auftauchen. Open Source Software ist nur Produkt. Gerade deshalb ist es wichtig zu verstehen, wie es möglich sein kann, dass gute Produkte kostenlos sind. Open Source Software durchbricht traditionelle Formen der Wertschöpfung in der Informationstechnologie und schafft ein neues, praktikables Modell. Wir wollen den geneigten Leser im Folgenden einladen, dieses Modell besser verstehen zu lernen, die Motivation der Entwickler zu ergründen und Chancen und Grenzen der Open Source Entwicklung auszuloten.

Kleine Modellanalyse der Open Source Entwicklung nach dem Anreizprinzip

Software ist ein immaterielles Gut. Man kann Software nicht kaufen oder verschenken, sondern nur die Nutzungsrechte per Lizenz überlassen. Entsprechend funktionieren die Wertschöpfungsketten in der Softwareindustrie anders als in der Produktion klassisch materieller Güter. Zur Produktion von Software braucht es nur einen Computer und einen Programmierer. Mit dem Internet kommt ein Distributionsmechanismus hinzu, der für den Anbieter nahezu kostenlos ist. Die Kombination dieser Faktoren macht ein neues Wertschöpfungsmodell möglich, das völlig andere Anreizstrukturen beinhaltet als kommerzielle Methoden: die Entwicklung von Open Source Software.

1. Annäherung an ein neues Modell

Nehmen wir einmal folgendes an: Sehr viele Menschen haben ein ähnliches Bedürfnis. Dieses ist mit einem Produkt zu befriedigen, dass beliebig reproduzierbar ist; unter Berücksichtigung der Skaleneffekte gehen die Kosten gegen null. Diese Menschen tun sich nun zusammen, um dieses Produkt gemeinsam zu entwickeln. Sie einigen sich darauf, dass jeder berechtigt ist, es mitzuentwickeln und mitzunutzen. Damit muss das Produkt so flexibel sein, dass es auf jedes Bedürfnis individuell angepasst werden kann.

Mit ein bisschen Arbeitseinsatz erhält so jeder die optimale Lösung für seine Ansprüche. Nun geht man noch einen Schritt weiter. Bis anhin Unbeteiligte bemerken, dass dieses Produkt auch ihre Probleme lösen könnte. Da bis zu diesem Zeitpunkt nur der freiwillige Arbeitseinsatz Kosten produziert hat, können sie es gratis erhalten, einsetzen und weiterentwickeln; zumindest solange, wie sie bereit sind, ihre Entwicklungen auch wiederum gratis abzugeben. Das alles klingt ziemlich theoretisch und – offen

gesagt – auch ziemlich absurd. Niemand würde seine Arbeitskraft und sein Wissen verschenken. Immerhin widerspricht das jeder ökonomischen Rationalität, wie man sie von Kindesbeinen an gelernt hat. Wer etwas tut, soll auch etwas dafür bekommen. Und etwas bekommen bedeutet, etwas Geldwertes in der Hand zu haben. Dennoch funktioniert dieses Modell im Internet, in der globalisierten Softwareentwicklung vom Betriebssystem bis zur Sicherheitslösung. Und es entwickelt sich rasant!

2. Selbstorganisation versus Anarchie

Nehmen wir ein einfaches Beispiel, um die Theorie ein bisschen fassbarer zu machen. Eine mittelalterliche Dorfgemeinschaft hat ein gemeinsames Bedürfnis: Sie will das geerntete Getreide nicht zerkauen, sondern mahlen und weiterverarbeiten. Aber es lohnt sich für keinen, selbst eine Mühle zu bauen. In diesem Fall hätte niemand mehr die Zeit, seine Felder zu bestellen. Also beschliessen sie, dies gemeinsam zu tun. Jedes Mitglied der Dorfgemeinschaft leistet seinen Beitrag, sei es durch Arbeit an der Mühle, sei es durch das Ausrüsten und Verpflegen der Handwerker. Alle Materialien werden selber hergestellt, Steine gehauen und Nägel geschmiedet, Balken gesägt und Ziegel gebrannt.

Nach vielen Mühen steht die Mühle, und die Mitglieder der Dorfgemeinschaft können gemahlenes Getreide beziehen. Dem Müller geben sie jeweils einen Anteil von – sagen wir – einem Zehntel, um ihn für seinen Aufwand zu entschädigen. Aber jeder hat ein unbeschränktes Nutzungsrecht. Ein Idealbild – denn sehr bald kommt ein Trittbrettfahrer. Er hat nicht mitgeholfen zu bauen und mahlt dennoch sein Getreide. Die Dorfgemeinschaft könnte ihn ausschliessen. Aber erinnern wir uns an den Anfang: Sie wollten gemahlenes Getreide, möglichst hohen individuellen Nutzen und nicht Ertrag für Aufwand. Sie definieren den Sinn der Mühle gerade damit, dass der Nutzen für alle

Mitglieder der Gemeinschaft offen sein soll. Damit sind die Trittbrettfahrer nicht relevant für das Bedürfnis der Gemeinschaft, solange die Mühle ihr Getreide weiterhin für sie mahlt. Ansonsten würden die Dörfler den Betreffenden nicht nur vom Nutzungsrecht, sondern auch aus der Gemeinschaft ausschliessen.

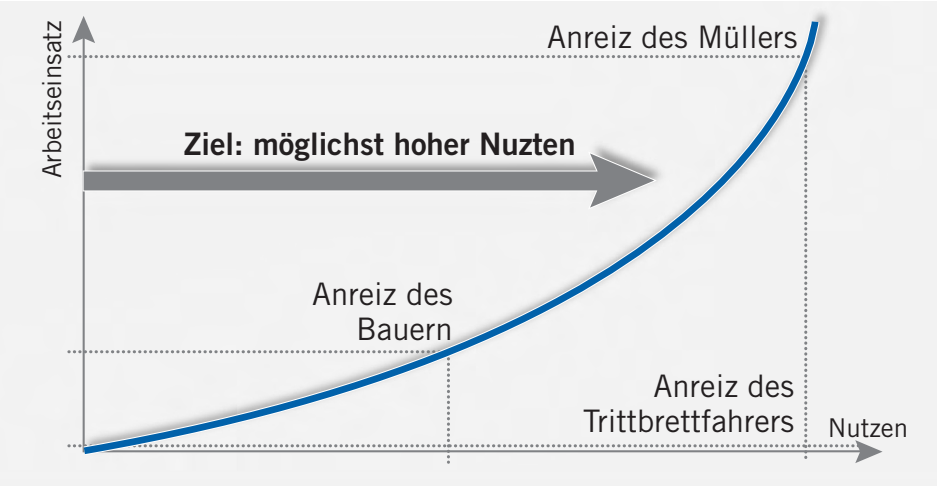
Reizen wir das Bild noch ein wenig weiter aus: Der Müller findet heraus, dass sich die Kraftübertragung mit Hilfe von Zahnrädern optimieren lässt. Diese Neuerung baut er in die Mühle ein und verlangt von der Dorfgemeinschaft nun 15%, weil man nicht mehr solange warten muss, bis das Getreide gemahlen ist. Was wird die Dorfgemeinschaft tun? Im besten Fall wird sie den Müller davonjagen, ansonsten wird sie ihm auf martialische Weise zeigen, was mit der Kraftübertragung noch so alles möglich ist. Alle haben gemeinsam für die Mühle gearbeitet. Wer also etwas verbessert, tut dies ebenso für alle, darauf hat man sich geeinigt.

2.1. Anreize bei der «freien Mühle»

Zurück in unser Jahrhundert. Nehmen wir die betriebswirtschaftliche Messlatte in die Hand und analysieren, was die Dörfler getan haben. Sie haben ein gemeinsames Bedürfnis identifiziert und gemeinsam befriedigt. Sie haben sich selbst organisiert und ein Modell geschaffen, wie es sich die moderne Mikroökonomie nicht besser ausdenken könnte. Nun, zumindest solange die Verarbeitungskapazität der Mühle nicht begrenzt ist. Aber Modelle brauchen Annahmen, sonst funktionieren sie nicht. Jeder Einzelne hatte den Anreiz, soviel zu arbeiten, wie es der Höhe seines Bedürfnisses entsprach, gemahlenes Getreide zu bekommen. Wenn der Trittbrettfahrer das Risiko zu tragen bereit ist, dass alle anderen plötzlich die Lust am Bauen verlieren, dann ist das seine Entscheidung. Er riskiert durch sein Verhalten, die Befriedigung seines Bedürfnisses niemals zu erreichen. Oder nehmen wir das andere Extrem: den Müller.

Er hat den grössten Anreiz, für die Errichtung der Mühle Aufwand in Kauf zu nehmen, denn sie sichert in Zukunft sein Einkommen. Also wird er so viele Balken sägen, wie er vermag, um möglichst schnell die Funktionsfähigkeit zu erreichen.

Es braucht also keinen Landvogt, der die Bauern zum Arbeiten zwingt. Die Dorfgemeinschaft organisiert sich selbst – und sie motiviert sich selbst. Wohl aber braucht es einen Zimmermann, der die Arbeiten koordiniert, jemanden, der das Endprodukt im Auge hat. Und die Bauern akzeptieren seine Führungsfunktion, weil sie sich im klaren sind, dass jemand den Überblick haben muss, damit die Mühle irgendwann funktioniert. Sonst können sie ihre eigene Mühle bauen, oder es zumindest versuchen. Jeder leistet also gemäss seinen individuellen Fähigkeiten. Das klingt nun schon weit weniger nach Anarchie, vielmehr nach Zielkonsens, nach Selbstorganisation operativer Einheiten und kooperativer Führung in Reinstform.



Anreizstrukturen Dorfgemeinschaft

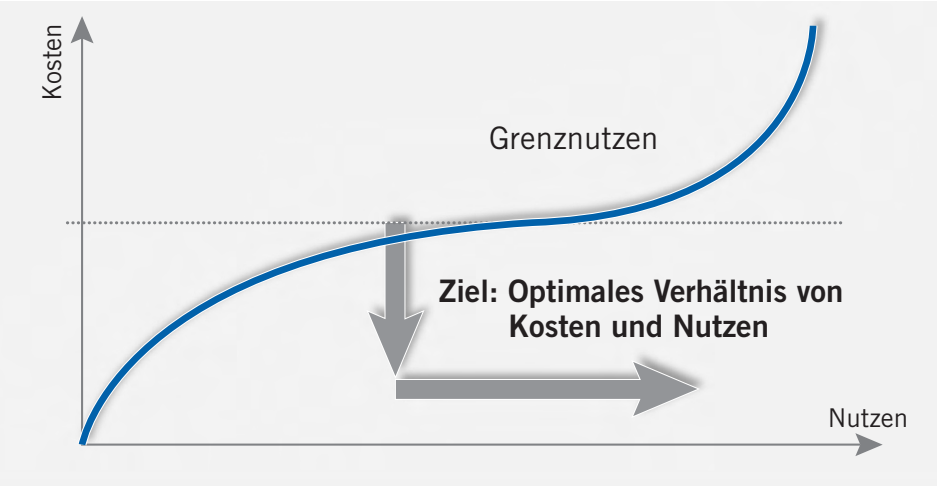
2.2. Die «proprietäre» Mühle

Gehen wir nun noch einmal in die finsteren Zeiten zurück: Was passiert, wenn ein findiger Handwerker auf die Idee kommt, all sein Hab und Gut zu verkaufen, um aus dem Erlös Tagelöhner zu bezahlen, die für ihn die Mühle errichten? Nehmen wir an, er sei ein Zimmermann. Er weiss also, wie man eine Mühle baut. Aber weiss er, wie man Steine behaut oder Nägel schmiedet? Wohl kaum. Aber er kann sie kaufen. Und das kostet. Er hat den Anreiz, so wenig von seinen Mitteln wie möglich zu verwenden, um letztendlich Einnahmen zu verzeichnen. Und sowieso, der Mauerstein ist ja schon da, man muss nur noch den Fels rundherum weghauen. Er leitet also seine Tagelöhner an, wie sie im Steinbruch zu funktionieren zu haben. Welchen Anreiz haben aber die Tagelöhner, gute Leistungen zu erbringen, selbst wenn sie Steinmetze sind? Sie haben das Bedürfnis, entlöhnt zu werden. Sie müssen also darauf schauen, dass der Zimmermann mit ihnen zufrieden ist. Und wenn er bei dem Mauerstein runde Ecken will, dann machen sie runde Ecken. Auch wenn sie wissen, dass einzig eckige Ecken Sinn machen. Dennoch steht das Gebäude irgendwann. Es knarzt ein wenig, aber es steht. Und man kann mahlen. Ein bisschen grob, aber man kann mahlen. Und die Leute kommen. Schliesslich müssten sie sonst in den Nachbarort und das wäre nicht zumutbar. Also mahlen sie beim Zimmermann.

Betrachten wir einmal das Fortschreiten nach Abschluss der Arbeiten. Ein Zimmermann weiss wie man eine Mühle baut. Aber weiss er wie man Getreide mahlt? Er kann nun einen Müller anstellen – und das wird er auch tun, doch dieser hat auch nur den Anreiz, bezahlt zu werden, nicht gut zu mahlen. Im Gegenteil: Verlangt er Verbesserungen im Prozess, kostet das den Chef bares Geld. Also ist auch er ein Tagelöhner, nichts anderes. Unser kleiner Monopolist merkt nun, dass er eigentlich auch 20% vom Getreide verlangen könnte. Die Leute kommen ja sowieso. Und sie tun es auch, bis ein

anderer Handwerker auch eine Mühle baut. Vielleicht sogar eine bessere, aber dennoch kommen die Dörfler nicht, weil sie es gewohnt sind, bei unserem Zimmermann zu mahlen. Der andere geht also über kurz oder lang bankrott. Und wenn nicht, entzündet sich auf wundersame Weise der Dachstuhl seiner Mühle oder er wird von unserem nun alteingesessenen und inzwischen kapitalkräftigen Zimmermann aufgekauft. Und wird zum Tagelöhner...

Der einzige Ausweg aus diesem Dilemma ist wiederum die Selbstorganisation der Dorfgemeinschaft. Es ist aufgrund unterschiedlicher Anreizstrukturen auf den Ebenen der Leistungserbringer nicht relevant, ob irgendein Handwerker bereits eine Mühle gebaut hat. Die Dörfler bringen gemeinsam immer das bessere Ergebnis zustande, weil ihre Leistung um den Faktor «soviel Ertrag wie möglich mit geringstmöglichem Aufwand» bereinigt ist. Sie sind nur an einem guten Ergebnis interessiert.



Anreizstrukturen proprietäre Mühle

3. Leistung, Motivation und Organisation

Noch ist der Transfer der Ergebnisse aus der Modellannahme recht abstrakt. Dennoch lässt er einige grundsätzliche Erkenntnisse zu, die, wie wir sehen werden, für die sich heute stellenden Probleme von überraschender Relevanz sind. Wir wollen hierzu unsere Metapher von der mittelalterlichen Dorfmühle verlassen und allgemein von einer Leistungserstellung ausgehen. Primär interessiert, was den Ausgangspunkt für die Leistungserbringung darstellt, wenn ihr keine kommerzielle Motivation zugrunde liegt. Warum wird überhaupt etwas geschaffen. Haben wir den Ausgangspunkt definiert, müssen wir uns fragen, ob es sich hier um einen sich immer wiederholenden Prozess handelt oder eine einmalige Aktion; ob also die Leistungserstellung einer Entwicklung unterliegt. Mit der Beantwortung dieser beiden Fragen ist die rein objektbezogene Dimension erfasst. Nun behandeln wir die subjektive und damit letztlich die Frage, wie die Übernahme von Aufgaben motiviert und wie sie organisiert ist. Wir müssen also folgende vier Kategorien behandeln:

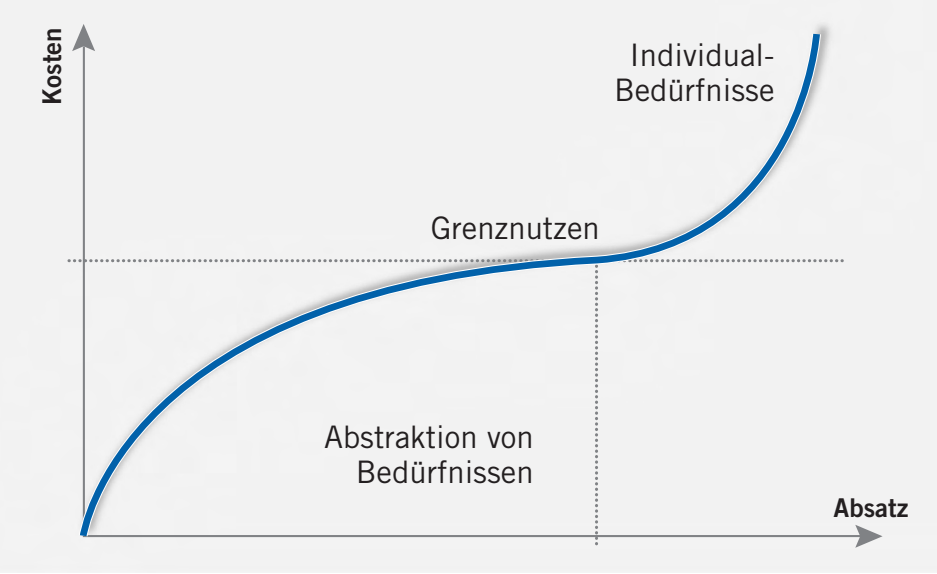
- Auf welcher Grundlage wird eine Leistung konzipiert?
- Wie entwickelt sich die Konzeption weiter?
- Wie ist die Leistungserstellung motiviert?
- Wie ist die Leistungserstellung organisiert?

Dabei wollen wir immer zwei Gegensatzpaare betrachten: Wir fragen uns, was real ist und was aus Sicht des Kunden und des Produzenten optimal wäre.

3.1. Bedürfnisse vieler als Integrationsansatz

Wenn ein Produkt auf klassischem Wege entwickelt wird, setzt dies die Definition der damit zu befriedigenden Bedürfnisse voraus. Beispielsweise muss eine Spülmaschine die Gläser sauber und ohne Kalkrückstände reinigen. Hierzu muss «Bauknecht wissen, was Frauen wünschen».

Die Entwickler treffen also eine Annahme darüber, was ihr Produkt alles können muss. Sollte jemand auf die Idee kommen, seinen neuen Kaschmirpullover in die Spülmaschine zu geben, wird er merken, dass sie dafür nicht konzipiert ist. Aber warum nicht?

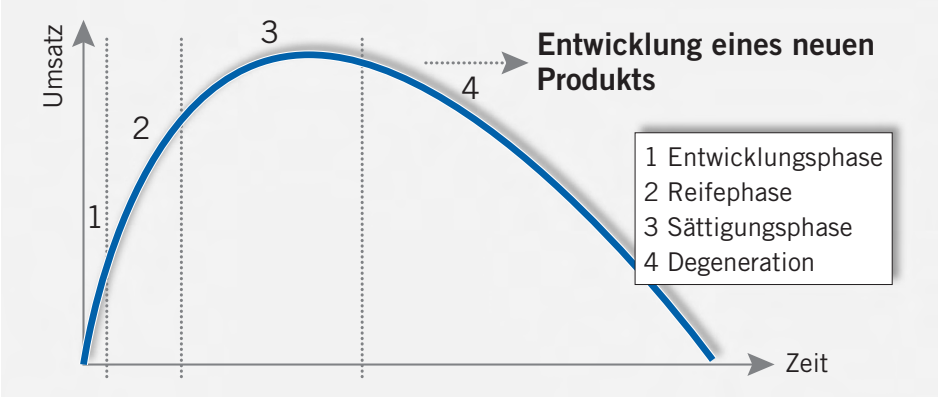


Grenznutzenproblematik Entwicklung

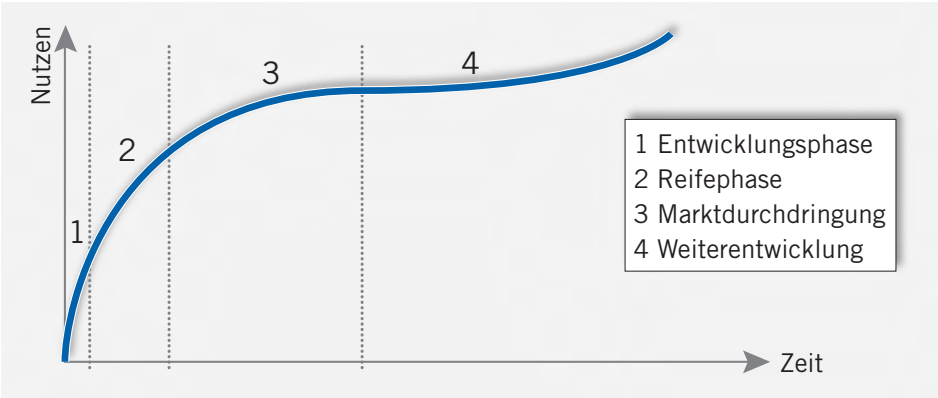
Die Abstraktion von Individualbedürfnissen hat einen sehr einfachen Grund: Je mehr Leute das Produkt kaufen, desto mehr Ertrag wird generiert. Hierbei ist aber der Grenznutzen für eine zusätzliche Funktion schnell erreicht. Wozu also Entwicklungsaufwand auf sich nehmen, nur um drei zusätzliche Spülmaschinen an Wollpulloverfanatiker zu verkaufen? Das rechnet sich nicht. Erst wenn nicht mehr von der Abstraktion, sondern ganzheitlich von den Individualbedürfnissen ausgegangen wird, steigt die Flexibilität des Produktes. Die Logik der kommerziellen Entwicklung wird ins Gegenteil verkehrt. Und vielleicht finden «die» Frauen ja heraus, dass das mit den Pullovern eine tolle Sache ist. Aus der Integration vieler Bedürfnisse in ein Produkt folgt multiple Einsetzbarkeit und damit ein direkter Mehrwert für den Benutzer. Wenn uns die Marketinggurus seit Jahren predigen, man müsse von den Kundenbedürfnissen ausgehen, so muss man nun sagen: Dann macht es auch konsequent und geht von Individualbedürfnissen aus!

3.2. Entwicklung als systemimmanenter Faktor

Eine Leistung unterliegt dem Produktlebenszyklus. Sie wird entwickelt, gelangt zur Reife und veraltet dann wieder. Wenn der Leistungserbringer nicht aus dem Markt geworfen werden will, muss er neue Entwicklungen präsentieren. Treibender Faktor in der Marktgerechtheit der Leistung ist die Nachfrage. Wir haben aber gesehen, dass die Nachfrage und damit die Bedürfnisse der Kunden der Einschätzung des Leistungserbringers und der Grenznutzenproblematik unterliegen. Damit ist die Qualität der Leistung aus individueller Kundensicht nicht nur suboptimal, sie ist sogar aus der ökonomischen Entwicklungslogik heraus veraltet, denn der Hersteller braucht Zeit, um Bedürfnisse zu identifizieren und daraus ein Produkt zu machen. Idealerweise führt das Kundenbedürfnis direkt zu einem (Weiter-)Entwicklungsprozess; sozusagen «realtime-to-market».



Klassischer Produktlebenszyklus



Idealer Produktentwicklungszyklus

Damit wird aus einem Produktlebenszyklus eine in Abhängigkeit vom Entwicklungsengagement stetig bis exponentiell ansteigende Funktion. Leistungen unterliegen keinem Lebenszyklus mehr, sondern einer stetigen Entwicklung.

3.3. Intrinsische Motivation der Leistungsersteller

Wir haben bis anhin den Produktentwicklern Unrecht getan. Sie führen ja nur aus, was ihnen vorgegeben wird. Der Ansatz für die beschränkte Einsetzbarkeit der Entwicklung ist ja wie gesagt nicht bei ihnen zu suchen. Versetzen wir uns nun in die Situation eines Entwicklers – sagen wir, eines Ingenieurs, der Turbinen entwickelt. Er hat nun nicht fröhlich irgendwelche Turbinen zu entwickeln, die ihm persönlich in Zukunft einen tieferen Strompreis sichern, – er muss sich an klare Inputparameter halten.

Nebst der technischen Realisierbarkeit sind dies nicht zuletzt Überlegungen aus ökonomischer Sicht. Angenommen, er findet heraus, eine Oberflächenbeschichtung könnte den Energieumsatz der Turbine merklich steigern – was wird passieren? Man wird ihm auf die Schulter klopfen und ihn bestenfalls befördern. Wenn er klug genug ist, wird er die Idee aber an einen anderen Turbinenhersteller veräussern und in Zukunft auf einer Dreissig-Meter-Yacht die Karibik durchkreuzen. Sein Antrieb, die Konstruktion der Turbinen zu verbessern, ist ausschliesslich ein ökonomischer. Er bleibt ein in seinen Schranken agierender Tagelöhner. Jetzt ist aber dieser Ingenieur nicht einfach ein Shareholder-Value- oder Homo-oeconomicus-gläubiger Egomane, nein, er sieht sich dem Fortschritt der Menschheit verpflichtet und zieht einen tieferen Sinn aus seiner Arbeit. Je besser der Energieumsatz seiner Turbine ist, desto grösser sind die Chancen, dass die Auswirkungen der Energieproduktion auf die Umwelt in Zukunft minimiert werden können. Warum nun aber führt diese Annahme – so selten sie in der Praxis auch antreffen sein wird – nicht zu einem Fortschritt? Selbst wenn sich unser ambitionierter Altruist noch so sehr anstrengt, ein optimales Produkt zu entwickeln, es wird immer den Restriktionen der Vermarktbarkeit unterliegen. Ist die Turbine im Energieumsatz besser als die der Konkurrenz, wird sie teurer und somit die Verwendbarkeit für viele Nutzer unmöglich. Ist sie besser, der Produktionsaufwand aber nicht amorti-

sierbar, wird sie niemals in Produktion gehen. Aus dieser Erkenntnis ergibt sich, dass nur eine Leistungserbringung, die aus der bestmöglichen intrinsischen Motivation heraus geschieht – nämlich Produzent und Konsument in Personalunion – eine individuell optimales Ergebnis liefert.

3.4. Hierarchie und Heterarchie als Organisationsprinzip

Wir wollen nun einen Blick auf den Prozess der Leistungserstellung und damit die organisatorischen Strukturen werfen. Die Diskussion, ob denn Führung – in welcher Form auch immer – überhaupt notwendig sei, wollen wir hier getrost beiseite lassen und annehmen, dass die Übernahme von Verantwortung bereits so etwas wie Führung ist. In einem sich selbst organisierenden System wird Autorität nicht verliehen, sondern erarbeitet. Sie ist damit kein gottgegebenes und somit beständiges, einklagbares Recht, sondern untersteht einem konsequenten Legitimationszwang. Aber Legitimation wodurch? Lange galt der charismatische Führer als ideales Führungskonzept. Er hält den Regenschirm in die Luft und die Schäflein folgen ihm. Aber kann Charisma eine Führungsfunktion legitimieren? Es besteht nämlich keinerlei Bezug zum Zweck der Organisation und damit wird ein neuer Führer mit sympatischerem Lächeln Ansprüche erheben, ohne Nutzen für die Gemeinschaft zu erzielen. Ideal ist demnach nicht Charisma als Legitimationsbasis, sondern Kompetenz und Engagement. Lässt eine Organisation die objektive Bewertung einer Fähigkeit zu, dann kann dies als Massstab von Autorität genommen werden.

Es sind zwei Dinge vonnöten, um das Konzept praktikabel zu machen:

- Das eine ist die strukturelle Flexibilität. Eine Führungsposition ist keine statische Verantwortung für eine Abteilung, Division oder Unternehmung. Autorität muss einem

dauerhaften Rechtfertigungsdruck unterstehen und die Nichterreichung von Zielen oder Nichterfüllung von Aufgaben muss zur Absetzung führen. Und zwar schnell.

- Weiterhin muss die Definition von Aufgaben und Zielen nicht nur klar sein, sondern auch von allen Betroffenen getragen werden. Dann, und nur dann, kann überhaupt aus der 360º-Perspektive beurteilt werden, denn alle Betroffenen tragen die Zielsetzungen aus intrinsischer Motivation – und das ganz ohne Leitbildprozess.

Beide Voraussetzungen können in herkömmlichen Organisationen als nicht gegeben betrachtet werden. Die strukturelle Flexibilität würde jede abbildbare Organisation unmöglich machen. Die gemeinsame Zieldefinition ist wohl angestrebtes Ziel fortschrittlicher Betriebswirtschaftslehre, die Mittel scheinen im Vergleich zu unserem Bild aber als Flickwerk. Erst wenn die Organisation sich aus einer gemeinsamen Zieldefinition heraus bildet, Aufgaben aus dieser Motivation heraus wahrgenommen und Führung als Beitrag zur Zielerreichung verstanden wird, hat das System eine Überlebenschance.

3.5. Die Umkehrung der ökonomischen Logik

Wir haben aus den Beispielen nun eine Summe von Postulaten abgeleitet. Alle fussten auf der Annahme, dass sich die ökonomischen Sachzwänge als Ausgangsbasis einer Leistungserstellung nur begrenzt eignen. Ist es aber möglich, auf einer anderen Grundlage ein Modell zu errichten, und wenn ja, wie soll das gehen? Es ist bereits klar geworden, dass wir nicht monokausal argumentieren können. Es reicht also nicht, dass wir versuchen, einen Nutzen in Geldwert zu quantifizieren. Wir müssen alle Dimensio-

nen von Nutzen erfassen, auch die immateriellen. Was also bewegt einen Menschen dazu, aktiv zu werden? Welche Bedürfnisse treiben ihn?

Ich will Ihnen an dieser Stelle ersparen, noch einmal die ganze maslov'sche Bedürfnispyramide zu besichtigen. Machen wir es uns ein wenig einfacher und gehen wieder von unserem Beispiel aus: Welche Dimensionen von Nutzen für einzelne Dörfler lassen sich beim Bau der Mühle erkennen? Sie wollen essen. Das Werk, das sie gemeinsam errichten, bringt jedem der daran teilhaben will, einen direkten Vorteil bei der Existenzsicherung. Jetzt muss das Getreide nicht mehr von Hand zerrieben werden, und die Dörfler haben Zeit für andere Dinge. Solange die Gemeinschaft besteht, ist das Risiko, diesen Nutzen von heute auf morgen zu verlieren, gering. Damit sind wir beim nächsten Problem: dem Sicherheitsbedürfnis. Geht einem Müller das Mahlwerk kaputt, steht er vor einem Problem, das zu lösen er unter Umständen alleine gar nicht in der Lage ist. Zerbricht der Dorfgemeinschaft der Mühlstein, wird dies zum Problem aller und somit auch gemeinsam gelöst. Damit wird das subjektive Sicherheitsempfinden deutlich erhöht. Die Leistung des Einzelnen wurde bis anhin gleichwertig eingeschätzt. Das ist selbstverständlich nicht ganz richtig. Einzelne erbringen qualitativ höherwertige Leistungen, der Zimmermann etwa. Er wird nun nicht für seine Leistung bezahlt. Was hat er also von seiner Arbeit? Sollte er mehr Getreide bekommen als ein Handlanger? Das würde die Motivationsachse im Modell verschieben. Der einzige Grund, mehr zu leisten als andere, ist das Geltungsbedürfnis des einzelnen. «Ich habe für die Gemeinschaft viel geleistet!» – damit lässt sich Ansehen und Respekt gewinnen.

Es sind also primär die drei Bedürfnisse nach Grundversorgung, nach Sicherheit und nach Ansehen, die eine breite Motivationsbasis für die Leistungserbringung bilden. Und alles ohne monetäre Anreize.

4. Modell der Open Source Softwareentwicklung

Inwieweit sich der Bau einer Mühle auf die Entwicklung von Software übertragen lässt, ist eine berechtigte Frage. Tatsächlich ist der Vergleich des Bedürfnisses nach Nahrung und nach Benutzung einer Software ein wenig weit hergeholt. Beginnen wir den Vergleich mit der Frage nach der proprietären Mühle. Für ein Bedürfnis, etwa einem Office-Paket, besteht ein proprietäres Angebot. Der Nutzer kann faktisch sogar zwischen vielen Lösungen auswählen und hat entsprechend dafür bezahlen.

Klar ist, dass damit sein Primärbedürfnis nach der Nutzung in dem Masse befriedigt ist, in dem die Software dies zu leisten vermag. Damit fangen die Probleme aber schon an. Was ist, wenn sie gewisse Funktionen nicht beherrscht? Wenn wir an den Kaschmirpullover zurückdenken, wird klar, dass niemals alle Bedürfnisse befriedigt werden. Der Nutzer kann eigentlich nur hoffen, dass eine kaufkräftige Benutzergruppe den Hersteller darum bittet, diese Funktion in den nächsten Release einzubauen – der notabene zu bezahlen ist. Der Benutzer unterstellt sich also auf Gedeih und Verderb der Willkür der Hersteller. Er wird abhängig, weil er selbst nicht in der Lage ist, die Entwicklung zu beeinflussen. Die Softwareschmieden schützen den Quellcode ihrer Programme wie ihren Augapfel, schliesslich ist er das Kapital der Gesellschaft. Daraus lässt sich ihnen kein Vorwurf machen, sie beschneiden damit aber die Autonomie der Benutzer. Es geht also im Kern um die Frage, welche Rechte der Anwender bezüglich des Produkts besitzt – um den Quellcode der Programme und damit um die Lizenz, der er unterstellt wird.

4.1. Lizenzen – Urheberrechte – Wirkungsabsicht

Eine proprietäre Software wird hergestellt, um damit Gewinne erwirtschaften zu können. Das mit einer Lizenz verbundene Urheberrecht hat somit nur eine Absicht: den Anspruch auf Einnahmen zu sichern. Entsprechend sind Unternehmungen auch bestrebt, von Lizenzrechtsverletzungen zu erfahren und die Softwarepiraten zu bestrafen. Das ist auch ihr gutes Recht. Das Urheberrecht lässt sich auch auf Geschriebenes beziehen. Wenn ein Leser also Teile dieses Manuskripts veröffentlichen möchte, ist dies nur unter Angabe der Quelle erlaubt. Genauso sieht es in der Open Source Entwicklung aus. Weil die Programmierer keinen geldwerten Nutzen aus ihrer Arbeit ziehen wollen, ist die Verwendung ihrer Arbeit gestattet, solange kenntlich gemacht wird, wer sie geleistet hat. Die OSS-Community, die Gemeinschaft der Programmierer im Netz, kennt eine ganze Reihe von Lizenzmodellen, die diesen Anspruch sichern sollen. An dieser Stelle soll der Einfachheit halber aber nur auf ein Basismodell eingegangen werden, das für das Verständnis vollauf genügt: die General Public License.

Das Lesen der GPL kann wärmstens empfohlen werden – an dieser Stelle genügt es, zentrale Aspekte kurz zu erläutern. Eine neue Software unter die GPL zu stellen, bedeutet sie gratis verfügbar zu machen. Der Programmierer gibt also seinen Anspruch auf Ertrag aus seiner Arbeit auf; er verlangt aber von allen Benutzern, dass sie dies in Verbindung mit dieser Software ebenso tun. Wer also auf der Grundlage der geleisteten Arbeit aufbaut, muss seine Arbeit wieder der GPL unterstellen. Damit kann gewährleistet werden, dass das, was Open Source begonnen hat, niemals zu proprietärer Software wird. Jeder, der den Quellcode verändert oder neuen hinzufügt, muss dies deutlich kommentieren. Es steht also beispielsweise im Quellcode, dass dieses Modul von Heinrich K. am 24.12.99 programmiert wurde. Diese Vorschrift dient dazu, möglichen Urheberrechtsverletzungen vorzubeugen. Was wäre aber an einer Verletzung so

dramatisch? Nun, eine Unternehmung könnte auf die Idee kommen, den ganzen Quellcode oder Teile daraus als ihr eigenes Werk zu verkaufen. Damit wäre das ganze OSS-Modell gefährdet. Ein weiteres zentrales Anliegen der GPL ist die Wegbedingung von Garantieansprüchen. Da keine Unternehmung hinter der Software steht, könnten nur die Programmierer haftbar gemacht werden. Dass diese keinerlei Interesse haben, für eine Gratisleistung auch noch für Haftungsansprüche geradezustehen, versteht sich von selbst. Die einzige zulässige Kommerzialisierung einer unter der GPL veröffentlichten Software ist die Abgeltung des Distributionausfwandes, also der Kosten für das Paket, das um die Software herum gebastelt wird. Auf diesen Punkt wird im folgenden noch genauer einzugehen sein.

Die Lizenz ist also die Definitionsbasis für das Modell der OSS-Entwicklung. Die Dorfgemeinschaft wird wohl kaum solch ausgeklügelte Bedingungen formuliert und unterzeichnet haben; doch wird jeder die Grundsätze gemeinsamer Nutzung und Entwicklung akzeptiert haben.

4.2. Die Bedürfnisse vieler als Integrationsansatz

Ausgangspunkt einer Entwicklung ist immer das Bedürfnis eines Einzelnen. Stück für Stück wird der Anspruch an die Lösung durch Ideen und Ansätze weiterer Benutzer erweitert. Nehmen wir ein weiteres Beispiel, um dies plastischer werden zu lassen. Ein Administrator eines mittelständischen Betriebes soll auf Verlangen der Geschäftsleitung eine Intranet-Plattform für den Betrieb realisieren. Sein Budget ist mit CHF 10’000.– für 50 Benutzer ziemlich dürftig.

Die Evaluation bestehender proprietärer Lösungen zeigt, dass sie weder vollauf den Bedürfnissen der Unternehmung entsprechen noch im Rahmen des Budgets reali-

sierbar sind. Entweder fehlen benötigte Kernfunktionen oder die Software ist in Anschaffung und Unterhalt zu teuer. Der Administrator wendet sich also im Internet an andere Administratoren, die das gleiche oder ein ähnliches Bedürfnis haben. Auf der Seite www.intranet.org sucht er nun nach bestehenden Modellen, die alle der GPL unterstehen. Er findet jedoch weder eine Software, die genau auf seine Anforderungsliste passt, noch könnte er im Alleingang eine solche programmieren.

Also eröffnet er ein Projekt im Internet und beschreibt seine Bedürfnisse. Die Basis des Programms hat er im Quellcode bereits geschrieben und hinterlegt sie am selben Ort. Dies finden nun andere Programmierer und interessieren sich für seinen Ansatz. Der eine programmiert ein Kalendermodul für die Intranetlösung, ein zweiter macht eine Content Management Lösung usw. Unser Systemadministrator hat als Initiator die Hoheit über das Projekt. Er kann entscheiden, welches Modul integriert wird und was zu verwerfen ist.

Steht eine funktionsfähige und getestete Lösung, wird sie veröffentlicht. Die Geschäftsleitung erhält für ihr Geld eine den Anforderungen entsprechende Lösung und die Community die Erlaubnis, sich der Lösung zu bedienen. Jetzt ist die ganze Entwicklung selbstverständlich nicht beendet. Das Projektforum im Internet besteht weiterhin, oft genug wird sogar eine eigene Webseite erstellt, wenn das Produkt erfolgreich ist. Hier tauschen Anwender und Programmierer Tipps und Tricks in der Anwendung und Bedürfnisse aus, die in der nächsten Version berücksichtigt werden sollten. Die Entwicklung wird damit zu einem iterativen, also sich wiederholenden Prozess, der die Qualität in allen Belangen verbessert. Wieso aber sollte dieses Verfahren der Entwicklung proprietärer Software überlegen sein – schliesslich bringen Hersteller von Software auch immer wieder neue Versionen ihres Produktes heraus?

Die folgenden vier Hauptgründe sprechen für das Open Source Modell:

- Für die Verwendung von Updates fallen keine Kosten an.
- Ansatzpunkt für die Integration eines Bedürfnisses ist nicht die Kaufkraft der Benutzergruppe, sondern die Sinnhaftigkeit der Integration.
- Die Basis der Programmierer und Tester des Produktes ist um ein vielfaches grösser und somit die Lösung stabiler und sicherer.
- Die Entwicklungsdynamik ist abhängig von der Anzahl und dem Engagement der Benutzer.

Es ist deutlich geworden, dass der kritische Erfolgsfaktor für die Integration vieler Bedürfnisse die Entwicklung ist. Wenn sich niemand mehr für eine Lösung interessiert, ist das Open Source-Projekt gestorben und der Benutzer steht mit einer statischen Lösung alleine da.

4.3. Entwicklung als systemimmanenter Faktor

Keine andere OS-Software ist erfolgreicher als das Betriebssystem Linux. Linux unterliegt einer enormen Entwicklungsdynamik und es gibt zahlreiche Unterprojekte, die es selbst bis zum Standard gebracht haben. Was liegt also näher, als dies zum Anschauungsobjekt zu erklären. Die Basis eines Betriebssystems ist der Kernel, also der Quellcode-Kern. Die Linux-Community entwickelt diesen unter der Leitung des Programmierers, der den Anstoss zur Entwicklung unter der GPL gegeben hat, kontinuierlich weiter. Schritt für Schritt werden neue Funktionen oder neugestaltete Module in einer Entwicklerversion veröffentlicht. Wer sich dafür interessiert, kann nun an diesem Kernel herumprobieren. Gleiches machen nun Unternehmungen, die von dieser Open Source Software leben, die also sogenannte Distributionen ihrer Linux-Version ver-

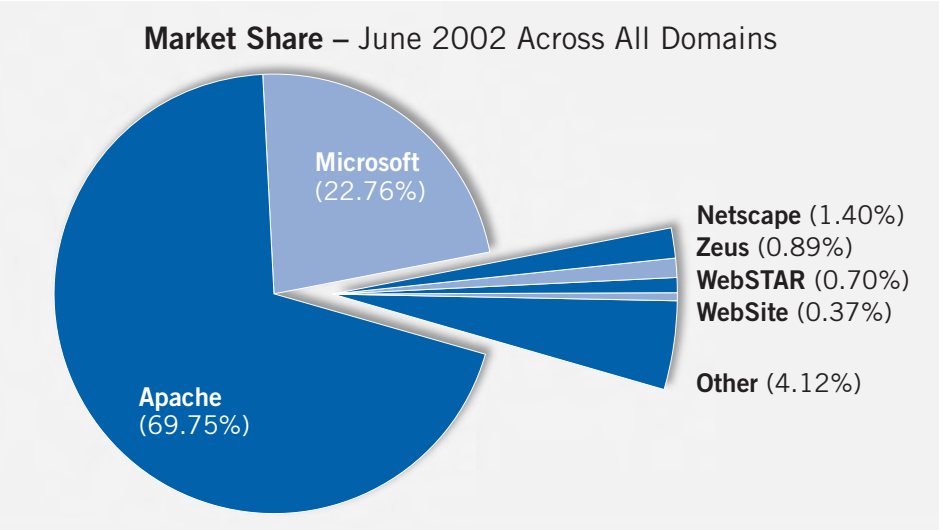
kaufen. Diese Firmen nehmen nun einen Kernel und bauen Installationsroutinen drumherum, fügen Software hinzu und verfassen Handbücher. Nach der GPL sind sie nur berechtigt, für ebendiese Leistung Geld zu verlangen, nicht aber für das Betriebssystem an sich. In der Folge kostet eine Linux-Distribution rund CHF 100.-, wogegen andere Betriebssysteme ziemlich blass aussehen. Auch die Distributoren sind an einer Weiterentwicklung ihrer Version interessiert und folgen daher der gleichen Logik. Mit diesem Vorgehen wird die Zeit zwischen zwei Veröffentlichungen minimiert.

Nun müsste man annehmen, dass dies den Benutzer nicht besserstellt, da er wiederum auf einen neuen Release warten muss. Dem ist selbstverständlich nicht so. Eine neue Version wird von der Community zuerst einmal kritisch unter die Lupe genommen und auf Fehler durchleuchtet. Das Ergebnis ist in geradezu revolutionär kurzer Zeit ein Lösung für entstehende Probleme. Anwender, die dazu in der Lage sind, beheben Fehler also, wenn sie entdeckt sind und stellen allen Benutzern diese «Patches», Programme zur Fehlerbehebung, zur Verfügung. Es wird deutlich, dass die Stärke von Linux nicht darin besteht, ein sauberes System frisch ab Presse zu bekommen, sondern darin, dass die Stabilität und Funktionalität eines Releases an Reife gewinnt. Die schrittweise Fortentwicklung erscheint ein überlegenes Modell zu sein.

4.4. Intrinsische Motivation

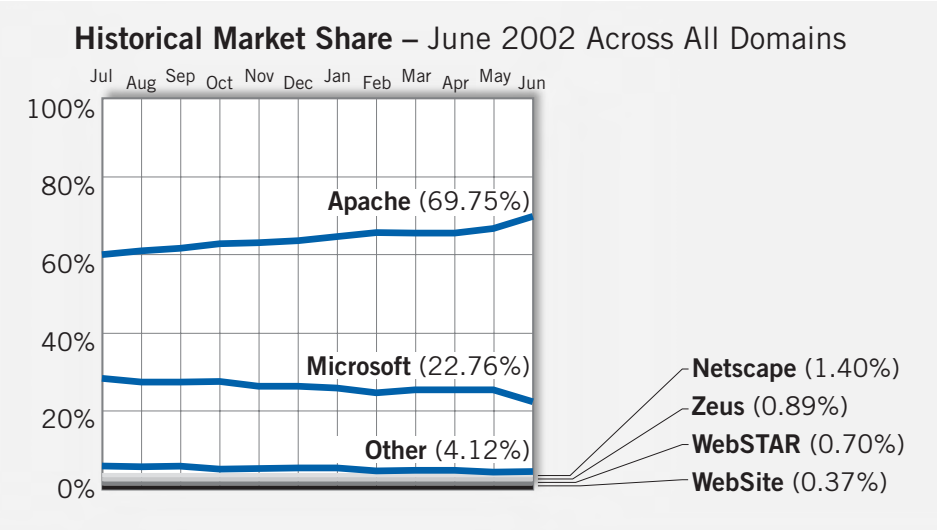
Im Vorangehenden wurde festgestellt, dass nur dann ein wirklich optimales Ergebnis herauskommt, wenn Leistungsersteller und Konsument ein und dieselbe Person sind. Die Entwickler von Open Source Software sind die primäre Zielgruppe ihrer eigenen Produkte und die Anreizstrukturen funktionieren.

Entsprechend sind die Programme am erfolgreichsten, die die IT-Profis ansprechen. Am eindrücklichsten beweist dies der Webserver Apache, ein Open Source Produkt mit heute fast 70% weltweitem Marktanteil. Soweit Open Source Produkte also diejenigen ansprechen, die etwas von der Materie verstehen, können sie die proprietäre Konkurrenz nachhaltig aus dem Feld schlagen, wie die Entwicklung der Marktanteile zeigt. Nun ist das Modell aber nicht der Heilsbringer dieser Welt, sondern bringt auch



Quelle: Securityspace.com

Probleme mit sich. Eines der bedeutendsten ist die manchmal fehlende Zielharmonie von Entwickler und Anwender. Projekte wie OpenOffice.org, dem freien Pendant zu Microsofts Office, haben bewiesen, dass die Community bereit ist, Software für den Endanwender zu programmieren. Es existiert aber noch kein Projekt, dass ähnlich komplexe Funtionen wie SAP oder Navision zu bieten in der Lage wäre. Alle Versuche sind bis anhin gescheitert, ein Open Source E-Business-System zu realisieren. Die Gründe sind vermutlich bei der Komplexität zu suchen; die wenigsten Programmierer verstehen etwas vom Management einer Unternehmung, geschweige denn eines Konzerns. Wenn also kein Interesse an einem Projekt besteht, stirbt es. Um neue Projekte initialisieren zu können, müssen sie sexy sein, also bei den zukünftigen Mitentwicklern Interesse wecken. Nur dann ist das Fundament für den Erfolg des Produktes gelegt.



Quelle: Securityspace.com

4.5. Hierarchie und Heterarchie

Die Entwicklung einer Software ist ein komplexes Unterfangen, weil eine Software ein fragiles und weitvernetztes System ist. Dass nicht einfach jeder hier und dort ein bisschen Code einfügen kann, ist einleuchtend. Lag der Fokus gerade noch auf der freiwilligen Mitarbeit, wird er jetzt um hierarchische Komponenten erweitert, denn es braucht nunmal die Übernahme von Verantwortung, mit anderen Worten: Führung. Der einfachste Weg, Verantwortung zu erlangen, ist ein Projekt zu starten. Damit behält man die Lufthoheit über dem Quellcode und die Legitimation, Releases zu autorisieren. Ein Einzelner kann dies aber nicht gewährleisten, da er selbst bei Vollbeschäftigung mit dem Thema nicht genügend Kapazität hätte. Der logische Schritt ist die Bildung niedrigerer Hierarchiestufen, die Verantwortung delegierbar machen. Diese Organisation muss nun die paradoxe Situation von freiwilliger Mitarbeit und Führung überwinden. Dies funktioniert nur dann, wenn der Geführte seinen Vorgesetzten als solchen akzeptiert, ansonsten verabschiedet er sich ohne Verlust für sich selbst, fügt der Gemeinschaft durch Kompetenz- und Kapazitätsverlust aber Schaden zu. Führung unterliegt also einem ständigen Legitimationszwang.

Aus Stufe 1 der Nutzendefinition lässt sich ableiten, dass ein technisch überlegener Führer anerkannt ist, da er die beste Gewähr für eine Bedürfnisbefriedigung darstellt. Wer also ein ausgezeichneter Entwickler und Kommunikator ist, ist für solche Positionen prädestiniert, solange seine Fähigkeiten den Erfordernissen entsprechen. Es ist einleuchtend, dass der Leiter eines Projektes primär fähig sein muss, aus einem unübersichtlichen Haufen von Code-Blöcken ein funktionierendes Ganzes zusammenzustellen. Ob er in der Lage wäre, den Code selbst zu schreiben, ist eher nebensächlich.

Das Sicherheitsbedürfnis – Stufe 2 – legitimiert Führung über den Zusammenhalt der Gemeinschaft. Kann die Führungskraft eine Atmosphäre gewährleisten, die die Mitar-

beit spannend und angenehm macht, wird es Entwicklern eine Freude sein, mitzuarbeiten. Damit entsteht ein Zusammengehörigkeitsgefühl und daraus die Sicherheit, dass aus diesem Projekt einmal ein Produkt wird, dass Bedürfnisse zu befriedigen in der Lage ist.

So richtig heiss wird Führung allerdings erst im Licht der 3. Stufe: dem Geltungsbedürfnis. Wer eine Hierarchiestufe aufsteigt, geniesst höheres Ansehen. Nun muss man daraus nicht unbedingt OSS-Darwinismus ableiten. Es ist aber richtig, dass eine Führungsfunktion ständig streitbar gemacht wird. Und das ist gut so, denn nur dadurch kann garantiert werden, dass Legitimation stattfindet und einer Dynamik unterliegt. Solange ein Nutzen aus Zusammenarbeit generiert wird, funktioniert dieses Führungsmodell. Eine Unternehmung hingegen fiele über kurz oder lang in ein Chaos, weil mit einer Position direkter, monetärer Nutzen verbunden ist.

5. Was bringt die Zukunft?

Das Modell der OS-Entwicklung wird entgegen der gerne verbreiteten Position das proprietäre Modell keineswegs hinwegfegen. Das Modell hat wohl viele Vorteile, aber eben auch Grenzen, die den Erfolg nicht in Frage stellen, aber ebensowenig das Verdrängen von Anbietern von Software prophezeien lassen.

Zentrale Frage für die Bewertung der Zukunftsaussichten ist die Breite des Angebotes und dessen Distribution. Je professioneller eine Software daherkommt und je einfacher der Zugang sich darstellt, desto grösser sind die Erfolgschancen. Will aber das Modell sich durchsetzen, müssen sukzessive alle Anwendungsbereiche durchdrungen werden, die auch den Endbenutzer von der Qualität überzeugen und somit nicht nur dem Individualnutzen, sondern dem Modell und seiner Überzeugungskraft zum Durchbruch verhelfen. Die Analyse des Modells auf Basis der Anreizstrukturen liefert

diesbezüglich nur ein mageres Ergebnis: Wenn die Motivation und das Engagement der Entwickler das Modell weiterhin tragen und seine Verbreitung befördern, wird es fortbestehen. Wenn es aber erfolgreich verwendbare Produkte hervorbringt, kann es die einseitig ökonomisierte Logik der globalisierten Gesellschaft auf einen grundlegenden Wert zurückführen, der das Menschsein prägt: Gemeinsamkeit.

Open Source Software – die wichtigsten Begriffe

Quellcode Damit ein Computer-Programm funktionieren kann, braucht es einen Bauplan. Im Quellcode stecken alle Befehle, die ein Programm ausführen kann, sein Aussehen und seine Funktionen. Bei den meisten Produkten ist dieser Quellcode für Benutzer nicht einsehbar.

Open Source Die Offenlegung des Quellcode ist eine Revolution in der Informationstechnologie. In der traditionellen Wertschöpfung ist der Code das Kapital einer Unternehmung. Eine Offenlegung würde bedeuten, das Produkt zu verschenken. Genau das ist die Absicht hinter Open Source Software: Jeder kann den Code und damit das Programm kostenlos besitzen und verwenden, den Code verändern und weitergeben.

Open Source Projekt Die Entwicklung einer Software ist ein komplexer Prozess. Open Source Software wird als virtuelles Projekt über das Internet entwickelt. Auch hier steht am Anfang die Definition der Bedürfnisse, es folgt ein Entwicklungskonzept und schliesslich der Code. Auf der Projektwebseite wird das Produkt dann weiterentwickelt, Versionen werden zum Download bereitgestellt und Supportdienstleistungen angeboten.

Proprietäre Software Nur wenn der Quellcode nicht einsehbar ist, kann eine Unternehmung Eigentumsrechte geltend machen. Proprietäre Software sind Produkte, die dem Anwender per Lizenz und gegen Entgelt zur Benutzung überlassen werden.

General Public License (GPL) Auch Open Source Software untersteht einer Lizenz. Die GPL ist die verbreitetste (www.gnu.org/licenses/gpl.html). Sie besagt jedoch nicht, dass der Benutzer für die Überlassung bezahlen müsste, sondern im Gegenteil, dass niemand Geld für die Software verlangen darf.

Idee, Konzept und Realisation: iframe Integrated Communications AG Heerbrugg, St. Gallen, Zürich