Info Bridge

Herausgeber: Object Engineering GmbH • www.objeng.ch • September 2019 • Nr. 12

Editorial

Liebe Leserinnen, liebe Leser

Wieder ist eine Weile vergangen, während der Sie und wir unseren Tätigkeiten nachgegangen sind und versucht haben "die Welt am Laufen" zu halten. Inzwischen war auch die Hype-Szene nicht müde neue Begriffe zu kreieren und zu verbreiten. Unüberhörbar sind Begriffe wie Digitalisierung, Artificial Intelligence, Machine Learning, um nur einige zu nennen. Ja sogar politische Grössen und natürlich die Medien, also die „Experten" unserer Zeit, blasen laut in das Hype-Horn. Aber Halt! Habe ich gesagt neue Begriffe? Nun, Digitalisierung und Artificial Intelligence sind doch keine neuen Begriffe, beide sind 30 bis 40 Jahre alt. Aber es scheint so, als würde man von Zeit zu Zeit diese Begriffe vergessen und wieder neu erfinden. Artificial Intelligence habe ich bereits in meinem Universitätsstudium vor fast 40 Jahren kennengelernt. Ich muss zugeben, es war nicht mein Lieblingsfach. Von Digitalisierung hat man bereits gesprochen, als man vor längerer Zeit analoge Signale in digital codierte Signale umwandelte. Ja selbst Sprache wurde auf PCM gewandelt, in etwa vergleichbar mit VoIP (Voice over IP). Und die Digitalisierung ging laufend weiter und wurde exponentiell seit den ersten mit Mikroprocessor gesteuerten Computern (u.a. Personal Computer) fortgeführt. Und jetzt soll es NEU sein?

Man könnte mir natürlich vorwerfen, dass ich hier nicht mehr mitreden kann und nicht mehr draus komme, schliesslich habe ich auch schon einige Jährchen auf dem Buckel. Ja gut, akzeptiert. Aber dann frage ich mal die Experten, ob sie mir mit einfachen Worten DIGITALISIERUNG erklären können. Ich habe diese Frage kürzlich einem Buchautor über dieses Thema gestellt, die peinliche Antwort möchte ich Ihnen ersparen.

Wenn wir uns an Wikipedia wenden, das auch nicht über alle Zweifel erhaben ist, finden wir jedoch eine vernünftige Definition (siehe Kasten). Diese Definition ist recht differenziert, wenn auch erweiterbar, doch zumindest recht zutreffend. Damit lassen sich Bild-Digitalisierung, Audio-Digitalisierung und Ereignis-Digitalisierung und einiges mehr abdecken. Einen wichtigen Faktor dazu möchte ich aber erwähnen, die Beschreibung von Digitalisierung als „Veränderungen von Prozessen". Dies ist recht allgemein gehalten, trotzdem wäre Prozess-Digitalisierung, was wohl meistens in der Geschäftswelt gemeint ist, der bessere Ausdruck, als einfach Digitalisierung. Nachdem man kurz nach dem Milleniumwechsel versucht hat Prozesse in Firmen zu automatisieren, dies aber oft an dem sich sträubenden Personal gescheitert ist, ist es verständlich, dass man einen neuen Anlauf nimmt und nun „Prozesse" als Wort weglässt und nur noch von Digitalisierung spricht.

> *Der Begriff **Digitalisierung** bezeichnet allgemein die Veränderung von Prozessen, Objekten und Ereignissen, die bei einer zunehmenden Nutzung digitaler Geräte erfolgt. Im ursprünglichen und engeren Sinne ist dies die Erstellung digitaler Repräsentationen von physischen Objekten, Ereignissen oder analogen Medien. Im weiteren (und heute meist üblichen) Sinn steht der Begriff insgesamt für den Wandel hin zu digitalen Prozessen mittels Informations- und Kommunikationstechnik. Aussagen zu „Digitalisierung" von Bildung, Wirtschaft und Gesellschaft sind dabei gleichbedeutend mit der digitalen Transformation oder Digitalen Revolution von Bildung, Wirtschaft, Kultur und Politik; dies wird unter den genannten Stichworten behandelt.* Quelle:Wikipedia.org

Natürlich dürfen sie mich ruhig als Lästerer abtun, aber ich bezeichne solche Hypes als Business-Terrorismus. Sie stiften Unruhe in der Arbeitswelt und in den Geschäftsleitungen vieler Unternehmen.

Kleines Beispiel: Ein wichtiger Grosskunde legt dem Lieferanten nahe, dass seine Prozesse unbedingt digitalisiert werden müssen, damit der Kunde besser und jederzeit über den aktuellen Stand in der Versorgungskette Bescheid wüsste. Der Lieferant, sich der Grösse und Wichtigkeit des Kunden bewusst, streckt sich nach der Decke und erfüllt den Wunsch des Kunden, wohlverstanden auf eigene Kosten. Andere Projekte wurden nach hinten geschoben, seine IT-Leute leisten Mehrarbeit, um Anforderungen und Termin zu erfüllen und damit auch den Kunden zu erhalten. Kleiner Makel an der ganzen Sache, testen konnte man nicht, da der Kunde für die Übernahme der digitalisierten Daten überhaupt nicht bereit war.

Nun Fortschritt muss sein, und doch sollte man diesen auch hin und wieder hinterfragen. Zum Beispiel: Wem dient das Vorantreiben einer schlecht definierten Idee? Wem dienen diese digital und möglichst in der Cloud in irgend einem Land gespeicherten Daten? Warum sprechen Politiker uni sono über Digitalisierung und wie wichtig es sei, den Anschluss nicht zu verlieren, ohne aber vergleichbar grosse Anstrengungen für die Weiterbildung aller Betroffenen zu unternehmen? Was steckt dahinter? Dienen diese ganzen Anstrengungen dem grössten Teil der Bevölkerung? Was ist der Einfluss auf Privatsphäre und Sicherheit?

Die Gedanken dazu überlasse ich Ihnen.

Angst vor Fortschritt muss man nicht haben, aber manchmal sollten etwas vertieftere Überlegungen gemacht werden, welche Konsequenzen daraus resultieren. Und nicht zuletzt sollte man sich die Frage stellen: Wie läuft mein Betrieb, wenn einmal das Internet nicht erreichbar ist, oder Hacker einen oder mehrere unserer Server lahm legen?

Lassen Sie sich nicht ins Boxhorn jagen, sondern verlangen Sie Antworten.

Mit freundlichen Grüssen
Andres Koch

Durch modulare Architektur technische Schulden reduzieren

Hintergrund

Gemäss Conveys Law passt sich die Architektur der Organisation an, das ist die eine Aussage. Man könnte genauso gut einen Umkehrsatz zu Conveys Law bilden und sagen «eine Architektur bestimmt die Organisationsform». So praktiziert im modernen Wohnungsbau, wo grosse Laster Fertig-Komponenten in Form von Wänden mit Fenster-Einlassung anliefern und ein grosser Wohnblock quasi zusammengeklebt wird. Diese Häuser entstehen im Sauseschritt.

> ***Das Gesetz von Conway*** *ist eine nach dem US-amerikanischen Informatiker Melvin Edward Conway benannte Beobachtung, dass die Strukturen von Systemen durch die Kommunikationsstrukturen der sie umsetzenden Organisationen vorbestimmt sind. Es wurde von Conway 1968 folgendermaßen formuliert: Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations. Zu Deutsch: Organisationen, die Systeme entwerfen, [...] sind gezwungen, Entwürfe zu erstellen, die die Kommunikationsstrukturen dieser Organisationen abbilden.* Quelle: Wikipedia.org

Ähnliches kennt man auch bestens aus dem IT-nahen Elektronik-Bereich, wo es ohne vorproduzierte Komponenten nicht mehr geht. Geräte aller Art könnten gar nicht in dieser Produktivität und schon gar nicht zu diesen tiefen Preisen entstehen, würden nicht hochintegrierte Schaltungen in Form von ICs (Integrated Circuit) verwendet. Agilität ist hier nicht gefragt, jedoch Effizienz und Geschwindigkeit. Wenn der Entscheid Fertigbau-Komponenten zu verwenden gefallen ist, hat dies natürlich Einfluss auf die Architektur. Das bedeutet, der/die Architekt/-in macht den Entwurf unter Berücksichtigung der vorhandenen Komponenten. Der Kreativität ist hier trotzdem keine Grenzen gesetzt, ja es braucht vielleicht sogar etwas mehr davon und vor allem mehr Überlegung, wie die Komponenten geschickt zum gewünschten Endprodukt zusammengestellt werden können.

Diese Musse zum Überlegen scheint in der agilen Software-Entwicklung nicht mehr üblich zu sein. Bei einer agilen Entwicklungskultur hat eine weitsichtige Architektur einen eher schweren Stand, ja es scheint, man verfalle dem Trugschluss, dass kostengünstige Systeme schneller gebaut werden könnten, wenn man Schicht für Schicht, oder Patch für Patch in hoher Frequenz anbringt. Time to Market ist das A&O. Aber wie würde nun eine komponenten-basierte Architektur zu einer agilen Entwicklungs-Organisation passen? Bestens, würde der Autor meinen, solange man Komponenten hätte und sich die Entwickler daranhalten würden diese einzusetzen, sowie die übergeordneten Architektur-Entscheide zu leben.

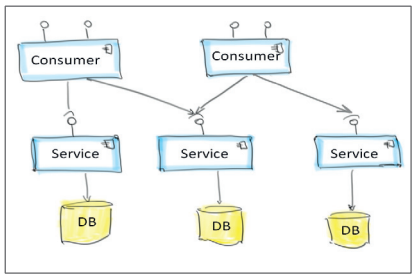
So würde der Umkehrsatz zum Zug kommen, pro Komponente, also für jede Applikations-Domäne (gemäss Domain Driven Design) würde ein Team die Arbeiten machen. Dieses Team hätte das spezifische Domänwissen und könnte trotzdem agil und effizient operieren.

Es braucht aber ein übergeordnetes Architektur- oder Business-Prozess-Team, das sich um die Prozesse und die Gesamtstruktur kümmert, auch wenn dies nicht Agilität auszustrahlen scheint. Es sind auch Regeln gefragt, welche die Interaktionen zwischen den Komponenten und auch die Erstellung der Komponenten definieren. Wenn jede Komponente nach dem Wohlbefinden der Entwickler gebaut würde, dann wäre die Zukunft dieses Gesamtsystems eher düster, denn man würde mittelfristig auf die Müllhalde hin arbeiten. In jeder Industrie wurde Effizienz und Wartbarkeit erst erreicht, als eine normierte Herstellung der Produkte eingeführt wurde. Leider hat sich da auch eine Wegwerf-Mentalität breitgemacht. Auch unvermeidbare, kundenspezifische Anpassungen wären dann sogar einfacher zu bewerkstelligen, als wenn ohne Norm gearbeitet wird. Aber in der Software-Industrie hat man das im Gegensatz zur Automobil-, Elektronik- und sogar Bau-Industrie noch nicht erkannt, oder es hat sich trotz vielen Versuchen zumindest noch nicht durchgesetzt.

Service-Architekturen

Jedem dritten Leser springt bei der Erwähnung von Komponenten unweigerlich der Gedanke von Micro-Services ins Bewusstsein. Das wäre eine gute Assoziation, wobei das Präfix "Micro" vorerst weggelassen werden kann. Wichtig ist eigentlich der Begriff Service. Wir könnten, damit wir alle abholen, auch von Stern-Services (*-Service, * kann für Web, SOA, CORBA, Micro, Nano oder irgendwas ersetzt werden) sprechen, damit alle damit denken können. Aber die Grundlage dazu wurde in IT-Urzeit von David L. Parnas in seinem ACM-Artikel «On The Criteria To Be Used In Decomposing Systems Into Modules» im Dezember 1972 beschrieben. Der Begriff Modul wurde später durch «Klasse» oder eben «Service» in der Realität umgesetzt. Sprachen wie ADA und alle objektorientierten Sprachen haben dieses Konzept gut unterstützt.

In einer heute typisch verteilten Umgebung ist ein Service die oben angesprochene Komponente oder das Modul. Ein Service ist das grundlegende



Bauelement eines jeden verteilten Systems.

Sieht man in System-Architekturen die Verwendung von *-Services vor, dann bekommt man eine Architektur, welche über viele Jahre den neuen Anforderungen angepasst und trotzdem gewartet werden kann. Zwanzig und mehr Jahre sollten da keine Ausnahme sein. Dazu muss man aber gewisse Entwurfs-Richtlinien einhalten, um verteiltes Kopfweh zu vermeiden.

> *Der Begriff* ***Service*** *(auch Dienst oder Daemon) beschreibt in der Informatik allgemein eine technische, autarke Einheit, die zusammenhängende Funktionalitäten zu einem Themenkomplex bündelt und über eine klar definierte Schnittstelle zur Verfügung stellt.* Quelle: Wikipedia.org
>
> *Seit gut 25 Jahren ist die Bezeichnung Service in der Software- und System-Architektur ein Begriff, auch wenn dieser Begriff nie genau definiert oder immer wieder umdefiniert wurde.*
>
> *Dazu kommen die verschiedenen Präfixes, die Service immer wieder vorangestellt wurden und dem Publikum den Eindruck vermittelt haben, es sei etwas total Neuartiges. Daraus entstanden CORBA-, Web-, SOA-, REST-, Mikro- oder sogar Nano-Services.*

Ein Service kapselt eine bestimmte, möglichst einer Business-Domäne entsprechende Funktionalität mit

oder ohne persistente Daten ein, und macht diese über eine oder mehrere Schnittstellen (pro Aspekt) für den Konsumenten zugreifbar. Ein Konsument (Consumer) kann einzelne oder mehrere über die Schnittstellen verfügbaren Funktionen nutzen.

Aber die Grundlage ist und bleibt das Folgende: Eine Software-Komponente mit einer (oder mehreren) klar definierten Schnittstellen (API), klar definierten Meldungsstrukturen mit Protokollen und klar definierten Funktionalitäten.

Ein Service hat kein Wissen über seine Schnittstelle hinaus, d.h. er kennt seine Konsumenten nicht. Services können von anderen Services als Konsumenten genutzt werden, wobei auch hier gewisse architekturbezogene Regelungen berücksichtigt werden müssen. Der Konsument bestimmt die Anforderungen an einen Service. Der Service (respektive sein Designer) hingegen bestimmt seine Schnittstelle. Der Domänen-getriebene Entwurf steht bei der Service-Funktionalität Pate. Trotz der Grundlagen von D. L. Parnas ist der Umfang des Services (Moduls) seit Jahrzehnten eine laufende und nicht enden wollende Diskussion. Aber hier unterstützt uns das Domain Driven Design in der Entscheidungsfindung gut.

Die Schnittstellen-Definition folgt grundlegenden Regeln, die übergeordnet durch die Architektur vorgegeben sein sollten. Zur Schnittstelle gehört die eigentliche, funktionale Schnittstellen-Definition (API), die definierten Meldungs-Formate, die ausgetauscht werden und das Protokoll auf der applikatorischen (nicht technischen) Ebene.

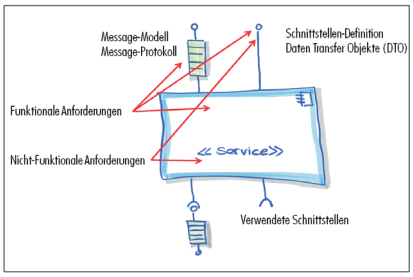
Die übergreifenden Interaktionen, Schnittstellen und die Modulaufteilung sollte Sache der Software Architektur sein. Dies erfordert natürlich wiederum ein sehr gutes Domain-Wissen, aber es darf, obwohl es übergeordnet ist, nicht im Elfenbeinturm entschieden werden, sondern durch fortlaufende Kommunikation mit den Komponenten-Teams und den Business-Anforderungs-Spezialisten. Die Implementation ist Aufgabe der Komponenten-Teams. Schnittstellen-Anpassungen müssen vom Architektur-Team einem Review unterzogen werden.

Effizienz ist wie oft der Dauerbrenner und in einem industriellen Umfeld wie Software-Entwicklung nur dann vorhanden, wenn gewisse Doppelspurigkeiten vermieden werden.

Es ist naheliegend gewisse Service-Eigenschaften wie Konfiguration, Logging, Monitoring und Interprozesskommunikation nur einmal, allgemein auf verständliche Art und Weise zu implementieren und zu dokumentieren.

Services sind die Building-Blocks einer Enterprise-Architektur, egal welche Art von Services es sind. Es sind eigenständige, meist lose gekoppelte Module.

Auch der Kommunikations-Austausch folgt einem bestimmten Muster. Während ein synchroner Aufruf immer eine Antwort zurückliefert, muss das bei einem asynchronen Aufruf nicht immer so sein, oder die Antwort wird verzögert oder gar nicht geschickt.



In einem verteilten System wird ein Service wie eine Blackbox angesteuert, unabhängig davon, ob dieser nun synchron oder asynchron aufgesetzt wird. Solche Systeme erfordern viele Software-Engineering-Skills der Beteiligten, unabhängig davon, ob diese lokal oder in der Cloud betrieben werden. Entwickler/-innen sind heute sehr auf den Code konzentriert und schreiben meist (optimistisch gesehen) guten und „schönen" Programm-Code. Was aber oft und immer mehr untergeht, je kleiner die Komponenten werden, ist das Bewusstsein für Design und die Sicht über den Tellerrand hinaus aufs Ganze. Insbesondere die Aufteilung in einzelne Services ist immer eine delikate Angelegenheit und kann nicht nach sturen Regeln, sondern nur in Bezug aufs Ganze sinnvoll gemacht werden.

Dies ist in der Elektronik ja eigentlich auch der Fall. Es gibt Teams, die darauf spezialisiert sind Komponenten zu entwerfen und deren Funktionalität mit grossem Geschick zu implementieren. Dann kommen aber die kunden-bezogenen System-Entwickler zum Zug, welche diese Komponenten geschickt kombinieren und zu einem ganzheitlichen System machen. Zwei recht unterschiedliche Disziplinen.

Genau diese Trennung, welche in der Elektronik und System-Technik erfolgreich und effizient angewendet wird, setzt sich in der Software-Entwicklung nicht so recht durch. Zumindest nicht im Bereich der Entwicklung von Individual-Systemen. Etwas besser sieht es aus, wenn Cloud-Anbieter fertige Services und Funktionalitäten anbieten, was einen Lichtblick für Software-Systeme aufblitzen lässt. Was bei Elektronik-Komponenten durch harte physikalische Grenzen einschränkt ist, müsste in der Software durch Disziplin und Normen erreicht werden. Ein hoffnungsloses Unterfangen behaupten die einen, die einzige Chance für die Zukunft von software-getriebenen Systemen sagen die anderen.

Migration und Modernisierung

Ein Top-Design in der Entwicklung von Software-Systemen ist gefragt und sollte eingefordert werden. Man kann dies Architektur nennen oder eben Design auf höherer Stufe. Diese Disziplin ist gerade bei der Migration oder Neuentwicklung von Applikationen mit Cloud-Fähigkeit notwendig. Auch wenn man bei einer Migration das innere Gefühl nicht los wird, es sei einfacher und kostengünstiger ein Individual-System durch ein neu entwickeltes Individual-System zu ersetzen, könnte das in einer ernüchternden Erfahrung resultieren. Man darf nicht vergessen, dass ein bestehendes System in der Regel viel Business-Know-how enthält, das man teilweise gar nicht mehr bewusst kennt. Natürlich können oft Teile einer Individual-Software durch neuere, generalisierte, erwerbbare Komponenten ersetzt werden.

Die einer Firma das Alleinstellungsmerkmal verleihenden Spezialitäten sind in der Regel nicht durch „Standard"-Komponenten ersetzbar. Die Frage, wo sich diese Spezialitäten in der bestehenden Software überhaupt befinden, kann bei einer vor zehn oder mehr Jahren entwickelten Applikation fast unbeantwortbar sein. Ebenso kann die Aufteilung einer monolithischen und grossen Applikation in viele kleinere *-Services sowohl technisch, wie auch applikatorisch fast ein Ding der Unmöglichkeit bedeuten wenn man es manuell beantworten will. Hier sind dann technische Hilfsmittel gefragt, welche die zig-Millionen Zeilen von Code und Datenbank-Strukturen durchsuchen und daraus die Grundlagen für ein Re-Design liefern und die Chance bieten, eine Modernisierung zu schaffen, ohne gleich alles wegwerfen zu müssen.

Aktuelles

Brückenwettbewerb

Hier die Wettbewerbsfrage: Wie nennen die Einheimischen die oben abgebildete Brücke und über welchen See führt diese Brücke? Hinweis: Die Schlussrunde an der diesjährigen Tour de Suisse führte kurz vor dem Etappenziel über diese Brücke. Ihre Antwort senden Sie bitte bis zum 30. September an: ***infobridge@objeng.ch***

Der Gewinner wird mit einer süssen Überraschung aus der Gegend der abgebildeten Brücke überrascht werden. ***Wir wünschen viel Glück!***

Lösung Wettbewerb InfoBridge 11

Bei unserer Wettbewerbs-Brücke handelt es sich um die ***Valtschielbrücke*** bei Donat (Graubünden). Sie gilt als ein Paradebeispiel der Brückenbaukunst des Schweizer Ingenieurs Robert Maillart. Sie wurde 1925 erbaut und ist denkmalgeschützt. Die zweite Frage war etwas schwieriger zu beantworten. Was hat diese Brücke mit der Object Engineering zu tun?

Bei gutem Geschäftsgang versuchen wir einen kleinen Gewinnanteil für gute Zwecke einzusetzen. Wir wollten unter anderem ein Projekt unterstützen, das etwas mit dem Symbol zu tun hat, das die Object Engineering seit Beginn an begleitet, das Symbol der Brücke. So fanden wir das Projekt der Sanierung der Valtschielbrücke. Object Engineering hat zur Film-Dokumentation der Sanierung einen Sponsoring-Beitrag geleistet.

Der Gewinner des Wettbewerbs war ***Wale Bucher*** von der Swisscom. Herzliche Gratulation!

Erfolgreicher Abschluss eines Refactoring-Projektes zusammen mit Bison (Schweiz) AG

BISON
www.bison-group.com

Im ersten Halbjahr 2019 haben wir in enger Zusammenarbeit mit unserem Kunden ***Bison (Schweiz) AG*** und dem Einsatz unseres Object Metadata Analyser Werkzeugkasten (***OMAN***) ein Refaktoring-Projekt erfolgreich abgeschlossen.

Das Projekt hatte das Ziel die Kern-Geschäftslogik aus einer umfangreichen monolithischen Vererbungshierarchie mit einem zusätzlichen Geflecht an Hilfsklassen neu zu strukturieren. In der bisherigen sequentiellen Code-Struktur war es eine der grössten Herausforderungen, bei Wartungs- und Erweiterungsarbeiten die jeweils richtigen Codestellen zu finden und sicherzustellen, dass Berechnungsregeln in der richtigen Reihenfolge ausgeführt werden. Mittels ***OMAN*** konnte der zu migrierende Code ermittelt und daraus neue Klassen für die neue Architektur generiert werden.

Aussage des Chef-Architekten Kai-Uwe Hermann: "Wir konnten den Ansatz für einige unserer Problemstellungen als eine pragmatische Methode erkennen, um für die Generierung sehr vieler gleichartiger Code-Elemente einen deutlichen Effizienzgewinn bei der Migration zu erreichen."

Am 6. Mai wurde das Projekt am «21. Workshop Software-Reengineering und -Evolution» in zwei Vorträgen vorgestellt. Beide Artikel dazu («Teilautomatisiertes Architektur-Reengineering in einem JavaEE Monolithen» aus Sicht des Kunden und «Automatisierte Code-Refaktorierung in der Praxis» aus Sicht des Lieferanten) finden Sie unter:
objeng.ch/de/wissen/whitepapers →

Haben Sie spezielle Fragen dazu? Unsere Spezialisten stehen Ihnen gerne zur Verfügung.

Neuer Mitarbeiter im Marketing

Möglicherweise haben Sie bereits eine Mail von ihm erhalten oder ein Telefonat mit ihm geführt.

Beni Holenstein hat seit kurzem unser Marketing übernommen, um die Software-Analyse und das -Refactoring den Software-Architekten, den Software-Herstellern und Softwareabteilungen in grösseren Firmen näherzubringen.

Er leitete beinahe 30 Jahre lang seine eigene IT-Unternehmung im Bereich des Adressmanagements, erstellte Software für die verschiedenen Aspekte und Problematiken, die Adressen mit sich bringen.

In der Freizeit spielt Beni Holenstein gerne Tennis, mag Schwimmen oder mit seinem edlen Mustang GT 5.0 Convertible auszufahren.

Object Engineering GmbH
Birmensdorferstr. 32
CH-8142 Uitikon-Waldegg

Tel: +41 (0) 44 400 47 00

www.object-engineering.ch
kontakt@object-engineering.ch