Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



# SoVeCoPlus

**Software-Vergabe bei Hard- und Software Co¬Entwick¬lung**
**KTI # 12475_1 PFES-ES**

# Software Vergabe und CoEntwicklung

# Vorgehen Tools Verträge

Versionen:

| Rev. | Datum | Autoren | Bemerkungen | Status |
|---|---|---|---|---|
| 0.1 | 13-07-22 | M. Jud | 1. Entwurf | in Arbeit |
| 0.2 | 14-05-26 | M. Jud | Erfahrungen Projektpartner eingearbeitet | in Arbeit |
| 0.3 | 14-08-11 | M. Jud / T. Koller | Struktur angepasst, comedia ergänzt, Checklisten | in Arbeit |
| 0.4 | 14-08-26 | M. Jud / T. Koller / R. Fanger | Vertragliche Elemente, Requirements & Checklisten ergänzt | verteilt MS III |
| 0.5 | 14-11-20 | M. Jud / T. Koller / R. Fanger | Neues Kap 2 eingefügt<br>Musterverträge eingefügt | In Arbeit für MS IV |
| 0.6 | 14-11-26 | M. Jud / T. Koller / R. Fanger | Redaktionell überarbeitet | verteilt MS IV |
| 0.7 | 14-12-30 | M. Jud / T. Koller / R. Fanger | Ergebnisse und Korrekturen aus MS IV eingearbeitet, Arbeitsanhänge entfernt | Release Candidate |
| 1.0 | 15-02-23 | M. Jud / T. Koller / R. Fanger | Tools ergänzt, Schlussredaktion | Release |

# Inhalt

# Einleitung

## 1.1. Kontext

Beim Projekt SoVeCo – Software-Vergabe bei Hard- und Software CoEntwicklung geht es um die Entwicklung eines innovativen interdisziplinären Kooperationsmodells und von Tools zu dessen Unterstützung.

SoVeCo wurde im Rahmen eines KTI-Projektes erarbeitet. Zur Zielerreichung haben wir auf das Begleiten und Analysieren eines realen CoEntwicklungsprojektes gesetzt. Die Idee dahinter ist

- einerseits, durch eine Aussensicht und über das Schulbuchwissen hinaus zu erkennen, welche Schritte oder Unterlassungen im Bereich der gemeinsamen Projektsteuerung und -kontrolle projekterfolgsrelevant sind.
- andererseits sollten die Vorstellungen und Ideen aus dem „Konzept CoEntwicklung", das im Workpackage 1 erarbeitet worden war, verifiziert bzw. falsifiziert werden.

Dass die Beobachtung einen Einfluss auf das beobachtete Objekt hat, ist nicht nur in der Quantentheorie bekannt; auch Menschen verhalten sich anders, wenn sie sich beobachtet fühlen. Ziel unserer Projektbegleitung ist aber, ein möglichst unverfälschtes Bild des Zusammenspiels der Partner bei einer CoEntwicklung zu erhalten.

Entsprechend haben wir versucht, möglichst wenig Präsenz zu markieren und dennoch möglichst viel Einblick in die Prozesse zu erlangen. Keinesfalls wollten wir als Advokat der einen oder der andern Seite auftreten. Und natürlich enthielten wir uns jeglicher beratenden Funktion.

Wir haben eine Reihe von teilstrukturierten Interviews getrennt mit beiden Seiten geführt und uns dazu Notizen gemacht. Wir hatten Gelegenheit bei Profin die Montage der Maschine zu sehen, auch waren wir bei der Präsentation einer Aufwandschätzung des Software-Partners als Beobachter dabei. Nach Abschluss der Entwicklung wurde im Sinne von Lessons learned ein Interview mit beiden Partnern der CoEntwicklung gemeinsam geführt.

In ungezählten Diskussionen und Gesprächen mit einzelnen Projektpartnern und weiteren Experten haben wir Problemfelder, Erfahrungen und Lösungsansätze aus unterschiedlichsten Perspektiven erläutert.

## 1.2. Zweck

Das vorliegende Dokument zeigt die Erkenntnisse aus dem KTI-Projekt SoVeCoPlus auf. Es stellt ein generalisiertes Vorgehen bei Software Vergabe und CoEntwicklung vor und bietet Werkzeuge zur Unterstützung dieses Vorgehens an. Ein Vertragsmodell, das dem erarbeiteten Vorgehen gerecht wird und die CoEntwicklung Partner in ihren Rollen unterstützt hilft, den für beide Seiten anspruchsvollen Weg rechtlich abzusichern.

# 2. CoEntwicklung?

## 2.1. Herausforderungen

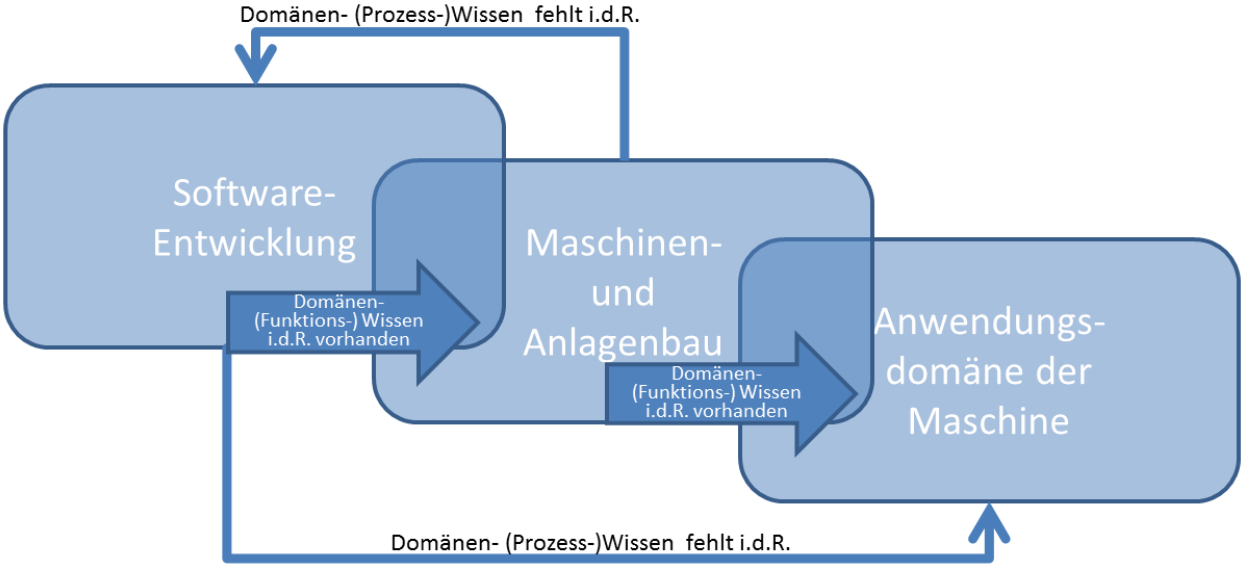
Viele Maschinen- und Anlagebauer sind KMU und haben keine eigene Software-Entwicklungsabteilung. Für die Entwicklungen neuer Produkte, aber auch für die Weiterentwicklung bestehender Produkte werden fallweise externe Software-Entwickler oder auf Softwareentwicklung spezialisierte Engineering-Firmen beigezogen.

In dieser Konstellation ergeben sich in der Regel zwei nicht triviale Herausforderungen für die Beteiligten: Wissensdefizite und Komplexität.

### Wissensdefizite

Der Maschinen- bzw. Anlagebauer hat in der Regel ein gutes Domänenwissen im Maschinenbau einerseits, aber auch im Anwendungsgebiet seiner Anlagen, sonst wären seine Produkte kaum erfolgreich.

Da Software z.B. für den Business-Bereich, oder für die IT-Infrastruktur, oder für mobile Endanwender oder für die Steuerung und Überwachung von Industrieanlagen mit je andern Werkzeugen erstellt, auf andern Rechnern betrieben und nach andern Kriterien bewertet wird, haben Software-Entwickler bzw. Software-Engineering Firmen in der Regel eine gute Fachkompetenz nur in einem Bereich des Software-Engineering.




- *Der Maschinen- bzw. Anlagebauer hat aber in der Regel kaum eigene Kapazität und eigenes Know-How was die Methoden und Vorgehensweisen der Software-Entwicklung angeht.*
- *Umgekehrt haben Software-Entwickler bzw. Software-Engineering Firmen in der Regel kaum Domänenwissen im spezifischen Anwendungsgebiet der Maschinen bzw. Anlagen ihres Auftraggebers.*

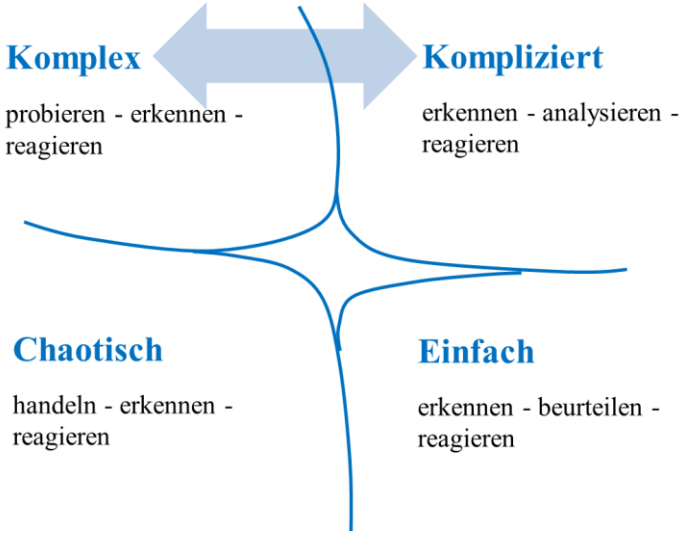
**Komplexität**

Die Entwicklung von Maschinen und Anlagen ist nicht einfach nur anspruchsvoll und kompliziert, sie ist komplex.

Komplexität zeichnet sich aus durch [Fun06]:

- eine grosse Anzahl von Variablen, die die Situation beschreiben
- eine starke Vernetzung der beteiligten Variablen – also gegenseitige Abhängigkeiten
- Polytelie, d.h. viele sich gegenseitig beeinflussende und sogar widersprechende Ziele
- Intransparenz im Hinblick auf die beteiligten Variablen und deren Abhängigkeiten sowie die Zielstellung
- hohe Dynamik der Problemsituation – d.h. unvorhersehbare Änderungen

➢ *Unter diesen – bei der Entwicklung von Maschinen und Anlagen typischen – Voraussetzungen ist eine genaue Planung gar nicht möglich, ein klassisches sequenzielles Vorgehen ist nicht zielführend und es eignet sich nicht für hohe Arbeitsteilung.*




CYNEFIN Strategiemodell [Sno00]

Weil Entwurfsprobleme inhärent unscharf sind, lassen sie sich nicht linear durch Sammeln und Zusammenführen von Informationen lösen. Der Architekt Richard MacCormac [McC76] sagt:

➢ *I don't think you can design anything just by absorbing information and then hoping to synthesize it into a solution. What you need to know about the problem only becomes apparent as you're trying to solve it.*

Zu Beginn ist nicht klar, welches die relevanten Informationen zur Lösung einer komplexen Aufgabe sind. Der Lösungsweg wird wesentlich durch die auf diesem Weg selbst gewonnenen Einsichten mit bestimmt. Design ist opportunistisch.

## 2.2. Falsche Annahmen, falsche Erwartungen

Auf den ersten Blick geht es nur darum, dass der Maschinen- und Anlagebauer die fehlende Softwareentwicklungs-Kompetenz und Kapazität zukauft.

Wenn die oben beschriebenen Herausforderungen, „Wissensdefizite" und „Komplexität" zutreffen, dann reicht der Zukauf von Softwareentwicklungs-Kompetenz und -Kapazität nicht, um erfolgreich neue Produkte zu entwickeln oder bestehende Produkte weiter zu entwickeln.

- Weil die Entwicklung von Maschinen und Anlagen eine komplexe Aufgabe ist, lässt sich im Vorfeld kein vollständiges Lastenheft formulieren, denn erst die auf dem Lösungsweg gewonnenen Einsichten bestimmen, welches die relevanten Informationen zur Lösung der Aufgabe sind.
- Da im Vorfeld weder die konkrete Lösung noch der dazu nötige Entwicklungsweg im Detail bekannt sind, ist eine genaue Planung gar nicht möglich.

Das heisst natürlich nicht, dass Lastenheft und Planung obsolet sind. Es bedeutet aber, dass im Vorfeld die benötigte Funktionalität nur grob skizziert werden kann und entsprechend auch der Realisierungsaufwand nur grob abgeschätzt werden kann.

Lastenheft und Planung sind daher nicht in Stein gemeisselt, sondern müssen entsprechend dem Wissenszuwachs im Projektverlauf nachgeführt werden. Sie sind wichtig zur Kommunikation zwischen Maschinenbauer und Softwareentwickler.

Die Intransparenz bezüglich der beteiligten Variablen und deren gegenseitige Abhängigkeiten verbietet eine hohe Arbeitsteilung in der Lösungsfindung und im Entwurfsprozess, denn weder Maschinenbauer noch Softwareentwickler können alleine entscheiden, ob ihre Entwurfsentscheidungen Konsequenzen für die jeweils andere Domäne haben.

➢ *Mit diesem erweiterten Verständnis der Ausgangslage bei Vergabe eines Softwareentwicklungs-Auftrages wird offensichtlich, dass ein erfolgversprechendes und zielführendes Vorgehen eine enge Kooperation zwischen Maschinenbau- und Software-Domäne erfordert.*

## 2.3. CoEntwicklung und Kommunikation

Eine CoEntwicklung, das heisst ein gemeinsames Erarbeiten der konkreten Lösung, ist der angemessene Umgang mit der vorliegenden Komplexität.

Eine CoEntwicklung macht aus Auftraggeber und Auftragnehmer auf inhaltlicher Ebene Partner auf Augenhöhe. Dabei werden sich die gegenseitigen Wissensdefizite wie oben aufgezeigt bemerkbar machen.

Nach unserer Erfahrung und unseren Erkenntnissen braucht es

- Ein minimales Know-How auf Seite des Maschinen- bzw. Anlagebauers was die Methoden und Vorgehensweisen der Software-Entwicklung angeht
- Ein minimales Domänenwissen auf Seiten des Softwareentwicklers im spezifischen Anwendungsgebiet ihres Auftraggebers.

Zumindest braucht es die Offenheit, das Fehlen dieses Wissens einzugestehen und die Bereitschaft die Wissenslücken gemeinsam zu schliessen [siehe auch Kap. 3.3 Vertrauen].

Zudem braucht es Mechanismen, die in der spezifischen Projektsituation von Hard- und Software CoEntwicklung die Zusammenarbeit und Kommunikation der Projektpartner unterstützen und verbessern.

Im folgenden Kapitel wird ein Kooperationsmodell skizziert, dass diesen Anforderungen Rechnung trägt und auf Vorgehensebene eine gemeinsame Begrifflichkeit anbietet.

# 3. Hintergrund: Das Kooperationsmodell

## 3.1. Anspruch

Das Kooperationsmodell ist ein auf Hard- und Software CoEntwicklungen ausgerichtetes Verfahren von der Vergabe über die Durchführung bis zur Wartung. Es gibt Empfehlungen zum effektiven Einsatz bekannter Softwarevorgehensmodelle (RUP, SCRUM, etc.) und für ein erfolgreiches Zusammenspiel dieser Modelle mit dem Entwicklungsvorgehen im Maschinenbau, das typischerweise intern wesentlich anders strukturiert ist.

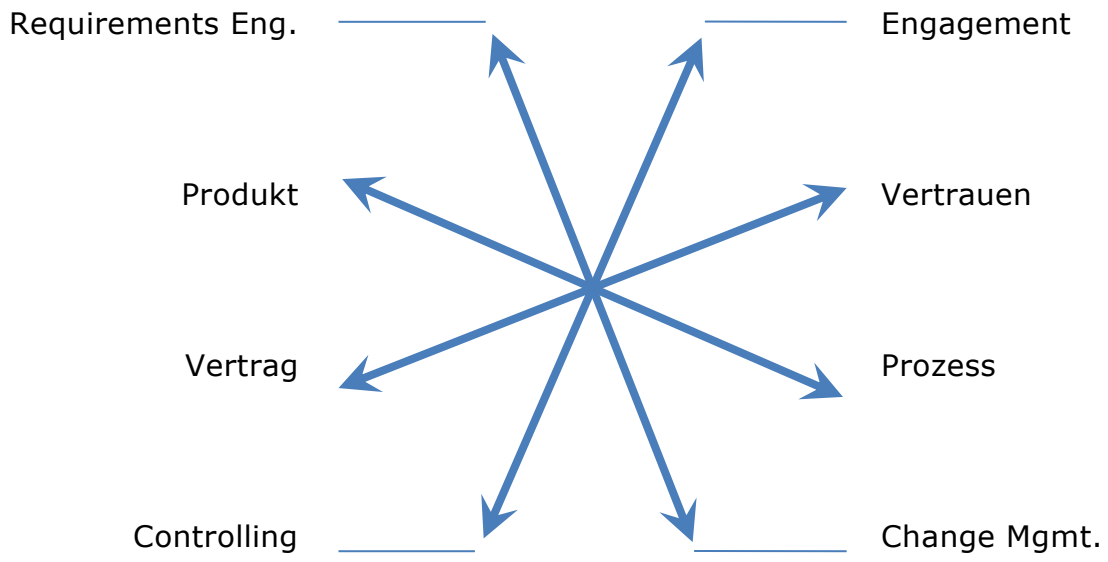
Es bietet Lösungsansätze, Richtlinien und Empfehlungen für die bislang ungelösten kritischen Punkte solcher Hard- und Software CoEntwicklungen:

- Für den Umgang mit sich im Laufe des Projekts *ändernde Anforderungen*
- Für die wegen der gegenseitigen Abhängigkeiten mit einer Ungewissheit versehene *Aufwandschätzungen*
- Für eine realistische *Fortschrittkontrolle* bei unterschiedlichen Vorgehensmodellen in Hard- und Software
- Für die *Qualitätssicherung* wichtiger Produkteigenschaften, deren die Qualität vom Zusammenspiel der Hard- und Softwarekomponenten abhängt
- Für die Inbetriebsetzung und *Abnahme*, insbesondere für das Sicherstellen der im Feld gewonnenen Erkenntnisse und der dort gemachten Änderungen
- Generell für die *Kommunikation* zwischen den Projektpartnern über die Fachdisziplinen hinweg

Das Kooperationsmodell zeigt also *Mechanismen* auf, die in der spezifischen Projektsituation von Hard- und Software CoEntwicklung die Zusammenarbeit der Projektpartner durch geeigneten Einsatz und Kombination bekannter Projekttechniken unterstützen und verbessern.

### 3.2. Duale Sicht der Schlüsselelemente

Dem Anspruch Lösungsansätze und Empfehlungen für die bislang ungelösten kritischen Punkte für Hard- und Software CoEntwicklungen zu präsentieren versuchen wir gerecht zu werden, indem wir die kritischen Problembereiche systematisch mithilfe einer dualen Sicht – nicht im Sinne von Gegensätzen, sondern mehr im Sinn von Yin und Yang, d.h. sich gegenseitig ergänzender Prinzipen – ausloten:




[Fig. 1] Netzdiagramm Polylemma der HW-/SW-CoEntwicklung

Engagement – Controlling

„Lösung und Problemverständnis gehen Hand in Hand" [Nig]: Ich kann die Lösung eines Problems nur delegieren, wenn ich mich damit wenigstens so weit auseinander setze, dass ich es dem Projektpartner verständlich machen kann. Nur dann kann ich den Prozess auch steuern und überprüfen.

Vertrauen – Vertrag

„Vertrauen ist die Grundlage jeder Beziehung" [S&T]: Ich kann mit meinem Projektpartner nur erfolgreich zusammenarbeiten, wenn wir uns gegenseitig vertrauen. Ein fairer und der spezifischen Situation der CoEntwicklung angemessener Vertrag hilft, dieses Vertrauen zu entwickeln.

Prozess – Produkt

„Erst der bewusste Einsatz des Problemlösekreislaufs in der Wissensarbeit sowie der offenen Kommunikation von vorhandenen Wissenslücken ermöglicht eine kooperative Zusammenarbeit im Team" [Dir]: In einer CoEntwicklung sind gegenseitige Wissenslücken alltäglich, der professionelle Umgang damit erfolgsentscheidend.

Change Management – Requirements Engineering

„Requirements Engineering als erster Schritt der Systementwicklung beeinflusst massgeblich den Erfolg eines Projektes" [IRE]. Es liegt aber in der Natur der Sache, dass sich diese Anforderungen im Verlauf einer CoEntwicklung ändern und auch dies muss beherrscht werden.

### 3.3. Vertrauen

Bei einer CoEntwicklung kommt dem Vertrauen zwischen den Projektpartnern eine zentrale Bedeutung zu. Aber gerade in der Situation zwischen CoEntwicklungspartnern ist Vertrauen nicht selbstverständlich, denn die Partner kommen aus unterschiedlichen Kulturen (z.B. Maschinen- / Anlagenbau und Informatik). Die darin begründete potentielle Diskordanz von Werten, Abläufen und Begriffen ist vielfach der Ausgangspunkt einer negativen Vertrauensentwicklung.

Psychologie des Vertrauens

In der Psychologie wird Vertrauen zum Teil als Persönlichkeitseigenschaft, sei es aufgrund frühkindlicher Erfahrung, „Urvertrauen" [Eri] oder im Laufe der Persönlichkeitsentwicklung erworbenes Vertrauen [Rot] gesehen. Andere [Deu] stellen mehr den situativen Aspekt des (bewussten) Vertrauen schenkens / brechens im spieltheoretischen Sinn in den Vordergrund. Die differentielle Vertrauenstheorie [Sch] integriert personale und situative Sicht.

Intra- und Interorganisationales Vertrauen

Betriebliche Organisationen als Gesamtheit können nicht vertrauen, Vertrauen kann nur von einzelnen Individuen entwickelt werden [Plö]. D.h. Vertrauen in einer CoEntwicklung ist Vertrauen zwischen den involvierten Personen. Für dieses haben experimentalpsychologische Untersuchungen [Zan] wirtschaftlich relevante positive Korrelate des Vertrauens belegt: *Danach verbessert Vertrauen vor allem die Qualität der Kommunikation in einer Organisation. Vertrauen entsteht durch Partizipation und Reduktion der Kontrolle.* Probleme werden effektiver gelöst. Umgekehrt leidet in asymmetrischen, nicht reziproken Verhältnissen, z.B. wenn ein Partner über eine höhere punitive Macht verfügt, als erstes die Qualität der Kommunikation: Informationen werden nicht mehr weiter gegeben. [DeD]

Aufbau von Vertrauen

Wie oben ausgeführt, profitiert die Kommunikation von Vertrauen respektive leidet sie unter mangelndem Vertrauen. Umgekehrt trägt vor allem das Kommunikationsverhalten zum Aufbau von Vertrauen bei. [Gri] Wesentliche Aspekte der Vertrauensgenese via Kommunikation sind die Kompetenz, Verlässlichkeit und Klarheit des Gesprächspartners.

Wichtig ist das persönliche Engagement und der persönliche Kontakt: „Erlebtes Vertrauen geht mit dem persönlichen Engagement das man für eine Institution investiert einher, Vertrauen findet seinen Niederschlag immer auch im Handeln". [S&T]

Schliesslich ist *Transparenz ein wesentlicher Baustein zum Aufbau von Vertrauen*: Wer genug über den Partner weiss, um dessen Verhalten vorhersagen zu können, muss sich nicht mehr durch Kontrolle und Bestrafungsandrohung absichern.

### Aufrechterhalten des Vertrauens

Vertrauen und Kommunikation stehen in einem wechselseitigen sich selbst stabilisierenden Verhältnis zueinander. Eine Gefahr stellt aufkeimende Unsicherheit dar [Bec]: diese führt zur Erosion von Vertrauen, was wiederum die Unsicherheit verstärkt. Im *persönlichen Kontakt* regelmässig gemeinsam die Zukunft planen gibt Sicherheit und stabilisiert so das Vertrauen.

Diese Kommunikation setzt eine gemeinsame Sprache voraus. Es ist zwingend notwendig, dass ein *gegenseitiges Domänenwissen* aufgebaut wird:

- Maschinen- bzw. Anlagebauer kommen nicht umhin, sich ein Stück weit mit der Problematik und den Eigenheiten der Software-Entwicklung auseinander zu setzten.
- Umgekehrt müssen sich die Software-Entwickler in die Funktionsweise der zu steuernden Maschine bzw. Anlage hineindenken und darüber hinaus auch ein Verständnis für die damit durchgeführten Prozesse beim Endkunden aufbauen. Denn keine auch noch so präzise Spezifikation kann im kontextfreien Raum alle Fragen beantworten.

Halbwissen – und als Folge davon Verhalten auf Grund falscher Annahmen – ist Gift für Kommunikation und Vertrauen.

### Kontrolle der Kommunikation

In den verschiedenen Phasen eines Co-Entwicklungsprojektes sind die einzelnen Rollen auf Auftraggeber- und Auftragnehmerseite unterschiedlich und in immer neuen Konstellationen gefordert was Intensität und Inhalte der Kommunikation angeht.

Eine regelmässige Abstimmung z.B. alle 14 Tage zwischen den Teams ist wichtig um eine Plattform zu schaffen für den niederschwelligen Austausch zwischen den Teams. Bei einer halbwegs offenen Kultur müssten hier allfällige „bad smells" relativ früh wahrnehmbar sein.

So kann man erkennen, wo es mit der Kommunikation harzt und eigenmächtig auf potentiell falschen Annahmen basierend gearbeitet wird. Das ermöglicht noch Gegensteuer zu geben, bevor Schaden entsteht und Termine verstreichen.

### Vertrauen vs. Vertrag

An der Abschlusstagung des SNF-Projektes »Vertrauen verstehen. Grundlagen, Formen und Grenzen des Vertrauens« führte der Verhaltensökonom Ernst Fehr aus, dass

> *in einer Gesellschaft die Institutionen der Vertragsdurchsetzung Schlüssel des Vertrauens zwischen den Akteuren sind.*
>
> *Wenn man reingelegt wird, kann man zu einem Richter gehen und das einklagen. Das ist eine Staatsleistung, ohne die die Marktwirtschaft nie so funktionieren würde, wie sie es tut.*

Auf unsere Situation der CoEntwicklungsprojekte übertragen, bedeutet dies, dass wir darauf achten müssen, dass die Vertragskonstrukte stabil genug sind, um eine allfällig berechtigte Klage der einen oder andern Seite zu stützen und so den Boden bereiten, für eine vertrauensvolle Kooperation.

*http://www.uzh.ch/news/articles/2013/vertrauen-verstehen.html*

### 3.4. Projektstruktur

Es herrscht kein Mangel an PM Literatur und an Vorgehensmodellen. Dennoch fehlt eine Sicht auf die spezifische Situation von CoEntwicklungen.

Die VDI-Richtlinie 2206 „Entwicklungsmethodik für mechatronische Systeme" beschreibt das Vorgehen bei Entwicklungen an denen mehrere Domänen (Maschinenbau, Elektrotechnik, Informatik) beteiligt sind. [VDI]

Oestereich ist einer der wenigen, die im agilen Scrum Hype den Kopf nicht verloren haben, er beschreibt ein IT-Projektmanagement, das aktuelle SW-Entwicklungsmethodik und professionelles PM verbindet. [Oes]

Auf Basis dieser Publikationen stellen wir eine Projektstruktur vor, welche die Aspekte CoEntwicklung, mehrere Domänen sowie iterativ-inkrementelles und (zumindest auf der Softwareseite) agiles Vorgehen unterstützt: *comedia*

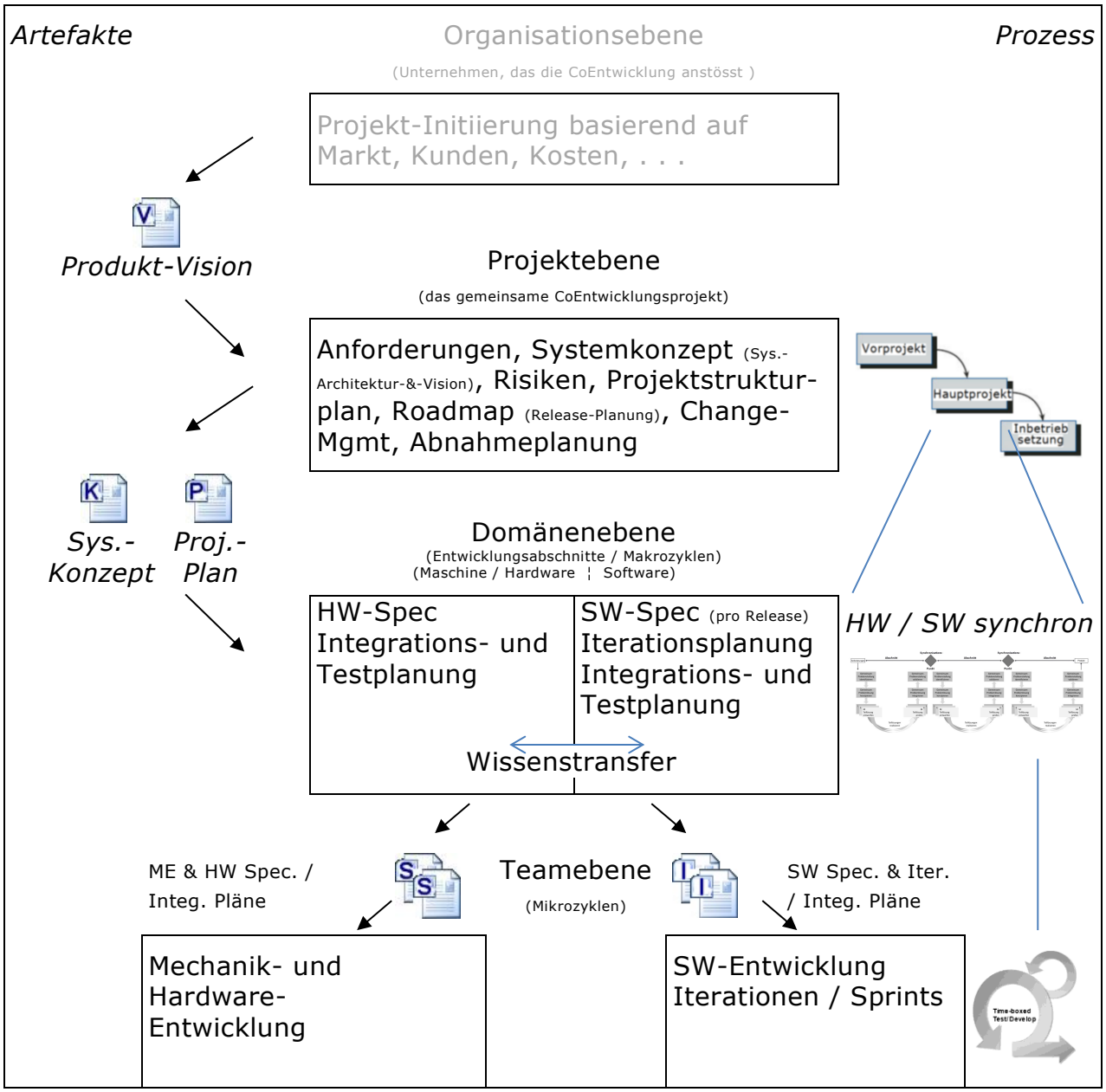
# 4. comedia

## 4.1. Anspruch

Es ist nicht das Ziel von SoVeCo ein weiteres Vorgehensmodell bzw. noch einen Entwicklungsprozess zu schaffen. Aber Maschinenbauer und Informatiker haben eine andere Sprache, andere Vorgehensweisen, andere Werte, eine je andere Projektkultur eben.

*comedia* soll eine neutrale gemeinsame Projektstruktur soweit beschreiben, dass wir eine gemeinsame Begrifflichkeit für das Vorgehen bei CoEntwicklungen haben und wir daran die wichtigen do's und don'ts, unsere Werkzeuge und Checklisten festmachen können. *comedia* ist aber bewusst so offen und einfach gehalten, dass die Projektpartner ihre je eigenen Vorgehensweisen beibehalten können.

## 4.2. *comedia* Ebenen, Artefakte und Prozess-Elemente



[Fig. 2] Übersicht *comedia* Ebenen und Artefakte

### 4.3. Gemeinsame Projektstruktur und Projektleitung

Ein CoEntwicklungsprojekt braucht eine gemeinsame Projektstruktur und eine gemeinsame Projektleitung.

Als Koordinator stimmt der *comedia* Projektleiter die Planung der Teilprojekte auf den Domänenebenen (Maschine / Steuerung / Software / . . .) untereinander ab und stellt sicher, dass zum frühest möglichen Zeitpunkt (Teil-)Integrationen gemacht werden weil nur so der (Software-)Fortschritt zuverlässig bestimmt und Risiken auf Grund von Missverständnissen frühzeitig ausgeräumt werden können.

Als Stratege sorgt der *comedia* Projektleiter für eine partnerschaftliche Arbeitsteilung zwischen Auftraggeber und –nehmer. Er erkennt, wenn z.B. Teilprojekte auf den Domänenebenen sich an lokalen Optima ausrichten und den gemeinsamen Projekterfolg aus dem Fokus verlieren.

Als Mentor gibt der *comedia* Projektleiter sein fachliches und sein Erfahrungswissen an weniger erfahrene Teammitglieder weiter. Er stellt den gegenseitigen Wissenstransfer zwischen den Projektpartnern sicher.

Als „Kümmerer" erkennt der *comedia* Projektleiter frühzeitig „bad smells", Bereiche, wo es mit der Kommunikation harzt und eigenmächtig auf potentiell falschen Annahmen basierend gearbeitet wird, oder wo Termine gefährdet sind, von denen die andern Domänen abhängen (Integrationspunkte), weil z.B. in Teilprojekten ohne Rücksprache mit den Partnern Prioritäten geändert werden.

Technische Geräte, Anlagen und Systeme erreichen oft eine Komplexität, welche die Beteiligten überfordert. Es gibt Menschen, die ein Sensorium für aufkommende Probleme haben, die spüren wo sie welche Fragen stellen müssen um potentielle Schwierigkeiten frühzeitig zu erkennen und zu entschärfen bevor sie zu „show stopper" werden. Der *comedia* Projektleiter hat entweder ein solches Gespür oder hat bewusst Menschen in seinem Umfeld, die über das entsprechende Sensorium verfügen.

### 4.4. Projekt- Domänen und Teamebene

*comedia* trägt der Situation Rechnung, dass in einem CoEntwicklungsprojekt Projektleitungsaufgaben auf unterschiedlichen Ebenen anstehen. Die *comedia*-Ebenen sind:

- Projektebene
  Leitung des gesamten CoEntwicklungsprojektes, typischerweise beim Maschinen-/Anlagenbauer. Hier wird der gesamte Rahmen von Vorprojekt, Hauptprojekt und Inbetriebsetzung Teilsystem- und Domänenübergreifend zusammengehalten.
- Domänenebene
  Leitung der domänenspezifischen Teilprojekte, typischerweise Teilprojektleitungen bei den CoEntwicklungspartnern. Hier werden in enger gegenseitiger Abstimmung die Domänenentwicklungen so geleitet, dass sie inhaltlich und terminlich in den übergeordneten Rahmen passen.
- Teamebene
  Hier erfolgt weitgehend voneinander unabhängig die domänenspezifische Entwicklung.

Die *comedia*-Ebenen kommunizieren über die Artefakte

Die Artefakte sind externe Mittel, die die Bildung eines gemeinsamen mentalen Modells des Projektes und des Produktes für alle Teammitglieder unterstützen.

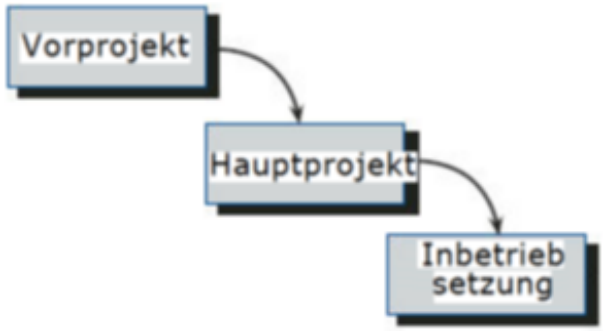
- Produkt-Strategie und -Vision
  Die Lancierung neuer Produkte und Produktlinen ist ein unternehmens-strategischer Entscheid. Das heisst nicht, das die Produktvision unbeleckt jeden technischen Verständnisses am Vorstandstisch entsteht: Die GL wird entsprechende Aufträge an die F&E vergeben zur Ausarbeitung der Produkt-Strategie und -Vision und diese nach einigen Iterationen absegnen. Damit ist sie dann Basis für die Arbeiten in der Projektebene.
- Systemkonzept und Projektphasenplan
  Die Ausarbeitung eines Systemkonzeptes setzt höchste Kompetenz in allen beteiligten Domänen voraus. Eine vernünftige Rahmenplanung ist abhängig von Marktterminen, Ressourcenverfügbarkeit (eigene und beim CoEntwicklungspartner) Materialverfügbarkeit und Werkstattkapazität für den Prototypenbau etc.
  Die Gesamtprojektleitung wird entsprechende Aufträge zur Abklärung an die Teilprojektleitungen vergeben, im Idealfall werden Konzept und Plan am Runden Tisch gemeinsam konsolidiert.
  Dabei werden nicht nur Lieferobjekte auf der Zeitachse abgebildet, es entsteht auch ein Kosten- und Ressourcenrahmen, so dass die Projektpartner auch auf der Zeitachse eine gemeinsame Vorstellung entwickeln, wann welche Ressourcen gebraucht werden und welche Kosten dabei entstehen.
- Spezifikationen, Integrationsplanung, Iterationsplanung
  Die Umsetzung innerhalb der Domänen erfordert spezifische Kompetenzen der einzelnen Mitarbeitenden und profitiert von deren Kreativität. Die Prozesse sind domänen- und firmenspezifisch. Entsprechend erfolgt die Erarbeitung der Detailspezifikationen und Planung in den jeweiligen Teams bzw. mit deren Unterstützung.

Diese Artefakte sind aber auch eine Art Verträge zwischen den *comedia*-Ebenen, sie helfen mit, sicher zu stellen, dass die Teilprojekte in den Domänen weder technisch noch terminlich auseinander laufen:

Stellen sich auf einer unteren Ebene Probleme, die nicht konform der Vereinbarung gelöst werden können, muss diese gemeinsam mit der nächsthöheren Ebene angepasst bzw. nachverhandelt werden, was wiederum bedeuten kann, dass derselbe Prozess u.U. mit der nochmal darüber liegenden Ebene abläuft.

### 4.5. comedia Vorgehensmodell

*comedia* teilt das CoEntwicklungsprojekt in drei Phasen:




- Vorprojekt / Initialisierungsphase
  -> fixer Aufwand, fixe Dauer
- Hauptprojekt
  -> Abschnittweise Projektfreigabe
- Inbetriebsetzung und Wartung
  -> im Voraus vereinbarter Umfang

[Fig. 3] *comedia* Phasen

Vorprojekt

Das Vorprojekt dient dazu den Scope abzuklären, die Machbarkeit zu prüfen und das Hauptprojekt zu planen, insbesondere auch in welchen Zyklen bzw. Abschnitten entwickelt wird und zu welchen Zeitpunkten sich somit die Hardware-Entwicklung und die Software Entwicklung synchronisieren kann und die bis zu diesem Zeitpunkt erstellte Software durch den Auftraggeber abgenommen werden kann. Das Vorprojekt wird mit einem fixen Aufwand und Dauer durchgeführt.

Kennen lernen: Das Vorprojekt dient aber auch dazu, dass sich die CoEntwicklungspartner kennen lernen. Auf der Seite des Software-Partners ist die nötige Fachkompetenz eine doppelte: Softwarekompetenz und Kompetenz in der Anwendungsdomäne. Deshalb ist es eine erste wichtige gemeinsame Aufgabe der Projektpartner von HW/SW-CoEntwicklungen diese Abgrenzung gemeinsam vorzunehmen und den Auftrag gemeinsam zu formulieren.

Anforderung: Es ist nicht realistisch alle Anforderungen an Maschine oder Anlagen von Beginn weg im Detail festzulegen. Zum einen weil der erforderliche Detaillierungsgrad von den Projektbeteiligten abhängig ist (wenn ein Entwicklungsteam vergleichbare Projekte schon gemacht hat, ist vieles selbstverständlich was einem Branchenfremden erläutert werden müsste). Zum andern, weil sich Problemfelder und Klärungsbedarf zum Teil erst im Verlauf der Entwicklung ergeben.

Roadmap: Im Vorprojekt müssen sich die CoEntwicklungspartner die Anforderungen soweit klären, dass der Umfang des Hauprojektes abgeschätzt werden kann und eine inhaltlich sinnvolle Aufteilung in Abschnitte geplant werden kann. Im Hauptprojekt werden dann vor den jeweiligen Entwicklungs-abschnitten die Anforderungen soweit detailliert, dass die CoEntwicklungspartner ihren Beitrag planen und umsetzen können

Ein der häufigsten Unterlassungssünden beim Projektstart ist das nicht Hinterfragen der Projektziele und Auftragsformulierung. Dies ist bei Software Vergaben und HW/SW-CoEntwicklungen doppelt wichtig, denn häufig wird Software unter falschen Annahmen an Dritte vergeben. [siehe Kap. 2.2]

Weitere wichtige Themen bzw. Quellen von Missverständnissen und falschen Annahmen die im Rahmen des Vorprojektes geklärt werden müssen sind:

Weiterentwicklung: Maschinen und Anlagen entstehen oft als Produktefamilien über mehrere Jahre. Für den Auftraggeber sind es zum Teil kleine Anpassungen und Ergänzungen von einem Typ zum nächsten. Auf der Softwareseite kann das aber fundamentale Veränderungen bewirken weil z.B. neu mehrere Prozesse zu synchronisieren sind, weil Daten nicht mehr lokal zu einer Maschine sind, weil Benutzerinteraktion nun auch remote möglich wird etc.

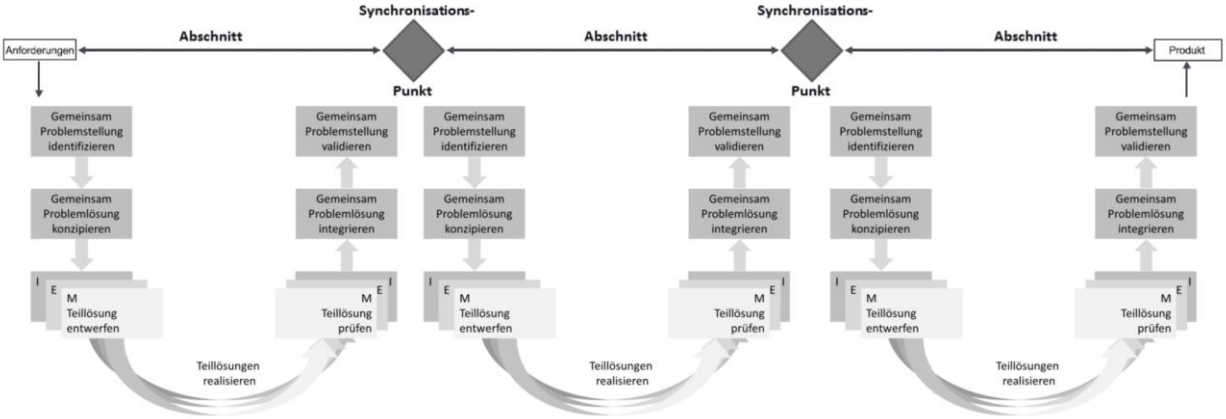
Konfiguration: Maschinen und Anlagen sind kundenspezifisch und kleine Änderungen im Maschinenbau können ernsthafte Konsequenzen für die Software haben, weil z.B. das Timing nicht mehr gleich ist oder eine leicht andere Geometrie neue Einschränkungen für Fahrwege verlangt etc.

Inbetriebnahme: Maschinen und Anlagen werden oft beim Endkunden in dessen spezifischen Konfiguration erstmalig in Betrieb genommen. Das geht in der Regel nicht ohne Inbetriebnahme-Unterstützung durch die Softwareseite.

Wartung: Vernünftige Wartung ist nur möglich wenn beide CoEntwicklungspartner ihr KnowHow aufrecht erhalten oder aber die Software (Quellcode) einschliesslich der Entwicklungsumgebung und der nötigen Dokumentation dem Maschinen- /Anlagebauer übergeben wird.

## Hauptprojekt

Die CoEntwicklung der Hardware und der Software im Hauptprojekt wird in mehreren Entwicklungsabschnitten unternommen, die jeweils in Synchronisationspunkten – d.h. gemeinsamen Meilensteinen bei beiden Entwicklungspartnern – enden. An diesen Meilensteinen wird die bis dahin entwickelte Software abgenommen. Dies bedingt, dass entweder ein Teil der Hardware zur Verfügung steht, auf der die Software getestet und verifiziert werden kann, oder dass dieser Teil vernünftig simuliert werden kann.




[Fig. 4] *comedia* Hauptprojekt: Entwicklungsabschnitte und Synchronisationspunkte

Der Aufwand zur Erstellung bis zum nächsten Abschnitt wird vorgängig geschätzt und das entsprechende Budget dazu freigegeben. Die Länge der Abschnitte ist gegeben durch die Entwicklung der Hardware, sollte jedoch ca. 3-5 Monate nicht überschreiten.

Innerhalb der Abschnitte wird die Software iterativ entwickelt mit Iterationen bzw. Sprints von 2-4 Wochen. Am Ende jeder Iteration steht ein Inkrement der Software zur Verfügung, das potentiell lauffähig wäre, jedoch vom Kunden – mangels Hardware – noch nicht abgenommen werden kann. Trotzdem findet am Ende jeder Iteration ein Treffen mit dem Kunden statt, indem die entwickelte

Software begutachtet wird und der aktuelle Stand, sowie allfällige Änderungen am Projektphasenplan besprochen werden.

Beim gemeinsamen Meilenstein am Ende jedes Entwicklungsabschnitts wird die Software abgenommen. Der Kunde muss sich dabei von der Funktionalität und der Qualität der erstellten Module überzeugen können. Abgenommene Software sollte im späteren Projektverlauf nicht mehr geändert werden müssen, ausser es gebe Änderungen an den Anforderungen.

### Inbetriebsetzung und Wartung

Mit Abschluss des letzten Entwicklungsabschnitts ist also die gesamte Software abgenommen. Insbesondere im Anlagenbau stehen aber nun bei jedem Endkunden des Maschinen- bzw. Anlagebauers wieder Abnahmen an. Oft ist für jede Anlage individuell Software anzupassen oder zumindest neu zu konfigurieren. Dabei können auch Mängel an der im Haus bereits abgenommenen Software zu Tage treten.

Art und Umfang der Unterstützung in dieser Phase ist stark vom Typ der Anlage und von der Organisation des Service beim Maschinen- bzw. Anlagebauer abhängig. Wesentlich ist daher, die benötigte Art und den zu erwartenden Umfang dieser Unterstützung mit dem CoEntwicklungspartner abzusprechen und geeignete Vereinbarungen rechtzeitig zu treffen.

# 5. CoControlling

## 5.1. Herausforderung

Bei der gleichzeitigen Entwicklung von Hard- und Software stellen Projektsteuerung und Controlling eine besondere Herausforderung dar:

- Der Projektfortschritt lässt sich oft nur schwer ermitteln, da die Software schon für sich wenig greifbar ist und in diesem Fall auch noch gegenseitige Abhängigkeiten bestehen.
  So kann zum Beispiel der Entwicklungsstand der Software nicht einfach überprüft werden, wenn die Hardware noch nicht zur Verfügung steht. Anderseits ist die Hardware ohne die zugehörige Software auch nur bedingt überprüfbar und einsatzfähig.
- Sollten Änderungen im Design einer Komponente, z.B. der Hardware notwendig sein – was in solchen Projekten nur schon wegen Änderungen der Anforderung zu erwarten ist – so muss mit grösster Wahrscheinlichkeit auch die Gegenkomponente, z.B. die zugehörige Steuerungssoftware angepasst werden.
  Schon kleine Änderungen im Design der einen Disziplin können grundlegende Anpassungen an der Struktur in der andern Disziplin auslösen

Das Co-Controlling umfasst für diese spezielle Problematik entwickelte Methoden und Tools zur Überwachung und Steuerung des Projektablaufs, die bei Hard- und Software CoEntwicklung die Zusammenarbeit der Projektpartner unterstützen und verbessern.

Gemeinsame Projektsteuerung in allen Phasen

Es gibt einen fundamentalen Unterschied zwischen dem Controlling von Hardware- und Software-Projekten:

<u>Hardware-Projekte</u> haben konventionelle Phasen und Meilensteine (weil Konzeption und Realisierung von unterschiedlichen Fachleuten gemacht werden und die Realisierung oft material- und zeitaufwändig ist). Die Arbeitsergebnisse sind sicht- und prüfbar, bis zu einem gewissen Grad durchaus auch von fachlichen Laien.

<u>Software-Projekte</u> haben iterativ-inkrementelle Vorgehensweisen (weil Konzeption und Realisierung von den gleichen Fachleuten gemacht werden und oft erst die Realisierung zeigt, welche konzeptionellen Fragen überhaupt geklärt werden müssen). Wichtige Deliverables sind nicht sichtbar und erst sinnvoll prüfbar, wenn lauffähige Software vorliegt und diese – möglichst auf der Zielhardware – ausgeführt werden kann.

⇨ ***Software-Controlling ist Blindflug bis zum ersten lauffähigen Prototyp***

*Eine gemeinsame Projektsteuerung ist deshalb nur möglich, wenn ein gemeinsames Projektverständnis aufgebaut wird, welches den fundamentalen Unterschied zwischen dem Controlling von Hardware- und Software-Projekten überbrückt.*

⇨ ***CoControlling setzt Kooperation gegenseitiger Mitwirkung voraus***

## 5.2. Synchronisationspunkte

Weil Software-Controlling bis zum ersten lauffähigen Prototyp Blindflug ist, ist es aus Controlling-Sicht von höchster Bedeutung, dass so früh wie irgend möglich lauffähige Software entsteht und diese auch möglichst realitätsnah (Funktionsmuster der Maschine) demonstriert und getestet werden kann. Erst ab diesem Moment ist der Blindflug beendet und ein effektiver Controlling-Prozess kann einsetzen.

Daraus ergeben sich aus übergeordneter CoControlling-Sicht zwei Ansprüche an die CoProjektleitung und –steuerung:

1. sicherzustellen, dass die Maschinenentwicklung möglichst früh geeignete Funktionsmuster bereit stellt und diese entsprechend dem Projektfortschritt laufend weiter ausbaut.
2. eine Rahmenplanung mit Kosten- und Ressourcenrahmen aufzusetzen, die möglichst gleichverteilt über die erwartete Projektdauer „gute" Meilensteine aufweist, an denen der Projektfortschritt beider Projektpartner effektiv ermittelt werden kann.

Gute Meilensteine

Ein Meilenstein ist ein geplanter Punkt im Projektablauf, an dem vorher festgelegte, messbare (Zwischen-)Ergebnisse vorliegen, die es erlauben, den Projektfortschritt zu festzustellen.

Bei einer Hard- und Software CoEntwicklung sind die zu erwartenden (Zwischen-)Ergebnisse beider Projektpartner naturgemäss unterschiedlich und insbesondere bei der Software nur unzuverlässig messbar, solange diese nicht auf einem Funktionsmuster der Maschine demonstriert und getestet werden kann.

Ein „guter" Meilenstein ist deshalb in einem CoEntwicklungsprojekt wenn immer möglich ein Hardware/Software-Rendez-vous. Er ist damit auch ein Kontrollpunkt, der Auskunft darüber gibt, ob die Projektpartner „aufeinander zu" oder „nebeneinander her" entwickeln. Diese Bedingung erfüllen die Synchronisationspunkte am Ende der Entwicklungsabschnitte.

## 5.3. Planung und Aufwandschätzungen

Referenz für das Controlling ist die Planung und eine Basis dazu sind wiederum die Aufwandschätzungen, die ihrerseits auf den Funktionsbeschreibungen der zu entwickelnden Komponenten basieren.

Die Planung einer CoEntwicklung hängt aber mindestens ebenso sehr von der Ressourcenverfügbarkeit der Partner ab: Viele Rollen in Konzeption, Umsetzung, Test und Inbetriebnahme sind sowohl auf Seite Maschinen-/Anlagebau als auch auf Seite Software-Engineering nur einfach besetzt und sind in der Regel kaum austauschbar, weil viel spezifisches Fachwissen erforderlich ist.

Planung, die einfach die geschätzten Aufwände aufaddiert und im Wesentlichen von einer parallelen Bearbeitung der Hard- und Softwareaufgaben ausgeht, greift zu kurz und kann in der Praxis einer Co-Entwicklung nie eingehalten werden.

Da die Auslastung der Entwickler auf Softwareseite in den frühen Phasen geringer ist, wird ein Software-Engineering Unternehmen seine Mitarbeitenden auch noch auf andern Projekten haben. Dies und der Anspruch, dass Meilensteine

Hardware/¬Software-Rendez-vous sein sollen, führen dazu, dass eine langfristige Detailplanung zum Scheitern verurteilt ist.

Am erfolgversprechendsten ist ein mehrstufiges Verfahren:

1. Rahmenplanung mit Kosten- und Ressourcenrahmen, die möglichst bei Projektstart die wesentlichen Meilensteine inhaltlich und zeitlich festlegt und als Grobplanung für den Ressourcenbedarf der Partner dient.
2. Rollende Planung, jeweils auf den nächsten Meilenstein.

   Der Maschinen-/Anlagebau wird hier eine Arbeitspaket-Planung machen, die Ressourcen, Abhängigkeiten, Materialbedarf und Aufwand in Einklang mit dem Meilenstein-Zielen bringt.

   Auf Softwareseite wird hier in der Regel eine Priorisierung und Konkretisierung des Produkt-Backlogs im Hinblick auf die Meilenstein-Ziele vorgenommen.

   Beide Seiten tun gut daran ihre Ressourcen nicht zu 100% zu verplanen. Mitarbeitende in Entwicklungsabteilungen können in der Regel nur zu etwa 70% am Entwicklungsprojekt arbeiten.
3. Abstimmung der Maschinen-/Anlagebau und Software Planung aus (2.) um das Hardware/Software Rendez-vous am Meilenstein sicher zu stellen. Gegebenenfalls Nachführen der Rahmenplanung mit Kosten- und Ressourcenrahmen aus (1.)

### 5.4. Controlling zwischen den Meilensteinen

In der Regel sind die Meilensteine zu weit auseinander, als dass Controlling und Synchronisierung der Hard- und Softwareentwicklung sich alleine darauf stützen könnten. Eine echte Fortschrittskontrolle ist zumindest auf Softwareseite allerdings nicht möglich, sinnvoller ist daher eine „Ausstandskontrolle". Diese zeigt, was alles noch zu tun bleibt – möglicherweise auch ungeplante Dinge.

Eine regelmässige Abstimmung z.B. alle 14 Tage zwischen den Teams ist aber auch wichtig um sicher zu stellen, dass die Entwicklung im Hinblick Hardware/-Software Rendez-vous synchron verläuft.

Aufgetauchte Probleme und Schwierigkeiten oder gefährdete Termine, von denen die andern Domänen abhängen, sollten hier angesprochen werden. So kann möglicherweise noch Gegensteuer gegeben werden und es lässt sich mit einer regelmässigen Abstimmung verhindern, dass Hard- oder Softwareentwicklung Dinge zurückstellen, die für eine Integration wesentlich sind.

### 5.5. Controlling an den Meilensteinen (Synchronisationspunkten)

Wirklich messen lässt sich der Projektfortschritt an den Meilensteine die echte Hardware/Software Rendez-vous sind, also am Ende der Entwicklungsabschnitte. Hier ist insbesondere auch abzulesen, wie weit die Softwareentwicklung tatsächlich ist. Im Anschluss an einen Meilenstein ist deshalb eine Retrospektive über die abgeschlossene Phase angezeigt (identifizieren von Verbesserungs-potential bezüglich Vorgehen und Kooperation im Projekt).

Controlling zeigt also nicht nur ein allfälliges Abweichen vom Plan, es muss den Regelkreis der Planung schliessen, indem es Ziele und Termine für alle Beteiligten koordiniert nachführt und so sicherstellt, dass immer ein gültiger und verbindlicher Plan existiert.

Am Ende eines Entwicklungsabschnitts wird die bis dahin entwickelte Software auf dem bis dahin entwickelten Hardware-Funktionsmuster abgenommen. Steht keine geeignete Hardware zur Verfügung, so muss diese simuliert werden.

Nur so kann die Software vernünftig getestet und verifiziert werden und nur so kann sich der Maschinen-/Anlagebauer von der Funktionalität und der Qualität der erstellten Softwaremodule überzeugen.

Abgenommene Software sollte im späteren Projektverlauf nicht mehr geändert werden müssen, ausser es gebe Änderungen an den Anforderungen.

Mögliche Ergebnisse und Konsequenzen bei Zwischenabnahmen:

a. [i.O.] Software kann vollständig abgenommen werden und die vereinbarte Funktionalität steht vollumfänglich zur Verfügung. Die Gesamtplanung wird angepasst (wie bei jedem Synchronisationspunkt)

b. [minimale Mängel] Der grösste Teil der Software wird abgenommen, kleinere Teile weisen Mängel auf oder können wegen Mangel an Hardware nicht vollständig getestet werden. Vorgehen: Diese Teile werden in die Planung für den nächsten Abschnitt aufgenommen und in diesem umgesetzt. Die Gesamtplanung wird angepasst (wie bei jedem Synchronisationspunkt)

c. [Hardware nicht verfügbar] Die Software wurde entwickelt, die Hardware steht jedoch nicht zeitgerecht zur Verfügung. Vorgehen: Der Gesamtplan wird um mindestens die Zeit, die es bis zur Erstellung der Hardware braucht, nach hinten geschoben. Falls sinnvoll, kann der nächste Abschnitt trotzdem geplant und durchgeführt werden. Bis zum Ende des nächsten Abschnitts muss die Hardware für den vorhergehenden jedoch zwingend zur Verfügung stehen, ansonsten wird keine weitere Software mehr entwickelt, i.e. es ist bezüglich Hardware höchstens ein Verzug von einem Synchronisationspunkt erlaubt, der Zeitplan wird aber auf jeden Fall gleich angepasst. Ein „Aufholen" der verlorenen Zeit ist nicht vorgesehen. Kann der nächste Entwicklungsabschnitt nicht in Angriff genommen werden, kann der Softwareentwickler ein anderes Projekt Dritter übernehmen oder Maschinenbauer vergütet für die Dauer der Untätigkeit dem Softwareentwickler einen reduzierten Stundenansatz.

d. [Software nicht fertig] Die Hardware wurde entwickelt, die Software wurde jedoch nicht mit der geforderten Funktionalität fertiggestellt oder weist grössere Mängel auf. Vorgehen: Der Abschnitt ist noch nicht beendet, dieser wird neu geplant und dann umgesetzt und abgenommen. Keine Weiterarbeit am nächsten Abschnitt. Neuplanung? Können mehr Ressourcen / Aufwand eingesetzt werden um „Aufzuholen". Falls nicht möglich und Verzögerungen an Gesamtprojekt -> Penalty

e. [Mischung] Funktionalität nicht erfüllt. Wiederholung bis zum Abschluss. Verschiebung des Gesamtprojekts da Risiko bestehen bleibt (auch wenn andere Möglichkeiten bestehen), ev. wieder Änderung/Anpassung der Planung nach Beseitigung des Risikos.

f. Kleiner / grösserer Mangel an Grösse der Aufwandschätzung zur Behebung. Mindestens beim nächsten Abnahmepunkt abgenommen.

### 5.6. Planung des nächsten Abschnitts

Nach dem Spiel ist vor dem Spiel

An den Meilensteinen ist einerseits der aktuelle Stand eines Projektes besonders klar zu erkennen, weil sie ja geplante Punkte im Projektablauf sind, an denen vorher festgelegte, messbare (Zwischen-)Ergebnisse vorliegen. Andererseits markieren sie den Beginn einer neuen Phase, die zwar in der Rahmenplanung / Kosten- und Ressourcenrahmen grob geplant und terminiert wurde, allerdings auf einem früheren und weniger detaillierten Wissensstand.

Effektiver Projektstand: Die nächste Phase muss nun also basierend auf dem effektiven Projektstand (offene Punkte / Nacharbeit) und den bis dahin gewonnenen Erfahrungen (wie kommt man vorwärts, wie läuft die Zusammenarbeit) und Erkenntnissen (wo sind die projektspezifischen Probleme und Risiken) im Rahmen der rollenden Planung detailliert geplant werden.

Anforderungen: Da im Vorprojekt die Anforderungen nur soweit geklärt wurden, wie es zur Abschätzung des Projektumfangs und zur Aufteilung in Abschnitte nötig war, müssen vor den jeweiligen Abschnitte die Anforderungen bedarfsgerecht weiter detailliert werden, damit die CoEntwicklungspartner ihren Beitrag planen und umsetzen können.

Anpassung des Rahmens: Daraus ergeben sich nicht selten auch Anpassungen an Zielen, Terminen und möglicherweise auch am Vorgehen in der CoEntwicklung. Weil der Rahmenplan / Kosten- und Ressourcenrahmen Basis ist für das Vorhalten von Ressourcen bei den Projektpartnern, ist es wichtig, den Regelkreis der Planung zu schliessen, indem Ziele und Termine für alle Beteiligten koordiniert nachführt werden und so sichergestellt wird, dass immer ein gültiger und verbindlicher Rahmenplan / Kosten- und Ressourcenrahmen existiert.

### 5.7. Co-ChangeManagement

In einem Entwicklungsprojekt sind nie alle Anforderungen schon zu Beginn erkennbar, die bekannten Anforderungen können aus verschiedensten Gründen im Projektverlauf ändern und neue Anforderungen können dazu kommen. Es braucht deshalb Mechanismen zum effektiven Umgang mit beiderseitigen Änderungen während der gemeinsamen Entwicklung.

Zu Projektbeginn ist die Flexibilität in Hard- und Software relativ gross. Im Projektverlauf wird es aber zunehmend schwieriger noch Änderungen in der Hardware einzuführen. Alle Anforderungen, die in der Hardware nicht oder nicht vollumfänglich gelöst werden können und praktisch alle Anforderungsänderungen versucht man deshalb mit der Software abzufangen.

Da für Auftraggeber und Projektleiter bei der Software – im Gegensatz zur Hardware – nicht augenfällig ist, welchen Umfang und welche Konsequenzen eine Änderung hat, geht es hier primär darum, dem Changemanager die nötigen Grundlagen für seine Entscheidungen zur Verfügung zu stellen.

Ein etappenweises Vorgehen, wie in *comedia* beschrieben, erleichtert den Umgang mit Änderungen. Denn zu Projektbeginn werden ja sowohl bezüglich der Planung als auch bezüglich der Anforderungen lediglich grobe Ziele und ein grober Rahmenplan festgeschrieben.

Die weitere Detaillierung der Anforderungen und Konkretisierung der Planung erfolgt rollend vor den einzelnen Entwicklungsabschnitten. Änderungen an noch

nicht realisierten Anforderungen und neue Anforderungen können so ohne aufwändiges Changemanagement einfliessen.

Dieser Mechanismus erlaubt es den Entwicklungspartnern, die in der spezifischen Projektsituation von Hard- und Software CoEntwicklung viele der durch Anforderungsänderungen von aussen oder durch Designentscheide in den Domänen gegenseitig verursachten Änderungen in kooperativer Weise zu beherrschen.

Einzig nachträgliche Änderungen, die bereits umgesetzte und abgenommene Funktionalität betreffen, führen zu echtem Mehraufwand.

# 6. Vertragliche Elemente

## 6.1. Allgemeines

Um der zeitgleichen Entwicklung von Hard- und Software auch auf der rechtlichen Ebene Rechnung zu tragen, sind die Ergebnisse des vorliegenden Forschungsprojekts in einen vertraglichen Rahmen zu bringen. Herkömmliche Vertragsmodelle vermögen agiler Projektentwicklung nicht gebührend Rechnung zu tragen ([Frö04], [H/M04], [Heu05], [Mor07], [Slo91], [Str03], [Str07] und [Str08]).
So wird den Beteiligten im Rahmen des vorliegenden Forschungsprojekts mittels Vertragsvorlagen ein Tool für eine CoEntwicklung nach dem in *comedia* und CoControlling beschriebenen Vorgehen überlassen. Ergänzt werden die Vertragsvorlagen mit den Erkenntnissen in Bezug auf die Kosten solcher Projekte aus den Vorschlägen von Dejaeger [Dej11], Oesterreich [Oes06], Auf der Maur/Steiner [A/M11] sowie Gruhn [Gru14].
Entsprechend der Zielsetzung der CoEntwicklung, zeitgleich eine lauffähige Maschine bzw. Anlage mit entsprechender Steuerungssoftware zu entwickeln, sind die Vertragsvorlagen für die Softwaredomäne bestellerfreundlich abgefasst. Bei Verwendung sind die Vertragsvorlagen detailliert auf das konkrete Projekt anzupassen.

Die hier vorgestellte vertragliche Struktur vermag den Anforderungen von *comedia* und CoControlling an das Projekt Softwareentwicklung gerecht zu werden, indem, nach Abschluss des Vorprojektvertrages und der Vorprojektsdurchführung, basierend auf den jeweiligen Erkenntnissen des Vorprojekts zwischen den Parteien ein Rahmenvertrag zur Regelung der grundsätzlichen, für das Projekt Softwareentwicklung geltenden Klauseln abgeschlossen wird. Dieser Rahmenvertrag Softwareentwicklung wird – analog den Vorschlägen des Verbundprojekts Salomo [VPS12] – um werkvertragliche Einzelverträge ergänzt, welche die spezifischen Entwicklungsabschnitte nach *comedia* abbilden.

Ein Gesamtprojekt Hard- und Software CoEntwicklung umfasst demnach in der Domäne der Software die Ebenen Vorprojekt (Vorprojektvertrag), Softwareentwicklung (Rahmenvertrag) und die dazugehörigen Entwicklungsabschnitte (Einzelverträge).

## 6.2. Vorprojekt

Im Rahmen des Vorprojekts ist die Ebene Softwareentwicklung zu planen, soweit und so detailliert dies bereits möglich ist [VPS12]. Nicht zuletzt sollen die Parteien aufgrund des Vorprojekts auch eine Entscheidungsgrundlage erhalten, ob eine tragfähige und produktive gemeinsame Zusammenarbeit im Softwareentwicklungsprojekt überhaupt möglich ist. Dabei sind insbesondere vor Start folgende Punkte vertraglich zu regeln und während des Vorprojekts zu klären:

- Vertragsgegenstand (Scope) Softwareentwicklung
- Planung CoEntwicklung (Anforderungen, weitere Voraussetzungen an Maschine, Hardware und Software, Lastenheft, Projektrisiken, Projektphasenplan [Definition Entwicklungsabschnitte sowie

Synchronisationspunkte Hardware/Software] und Aufwandschätzung) sowie Wissenstransfer vom Besteller zum (Software-)Unternehmer

- Dauer Vorprojekt (befristet)
- Abnahme Ergebnisse Vorprojekt
- Fixpreis (5% Projektsumme [Sockelwert CHF 50'000] zuzüglich Anwaltskosten für individualisierte Ausgestaltung Rahmenvertrag und Einzelverträge: ca. CHF 5-10'000, Kostendach CHF 20'000)

### 6.3. Softwareentwicklung: Rahmenvertrag

Der Rahmenvertrag regelt die Softwareentwicklung und damit das Grundverhältnis zwischen den Vertragsparteien, indem allgemeingültige Regelungen für das Projekt unabhängig von der Art und Anzahl der Einzelverträge definiert werden [VPS12]. In Anwendung eines Rahmenvertrages soll zumindest ein Einzelvertrag für einen Entwicklungsabschnitt abgeschlossen werden [VPS12], so dass die beiden Vertragsebenen zusammen die Domäne Software vertraglich regeln.

Dabei sind insbesondere folgende Punkte vorgängig vertraglich zu regeln:

- *Vertragsgegenstand* (Scope Softwareentwicklung sowie Leistungen [Software-]Unternehmer)
- *Vertragsbestandteile* (Ergebnisse Vorprojekt, Einzelverträge Entwicklungsabschnitte, Begriffsdefinitionen, Checklisten, Kostenplan, Kostenschätzung, Protokolle Projektstandsitzungen, Abnahmeprotokolle)
- *Rangfolge*
- *Projektorganisation* (Projektphasenplan, Abschnittsplanung Entwicklungsabschnitte [Projektstand, Anforderungen an Entwicklungsabschnitte], Weiterentwicklung/Folgeprojekt, Beratung, Terminkontrolle, Kostenkontrolle, Qualitätskontrolle, periodisches Reporting Projektfortschritt und Kosten)
- *Kommunikation und Zusammenarbeit* (Zusammenarbeit Parteien, Projektleiter und Stellvertreter, Projektstandsitzungen, Anzeigepflicht)
- *Termine* (terminliche Festlegung Entwicklungsabschnitte)
- *Abnahme* (Abnahmeperiode, Synchronisationspunkte als Meilensteine [Teilabnahmen Ergebnisse Entwicklungsabschnitte], Schlussabnahme, Vorgehen [keine Mängel/unwesentliche Mängel/Hardware nicht verfügbar/wesentliche Mängel], Abnahmefiktion)
- *Finanzielles* (Agiler Festpreis Softwareentwicklung: Schätzpreis mit Kostendach und Aufwandpreiselementen sowie agiler Festpreis pro Entwicklungsabschnitt mit Teilschätzpreis, Teilkostendach und Aufwandpreiselementen. Überlegungen dazu siehe Dejaeger [Dej11], Oesterreich [Oes06], Auf der Maur/Steiner [A/M11] sowie Gruhn [Gru14].)
- *Verzug* (Schuldnerverzug, Gläubigerverzug)
- *Beizug Dritter* (Unterakkordanten)

- *Urheber- und Nutzungsrechte (Lizenzen)* [Übertragung gewerbliche Schutzrechte an Besteller inkl. Ausschluss Miturheberschaft, Variante Nutzungsrechtsübertragung an Besteller]
- *Quellcode* (Aushändigung/Hinterlegung Quellecode, Installationstools, Entwicklungsumgebung)
- *Dokumentation*
- *Gewährleistung und Garantie*
- *Haftung*
- *Vertragsdauer*
- *Wartung und Weiterentwicklung* (Wartungsverpflichtung, Weiterentwicklungsverpflichtung [Offertstellung Detailkonzept, Zeitplan und Kostenschätzung])
- *Geheimhaltungs- und Rückgabepflicht*
- *Datenschutz- und IT-Sicherheit*
- *Krisenmanagement* (Verpflichtung zu Krisensitzungen u. Krisenmanagement durch unabhängige Dritte)
- *Störung Vertragsabwicklung* (Verpflichtung zur Aufnahme von Verhandlungen bei Störungen in der Vertragsabwicklung)
- *Beweismittel*
- *Vertragsänderungen*
- *Teilnichtigkeit und Vertragslücken*
- *Übertragung von Rechten und Pflichten*
- *Anwendbares Recht und Gerichtsstand*

### 6.4. Entwicklungsabschnitt(e): Einzelverträge 1-n

Während der Rahmenvertrag die allgemeingültigen Vereinbarungen des Vertragswerks umfasst, regelt der Einzelvertrag die Erstellung von Software [VPS12] im Rahmen eines Entwicklungsabschnitts zwischen Synchronisationspunkten oder die Erbringung von Wartungsleistungen nach Abschluss der Entwicklung oder die Erbringung von Beratungsdienstleistungen.

Dadurch ist die flexible vertragliche Ausgestaltung der Softwareentwicklung mittels Entwicklungsabschnitten zwischen den verschiedenen Synchronisationspunkten gewährleistet und den unterschiedli-chen Parteiinteressen kann Rechnung getragen werden.

Die vorliegend erstellte Vertragsvorlage fokussiert auf die Variante der Softwareerstellung, kann aber mit für Wartung, Weiterentwicklung oder Beratung entsprechend angepasst werden.

Zu regelnde Punkte (je nach Gegenstand des Einzelvertrages; keine abschliessende Aufzählung):

- *Vertragsgegenstand* (Scope Entwicklungsabschnitt)
- *Leistungen (Software-)Unternehmer* (Erstellung Software sowie für Anwendung, Inbetriebnahme, Ausserbetriebnahme und Konfiguration

erforderliche Unterlagen, weitere zur erbringende Leistungen, Projektleitung)

- *Lastenheft Entwicklungsabschnitt* (Anforderungen an Softwarekomponente, Erstellung durch [Software-]Unternehmer)
- *Entwicklungsplan* (Festlegung Schlusstermin, Fertigstellungstermin, Vorbereitungszeitraum, Prüfungszeitraum und Zeitraum Probebetrieb)
- *Vergütung* (Agiler Festpreis Entwicklungsabschnitt: Teilkostendach mit Teilschätzpreis und Aufwandpreiselement)
- *Mitwirkung des Bestellers*
- *Schlusstermin*
- *Schlussbestimmungen*

### 6.5. Weiterentwicklung

Die Weiterentwicklung der Maschinen und Anlagen kann vertraglich als neues Softwareentwicklungsprojekt (Rahmenvertrag) mit einem oder mehreren Entwicklungsabschnitten (Einzelverträge) gegliedert geregelt werden, sollten die Grundanforderungen unterschiedlich zum Rahmenvertrag ausfallen. Besteht hingegen keine inhaltliche Abweichung zum bisherigen Rahmenvertrag, kann die Weiterentwicklung unter diesem Rahmenvertrag mittels Einzelvertrag – bzw. bei grösseren Weiterentwicklungen – serieller Einzelverträge geregelt werden, entsprechend den vorgesehenen Entwicklungsabschnitten.

# 7. Hilfsmittel und Werkzeuge

## 7.1. Grundsätzliche Überlegungen

Im Bereich Projektführung gibt es eine Vielzahl von Tools. Welche davon nutzbringend eingesetzt werden, ist stark personen- und organisationsabhängig.

Vielfach lenkt die Beschäftigung mit den Tools von der Beschäftigung mit den Menschen und den wirklichen Problemen ab. Unterschiedliche Werkzeuge bei den Projektpartnern behindern möglicherweise eine gemeinsame Sicht auf das Projekt.

Wir sind daher der Überzeugung, dass weniger oft mehr ist und stellen entlang der Phasen der oben vorgestellten gemeinsamen Projektstruktur *comedia* sowie zum CoControlling eine Reihe von Checklisten zu Verfügung.

So können die Projektpartner ihre jeweiligen Prozesse und Werkzeuge nutzen und dennoch sicherstellen, dass im CoEntwicklungsprojekt die kritischen Punkte berücksichtigt werden.

## 7.2. Checklisten

1) Checkliste Vorprojekt
   a) Scope des Projekts vereinbart und festgelegt
   b) Visions Dokument geschrieben
   c) Anforderungen soweit erfasst, siehe Checkliste Anforderungen
   d) Risiken erfasst, bewertet und Massnahmen vereinbart
   e) Machbarkeit geklärt
   f) Erste Aufwandschätzung und Grobplanung des Projektverlaufs erstellt
   g) „Synchronisationspunkte" (ca. 3 monatlich, Zwischenabnahmen) festgelegt Abnahme-Kriterien & -Zeitplan > Testsysteme (> Zusatzaufwand)
   h) Wichtige Rollen und Verantwortlichkeiten geklärt
   i) Vertrag für den weiteren Vorgehen aufgesetzt, siehe Template Vertrag
   j) Ansprüche bezüglich Wartung geklärt, siehe Checkliste Wartungsvereinbarung
   k) (ev. Aufsetzen der Tools für die nachfolgende Phase)

2) Checkliste Anforderungen
   a) Projektziele und Rahmenbedingungen sind den CoEntwicklungspartnern bekannt (wird im Vorprojekt ermittelt)
   b) Grobe Anforderungen an Maschine / Anlage und deren Komponenten soweit geklärt und festgehalten, dass eine grobe Aufwandabschätzung und Grobplanung des Hauptprojektes sowie eine sinnvolle Aufteilung in Abschnitte gemacht werden kann (im Vorprojekt )

c) Anforderungen an Maschine / Anlage und deren Komponenten soweit detailliert und ergänzt dass Aufwandabschätzung und Planung des nächsten Projekt-Abschnitts gemacht werden können (im Hauptprojekt Synchronisationspunkten)

d) Anforderungen durch Vorbild: Werden Anforderungen nicht aufgeschrieben sondern dadurch gegeben, dass eine Komponente gleich funktionieren soll, wie ein bereits bestehendes Produkt, muss den Beteiligten klar sein, welche Aspekte des Vorbilds Anforderungscharakter haben (z.B. Farbe, Gewicht, Form, zeitliches Verhalten . . .) sonst ist diese Art der Spezifikation unbrauchbar.

e) Für die Software ist es oft wesentlich
- time constraints
- Mengengerüste der Daten und Häufigkeit mit der diese anfallen
- Kommunikationsart (push/poll/interrupt Änderung/Istwert etc.)
mit den Ansprüchen der Maschine / Anlage abzustimmen

3) Checkliste Kommunikation

a) Sitzungsstruktur und Rhythmus vereinbart, z. Bsp. monatlich bei Sprints des Software Entwicklers

b) Regelmässige Abstimmung vereinbart (innerhalb Sprints, wöchentlich)

c) Behandelte Themen bei Abstimmung vereinbart, zum Beispiel angelehnt an Scrum Stand-Up Meeting:

- Was wurde seit dem letzten Meeting gemacht
- Was ist bis zum nächsten Meeting geplant
- Was sind die aktuellen Probleme

d) gegenseitige Ansprechpartner vereinbart

e) Themen Abstimmung

4) Checkliste Planung

a) Grobplan erstellt

b) Synchronisationspunkte festgelegt

c) Iterationszyklen der Software Entwicklung festgelegt (Sprints)

d) Kommunikation vereinbart (siehe Checkliste Kommunikation)

e) Tool / Methode zur Erfassung des Aufwands festgelegt

5) Checkliste bad smells / Code Qualität / Architektur

a) Ist die Software-Architektur dokumentiert? (Beschreibung durch Architektur Sichten)

b) Ist die Architektur einfach und verständlich

c) Ist die Beziehung zwischen Anforderungen und der entwickelten Architektur klar?

d) Ist die Architektur vollständig? Sind alle Anforderungen damit abgedeckt?

e) Sind die Komponenten genügend beschrieben? (Interfaces, Abhängigkeiten, Beziehungen etc.)

f) Es gibt keine zyklischen Abhängigkeiten zwischen Paketen / Komponenten

g) Coding Guidelines vorhanden

h) Metriken vorhanden und mittels Werkzeugen überprüft

i) Regelmässige Code Reviews sind vereinbart und werden durchgeführt

j) Klassische Bad Smells werden mit Tools oder mit Code Reviews regelmässig überprüft (Code Duplizierung, Lange Methode, Grosse Klasse, Zuviele Parameter, Feature Envy, Parallele Vererbungshierarchien, Lazy Class, etc.)

k) Regelmässige Refactoring wird geplant und durchgeführt.

6) Checkliste Wartungsvereinbarung

a) Supportlevel vereinbart?

b) Bereitschaftszeiten vereinbart?

c) Responsezeiten vereinbart?

d) Inbetriebsetzungsunterstützung vereinbart?

e) Vorlaufzeiten für Entwicklung neuer Funktionalität vereinbart?

f) Rechte am Quellcode und dessen Verwahrung vereinbart?

g) Verfügbarkeit der Entwicklungsumgebung sichergestellt?

h) Möglichkeit der Weiterentwicklung durch Dritte geklärt?

7) Checkliste SynchPunkt

a) Hardware/Software-Rendez-vous organisiert
(Verfügbarkeit von Hardware und Software, personell, terminlich)

b) Software & Hardware wurden abgenommen
=> Checklist Abnahme

c) Planung (inhaltlich, personell, terminlich, Budget)
bis zum nächsten Meilenstein wurde erstellt

d) Gesamtplanung wurde angepasst

e) nächster Entwicklungsabschnitt freigegeben, Kickoff terminiert

f) Projektvorgehen reviewed und gegebenenfalls Anpassungen beschlossen

8) Checkliste Projektstart

a) Planung siehe Checkliste Planung

b) Sind alle Ergebnisse des Vorprojekts verfügbar und wurden kommuniziert?

c) Gab es in der Zeit zwischen Ende des Vorprojekt und Projektstart Änderungen? Wenn ja sind diese in die Projektergebnisse des Vorprojekts und in die Planung eingeflossen?

d) Ist das weitere Vorgehen klar für alle Beteiligten?

e) Alle Stakeholders sind identifiziert und sind vertreten

f) (Projekt Sponsoring aufgesetzt?)

9) Checkliste Projektabschluss

a) Wurden alle Projektergebnisse inkl. Dokumentation an den Auftraggeber abgegeben?

b) Wurde die Abnahme formell durchgeführt und das Abnahme Dokument unterzeichnet?

c) Wurde ein Projektabschlussbericht erstellt?

d) Wurde ein Projekt Abschluss Workshop durchgeführt (alle Parteien) ? Kritische Bewertung / Würdigung, Zusammenarbeit, Projektverlauf, Projektergebnis, Identifikation von Restaufgaben, Gemeinsame Analyse der aufgetretenen Schwierigkeiten und Planänderungen

e) Wurde ein Projektreview durchgeführt (Erfahrungssicherung für weitere Projekte, einzelne Parteien): Analyse der ereichten Ergebnisse, Diskussion von Problemen / Schwachstellen, was war gut, was lief nicht gut, welche Planänderungen gab es und was waren die Ursachen, welche Checklisten / Templates sind zu ergänzen

f) Abschlussevent

10) Checkliste externes Audit

a) Prüfung Projekt Ergebnisse: Wurden die Abnahmen entsprechend durchgeführt (Checkliste Abnahme) und ist demzufolge die Qualität der Ergebnisse wie definiert

b) Prüfung Stand des Projekts: Entsprechen die Projektergebnisse zu diesem Zeitpunkt den geplanten Ergebnissen? Wenn nicht, was sind die Differenzen und was sind die Gründe?

c) Prüfung der Planung: Ist die Planung noch aktuell bzw. nachgeführt? Wurden SynchPunkte und Sprints eingehalten?

d) Prüfung Vorgehen: Wurde im Projekt gemäss diesen Vorgaben vorgegangen? Wo gab es Abweichungen? Wie ist die Kommunikation zwischen den Projekt Partnern

e) Prüfungen Anforderungen: Ist der Scope des Projekts noch der selbe

f) Wurden für das Audit Gespräche mit den Projekt Partnern unabhängig und einzeln durchgeführt?

11) Checkliste Abnahme (auch für Teilabnahmen an SynchPunkten)

a) Die Software ist lauffähig und testbar (Hardware verfügbar oder simuliert).

b) Abnahmetests für die Software wurden erfolgreich durchgeführt.

c) Die Software Spezifikation (Architektur Dokument) wurde ergänzt.

d) Es wurde ein Code Review durchgeführt.

e) Die Software Qualität wurde geprüft (?) (Metriken, Code Review, ...)

### 7.3. Tool-Unterstützung

Aus Sicht des Maschinen-/Anlagebauers besteht der grösste Bedarf für ein Werkzeug sicher im Bereich des Controllings.

Wie in Kap. 5 beschrieben, ist allerdings Software-Controlling Blindflug bis zu ersten lauffähigen Prototypen. In diesem Sinne ist das verlässlichste Werkzeug die Etappierung in Entwicklungsabschnitte und stufenweise Abnahme der Software an den Synchronisationspunkten.

Zur Unterstützung der Kommunikation zwischen Maschinen-/Anlagebauer und Softwareentwickler ist ein einfaches (aber aufwändig zu pflegendes) Werkzeug eine Liste mit allen Funktionsbeschrieben (FUB) der Maschine und den zugehörigen Software-Spezifikationen (SDD).

Dabei geht es v.a. darum, dass

- leicht erkennbar ist, wenn zu einer Funktion die FUB oder die SDD fehlt
- gegenseitig bestätigt werden muss, dass FUB bzw. SDD in Ordnung sind
- offene Fragen verfolgt werden können

| Nr. | FUB (Funktionsbeschrieb Maschine) | ok | SDD (Software Spezifikation) | ok | offene Fragen |
|---|---|---|---|---|---|
| 1 | Link oder Beschrieb | | Link oder Beschrieb | | |

^ ok durch SW ^ok durch Masch.

In der Regel müsste wohl der Maschinenbauer eine solche Liste führen und der Software-Partner die Verständlichkeit der FUB bestätigen, SDD Links einfügen bzw. bei Bedarf offene Fragen formulieren. Umgekehrt würde der Maschinenbauer die Verständlichkeit der SDD bestätigen bzw. ebenfalls bei Bedarf offene Fragen formulieren.

Zeilen, welche z.B. noch keine SDD haben, oder bei denen die FUB noch nicht bestätigt wurde, müssen leicht gefunden / ausgewählt werden können.

#### Allgemeine Tools

Allgemein gibt es eine Reihe möglicher Tools für Planung und Controlling:

- Jira
  webbasiertes all-in-one Projektverfolgungstool für Teams zur Fehler-verwaltung, Problembehandlung und operativem Projektmanagement.

- Redmine
  freie, webbasierte Projektmanagementsoftware. Kann für Projektplanung, Issue-Tracking, Diskussionsforen, Wikis, etc. eingesetzt werden.

- ScrumDo
  webbasiertes Scrum-Board mit Unterstützung für Story-Verwaltung Sprint- und Releaseplanung, Statistik etc.

- Kanbanery
  einfaches, webbasiertes Kanban Aufgaben- bzw. Projekt-Management und Verwaltungswerkzeug das auch Problembehandlung unterstützt.

# 8. Abkürzungen und Begriffe

CoEntwicklungspartner kommen aus unterschiedlichen Kulturen: Maschinen- / Anlagenbau und Informatik haben unterschiedliche Werte, Abläufe und Begriffe. Wenigstens die Begriffe wollen wir hier klären.

| | |
|---|---|
| agil | hier: flexible und schlanke Vorgehensweise in der (Software-)Entwicklung, die mehr auf technische und soziale Aspekte achtet als die klassischen / rigiden Vorgehensweisen. |
| Artefakt | hier: ein Produkt (Dokument, Diagramm, Code etc.), das als Zwischen- oder Endergebnis in der (Software-) Entwicklung entsteht. |
| bad smell | (englisch für schlechter Geruch) Dinge im Projekt die nicht gut laufen. Projektverantwortliche, die eine „gute Nase" haben gehen diesen Dingen nach bevor es schlimmer kommt. |
| Change Management | Prozess der festlegt, wie Änderungen und Erweiterungen von Produktanforderungen während der Entwicklung gehandhabt werden. |
| CoEntwicklung | Gemeinsame parallele Entwicklung unterschiedlicher Teilsysteme durch mehrere Entwicklungspartner. |
| comedia | Neutrale, Projektstruktur, die den Partnern einer CoEntwicklung einen gemeinsamen Rahmen gibt, ohne ihnen eine bestimmte Vorgehensweise aufzuzwingen. |
| Deliverables | (englisch für Lieferobjekte) Projekt(zwischen)-Ergebnisse die an den Kunden geliefert werden. |
| Domäne | Fachbereich bzw. Wissensgebiet, Themenfeld das Gegenstand einer inhaltlichen Spezialisierung ist.<br>Hier: der Maschinen- bzw. Anlagenbau einerseits und die Softwareentwicklung andererseits. |
| Endkunde | hier: Der Kunde des Maschinen- bzw. Anlagenbauers (der wiederum Kunde des Softwareentwicklers ist). |
| Entwicklungsabschnitt | Etappe in einem CoEntwicklungsprojekt von ca. 3 bis 5 Monaten, in der Hard- und Software so koordiniert entwickelt werden, dass am Meilenstein an deren Ende die Software auf Hardware abgenommen werden kann. |
| F&E | Forschung und Entwicklung. |
| Funktionsmuster | Ein Funktionsmuster dient zur Versuchsdurchführung und zum Test einzelner Teilfunktionen des projektierten Seriengerätes. Dies im Gegensatz zum Prototyp der (im Maschinenbau) ein seriennahes Gerät ist, das in Form, Gestalt, Bedienung und Herstellung dem Endprodukt bereits nahe kommt. |

| | |
|---|---|
| Hardware | hier: Maschine inkl. Elektronik (im Gegensatz zur Software). |
| HW/SW-CoEntwicklung | Hardware- / Software-CoEntwicklung, gemeinsame parallele Entwicklung von Maschine (inkl. Elektronik) und Software durch mehrere Entwicklungspartner, i.d.R. den Maschinen- bzw. Anlagenbauer und den Software-entwickler. |
| HW/SW-Rendez-vous | Hardware- / Software-Rendez-vous, Meilenstein am Ende eines Entwicklungsabschnitts, in dem Hard- und Software so koordiniert entwickelt wurden, dass die Software auf Hardware abgenommen werden kann. |
| Inkrement | Zuwachs an laufähiger Software die in einer Iteration (in einem Sprint) entwickelt und getestet wurde. |
| iteratives Vorgehen | Methode im Projektmanagement v.a. im Software-bereich, die Aufgabenabfolge Analyse – Design – Implementierung – Test wird mehrfach durchlaufen und das Produkt entsteht inkrementell in einer Abfolge von Prototypen. (vgl. sequenzielles Vorgehen) |
| Konfiguration | Eine spezifische Kombination von konkreten Versionen von Hardware, Firmware und Software die zusammen in einem Produkt oder System funktionieren. |
| KTI | Kommission für Technologie und Innovation, Förderagentur für Innovation des Bundes. |
| Lastenheft | Anforderungsspezifikation des Auftraggebers an die Lieferungen und Leistungen eines Auftragnehmers. (vgl. Pflichtenheft) |
| Pflichtenheft | Vom Auftragnehmer auf Grundlage des Lastenheftes erstellte Spezifikation, die beschreibt, wie er die Anforderungen im Lastenheft zu lösen gedenkt. (vgl. Lastenheft) |
| Prototyp | In der Software wird jedes Funktionsmuster – sei es zum Test einzelner Teilfunktionen oder sei es ein seriennahes Stück Software – als Prototyp bezeichnet. |
| Requirements Engineering | Prozess der festlegt, wie Produktanforderungen bei einer Entwicklung ermittelt, dokumentiert, abgestimmt und verwaltet werden. |
| Road map | (englisch für Strassenkarte) im übertragenen Sinn Planskizze für ein prozesshaftes Vorhaben. |
| RUP | Rational Unified Process, iteratives Vorgehensmodell in der Softwareentwicklung. |
| Scope | (englisch für Spielraum, Umfang, Rahmen) Abgrenzung der Entwicklungsaufgabe im (Teil-) Projekt. |
| Scrum | (englisch für Gedränge) iteratives Vorgehens-Framework in der Softwareentwicklung. |
| SNF | Der Schweizerische Nationalfonds ist eine privat-rechtliche Stiftung, die im Auftrag des Bundes die |

| | |
|---|---|
| | Grundlagenforschung in allen wissenschaftlichen Disziplinen fördert. |
| SoVeCo | Software Vergabe und CoEntwicklung, Name des ursprünglich eingegebenen (aber von der KTI abgelehnten) Projektes. |
| SoVeCoPlus | Software Vergabe und CoEntwicklung Plus, Name des überarbeiteten KTI Projektes, dessen Ergebnisse hier vorliegen. |
| sequenzielles Vorgehen | Klassische Methode im Projektmanagement, die Phasen stehen für Aufgaben die nacheinander abgearbeitet werden. Im Maschinebau Anforderungen – Funktionsstruktur – Konstruktion – Produktion und Test, in der Softwareentwicklung Analyse – Design – Implementierung – Test. (vgl. inkrementelles Vorgehen) |

# 9. Literaturverzeichnis


[Alt06] Altmann D. et. al., Cooperative Product Engineering, 2006, Proceedings of the 35. IGIP annual Symposium, ieee asee sefi, Tallinn.

[A/M11] Auf der Maur, Rolf / Steiner, Thomas, Die Quadratur des Kreises, in: Swiss Made Software - Das Buch 2011, S. 102 f.

[BMB06] Bundesministerium für Bildung und Forschung, 2008, http://www.softwarefoerderung.de.

[Dej11] Dejaeger, Glenn, Scrum and fixed price; impossible? about: software development, 30. Januar 2011.

[Frö04] Fröhlich-Bleuler, Gianni, Softwareverträge, Bern 2004.

[Fun06] Funke, Joachim, Komplexes Problemlösen. In J. Funke (Ed.), Denken und Problemlösen, Hogrefe ,Göttingen 2006.

[Gru14] Gruhn, Volker, Unberechenbares Berechnen, in www.dotnetpro.de, 11/2014, S. 9 ff.

[H/M04] Heusler, Bernhard / Mathys, Roland, IT-Vertragsrecht, Zürich 2004.

[Hei05] Heinzelmann Elsbeth, Der kürzere Weg zum mechatronischen System. In: Mechatronik F&M, München Carl Hanser Verlag 8-9/2005, S. 53ff.

[Heu05] Heusler, Bernhard, Der Software-Entwicklungsvertrag, Tagungsbeitrag Tagung zum Internet-Recht und IT-Verträge vom 11.05.2004, in: Jörg, Florian S./Arter Oliver [Hsg.], Internet-Recht und IT-Verträge, Bern 2005, S. 49-125.

[Jud09] Jud, Martin, Kosten und Nutzen von Vorgehensmodellen, 2009, in OBJEKTspektrum 01 2009, http://www.sigs.de/publications/os/2009/01/jud_OS_01_09.pdf.

[McC76] MacCormac, Richard C., Design is...(Interview with N. Cross), BBC/Open University TV broadcast 1976.

[Moo06] Moore, James W., Converging Software and Systems Engineering Standards, IEEE Computer, September 2006.

[Mor07] Morscher, Lukas, Leistungsbeschrieb, Gewährleistung und Haftung in IT-Verträgen, Tagungsbeitrag Tagung IT-Verträge vom 17.11.2006, in: in: Jörg, Florian S./Arter Oliver [Hsg.], IT-Verträge, Bern 2007, S. 73-114.

[Oes06] Oestereich, Bernd, Der agile Festpreis und andere Preis- und Vertragsmodelle, OBJEKTspektrum, 01/2006 p30ff.

[Slo91] Slongo, Doris, Der Softwareherstellungsvertrag, [Diss.] Zürich 1991.

[Sno00] Snowden, David J, Cynefin, A Sense of Time and Place: an Ecological Approach to Sense Making and Learning in Formal and Informal Communities, conference proceedings of KMAC at the University of Aston, July 2000.

[Str03] Straub, Wolfgang,Informatikrecht, Zürich 2003.

[Str07] Straub, Wolfgang, Kostenüberschreitungen in IT-Verträgen, Tagungsbeitrag Tagung IT-Verträge vom 17.11.2006, in: in: Jörg, Florian S./Arter Oliver [Hsg.], IT-Verträge, Bern 2007, S. 115-154.

[Str08] Straub, Wolfgang, Verantwortung für Informationstechnologie, Zürich/St. Gallen 2008.

[VDI06] VDI, Entwicklungsmethodik für mechatronische Systeme, 2006, Beuth Verlag, Berlin.

[VPS12] Verbundprojekt Salomo, Erleichterung der Gestaltung und Abwicklung von Softwareerstellungs-Verträgen zwischen KMU (Maschinenlesbare Spezifikationen von bestimmten Anforderungen und deren automatische Überprüfbarkeit durch vereinbarte Werkzeuge), Freiburg/Saarbrücken, 28. Juni 2012.

[Wei06] Weisshaupt Bruno, SystemInnovation, 2006, Orell Fuessli.

[Wen03] Wendt, Siegfried, Initiative „Kommunikation in der Softwareentwicklung“ Position Statements, Dez. 2003, http://www.fmc-modeling.org/download/projects/kommse/KommSE-Position-Statements.pdf.

[Züh08] Zühlke, Prozess für Produktentwicklung, 2008, http://www.zuehlke.com/fileadmin/pdf/flyers/fl_055_d_produktentwicklungsprozess.pdf.