Agilität in Großprojekten durch „Integration Driven Design"

Ein Erfahrungsbericht

Stephan Jacobs, FH Aachen, jacobs@fh-aachen.de
17.06.2010

 www.fh-aachen.de

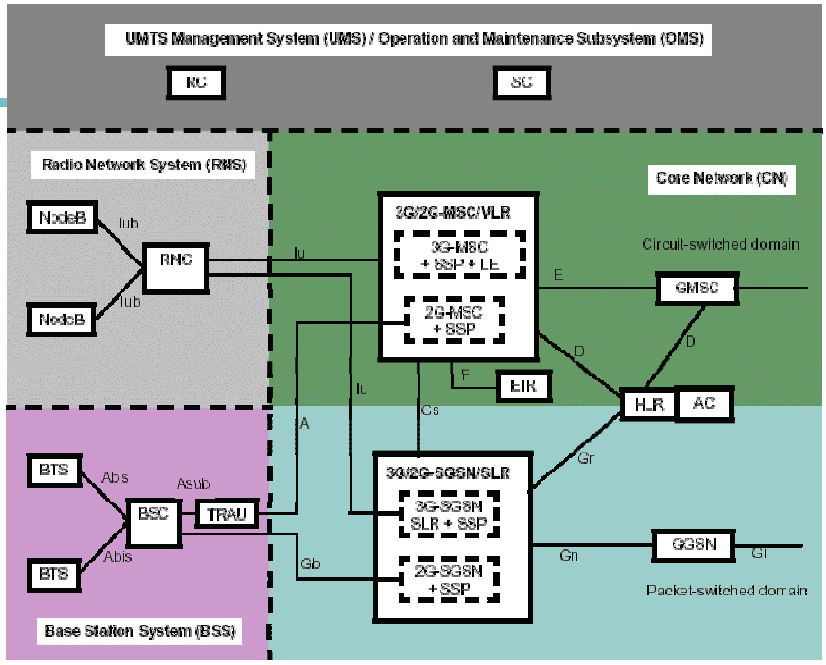
FH AACHEN UNIVERSITY OF APPLIED SCIENCES

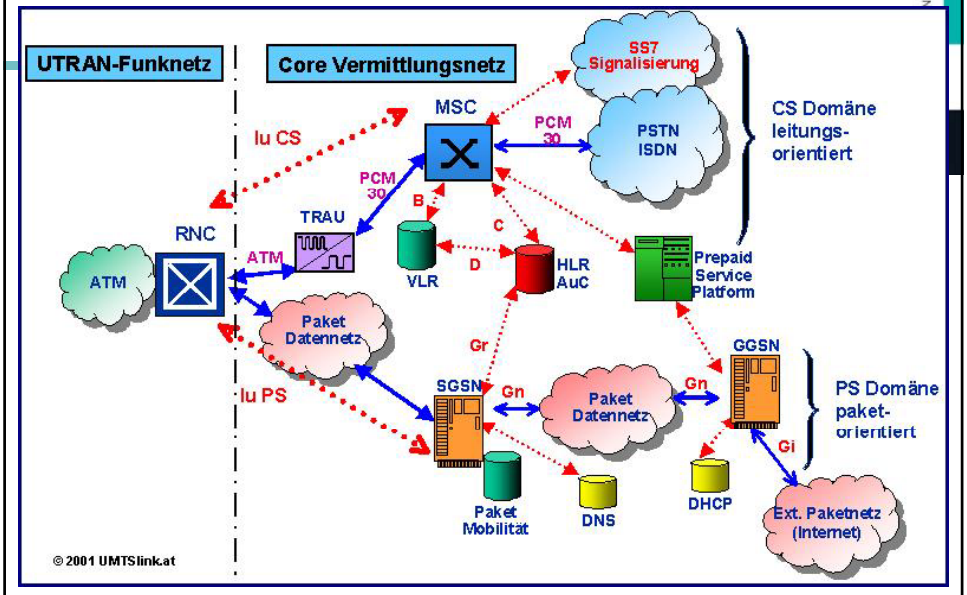
Überblick

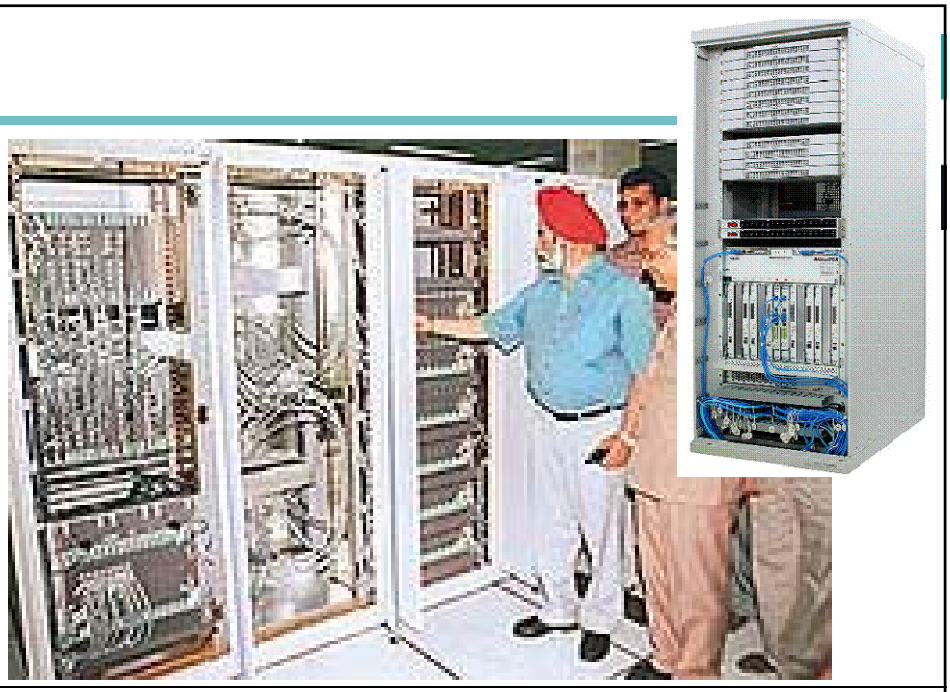
- Großprojekte bei Ericsson
- Agilität
  - Was ist das? Was bedeutet das insbesondere fürs Testen?
- Entwicklung des SW-Entwicklungsprozesses bei Ericsson
  - Am Anfang: Wasserfall
  - Dann: Moderne Paradigmen halten Einzug
  - Später: Integration wird zum bestimmenden Element
- Erfolg, Probleme und „Lessons Learned"

Netzwerke in der Telekommunikation

- Komplex
  - Telekommunikationsnetzwerke bestehen aus verschiedenen, komplexen Netzknoten unterschiedlicher Funktionalität
- Groß
  - Netzwerke bestehen aus mehreren tausend Komponenten
  - Erstrecken sich über Ländergrenzen hinweg
  - Verbinden Millionen Nutzer
  - Die Standards, die das Netzwerk spezifizieren, sind mehrere tausend Seiten groß
- Verschieden
  - Trotz Standards der Technologie sind die von den einzelnen Betreibern verwendeten Netzwerke sehr unterschiedlich in Aufbau, Größe, Verwendung

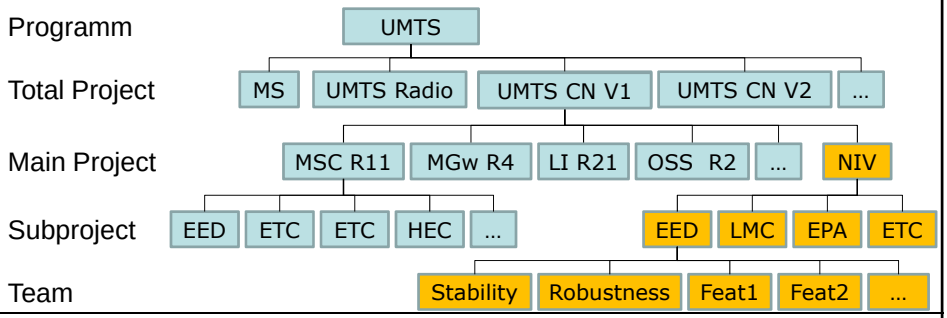



UTRAN-Funknetz
Core Vermittlungsnetz
SS7 Signalisierung
MSC
PCM 30
PSTN ISDN
CS Domäne leitungs-orientiert
Iu CS
PCM 30
B
C
TRAU
RNC
ATM
ATM
VLR
D
HLR AuC
Prepaid Service Platform
Paket Datennetz
Gr
GGSN
SGSN
Gn
Paket Datennetz
Gn
PS Domäne paket-orientiert
Iu PS
Gi
Paket Mobilität
DNS
DHCP
Ext. Paketnetz (Internet)
© 2001 UMTSlink.at



Großprojekte

- Große Produkte werden in großen Projekten entwickelt
  - Die einzelnen Knoten werden von eigenen (Main-)Projekten entwickelt
    - Main-Project zwischen 10 und mehreren 100 Programmieren groß
    - Main-Projects führen in der Regel einen Systemtest des eigenen Knotens durch
  - Teilprojekte sitzen an verschiedenen Standorten



Großprojekte

- Frage:
  - Kann man in solch großen Projekten auch agil programmieren?
  - Was genau bedeutet das?
  - Welche Methoden und Prinzipien der agilen Programmierung lassen sich für große Projekt skalieren
  - Welche Probleme treten dann auf? Wie kann man diesen Problemen begegnen?

  - Und ganz wichtig: Was bedeutet das fürs Testen?
    … (wir sind ja auf der TAV)

Agilität in der Software-Entwicklung

- Ziel
  - Softwareprozesse flexibler und schlanker
- Agiles Manifest
  - Individuen und Interaktion vs. Prozess und Tool
  - Funktionierende Programme vs. ausführliche Dokumentation
  - Stetige Zusammenarbeit mit Kunden vs. Verträge
  - Mut und Offenheit für Änderungen vs. fester Plan
- Agile Prinzipien und Methoden
  - Paarprogrammierung
  - einfach
  - ständige Refaktorisierung
  - Story Cards
  - ständig lauffähiges Produkt
  - testgetriebene Entwicklung
  - gemeinsamer Codebesitz
  - Entwicklung in Iterationen bzw. Zyklen
  - ständige Integration (Daily Build)
  - …
- Agile Prozesse → Extreme Programming, Scrum, …

Agilität in den Großprojekten bei Ericsson

- Es wurde nicht versucht, von den agilen Ansätzen zu lernen!
  - Agile Ansätze waren relativ unbekannt
  - Ericsson benutzt(e) …
    - … eigene Entwicklungsprozesse,
    - … eigene, selbstentwickelte Werkzeuge,
    - … (teilweise) eigene Programmiersprachen,
    - …
- Es fanden Analysen statt, …
  - … wo die SW-Entwicklung nicht optimal läuft
  - … wo Zeit und Ressourcen verschwendet wurden
  - … wo Probleme auftraten

Ausgangssituation

- 90‘er Jahre (GSM, 2G)
  - Fokus auf einzelnen Netzknoten – nicht auf dem gesamten Netzwerk
  - Klassisches Wasserfallmodell
    - Großes Prozessmodelle (je nach Knoten), mehrere 100 Seiten Dokumentation
  - CMM Fokus
    - Ericsson setzt CMM als Verbesserungsmodell ein
    - Verstärkt das Gewicht auf schwergewichtige, dokumentenorientierte Prozessen
  - Allerdings gibt es auch einige kleine Projekte, die leichtgewichtige Methoden ausprobieren konnten

  → Arbeitsweise ist etabliert, Produkte sind erfolgreich

Und dann …

- Ende 90‘er Jahre
  - Netze werden komplizierter
    - Kunden sind nicht länger bereit, Knoten selber zu integrieren → Netzwerktest (Netzwerkintegration, Netzwerktest)
  - Es kommt ein neue Generation von Mobilfunknetzen
    - GPRS (2.5G)
    - UMTS (3G, WCDMA)
- Für diese neuen Netzen müssen …
  - … alte Knoten weiterentwickelt werden (→ alte Prozesse)
  - … vollständig neue Knoten entwickelt werden
    - Da hier auf eine neue Technologie gesetzt wurde, waren auch neue Prozesse, Werkzeuge, etc. notwendig
    - Rational wird Werkzeuglieferant

Neue Herausforderung

- Ende 90‘er Jahre
  - GPRS-Projekt
    - Insgesamt 100 Entwickler an drei Standorten
    - Neues Produkt, neue Programmiersprache, neuer Prozess, neue Werkzeuge → Hohes Risiko
  - Ansatz
    - Projekt in mehrere Iterationen unterteilt
    - Daily Build zu ambitiös, deshalb Weekly Build
      - Build dauert länger als 24 Stunden
      - Kein sauber definierter Smoke Test
    - Modernes Versions- und Konfigurationsmanagement
      - Check-In / Check-Out
      - Branching und Merging
      - Kein privater Code, regelmäßiges einchecken, regelmäßiges integrieren

Probleme (1)

- Große Problem beim Versions- und Konfigurations-management
  - Keine „Public-Code-Kultur“ vorhanden
    - Continuous Integration, Public Code, … wird nicht verstanden
  - Konsequenz
    - Der Build-Prozess kommt ins Stocken, irgendwann steht das gesamte Projekt
    - Keine lauffähige Software vorhanden → kein Test möglich
- Kinderkrankheiten bei der Einführung von Iterationen
  - Iterationen werden vor allem über das Datum, weniger über den Inhalt definiert
    - Große Teile des Inhalts rutschen in die letzte Iteration → Es kommt hinten zum Big Bang, der eigentlich vermieden werden sollte
    - Bei den ersten Iterationen wird nur Software geliefert
      - Keine Installationsprozeduren („Die werden erst in Iteration 6 implementiert“)
      - Keine Dokumentation („Die schreiben wir nach der Implementierung“)

Probleme (2)

- Zweifelhafte Strategie: Erst alle Funktionen implementieren anschließend das Produkt stabil testen
  - Große Probleme bei der Integration
    - Die einzelnen Produkte laufen nicht
    - Eine Integration der Produkt ist nicht möglich
    - Ein Test des Netzwerks noch weniger
  - Ernsthafte Stabilitätstests können auf Netzwerkebene nicht durchgeführt werden.
- Un-Testbare Produkte – Nicht nur wegen mangelnder Stabilität
  - Kein „Design for Testability"
    - Systeme sind schwierig zu installieren und zu konfigurieren
    - Systeme haben keine Befehle, um einfach den Systemzustand abzufragen
    - Keine Versionskontrolle auf dem System → Eine Zuordnung der Fehler auf Versionen ist nicht direkt über das System möglich.

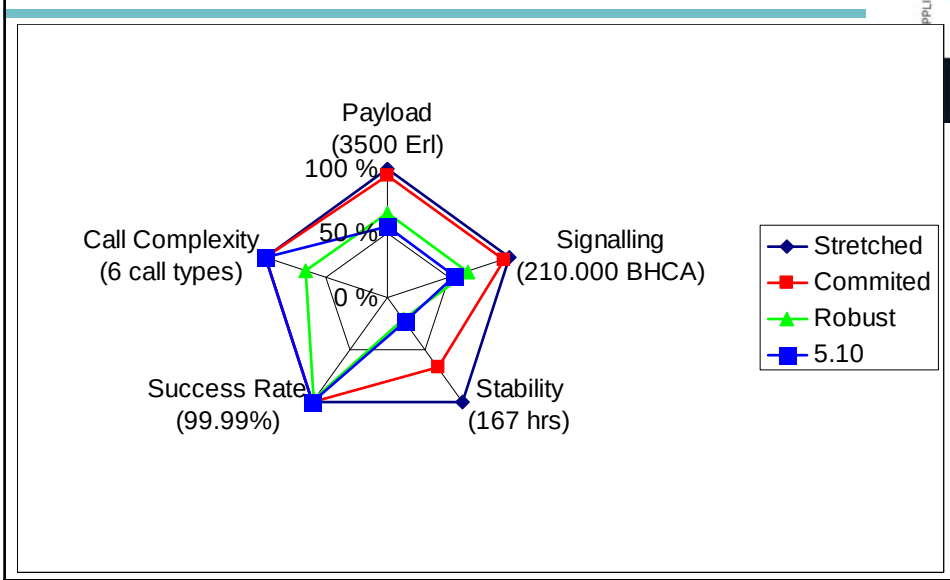
Lessons Learned

- Die grundsätzlichen Ansätze (Daily Build bzw. Weekly Build und Entwicklung in Iterationen) werden trotz der Probleme im ersten Projekt akzeptiert
  - → Kompetenz im Bereich Versions- und Konfigurationsmanagement wird aufgebaut
  - → Der Build-Prozess wird vereinfacht, besser kommuniziert, stärker überwacht
- Ein schwacher Test (eine schwache Testorganisation!!!) am Ende der Projekts steht auf verlorenen Posten
  - → Mit Hilfe von Metriken wird versucht Überzeugungsarbeit zu leisten
    - → Zeit, die verging, bis das Produkt stabil wurde
    - → Zeit zum Testen im Verhältnis zur Zeit zur Installation und Konfiguration
    - → Zeit, die verschwendet wurde, da keine lauffähig (testfähiges) Produkt vorliegt

Neues Spiel, neues Glück …

- Beim nächsten Projekt (Version2) wurden folgende Änderungen seitens des Tests durchgeführt
  - Der Test legt Qualitätskriterien fest, die erfüllt werden müssen, damit eine Version des Produkts angenommen wird
    - Software läuft problemlos im „Idle-Mode"
    - Software lässt sich problemlos installieren
    - Bekannte Fehler sind dokumentiert
    - Notwendige Dokumentation ist vorhanden
    - Testberichte werden vorgelegt (weniger aus Misstrauen, vielmehr um davon zu lernen)
  - Es wird ein „Pre-Test" durchgeführt
    - Erst wenn dieser Pre-Test erfolgreich absolviert wird, wird die Lieferung akzeptiert und die neue Version der SW im Test benutzt
  - Stabilität wird von Anfang getestet bzw. gemessen

Stabilität



Lessons Learned

- Im Folgeprojekt klappte vieles besser
  - Iterationen waren stabiler
    - Projektperspektive: Inhalte und Termine deutlich besser eingehalten
    - Qualitätsperspektive: SW läuft
  - Konfigurations- und Versionsmanagement nicht nur verstanden sondern gelebt
    - Continuous Integration, Public Code, …
- Natürlich noch nicht alles in Ordnung aber …

Die nächste Herausforderung …

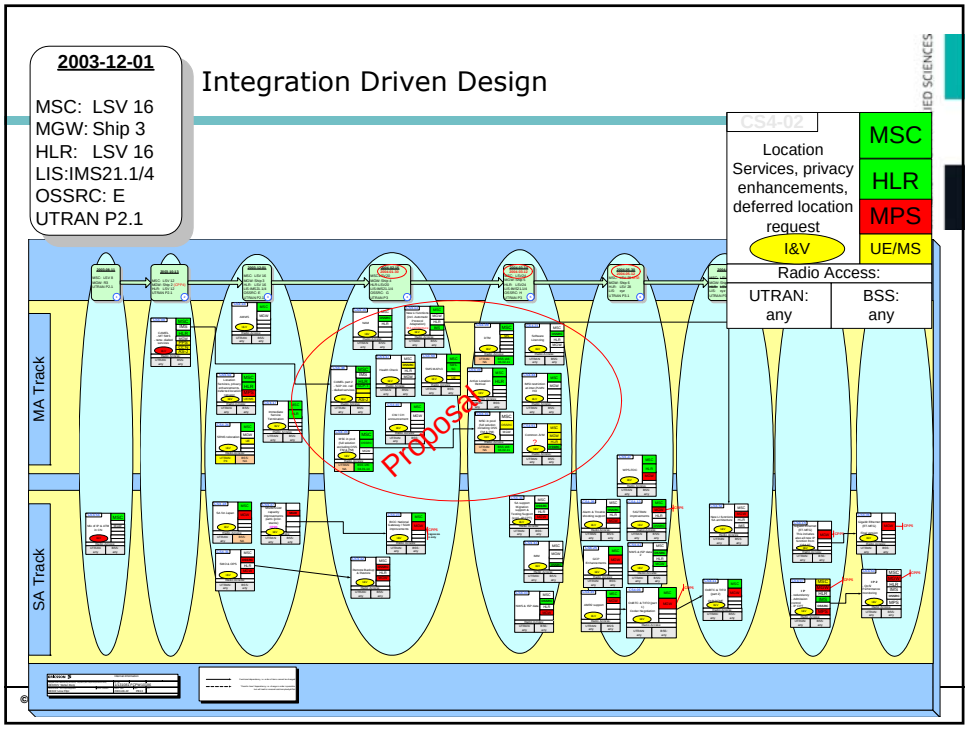
- UMTS 2.0 Projekt → Erstes kommerzielles UMTS
- Deutlich größeres Netzwerk (alte GSM Funktionalität)
  - Regressions Tests
  - Komplexere Funktionen → Mehr Knoten beteiligt (z.B. Prepaid)

  → Die Planung der Iterationen wird schwieriger
- Neuer Ansatz → Integration Driven Design
  - Bisher: Die Entwicklungsorganisation bestimmt, welche Iteration welche Funktionen enthält
  - Jetzt: Das Testprojekt – zuständig für Integration – bestimmt in welcher Iteration, was geliefert wird
  - Ziel: In allen Iterationen gibt es neue Funktionen, keine Big Bang Integration am Ende des Projekts

Integration Driven Design

- Zentrales Planungs- und Kommunikationsinstrument ist die Software-Anatomy (nächste Seite)
  - Stellt die einzelnen Iterationen dar
  - Legt fest, welche Features wann geliefert werden
  - Führt daher zu einem Fokus der einzelnen Features im Test
  - Legt fest, welcher Knoten etwas liefern muss
  - Dient als Ampel, wo kritische Lieferungen vorliegen

- Kommunikation
  - Die Software-Anatomy wurde einmal wöchentlich mit allen Projektleitern besprochen (Status, Verschiebungen, Eskalationen)
  - Die Software-Anatomy diente als Ausgangspunkt um, …
    - … Qualitätskriterien für die Lieferungen festzulegen
    - … Wünsche von den liefernden Projekten bzgl. Testfokus zu spezfizieren



Integration Driven Design

- Integration Driven Design bedeutet
  - Die Gliederung eines Projekts in mehrere Iterationen
  - Absprache, in welcher Iteration wer was liefern muss
  - Spezifikation von Qualitätskriterien bzgl. der einzelnen Lieferungen
  - Klärung der formalen Kommunikation zwischen Test und Entwicklung
    - Fehlermanagement
    - Korrekturzeiten
    - Patchhandling
  - Fokus auf Stabilität (im Netzwerk) von Beginn des Projektes an
  - Absprache einer gemeinsamen Teststrategie mit den einzelnen Projekten, die die Knoten entwickeln
- Last but not least: Integration Driven Design bedeutet, dass Test keine nachgelagerte Funktion ist, sondern DIE zentrale Instanz im Projekt!!!

Integration Driven Design

- Integration Driven Design erfordert …
  - … ein starkes Testteam
  - … viel Kommunikation mit den Entwicklungsprojekten
  - … Rückendeckung vom Senior Management
  -
- Mit Integration Driven Design waren wir in der Lage, das Gesamtprojekt (UMTS 2.0, Core Network) zu stemmen
  - In guter Qualität (= Telekom-Stabilität)
  - Mit wenig Verspätung
  - Zu den geplanten Kosten

Agilität und Testen in Großprojekten

Agiles Prinzip / Methode	Integration Driven Design
Pair Programming	Nein
Testgetriebene Entwicklung	Jain, vorgegebene Qualitätsstandards, Entwicklungsreihenfolge wird durch Test festgelegt
Featuredriven Development	Ja, Test und Integration basieren auf Features, die Integration wird auf der Basis von Features organisiert
Gemeinsamer Codebesitz	Nein, nur innerhalb der Entwicklungsprojekte
Ständige Refaktorierung	Nein, nicht auf Netzwerkebene
Ständige Integration (Daily Build)	Ja, allerdings sind die Zyklen deutlich länger. Der Ansatz lässt sich hervorragend kombinieren mit Daily Build im Entwicklungsprojekt
Entwicklung in Zyklen	Ja, Zyklen werden vom Test bestimmt.
Ständig lauffähiges Produkt	Ja, Stabilität hat zentrale Bedeutung auch auf Netzwerkebene
Story Cards	Nein, kein Kundenprojekt

Fazit

- Einige wesentliche Punkte, die mit Agiler Entwicklung assoziiert werden, spielen in diesem Ansatz keine Rolle
  - Kundenbeteiligung
  - Wenig Prozesse und Tools
- Anderer Merkmale lassen sich wenn auch anders skaliert auf den Test in Großprojekten übertragen
  - Iterationen
  - Daily Build → Monthly Integrations
  - Testgetriebene Entwicklung
  - Featuredriven Development

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Vielen Dank für die Aufmerksamkeit!

Gibt‘s noch Fragen?