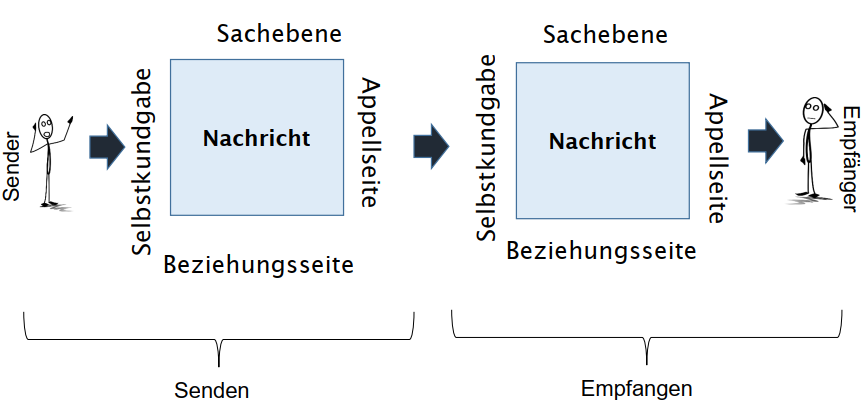
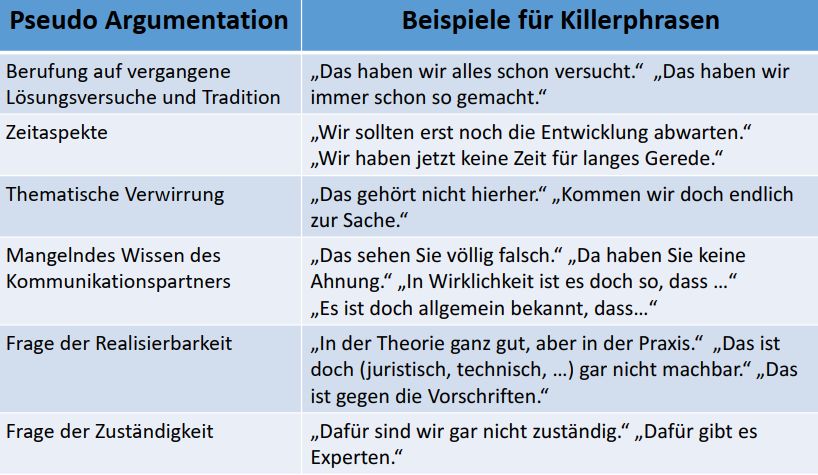
**Zusammenfassung MST**

**Kapitel 2: Kommunikation**

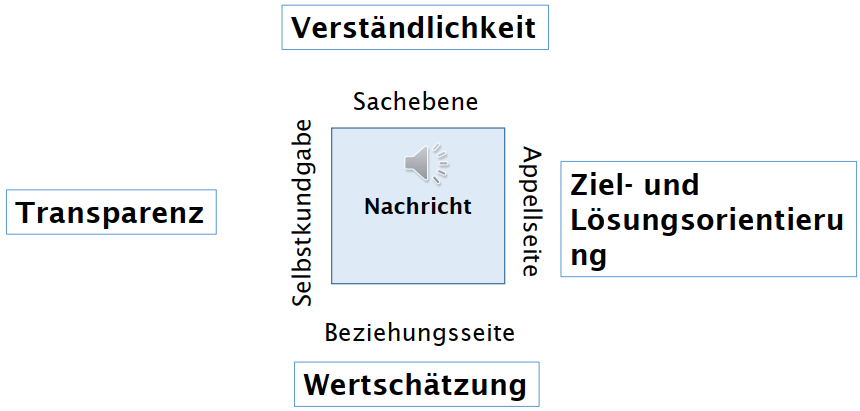
* Kommunikationsmodell Friedemann Schulz von Thun  
  🡪Vier Seiten einer Nachricht; auf jeder Seite kann es zu Missverständnissen/Konflikten kommen

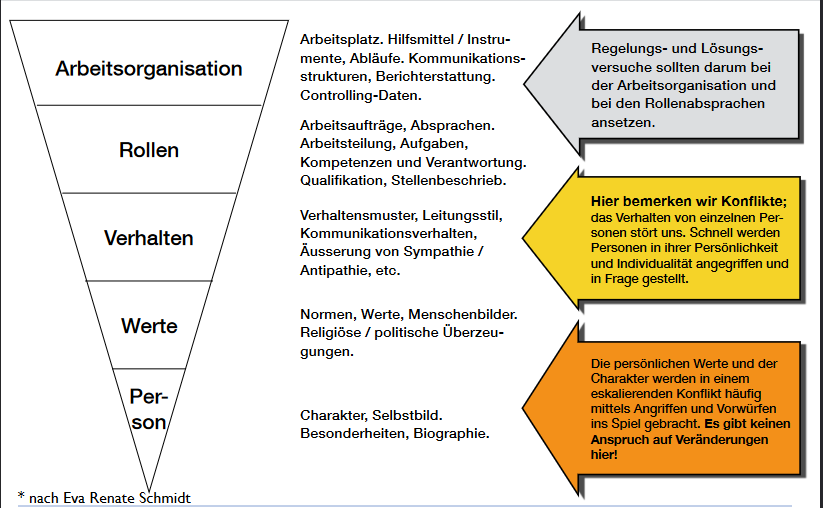
🡪4 Aspekte: Auf die Sache bezogen (Sachinhalt); auf den Sprecher bezogen (Selbstoffenbarung); auf die Beziehung bezogen (Beziehung); auf die beabsichtigte Wirkung bezogen (Appell)

* Weitere Aspekte Kommunikation:
* Explizite und implizite Botschaften (implizit oft auf nonverbalem Weg; Sarkasmus; Mimik/Gestik)
* Nonverbale Kommunikation (Mimik/Gestik/Emoticons für Text)
* Kongruente und inkongruente Nachrichten („mir geht’s gut“ während man weint; viel Schaden vor allem bei Kindern oder Tieren anrichtbar damit)
* Man kann nicht nicht kommunizieren (nichts sagen ist auch eine Kommunikation; auch nonverbal!)
* Beispiele für Kommunikationsprobleme:
* Einseitige Empfangsgewohnheit auf nur einer Seite des Modells (Taubheit auf Appell-Ohr)
* Winning an argument by TKO: komplizierte Wörter benutzen, bis das gegenüber aufgibt; Bullshit-Bingo
* Killerphrasen:

Unterbinden von Diskussion; Ziel ist nur mundtot zu machen; Gegenmittel: nachfragen! Was haben Sie denn schon versucht?

* Kennzeichen gelungener Kommunikation:



* Schichtenmodell zur Bearbeitung von Konflikten:  
  🡪Auf welcher Ebene steht der Konflikt?

**Kapitel 3.1: Konfiguration Management allgemein**

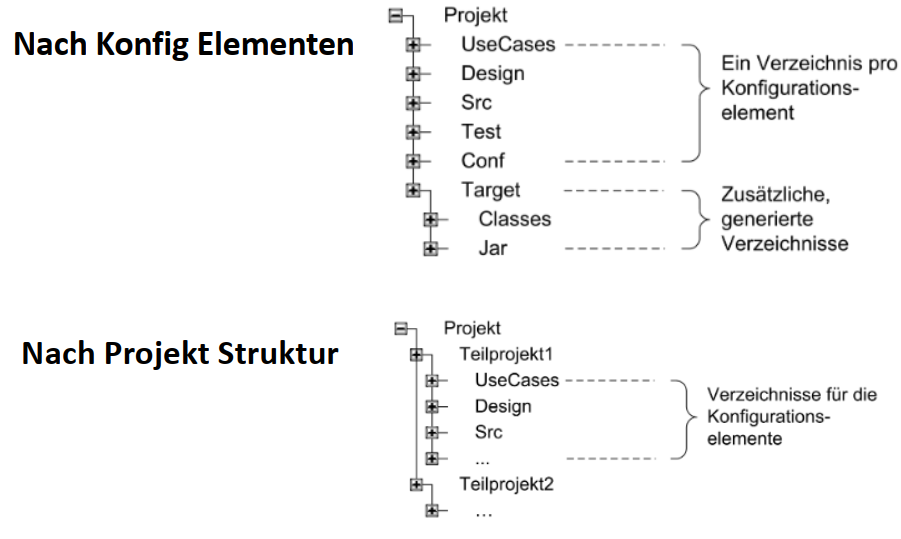
* Konfig Management: Projektergebnisse verwalten und den Teammitgliedern kontrolliert Zugriff gewähren
* Ziele Konfigurationsmanagement:
* Änderungen kontrollieren (Versionsmanagement, Rechtevergabe)
* Kommunikation vereinfachen/Transparenz verbessern (Change Management)
* Qualität sicherstellen (Testautomatisierung, Versionsmanagement)
* Produktivität steigern (Entwickler können sich auf Aufgabe konzentrieren)

Aufgaben des Kernprozesses:

* **Definition Konfig Element:** Alle Elemente, die zur Erstellung des Produkts nötig sind; Zeug zur Steuerung des Projekts wie Projektpläne, Kommunikationspläne, Protokolle gehören nicht dazu
* Eindeutige Konfigurationselemente: Quelltext; Anforderungsdokumente (Use Cases); Architektur und Design Dokumente; Testspezifikationen und Testdaten; Installationsanleitungen, Release Notes
* Mögliche Konfig Elemente: Werkzeuge (Compiler, IDE, Tools); Bibliotheken; Frameworks

🡪je nach Auslegung und Projekt können sie dazugehören; wenn z.B eine spezifische Version einer Bibliothek oder IDE benötigt wird, kann diese mit dazugepackt werden und auch zur Versionsverwaltung gehören; kann redundant sein, aber den Prozess erleichtern

* Keine Konfig Elemente: Meetingprotokolle, Projektpläne, generierte Dateien (kompilierte Quelltexte)
* **KM-Handbuch erstellen**; gibt Knigge vor für Dokumentationen, Code-Stil, Fehlermeldungen, Dokumentennamen (gute Darstellung von Beziehungen damit möglich), etc.



* **Festlegen der Projektstruktur**: Projektstruktur nach Konfig Elementen, typischer Projektstruktur oder nach Software Architektur
* **Verwalten der Konfig Elemente**: Verfügbarkeit der Dateien sicherstellen; Integrität gewährleisten; nur berechtigte Zugriffe; Nachvollziehbarkeit von Änderungen 🡺 **Benutzen von Repositories**
  + Lokale vs. globale Versionierung: bei lokal kriegt eine Datei einzeln einen neuen Versionsstand; bei global wird alles bei jedem checkin hochgezählt; svn und git sind global; cvs hatte lokal
  + Zwei Ansätze bei parallelen Änderungen:
    - Lock-Modify-Unlock (Reserved Checkout)
      * + Vermeidung von Konflikten, auch für binäre Dateien gangbar
    - Copy-Modify-Merge (Unreserved Checkout) (bei git so; bei svn pro Konfig element einzeln steuerbar)
      * + niemand wird blockiert
  + Tags bieten für Menschen merkbare Versionsnummern; Baselines sind bedeutsame Ereignisse, technisch aber identisch mit Tags; Release ist Auslieferung der Software; für jedes Release wird eine baseline erstellt, aber nicht umgekehrt
  + Branching and merging
* **Projektautomatisierung**: Build prozess automatisch; so viel wie möglich automatisieren; Varianten: Entwicklerbuild (lokal), Integrationsbuild (regelmäßiger build auf Integrationsumgebung), Releasebuild (Integrationsbuild + Setzen von Release Tag); Umsetzung über Shell Skripte, Ant, Maven, Gradle, etc.
* **Änderungs- und Fehlermanagement**: Änderungswunsch (Change Request) wird von Änderungsmanager geprüft, bewertet, dann vom Change Control Board (Gremium) überprüft und an das entsprechende Team geleitet zur Umsetzung + Überprüfung;
* **Aufgabe**: Entwickeln Sie ein sinnvolles Statusmodell für einen Changerequest und stellen Sie es in Form eines UML-Zustandsdiagramms dar

**Kapitel 3.2: VCS\_SVN**

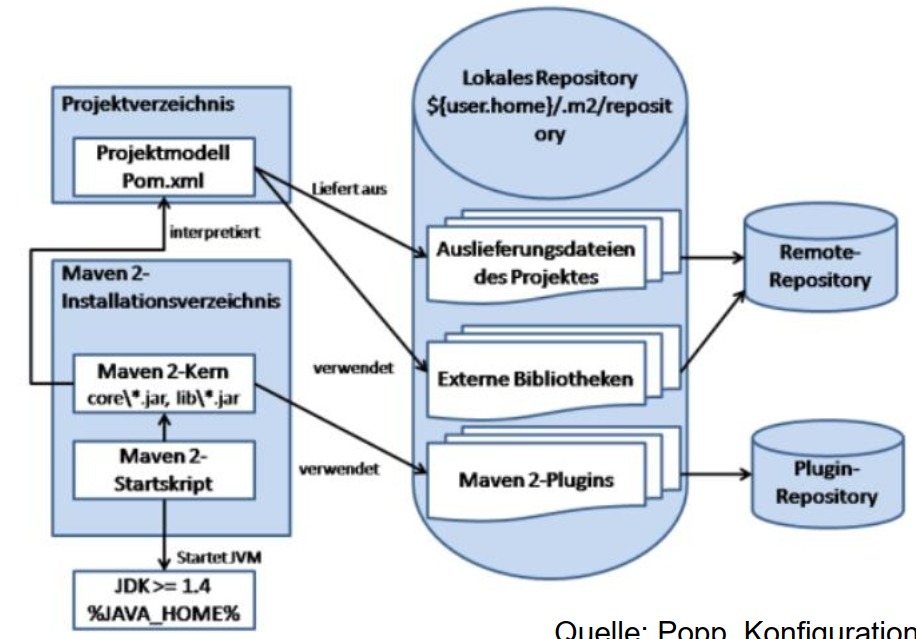
* SVN: sehr etabliert, kostenfrei, einfach, open source, gut für CI nutzbar, komplexe Szenarien abbildbar
  + Atomare Check-Ins; Versionierung von Dateien + Verzeichnissen und Bearbeitungsvorgängen; Rechtesystem vorhanden;
* Entscheiden zwischen zentralem oder projektspezifischem Repository (Empfehlung: Ein Repository pro Projekt)
* Arbeitsweise Repository:
  + Revisionen: für jede Transaktion eine Reversionsnummer (globale Versionierung)
  + Changeset: Alle im Rahmen der Transaktion getätigten Änderungen (Changeset kann Revision zugeordnet werden und umgekehrt)
* Standardmäßig ist Passwort in Klartext gespeichert
* In die Projektstruktur fließt mit ein: Konfig Elemente, gewünschte logische Struktur, tagging/branching/release Konzept
* Svn kennt keine echten tags und branches, beides wird als normales Verzeichnis abgebildet 🡪 Repository Struktur + Rechtevergabe um Tags nicht zu verändern und branches richtig abzubilden
* SVN kann durch Kommunikation mit dem Repo feststellen, in welchen Zustand sich eine Arbeitsdatei befindet (Unverändert+aktuell; lokal geändert+aktuell; unverändert+veraltet; lokal geändert+veraltet)
* Arbeitsprozess:
  + Arbeitsbereich aktualisieren
  + Prüfung auf Probleme
  + Änderungen durchführen
  + Auf Änderungen im Repository prüfen
  + Änderungen ins Repository schreiben
* Zusammenfassung: **Zentrales** Version Control System; sehr etabliert, sicher; Für CI sehr gut geeignet

**Kapitel 3.3: Git**

* SVN hat EINEN Server und EIN Repository
* Kein dedicated Server bei Git, Repository auf jedem Client/Server
* Dezentrale Versionsverwaltung:
  + Vorteile: hohe performance, effiziente Arbeitsweisen, offline Fähigkeiten, Flexibilität der Entwicklungsprozesse, Backup, Wartbarkeit
* Git verwaltet Versionen von Software; jede Version wird als Commit in einem Repo abgelegt; Commit umfasst immer das ganze Projekt; Mit jedem Commit wird für jede einzelne Datei im Projekt eine Kopie im Repository angelegt
* Nicht auf Remote Tracking Branches arbeiten
* …

**Kapitel 3.4: Maven**

* Maven is essentially a project management and comprehension tool and as such provides a way to help with managing: Builds, Documentation, Reporting, Dependencies, SCMs, Releases, Distribution; POM = Project Object Model
* Idee: Modellbasierter, deklarativer Ansatz zur Buildautomatisierung
* Ant: Toolbox 🡸 🡺Maven: Anwendung von Patterns
* Historie; Jedes Projekt besitzt eigene Ablagestruktur, jede Bib wird einzeln eingebunden & in jedem Projekt wird der build Prozess neu definiert
* Stattdessen in Maven: Unterstützung vernünftiger default Werte. Z.B. für Ablage der Sourcen, der Tests, der Kompilate, etc.
* Mit Maven; Einigung auf ein einheitliches Interface um Projekte zu bauen (statt eigenes build system für jedes Projekt)
  + Maven beantwortet, was nötig ist um das Projekt zu bauen, welche Libraries man downloaden muss und wo die Libraries abgelegt werden
* Build Prozess ist deklariert und wird druch Plugins implementiert; 🡪Änderungen in den Plugins ändern nicht das Build System
* Definition eines konzeptionellen Modells ermöglicht: Dependency Management; Remote Repositories; Wiederverwendung der Build Logik; Tool Integration; Einfache Suche nach Artefakten
* **Funktionsumfang Maven**:
  + Durchführung des Build Prozesses
  + Verwaltung der Abhängigkeiten von externen Bibliotheken
  + Erstellen der Projektdokumentation
* **Architektur!**

****

* Installation simpel und schnell; geringe Größe, da alles bei Bedarf dazu geladen wird