Überführung von Altprojekten und „legacy“-Code zu modernen und zeitgemäßen Standards der Softwareentwicklung

ROHDE&SCHWARZ

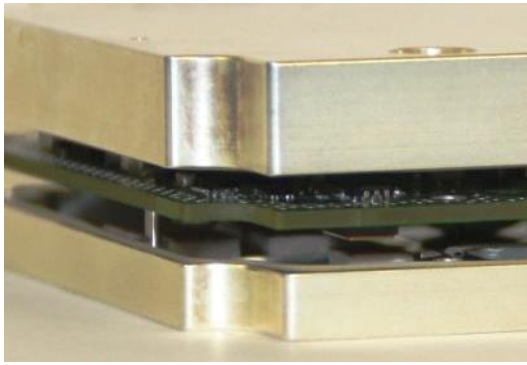
Wo arbeite ich; wenige Worte zur Firma…

Rohde & Schwarz SIT GmbH
Überblick

- Gegründet 1991
- 130 Mitarbeiter (Mathematiker, Informatiker und Ingenieure)
- Standorte in Berlin und Stuttgart
- 100%ige Rohde & Schwarz-Tochter
- VS-Produktion in den Werken von Rohde & Schwarz
- Weltweiter Service, Support und Vertrieb durch die Rohde & Schwarz-Gruppe

Geschäftsfelder der Rohde & Schwarz SIT

- Ende-zu-Ende-Absicherung
  - Schutz der Sprach- und Datenkommunikation von Endgerät zu Endgerät
  - Gewährleisten der Endgeräte-Sicherheit
- Netzwerksicherheit
  - Schutz der Kommunikation über öffentliche Netzwerke (z.B. Internet)
  - Absichern des Internetzugangs vor Einbrüchen ins Netzwerk und Datendiebstahl
- Kryptomodule
  - Universelle Hardware Sicherheits-Module
  - Entwickeln von Embedded-Kryptomodulen



Ende-zu-Ende-Absicherung
Produktüberblick

TopSec Mobile
VoIP-Verschlüsselung für Smartphones via GSM/UMTS/LTE, (W)LAN

ELCRODAT 6-2
ISDN, IP-Netze, Satellitenkommunikation
Behörden, Bw, EU, NATO

R&S®MKS 9680
ISDN, PSTN, IP-Netze, Satellitenkommunikation, weltweiter Einsatz

ELCRODAT 5-4
ISDN, PSTN, Satellitenkommunikation, Bw, NATO

ELCRODAT 4-2
HF, VHF/UHF, Festverbindungen, Bw, NATO

R&S®MMC3000
HF, VHF/UHF, Festverbindungen, Out-of-Area, non-NATO

Netzwerksicherheit
Produktüberblick

R&S®SITLine ETH100
Ethernet-Verschlüsseler mit 1x, 2x, 4x 100Mbit/s, Punkt-zu-Punkt, Multipunkt

R&S®SITLine ETH1G
Ethernet-Verschlüsseler mit 1Gbit/s, Punkt-zu-Punkt, Multipunkt

R&S®SITLine ETH50
Ethernet-Verschlüsseler mit 25, 50, 100Mbit/s, Punkt-zu-Punkt, Multipunkt, erweiterter Temperaturbereich -20/+70 C

R&S®SITGate S Serie
Next-generation Firewall, Applikationskontrolle, Antivirus, IDS/IPS und Webfilter
für bis zu 100 Benutzer

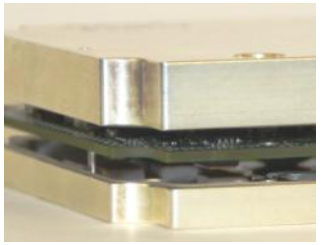
R&S®SITGate M Serie
Next-generation Firewall, Applikationskontrolle, Antivirus, IDS/IPS und Webfilter
für 500 bis 1.500 Benutzer

R&S®SITGate L Serie
Next-generation Firewall, Applikationskontrolle, Antivirus, IDS/IPS und Webfilter
für 2.500 bis 10.000 Benutzer

Kryptomodule
Produktüberblick

R&S®CryptoServer
Hochleistungs-HSM mit AES, ECC und RSA, CC EAL4+, Tamperschutz, ZKA und NIST zertifiziert

Deutschland-HSM
PC-Karten - HSM mit BSI-Zulassung für Reisepass und neuen Personalausweis



Embedded Kryptomodule
mit Standard- oder kundenspezifischen Algorithmen und software-definierter Kryptografie

Forschung & Entwicklung, Ausbildung

- Bundesweite Forschungsprojekte
- Studien im Bereich Sicherheitstechnik
- Bachelor & Masterarbeiten, Praktika
  - Studenten sind keine Code-Monkeys!
  - Einbindung in Projektteams
  - SmartCards, Bluetooth, Mobilfunk, HSM, Raspberry Pi Prototypen-Entw.
  - Gern auch mit Übernahme ☺
- Deutschlandweite Kooperation mit Hochschulen
  - Vorlesung „IT-Sicherheit in der Nachrichtentechnik“ oder Fachvorträge
- IHK-Ausbildung „Fachinformatiker für Anwendungsentwicklung“
  - Sollen u.a. das lernen was in diesem Vortrag aufgezeigt wird…

1. Arbeitstitel dieses Vortrags war: „von steinzeitlicher zu zeitgemäßer Softwareentwicklung“

Warum dieser Vortrag?

- „wie fühlt man sich als idealistischer Uni-Absolvent in der Informatik-Branche“
- „Dos & Don‘ts“
- Reality-Check?
- Was erwartet Euch vielleicht…
- Was erwartet Euch hoffentlich nicht…
- Was gibt es für Euch zu tun, wenn Ihr eine „Steinzeit“ vorfindet?
  - Engagiert Euch!

Disclaimer

- Keine böswillige Kritik
- Keine Werbung für genau ein Produkt/Tool
- Zusammengefasste Erfahrung aus verschiedensten Projekten, Kontexten und Firmen/Partnern
- Alle Produktnamen ausgedacht, Korrelationen rein zufällig ;-)

Agenda

- Versionsverwaltung
- Testgetriebene und testautomatisierte Softwareentwicklung
- Software(code)qualität
- Continuous Integration, Continuous Build
- openSource im professionellen Einsatz
- Plattformübergreifende Softwareentwicklung
- Agile Softwareentwicklung
- Ticketsysteme / Requirements-Engineering

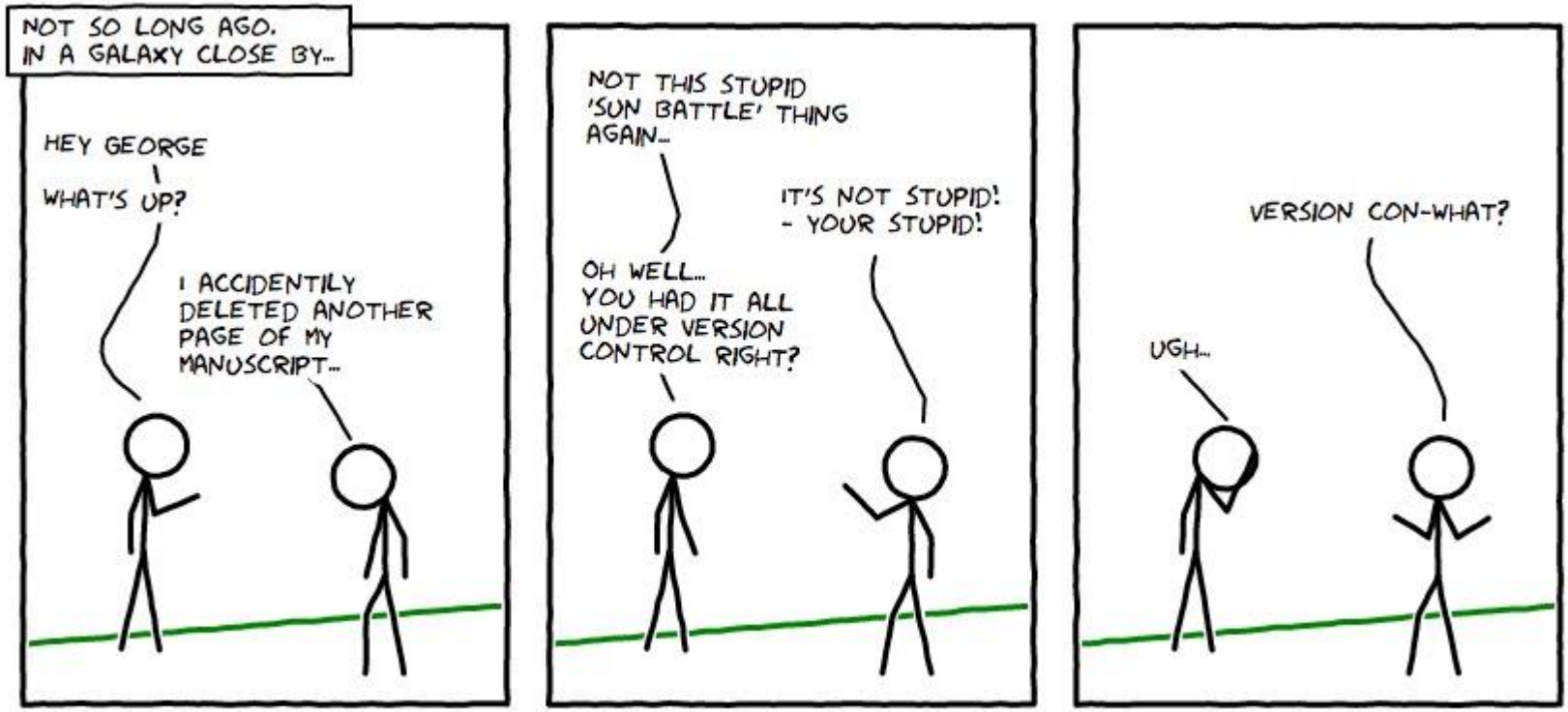
- Bonus: Angewandte Kryptografie für Behörden & Co.
- Bonus: Einsatz von DSLs in der „Industrie“ und plattformunabhängige Softwareentwicklung

- Einladung zur Diskussion!

Versionsverwaltung

```
D:\versioncontrolled\secret_project_without_name>dir

11.06.2013  13:41    <DIR>          _oldFiles
13.04.2014  17:23    <DIR>          _oldFiles2
22.02.2013  13:41    <DIR>          doc
05.10.2013  02:56           107.520 martins_requirements_mail.txt
30.09.2014  07:37    <DIR>          privat
24.08.2013  15:16           107.520 Requiremnts_20130824.txt
03.10.2013  13:25           142.924 Requiremnts_20131003_new.txt
11.06.2013  08:39           183.392 Requiremnts_20131106_merged.txt
13.04.2014  18:27           343.372 Requiremnts_20140413_v2.txt
19.04.2014  12:19           548.493 Requiremnts_20140413_v2_reviewed.txt
27.04.2014  17:36           752.832 Requiremnts_20140413_v2_reviewed_final.txt
30.04.2014  20:15           792.287 Requiremnts_20140413_v2_reviewed_final2.txt
18.02.2002  07:38               113 Thumbs.db
30.09.2014  23:42    <DIR>          src
               8 Datei(en),  1.987.400.090 Bytes
               5 Verzeichnis(se), 21.033.357.312 Bytes frei

D:\versioncontrolled\secret_project_without_name>
```

Versionsverwaltung



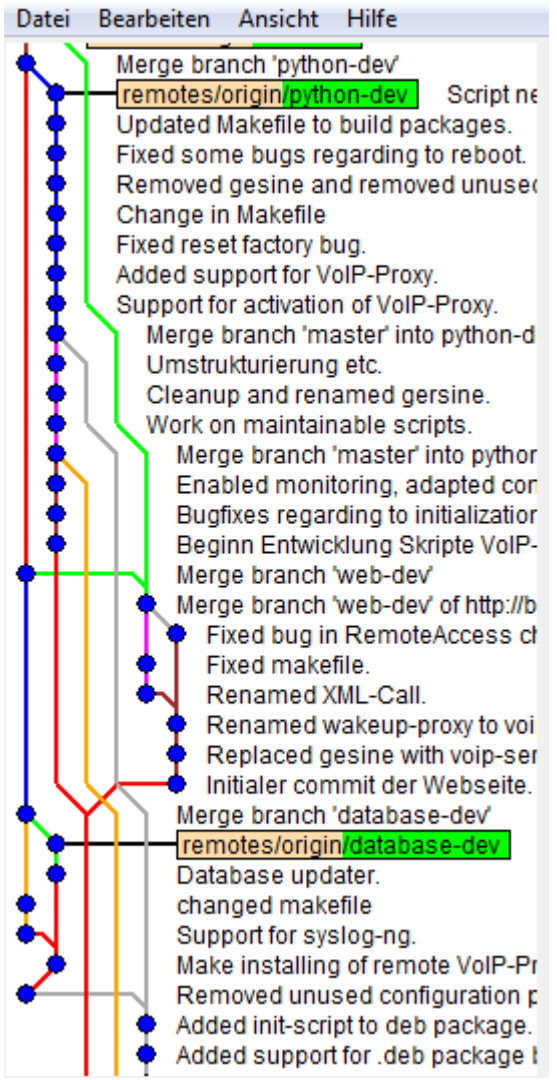
Quelle (CC BY-SA 3.0): http://smutch.github.io/VersionControlTutorial/_images/vc-xkcd.jpg

Versionsverwaltung à la 1992 und wie man es überführt

- Was habe ich vorgefunden?
- IBM ClearCase
  - Max. 8 repositories
  - Versionen werden pro Datei gespeichert
  - Keine gemeinsamen Commits
    - Versuch einen Build nachzubauen ggf. schwierig…
  - „tool“ frisst sich ins Betriebssystem ein

- Meine Wahl: git
  - Dezentral
  - Leichtgewichtig
  - Staging, Commits, vgl. Linux-Kernel-Entwicklung
  - Weit verbreitet – auch bei neuen Mitarbeitern

- Überführung von Clear Case zu git: cc2svn, svn2git



Software(code)qualität

- Erfahrung:
  - 100000+ Zeilen monolithischer Code sind schwer (und somit teurer) zu warten
  - 1.500+ Zeilen Switch/Case-Konstrukt mit min. 5 Ebenen (IF, weitere S/C)
  - Refaktorisierung bitter notwendig

- Empfehlung 1: Clean Code Developer
- Empfehlung 2: Vorlesung „Design Patterns & Frameworks“
  - Auch für Embedded-Entwickler brauchbar!

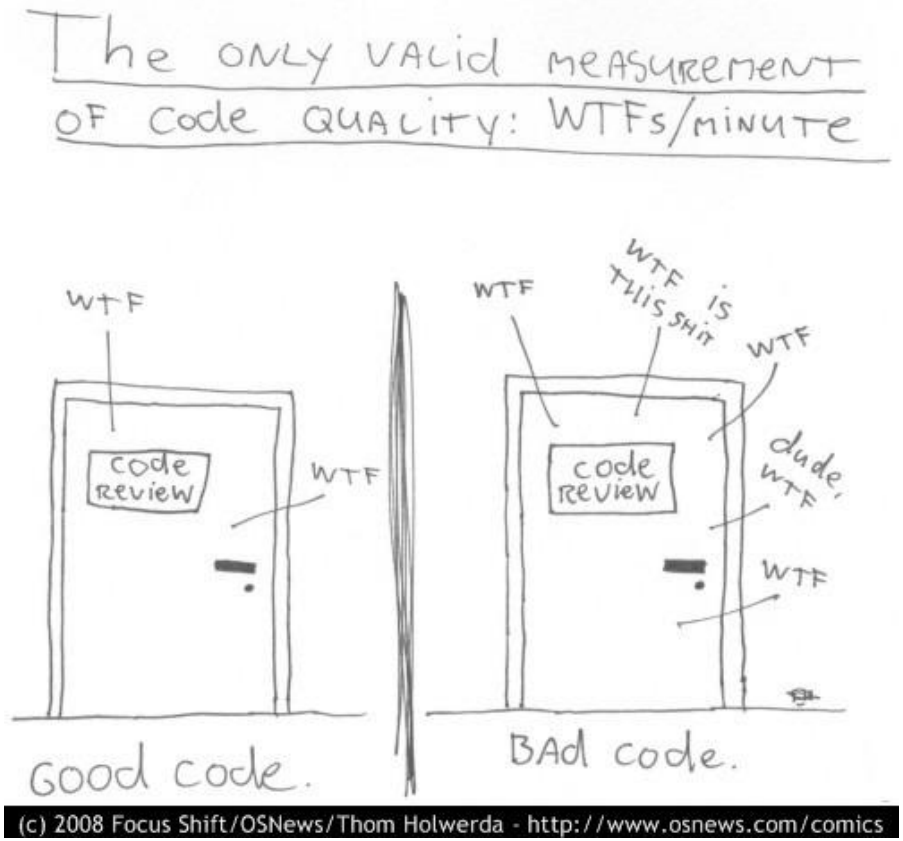
- Informatiker sind nicht „nur“ Programmierer sondern Software-Architekten
  - Projekte werden immer komplexer
  - Immer mehr Interaktionen zwischen Bestandsprojekten

Software(code)qualität

- Statische Code Analyse (bspw. Sonar oder WTF/s)
  - Wie viele bekannte Vergehen können frühzeitig erkannt/beseitigt werden?

- Ein Team/Projekt kann sich Ziele bzgl. Fatal/Major/Minor setzen
  - „bei mehr als k% Major werden zunächst Korrekturen durchgeführt bevor weitere Features hinzugefügt werden“

- Metrik unterstützt auch das Überzeugen von Vorgesetzten bzgl. Notwendigkeit von Refactoring

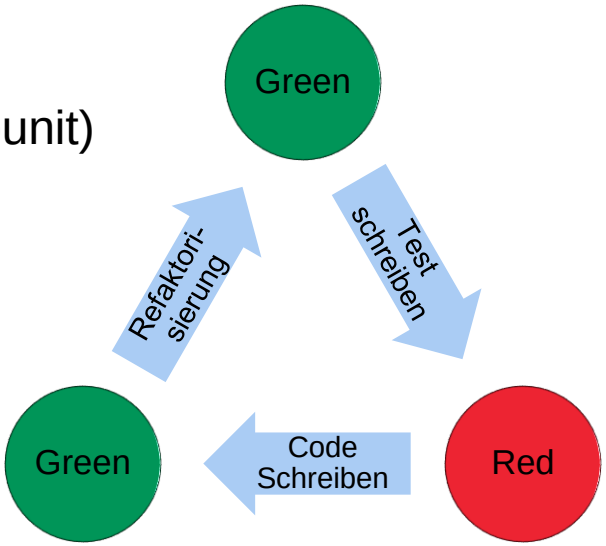


Testgetriebene und testautomatisierte Softwareentwicklung

- Aussage: „Für ein Release brauchen wir 4 Mannwochen für die Tests“
  - ☹
  - Es dauert nicht nur Zeit – es bindet auch Entwickler!
    - Keine weiteren/neuen Features im gleichen Zeitraum

- Erinnerung an SWT 3. Semester → JUnit & Co.
  - Es lohnt sich!
  - Unit-Tests auch für andere Sprachen (googletest, cppunit)

- GUI-Automatisierung spart wertvolle Zeit
  - SWTBot
  - Selenium
  - u.v.m.
- und erzeugt ggf. Screenshots für automatisierte Handbuch-Erstellung



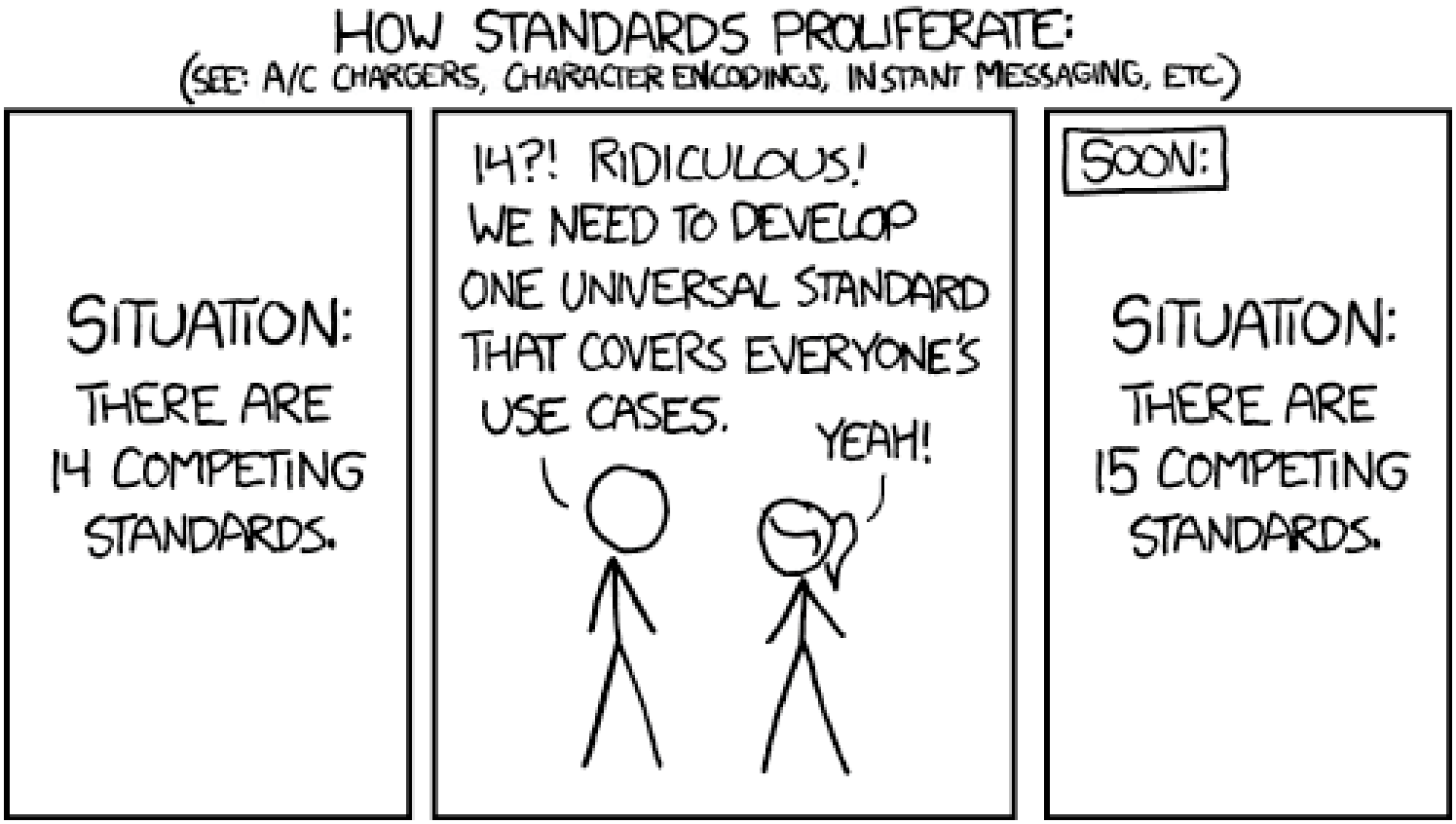
Continuous Integration, Continuous Build

- Aussage: „Die Release baut bei uns immer $name“
- Aussage: „wir nehmen einfach die Datei aus dem Projekt und kopieren sie dann dahin“

- Was ist wenn $name mal krank ist?

- Dringende Empfehlung: Buildsystem, z.B. Jenkins (o.Ä.)
  - Auführen von Build-scripten (Gradle, make, ant, …)
  - Bezieht Sourcen aus Repositories
  - Bezieht ggf. Artefakte von Artefakt-Servern
  - Frühzeitig Gedanken zum Buildsystem machen!
  - optional: Tests automatisch durchführen und Build nur bei Erfolg fortsetzen
  - optional: Statische Codeanalyse durchführen
  - optional: Code-Review einfordern (z.B. gerrit)

openSource im professionellen Einsatz



Quelle (CC BY-NC 2.5): http://imgs.xkcd.com/comics/standards.png

openSource im professionellen Einsatz

- Warum das Rad neu erfinden?

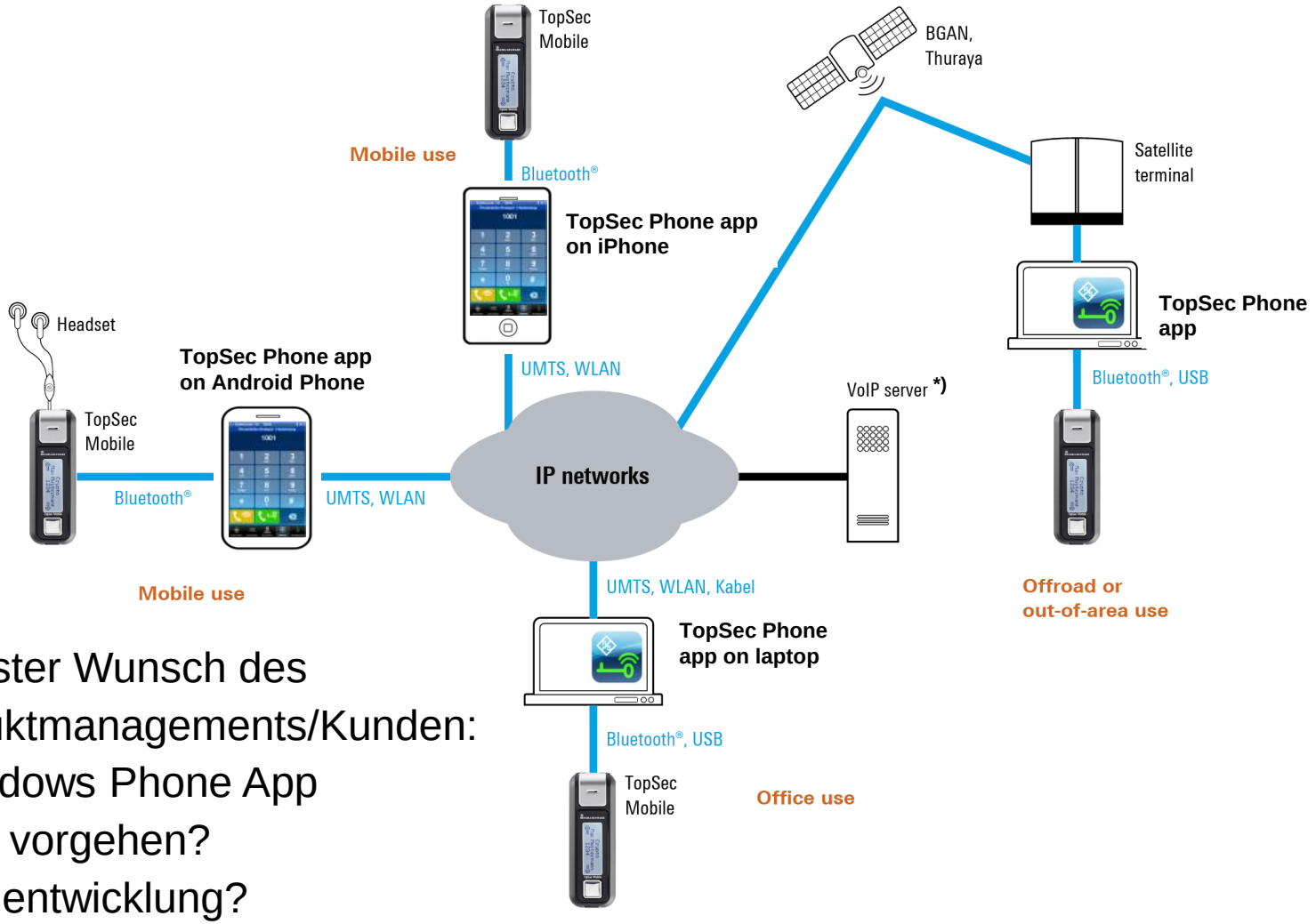
- Lizenz prüfen!
  - MIT, BSD, Apache, GPL,…
  - Was darf verwendet werden..
  - Welche Bedingungen/Konsquenzen gelten…

- Auch wenn openSource „im Netz verfügbar“ ist:
  - Ggf. Versionsverwaltung/Artefaktserver in Betracht ziehen
  - Archivierung nach (Konzern)Prozessen
  - Wiederherstellbarkeit sicherstellen

- openSource-Bibliotheken regelmäig aktualisieren

- Contribute!

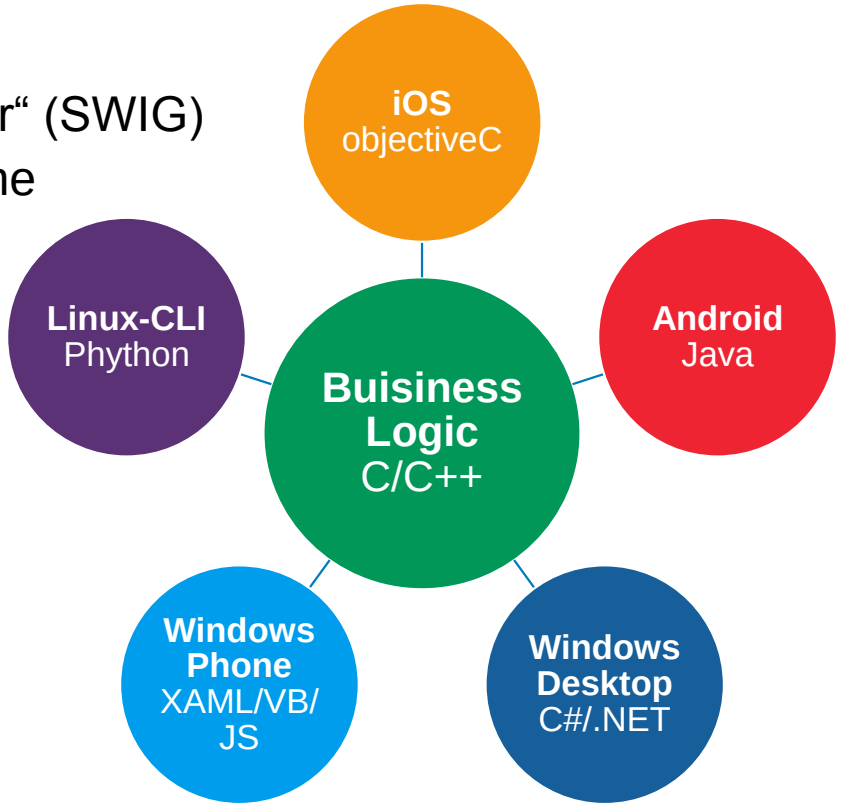
Plattformübergreifende Softwareentwicklung



- Nächster Wunsch des Produktmanagements/Kunden:
  - Windows Phone App
  - Wie vorgehen?
  - Neuentwicklung?

Plattformübergreifende Softwareentwicklung

- Wie entwickle ich effizient eine Anwendung für verschiedene Betriebssysteme?
  - Gleiche Algorithmen nicht mehrfach implementieren
    - Und auch nicht hin und her kopieren ;-)
- „Simplified Wrapper and Interface Generator“ (SWIG)
  - Gemeiname Datentypen über verschiedene Sprachen
- Wie nutze ich IDEs auf den Zielplattfomen?
  - CMAKE erzeugt Projekte zur effizienten Nutzung
- SWIG und CMAKE sind openSource



Agile Softwareentwicklung

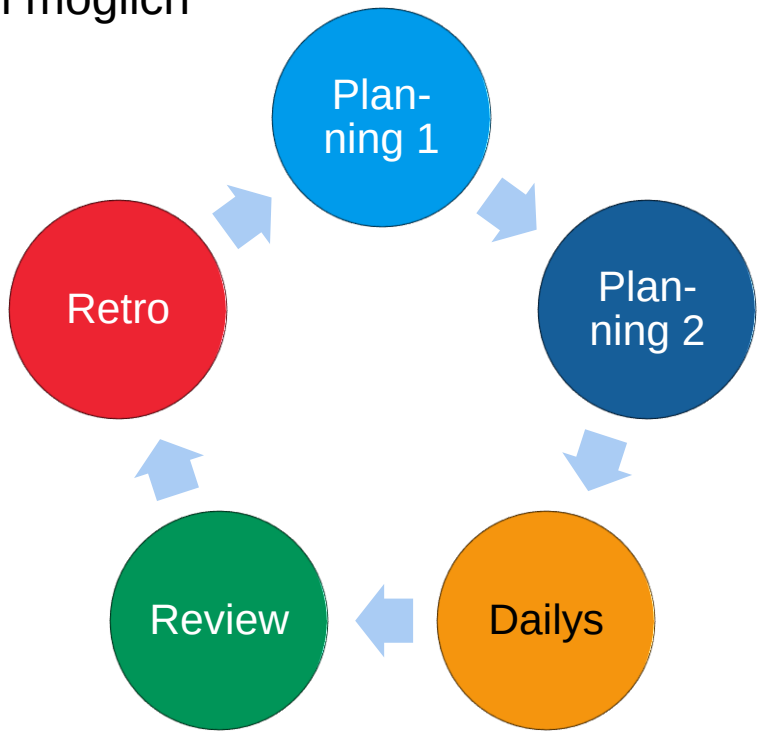
- Warum agile Entwicklung? Yet another Buzzword? „Scrum-Aber“?

- Erfahrungs-/Leidensbericht:
  - Wöchentliches „Change Controlboard über alle Tickets“ → 2h
  - Planung und Festsetzung von Terminen im Zeitraum von 1-2 Jahren mit dynamischen neuen Eingebungen von Produktmanagement/Vorgesetzten
  - Projekte mit 1-2 Jahre Requirementsphase und anschließender Verwerfung
  - Je früher Scrum implementiert wird, desto besser

- 1. Schritt: Chef überzeugen, dass agiles Vorgehen im Rahmen eines kleinen Projekts pilotmäßig getestet wird
- 2. Schritt: Erfolg ☺
- 3. Schritt: Schulung, ggf. Zertifizierung Scrummaster
- 4. Schritt: Aufbau/Entwicklung eines Scrum-Teams

Agile Softwareentwicklung

- Zusammenfassung
  - Keine 1-2 jährige Requirementsphase, „es geht direkt los“
  - Hochtransparent, hoher Wissenstransfer
  - Dynamik: schnelles Reagieren im Problemfall möglich
  - Fördert Team-Kommunikation
  - Ideal kombinierbar mit „pair-programming“
    - auch bei uns Wunschvorstellung
- Unterstützende Tools: s.u.
- Herausforderung:
  - Anpassung an Konzern-Prozesse
- Andere Agile Methoden: z.B. Kan Ban
  - Mehr Folien zu Scrum im Backup



Ticketsysteme / Requirements-Engineering

- 1. Überraschung:
  - MS Excel
  - Gern auch mit Suffixen per E-Mail verschickt:
    `requirements_v3_new_mw_final2_revised.xls`

- 2. Überraschung: IBM Rational Doors
  - Tabellen-artige, hierarchische Erfassung von Anforderungen
  - Zeilen/Zellen werden mit Volltexten gefüttert
  - gute Nachricht: es gibt MS Word-Export </ironie>

- Es stellt sich die Frage: wie kann die Erfüllung sinnvoll nachvollzogen werden?
  - Diskussionen über Erfüllung
  - Streit über Bedeutung der Requirements/Volltexte

Ticketsysteme / Requirements-Engineering

- Unser Ansatz: Redmine (Jira, etc.)
  - Verlinkt mit GIT-Repo: Commits sind mit Tickets/Features verbunden
  - Wiki für Hinweise Dokumentation
  - Modulares/erweiterbares System (Story Points, Criterien, etc.)

- Wie schreibe ich eine Story? Mit Grammatik!
  - As a **[role]** I want to **[feature]** so that **[benefit]**

- As a **User** I want to **see the battery status** so that **I know if I need to charge my TSM**.
- As a **Dev** I want to **set up a CI system** so that **we are able to introduce continuous integration**.

- Ganz wichtig: Akzeptanz Criterien, sowie Definition of Done
  - 5 parallele Verbindungen
  - Getestet, Dokumentiert, Gemerged, etc.

Ihr habt schon über 20 Slides überstanden, nun kommt Bonus ☺

Bonus: Angewandte Kryptografie für Behörden & Co.

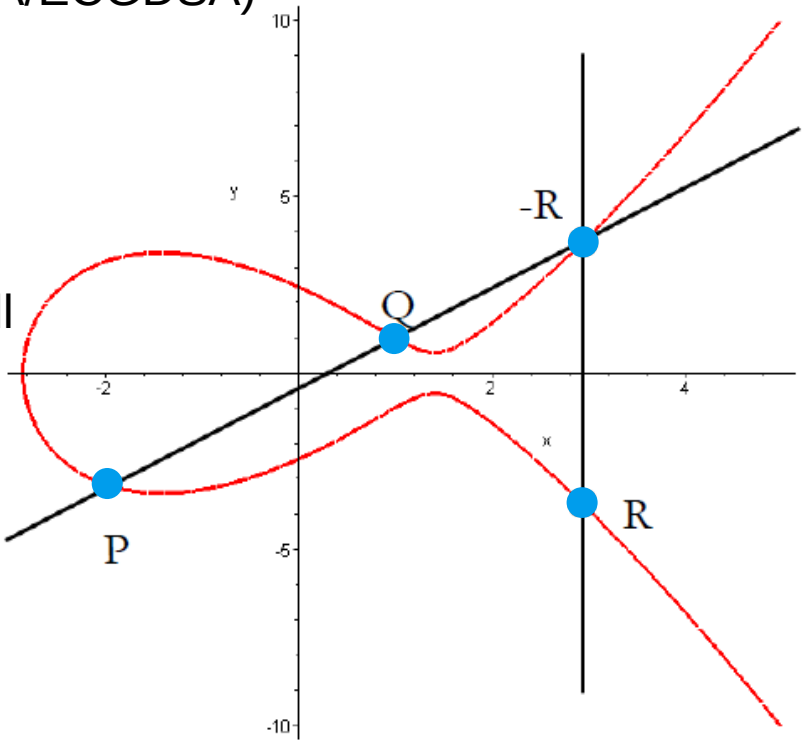
- Häufiger Projektpartner: Bundesamt für Sicherheit in der Informationstechnik
- Entwicklungsgegenstand: Kommunikationsmittel für Verschlussachen
  - „Nur für den Dienstgebrauch“ bis „STRENG GEHEIM“

- Entwicklung erfolgt in speziell zugelassenen Zonen (Riesentresor)
  - Phyisch getrennte Netzwerkstrukturen
  - Entwicklungsequipment auf Abstrahlung geprüft
    - PCs, Maus, Tastatur

- Entwicklung und Fertigung sicherheitsrelevanter Produkte erfolgt durch überprüfte Mitarbeiter
  - R&S fertigt diese Komponenten in Deutschland („VS-Produktion“)

Bonus: Angewandte Kryptografie für Behörden & Co.

- Evaluierung nach internationalem Standard: Common Criteria
  - Evaluation Assurance Level
- Implementierung Elliptischer Kurven (ECDSA/ECGDSA)
  - Sehr guter Vortrag auf 31C3
- Implementierung eingestufter Algorithmen
- Forschung/Implementierung von guten Zufall für Schlüsselmaterial
- Notlöschung von Geräten/Komponenten
- Erkennung physischer Angriffe



Bonus: Einsatz von DSLs in der „Industrie“ und plattformunabhängige Softwareentwicklung

- Entwurf einer eignen Grammatik für Softwareentwicklung für Verschlüsselungsgeräte
  - Requirements aus früheren Projekten in „IBM Rational Doors“ vorhanden
    - Wie vollständig sind die Requirements

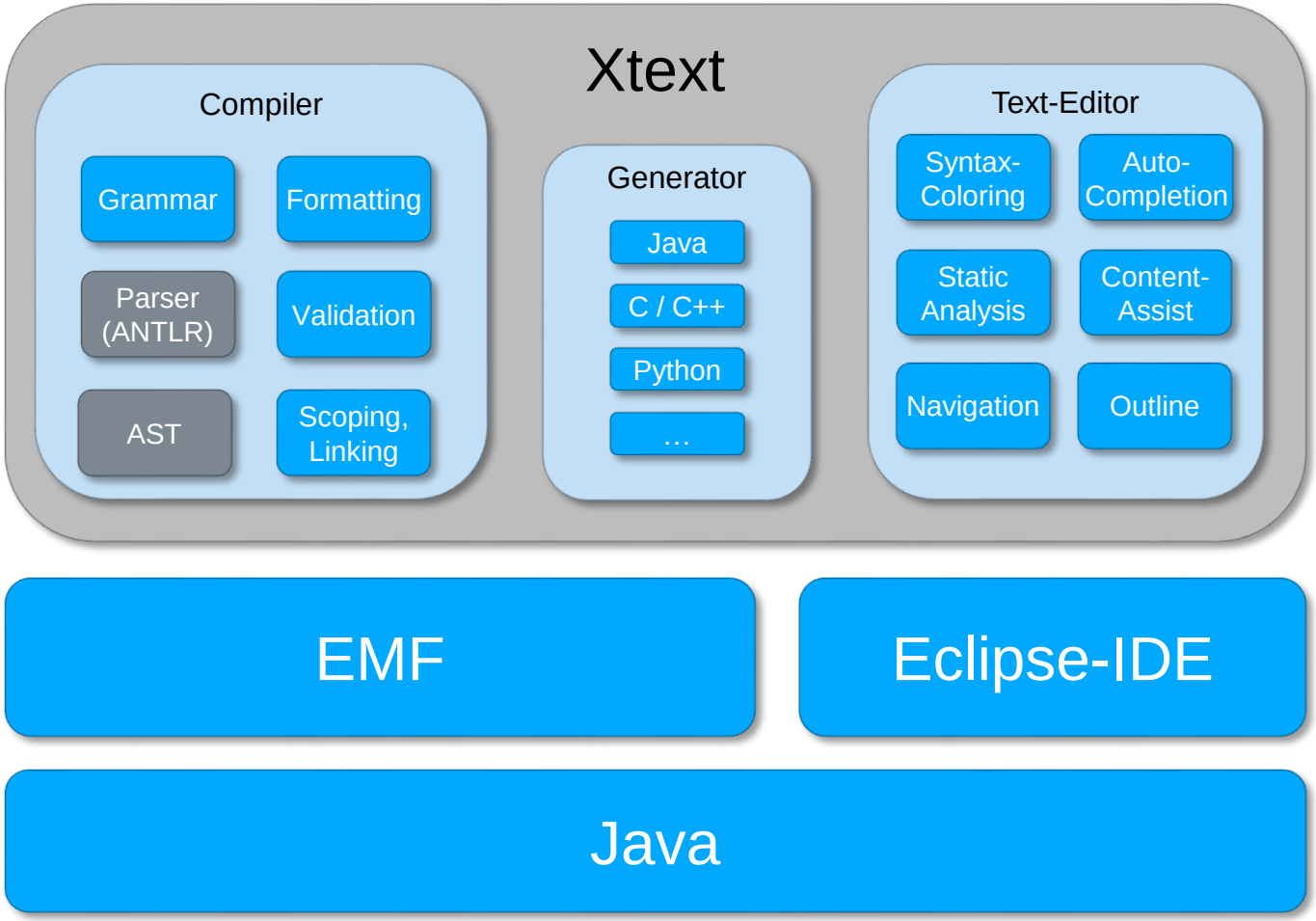
- Requirements über Usecases vervollständigen
  - Usecases erstellt in Prosa (DOORS)
  - In regelmäßigen Reviews immer wieder Verständnisprobleme
  - Unklare Begrifflichkeiten (Akteur X, Komponente Y…, was ist mit bestimmten Verben gemeint „Komponente X verarbeitet“)
  - Erstellung Definitionsbeschreibung
  - Ständige Nachbesserungen

Bonus: Einsatz von DSLs in der „Industrie“ und plattformunabhängige Softwareentwicklung

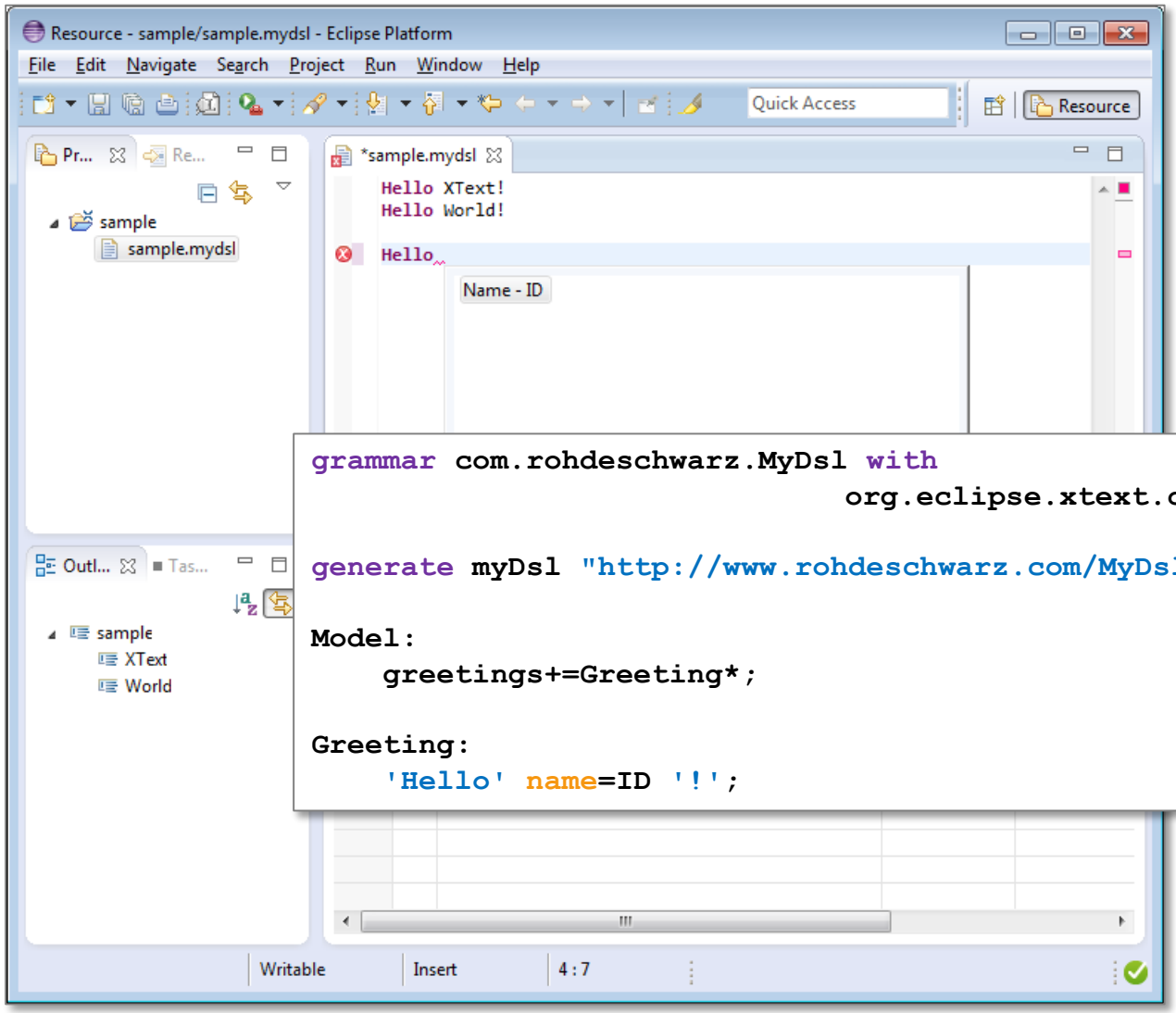
- einheitliche Sicht auf das System
- Mehr **formalere** Beschreibung der **Usecases**
- Klare Definition von Begrifflichkeiten:
  - Welche **Komponenten/Akteuere** gibt es?
  - Welche Verantwortung haben sie?
  - Wer interagiert mit wem?
- Geeignete Toolunterstützung bei Änderungen

→ DSL zur Beschreibung von Usecases

openArchitectureWare XText



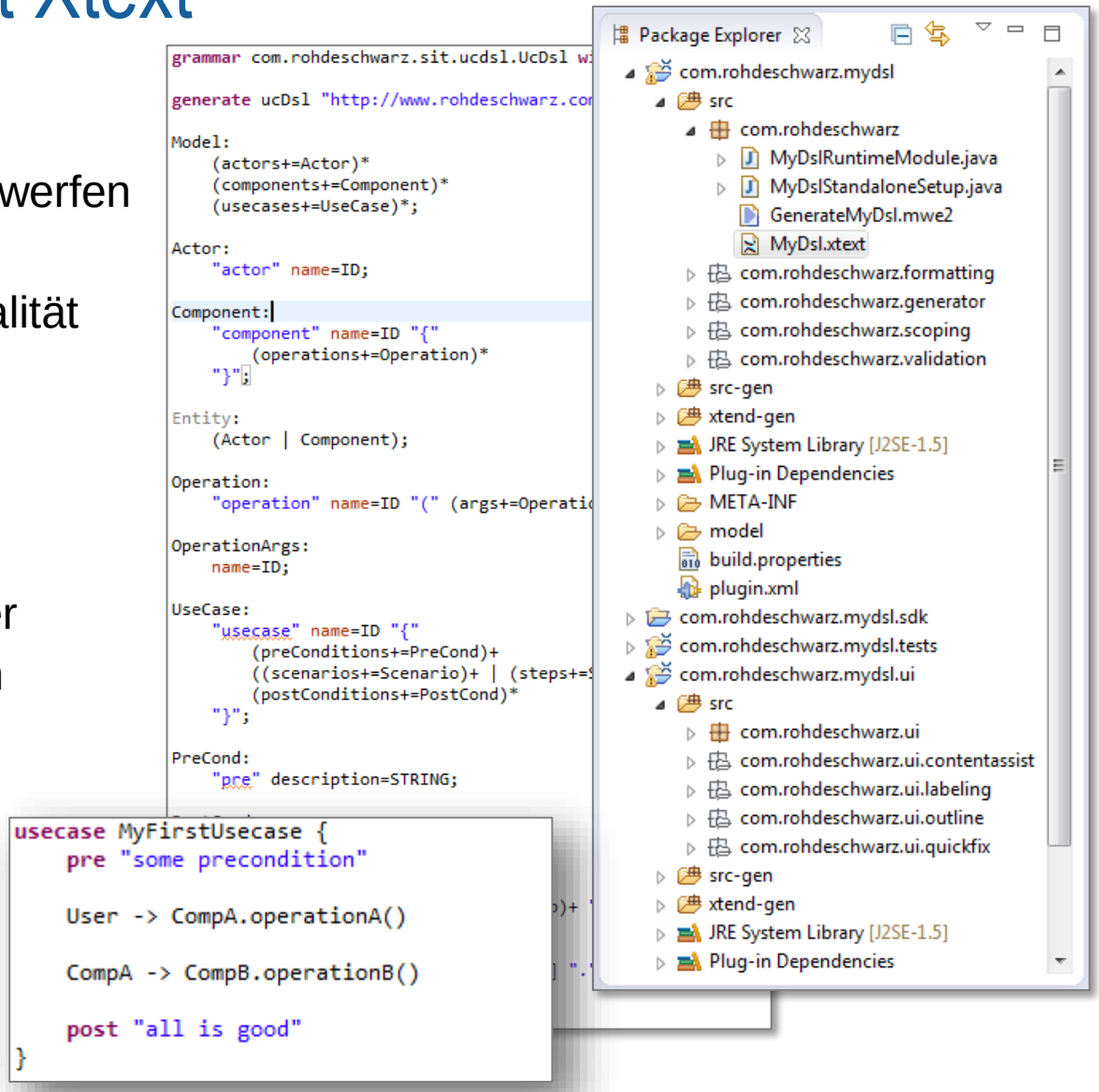
XText-Beispiel




```
grammar com.rohdeschwarz.MyDsl with
                                org.eclipse.xtext.common.Terminals

generate myDsl "http://www.rohdeschwarz.com/MyDsl"

Model:
    greetings+=Greeting*;

Greeting:
    'Hello' name=ID '!';
```

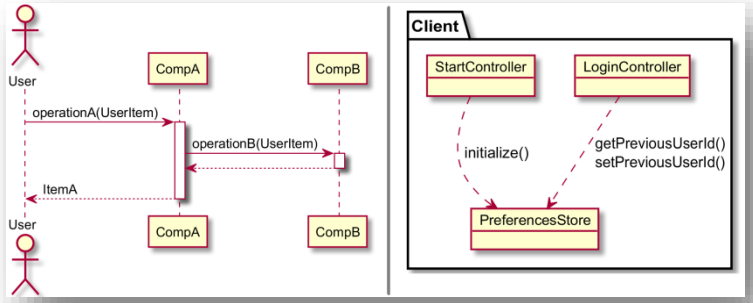
DSL Erstellung mit Xtext

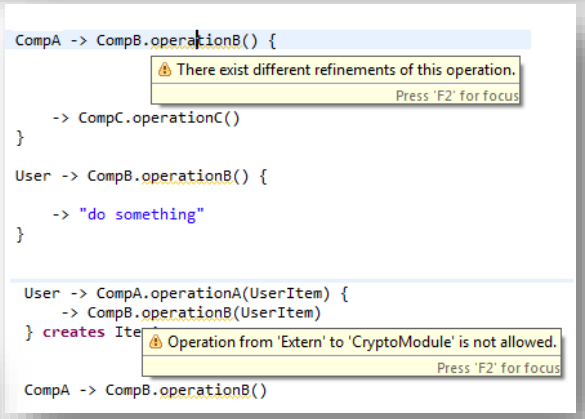
- Grammatik mit Regeln entwerfen
- Syntax-/Semantikfunktionalität erweitern
- Benutzerkomfort erstellen
- schrittweise Ergänzung der Grammatik mit Anwendern
- Bereitstellung über Eclipse-Update-Site (P2)

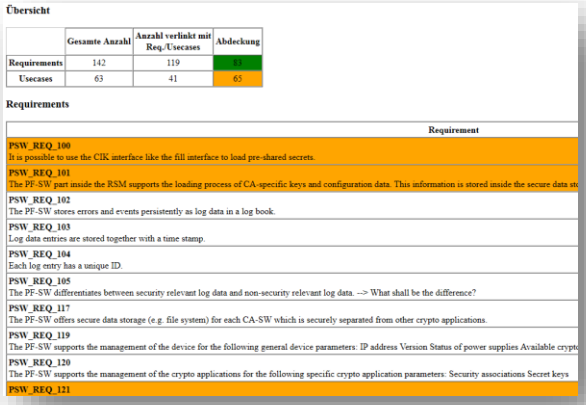


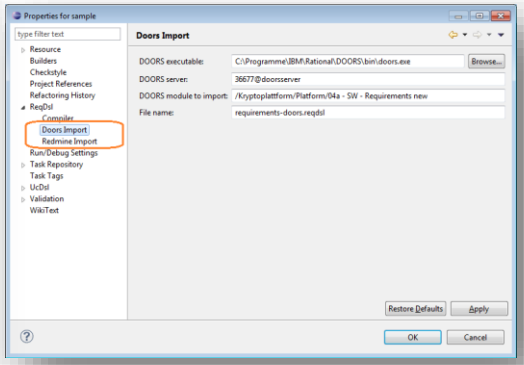
DSL Erstellung – Lust auf mehr

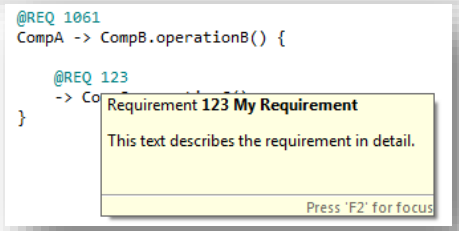
- Visualisierung mit UML (PlantUML)
- fachliche Validierungen
- Requirements-Tracking
- Requirements-Import (DOORS/Redmine)
- Reports erstellen











DSLs: Lessons Learned

- DSLs besonders an fachlichen Schnittstellen sinnvoll
  - z.B. für Test-Skripte, UI-Layout

- DSL-Design: immer fokussiert bleiben auf Problem-Domäne

- DSLs im konkreten Fall genutzt um
  - präziser zu sein in der Formulierung (semi-formal)
  - Einheitliches Verständnis zu entwickeln
  - Architekturen zu finden/entwickeln
    z.B. „Komponenten mit ihren Verantwortlichkeiten ergeben sich aus Usecases“
  - fachliche Validierungen frühzeitig vorzunehmen
    z.B. „Darf Komponente X mit Komponente Y reden“
  - toolgestützte Änderung-/Nachverfolgbarkeit zu gewährleisten
  - Codegenerierung durchzuführen

tl;dr

Zusammenfassung

tl;dr

- „Dezentrale, schlanke Versionsverwaltung“ vs. Monstertools
- „Testautomatisierung“ vs. „von Hand testen“
- Clean Code, Design Patterns & Metriken
- „Continuous Integration“ vs. „$name erzeugt immer das Release“
- „openSource“ vs. „das Rad neu erfinden“
- Plattformübergreifende Softwareentwicklung
- Agile Softwareentwicklung
- „Excel-like“ vs. Ticket-System

- Kryptografie für Behörden sind sehr anspruchsvolle Projekte mit besonderen Rahmenbedingungen
- DSLs werden auch in der Praxis verwendet

- Engagiert Euch ☺
  - Es wird vielleicht nicht so einfach; je früher desto besser!

Vielen Dank für Eure Aufmerksamkeit!

Fragen?