QUALITÄTSKRITERIEN SAFETY UND RELIABILITY IN RAUMFAHRT-FLUGSOFTWARE

Daniel Lüdtke

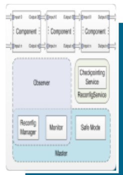
DLR Institut für Softwaretechnologie

Forschung und Entwicklung für Flugsoftware

Forschungsschwerpunkte:

- Softwareentwicklung für Flugsoftware
- Software-Produktsicherung für Flugsoftware
- Fehlertolerante und zuverlässige Software
- Rekonfigurierbare verteilte (Multicore-)Bordcomputer
- Echtzeit-Analyse und –Ausführungsplattformen
- Safety und Security von Flugsoftware
- Bewertung von modernen Programmiersprachen/-paradigmen für sicherheitskritische Systeme

On-board Applications



Middleware

(Real-time) Operating Systems

Embedded Programming

Mission:

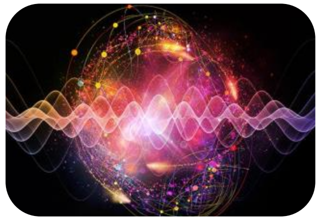
- Wir entwicklen zuverlässige und resiliente Echtzeit-Software für Luft- und Raumfahrtsysteme

Laufende Projekte und Missionen

ADMIRE

BECCAL

CALLISTO



CARIOQA Phase A

CARIOQA-PMP

COMPASSO

cRustacea in Space

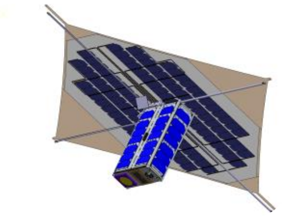
EnVision

HAP

MAIUS

MMX

PLATO



PLUTO

QYRO

Wiederstellung
Überwachung
Abwehr

RESILIENCE

ReFEx

SaiNSOR

ScOSA FE

SIDE

TITENT

VERITAS

SAFETY UND RELIABILITY IN DER RAUMFAHRT – STELLENWERT VON SOFTWARE

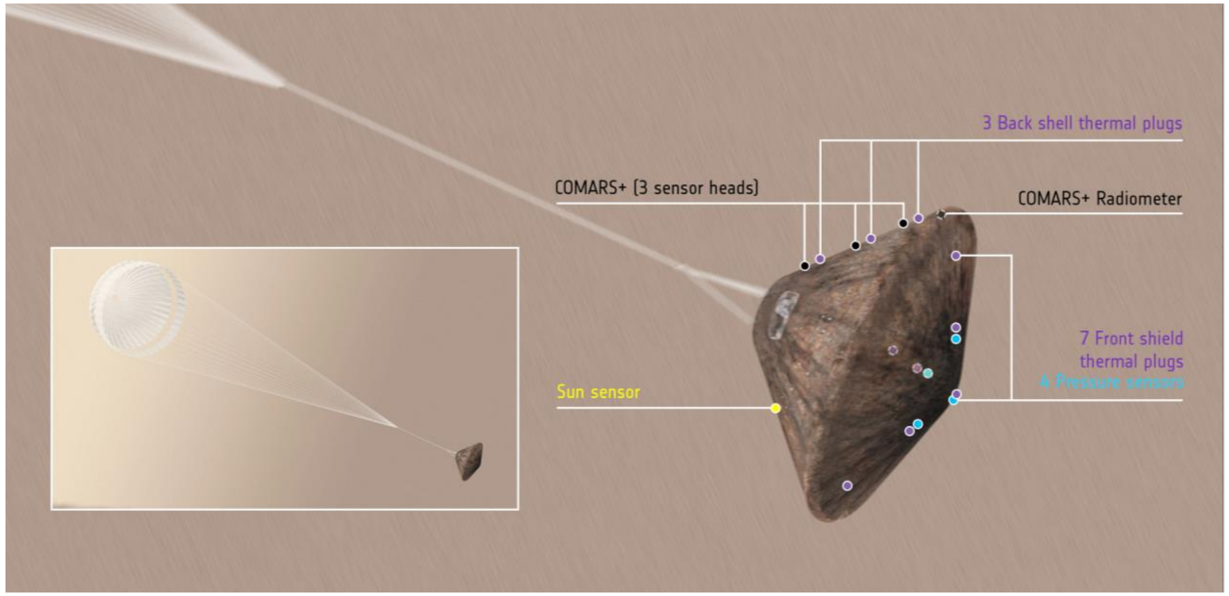
https://www.esa.int/ESA_Multimedia/Images/2009/09/Explosion_of_first_Ariane_5_flight_June_4_1996

EXOMARS 2016 - Schiaparelli Anomaly
a.k.a. „New Crater on Mars“

The following root causes for the mishap have been identified:

- Insufficient uncertainty and configuration management in the modelling of the parachute dynamics which led to expect much lower dynamics than observed in flight;
- Inadequate persistence time of the IMU saturation flag and inadequate handling of IMU saturation by the GNC;
- Insufficient approach to Failure Detection, Isolation and Recovery and design robustness;
- Mishap in management of subcontractors and acceptance of hardware

Source: http://exploration.esa.int/science-e/www/object/doc.cfm?fobjectid=59175

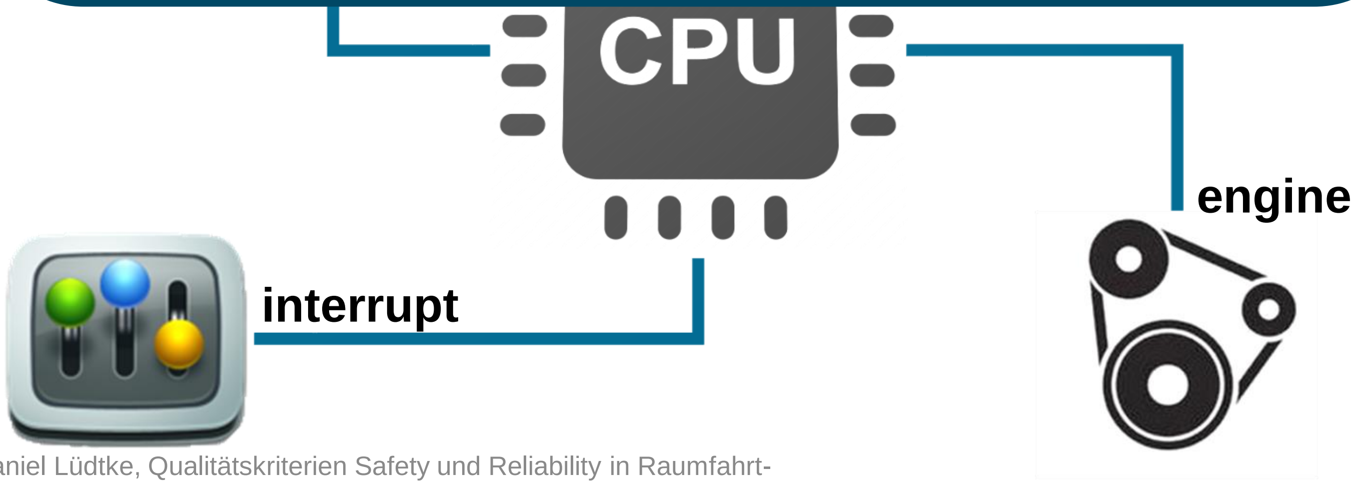


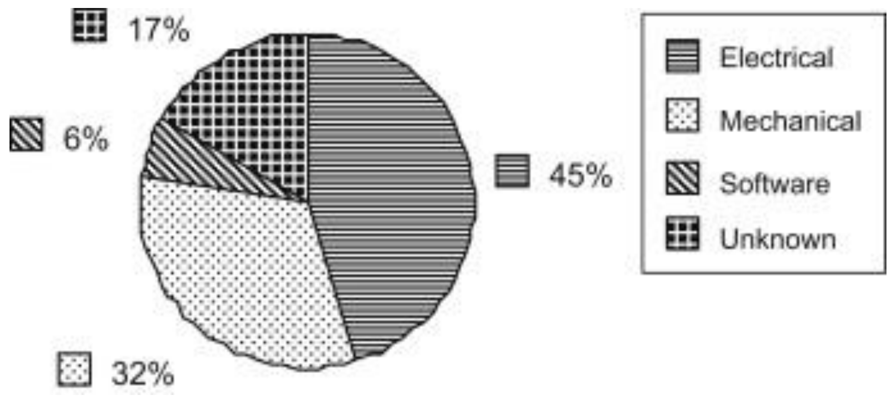
https://www.esa.int/ESA_Multimedia/Images/2017/05/Heatshield_sensors

Ausfälle, die durch Software ausgelöst werden

Softwarebedingte Unfälle werden in der Regel durch fehlerhafte Anforderungen verursacht:

- Unvollständige oder falsche Annahmen über den Betrieb des kontrollierten Systems oder den erforderlichen Betrieb des Computers
- Unbehandelte Zustände des kontrollierten Systems und Umgebungsbedingungen





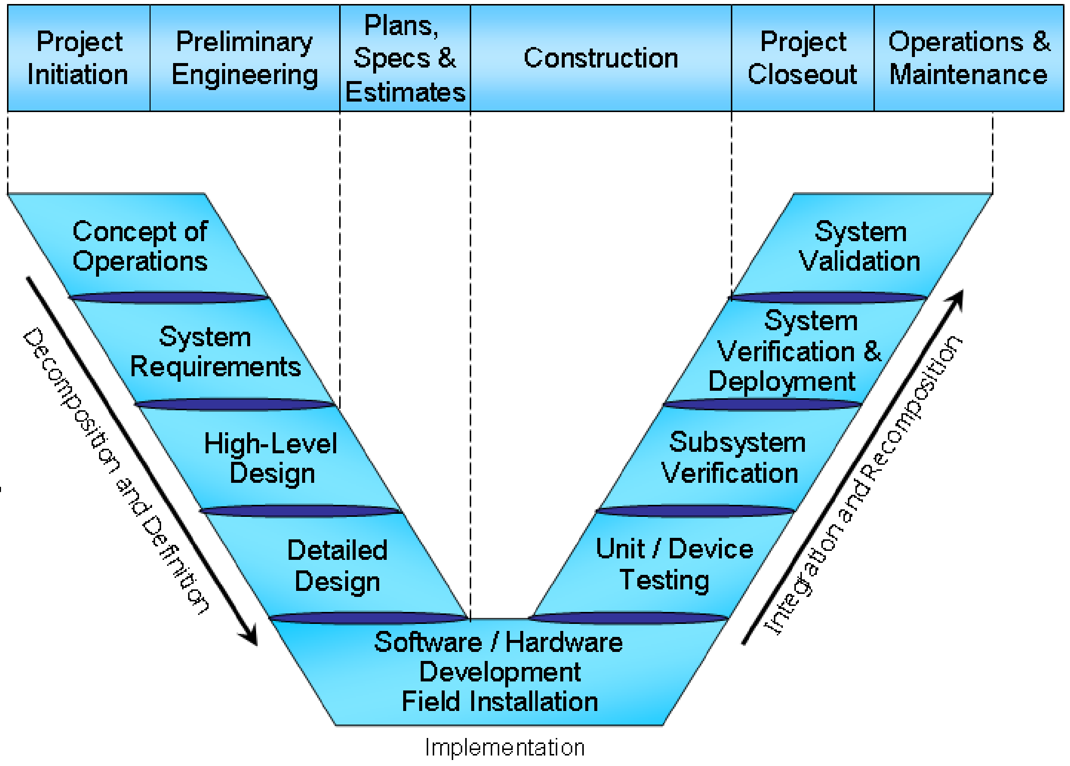
A study of on-orbit spacecraft failures
Mak Tafazoli, 2009

Concurrent Engineering Facility, DLR Insitut für Raumfahrtsysteme, Bremen

RAUMFAHRT-ENTWICKLUNG

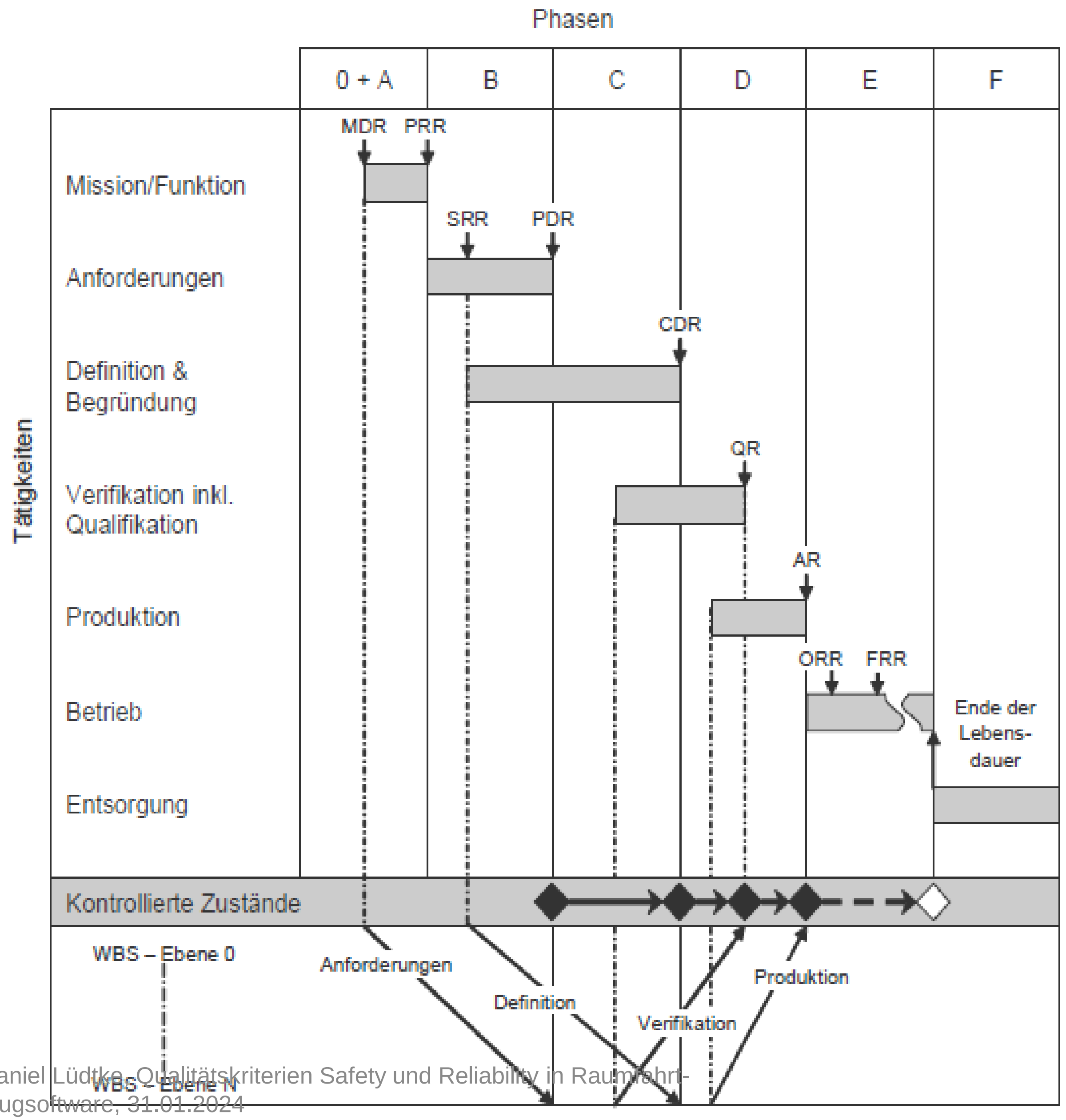
Wie baut man einen Satelliten?
V-Modell und ECSS

- Raumfahrtmissionen folgen oft dem V-Modell
- *European Cooperation for Space Standardization* (ECSS) definiert alle für Raumfahrtsysteme relevanten Normen
- Zweige M(anagement), E(ngineering), Q(ualität)
- Tayloring der Standards notwendig!
- Entwickelt von Experten der ESA, nationaler Agenturen und der Industrie
- Ähnliche Normen bei NASA/JAXA oder für Galileo



Source: US Department of Transportation, 2006

Lebenszyklus einer Raumfahrt-Mission



- **MDR** = Mission Definition Review
- **PRR** = Preliminary Requirements Review
- **SRR** = Systems Requirements Review
- **PDR** = Preliminary Design Review
- **CDR** = Critical Design Review
- **QR** = Qualification Review
- **AR** = Acceptance Review
- **ORR** = Operational Readiness Review
- **FRR** = Flight Readiness Review

ECSS Standards regarding Software

ECSS-Q-ST-80C Rev.1
15 February 2017

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space product assurance

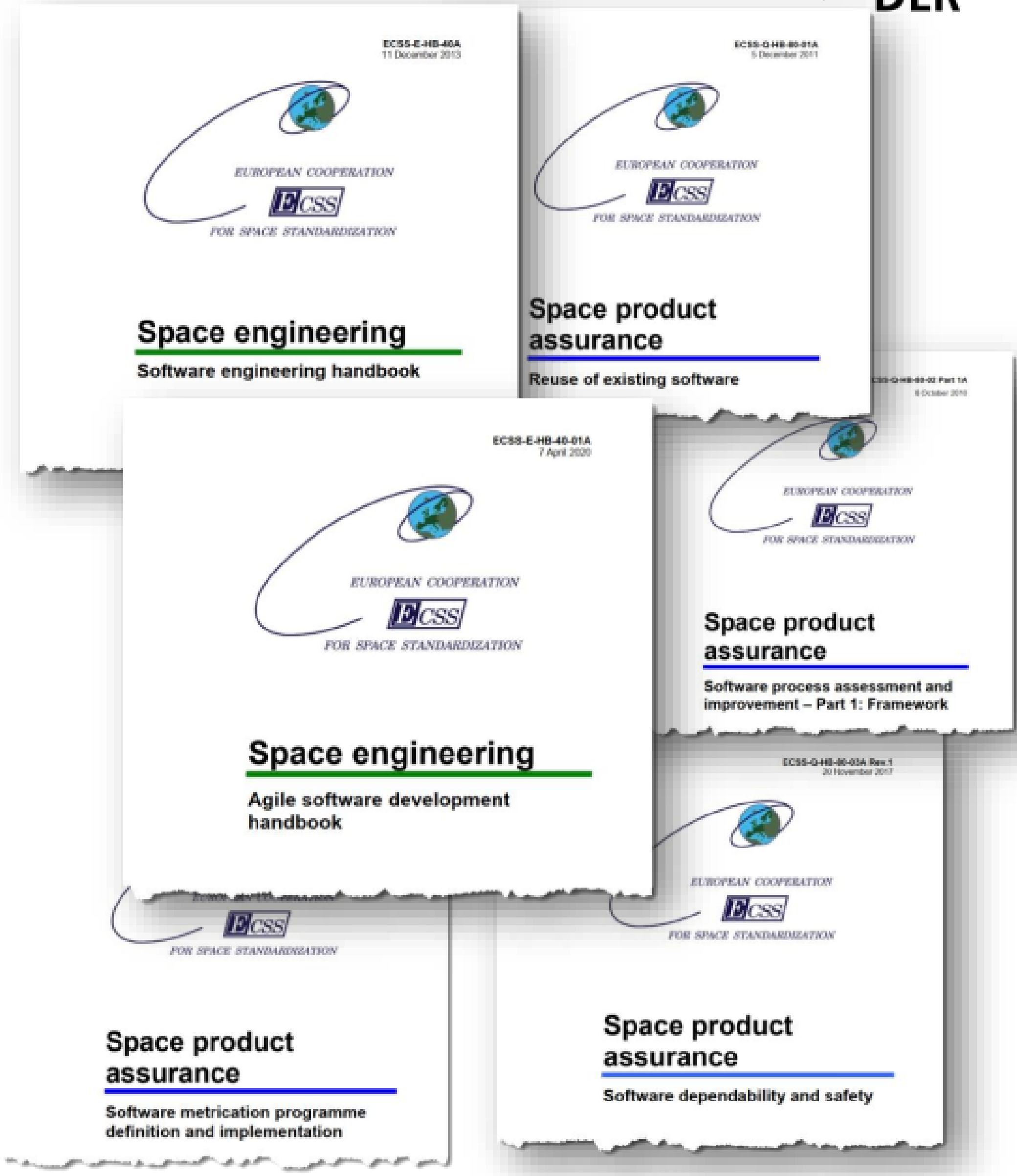
Software product assurance

ECSS-E-ST-40C
6 March 2009

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space engineering

Software



ECSS-E-HB-40A
11 December 2013

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space engineering

Software engineering handbook

ECSS-Q-HB-80-01A
5 December 2011

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space product assurance

Reuse of existing software

CSS-Q-HB-80-02 Part 1A
6 October 2010

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space product assurance

Software process assessment and improvement – Part 1: Framework

ECSS-E-HB-40-01A
7 April 2020

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space engineering

Agile software development handbook

ECSS
FOR SPACE STANDARDIZATION

Space product assurance

Software metrication programme definition and implementation

ECSS-Q-HB-80-03A Rev.1
20 November 2017

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space product assurance

Software dependability and safety

Quelle: Christian Prause, DLR Raumfahrtagentur

SAFETY UND RELIABILITY

EnMAP, Soiurce: OHB System AG/DLR

Safety vs. Reliability

- **Reliability:** ist die Wahrscheinlichkeit, dass eine Funktion unter bestimmten Umweltbedingungen und über einen bestimmten Zeitraum erfüllt wird
- **Safety:** kein inakzeptables Risiko von Personen- oder Sachschäden (unabhängig von der Korrektheit des Systems)

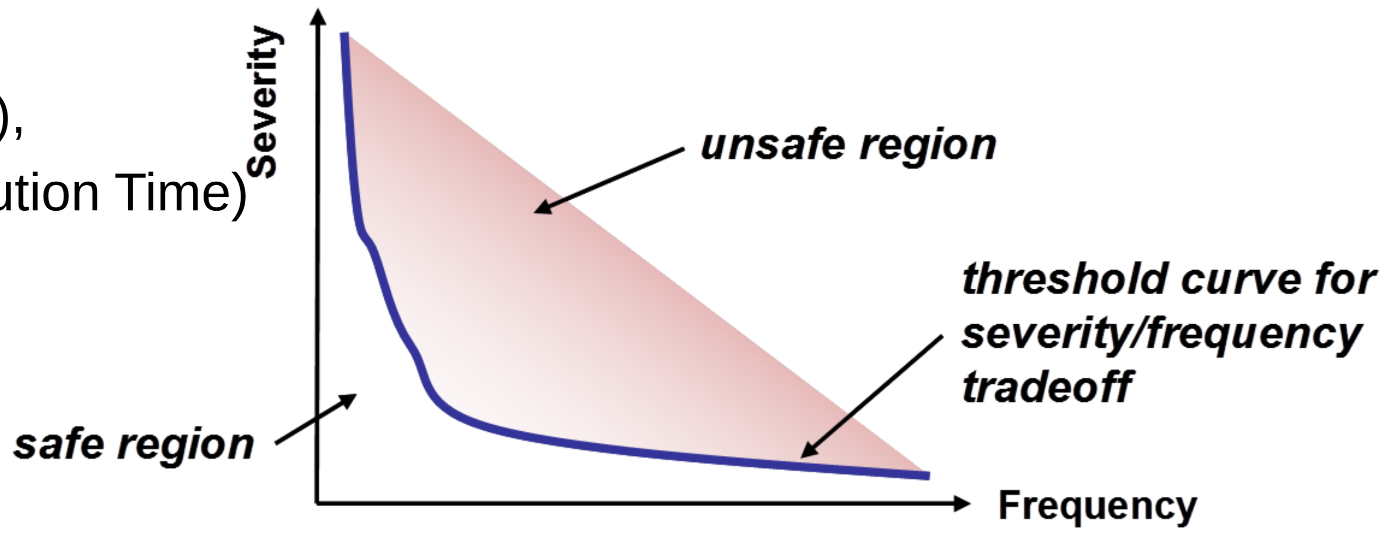
Reliability ≠ Safety

Safety lässt sich nur durch Reliability erreichen

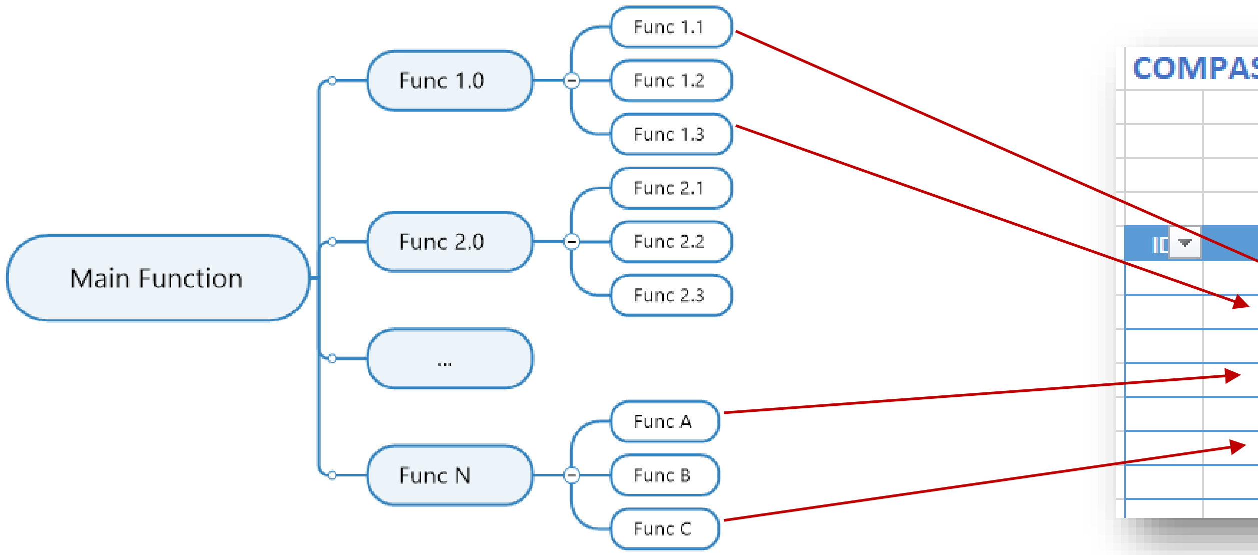
Design für Safety

- Das Risiko ist durch zwei Eigenschaften gekennzeichnet
  - Häufigkeit von gefährlichen Ereignissen
  - Schwere der gefährlichen Ereignisse

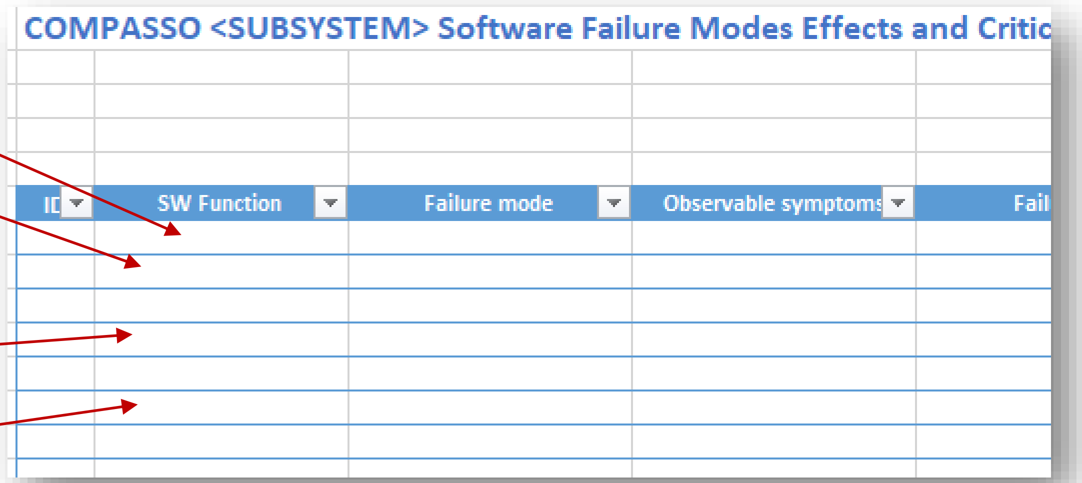
- Beispiele für Methoden zur Bestimmung der Häufigkeit und Schwere
  - Failure Mode, Effects, (and Diagnostics) Analysis (FME(D)A),
  - Fault Tree Analysis (FTA),
  - Fault Injection Test (Coverage-Ziele),
  - Timing Analysen (Worst Case Execution Time)



Beispiel Software-FMEA



Failure Modes and Effects Analysis (SW-FMEA)


COMPASSO <SUBSYSTEM> Software Failure Modes Effects and Critic

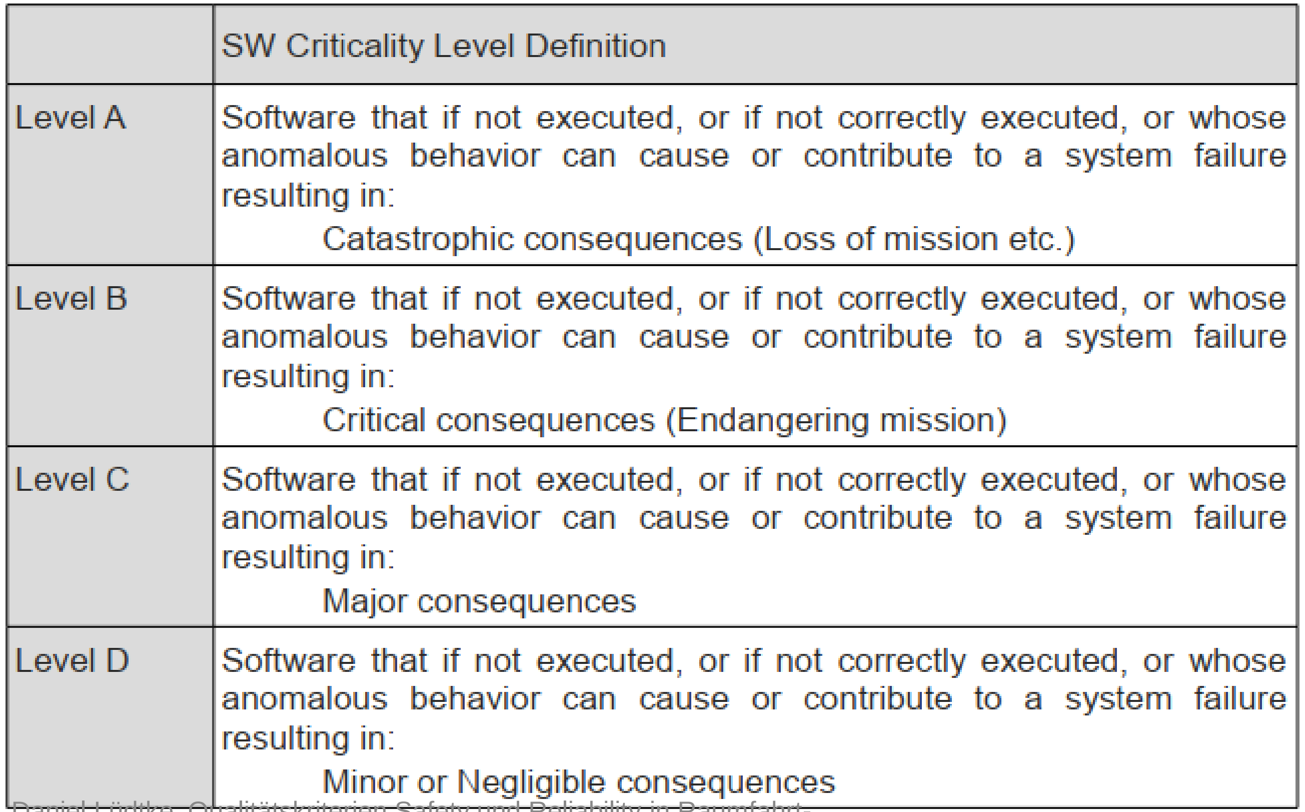
ID	SW Function	Failure mode	Observable symptoms	Fail

- Beobachtbare Symptome
- Ausfallursache Auswirkung auf das System
- Schwere und Kritikalität des Ausfalls
- Kompensationsmaßnahmen
- Kritikalitätsreduzierung der Kompensationsmaßnahmen

Software-Kritikalitätsanalyse

Aktivitäten nach ECSS hängen von der Kritikalität ab
(kritischer → mehr Anforderungen)

Table 11.1: Software criticality levels.
© ECSS-Q-ST-80C, Annex D

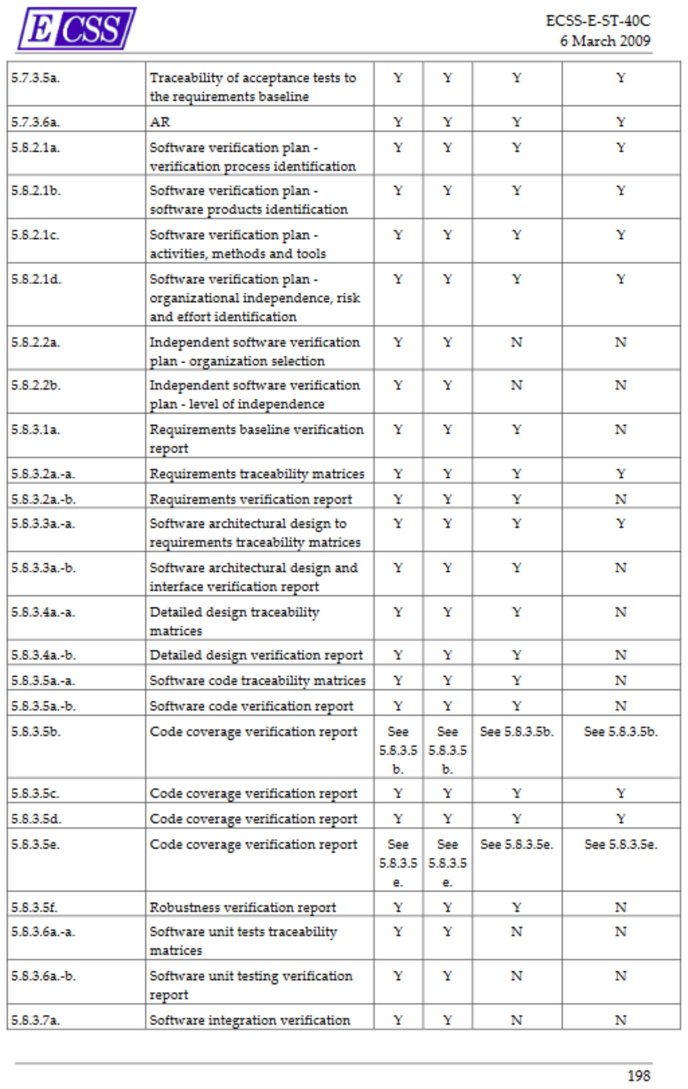


	SW Criticality Level Definition
Level A	Software that if not executed, or if not correctly executed, or whose anomalous behavior can cause or contribute to a system failure resulting in: Catastrophic consequences (Loss of mission etc.)
Level B	Software that if not executed, or if not correctly executed, or whose anomalous behavior can cause or contribute to a system failure resulting in: Critical consequences (Endangering mission)
Level C	Software that if not executed, or if not correctly executed, or whose anomalous behavior can cause or contribute to a system failure resulting in: Major consequences
Level D	Software that if not executed, or if not correctly executed, or whose anomalous behavior can cause or contribute to a system failure resulting in: Minor or Negligible consequences

Anomales Verhalten von Software wird definiert als:
- Softwarefunktion reagiert mit **falschem Timing**,
- Softwarefunktion reagiert mit **falschem Ergebnis**,
- Softwarefunktion trägt zum **Systemausfall** bei.

Eine ähnliche Klassifizierung findet sich in der DO178 mit der Bezeichnung "Certification levels" und im Galileo SW Standard, genannt “Development Assurance Levels” – im Bereich von DAL A bis DAL E.

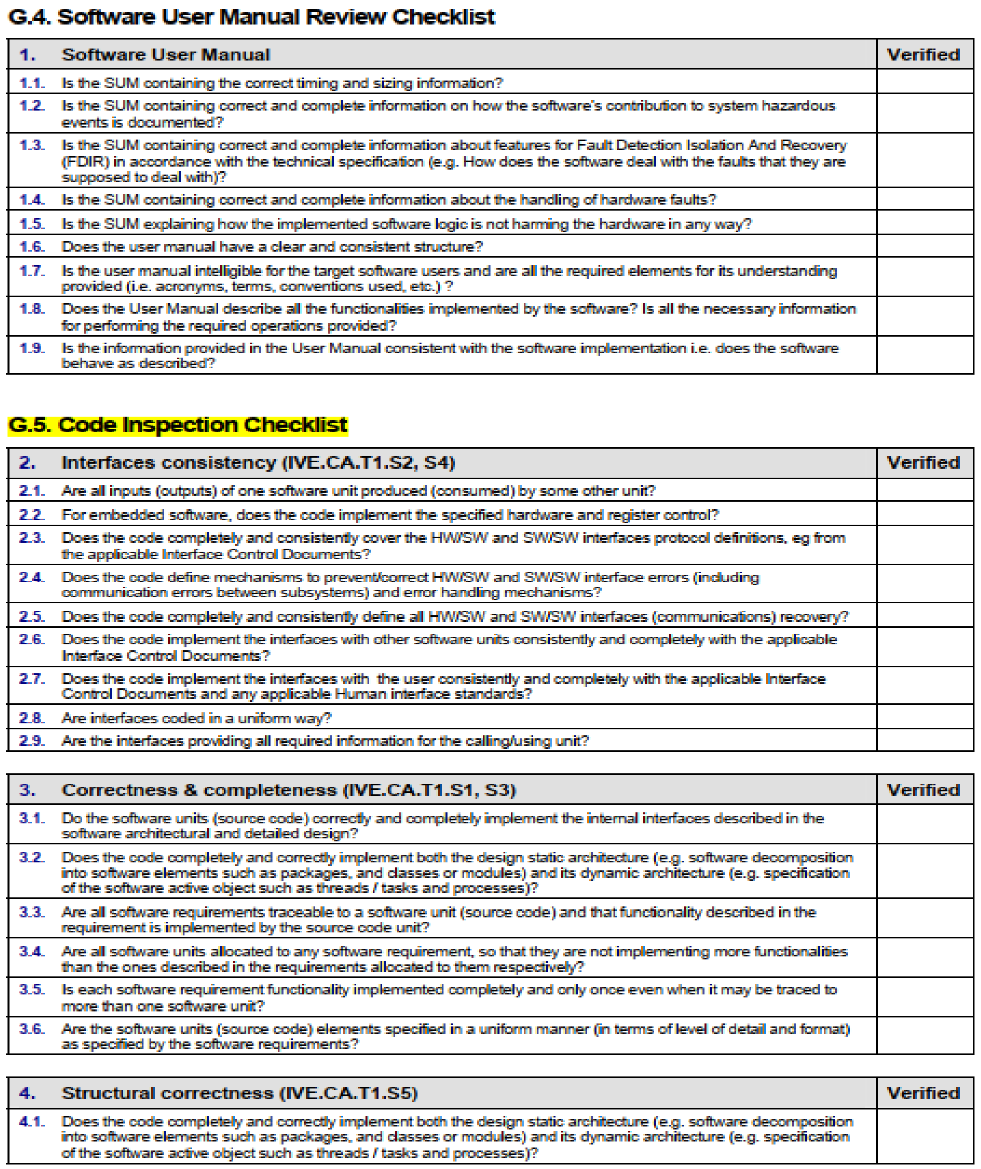
Tailoring entsprechend der Kritikalitätsklasse und Independent Software Verification and Validation (ISVV)



ECSS-E-ST-40C
6 March 2009

5.7.3.5a.	Traceability of acceptance tests to the requirements baseline	Y	Y	Y	Y
5.7.3.6a.	AR	Y	Y	Y	Y
5.8.2.1a.	Software verification plan - verification process identification	Y	Y	Y	Y
5.8.2.1b.	Software verification plan - software products identification	Y	Y	Y	Y
5.8.2.1c.	Software verification plan - activities, methods and tools	Y	Y	Y	Y
5.8.2.1d.	Software verification plan - organizational independence, risk and effort identification	Y	Y	Y	Y
5.8.2.2a.	Independent software verification plan - organization selection	Y	Y	N	N
5.8.2.2b.	Independent software verification plan - level of independence	Y	Y	N	N
5.8.3.1a.	Requirements baseline verification report	Y	Y	Y	N
5.8.3.2a.-a.	Requirements traceability matrices	Y	Y	Y	Y
5.8.3.2a.-b.	Requirements verification report	Y	Y	Y	N
5.8.3.3a.-a.	Software architectural design to requirements traceability matrices	Y	Y	Y	Y
5.8.3.3a.-b.	Software architectural design and interface verification report	Y	Y	Y	N
5.8.3.4a.-a.	Detailed design traceability matrices	Y	Y	Y	N
5.8.3.4a.-b.	Detailed design verification report	Y	Y	Y	N
5.8.3.5a.-a.	Software code traceability matrices	Y	Y	Y	N
5.8.3.5a.-b.	Software code verification report	Y	Y	Y	N
5.8.3.5b.	Code coverage verification report	See 5.8.3.5 b.	See 5.8.3.5 b.	See 5.8.3.5b.	See 5.8.3.5b.
5.8.3.5c.	Code coverage verification report	Y	Y	Y	Y
5.8.3.5d.	Code coverage verification report	Y	Y	Y	Y
5.8.3.5e.	Code coverage verification report	See 5.8.3.5 e.	See 5.8.3.5 e.	See 5.8.3.5e.	See 5.8.3.5e.
5.8.3.5f.	Robustness verification report	Y	Y	Y	N
5.8.3.6a.-a.	Software unit tests traceability matrices	Y	Y	N	N
5.8.3.6a.-b.	Software unit testing verification report	Y	Y	N	N
5.8.3.7a.	Software integration verification	Y	Y	N	N

198



G.4. Software User Manual Review Checklist

1.	Software User Manual	Verified
1.1.	Is the SUM containing the correct timing and sizing information?	
1.2.	Is the SUM containing correct and complete information on how the software's contribution to system hazardous events is documented?	
1.3.	Is the SUM containing correct and complete information about features for Fault Detection Isolation And Recovery (FDIR) in accordance with the technical specification (e.g. How does the software deal with the faults that they are supposed to deal with)?	
1.4.	Is the SUM containing correct and complete information about the handling of hardware faults?	
1.5.	Is the SUM explaining how the implemented software logic is not harming the hardware in any way?	
1.6.	Does the user manual have a clear and consistent structure?	
1.7.	Is the user manual intelligible for the target software users and are all the required elements for its understanding provided (i.e. acronyms, terms, conventions used, etc.) ?	
1.8.	Does the User Manual describe all the functionalities implemented by the software? Is all the necessary information for performing the required operations provided?	
1.9.	Is the information provided in the User Manual consistent with the software implementation i.e. does the software behave as described?	

G.5. Code Inspection Checklist

2.	Interfaces consistency (IVE.CA.T1.S2, S4)	Verified
2.1.	Are all inputs (outputs) of one software unit produced (consumed) by some other unit?	
2.2.	For embedded software, does the code implement the specified hardware and register control?	
2.3.	Does the code completely and consistently cover the HW/SW and SW/SW interfaces protocol definitions, eg from the applicable Interface Control Documents?	
2.4.	Does the code define mechanisms to prevent/correct HW/SW and SW/SW interface errors (including communication errors between subsystems) and error handling mechanisms?	
2.5.	Does the code completely and consistently define all HW/SW and SW/SW interfaces (communications) recovery?	
2.6.	Does the code implement the interfaces with other software units consistently and completely with the applicable Interface Control Documents?	
2.7.	Does the code implement the interfaces with the user consistently and completely with the applicable Interface Control Documents and any applicable Human interface standards?	
2.8.	Are interfaces coded in a uniform way?	
2.9.	Are the interfaces providing all required information for the calling/using unit?	

3.	Correctness & completeness (IVE.CA.T1.S1, S3)	Verified
3.1.	Do the software units (source code) correctly and completely implement the internal interfaces described in the software architectural and detailed design?	
3.2.	Does the code completely and correctly implement both the design static architecture (e.g. software decomposition into software elements such as packages, and classes or modules) and its dynamic architecture (e.g. specification of the software active object such as threads / tasks and processes)?	
3.3.	Are all software requirements traceable to a software unit (source code) and that functionality described in the requirement is implemented by the source code unit?	
3.4.	Are all software units allocated to any software requirement, so that they are not implementing more functionalities than the ones described in the requirements allocated to them respectively?	
3.5.	Is each software requirement functionality implemented completely and only once even when it may be traced to more than one software unit?	
3.6.	Are the software units (source code) elements specified in a uniform manner (in terms of level of detail and format) as specified by the software requirements?	

4.	Structural correctness (IVE.CA.T1.S5)	Verified
4.1.	Does the code completely and correctly implement both the design static architecture (e.g. software decomposition into software elements such as packages, and classes or modules) and its dynamic architecture (e.g. specification of the software active object such as threads / tasks and processes)?	

25 Seiten von Checklisten

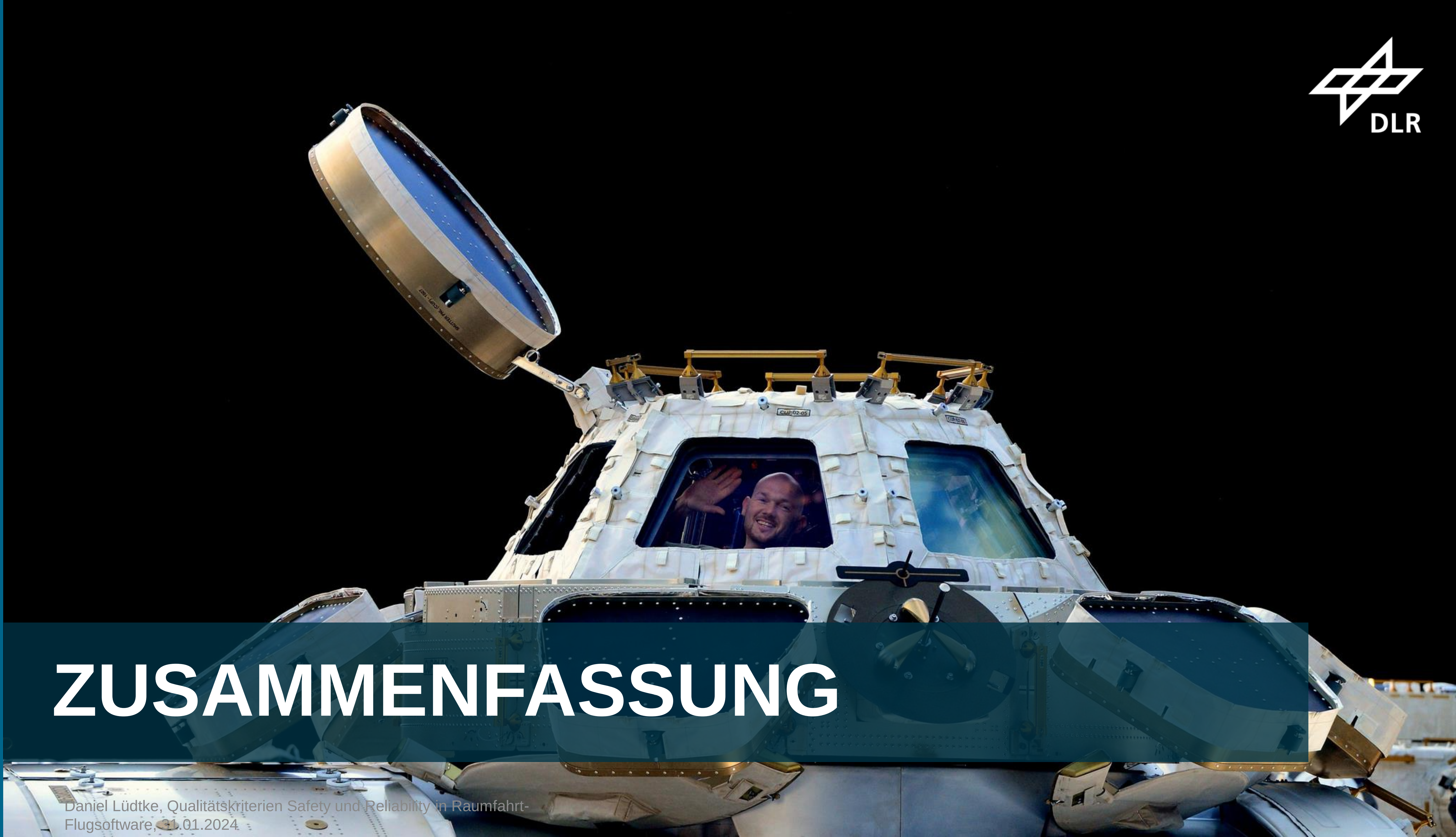
LESSONS LEARNED

Erfahrungen aus ca. 14-24 Jahren Flugsoftware im Forschungsumfeld

- Software-Entwickler ins Team integrieren! Enge Zusammenarbeit mit
  - Systems-Engineering
  - Assembly, Integration und Test
  - Operations

- Agile Software-Entwicklung ist auch im V-Modell möglich
- Continuous Integration/Delivery ist auch für sicherheitskritische eingebettete System hilfreich
- Reviews sind sehr wertvoll!
- Dokumentation ist extrem wichtig, muss aber nicht in Form von (digitalem) Papier sein

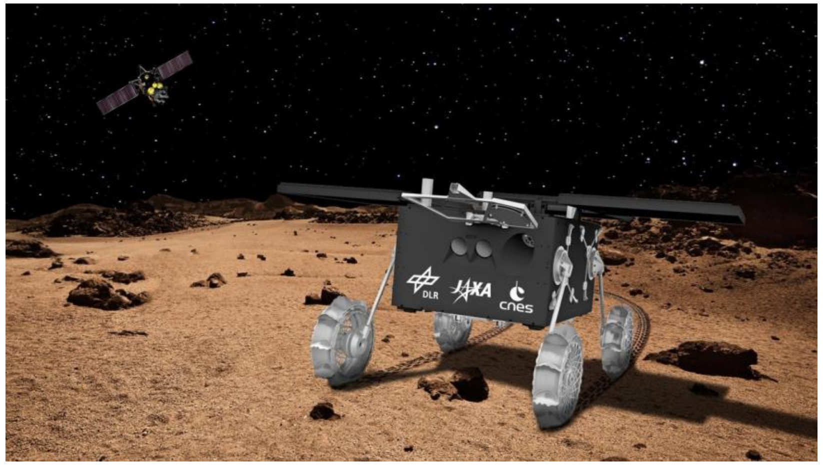
MAIUS-2-Team vor der Rakete, Credit: ZARM



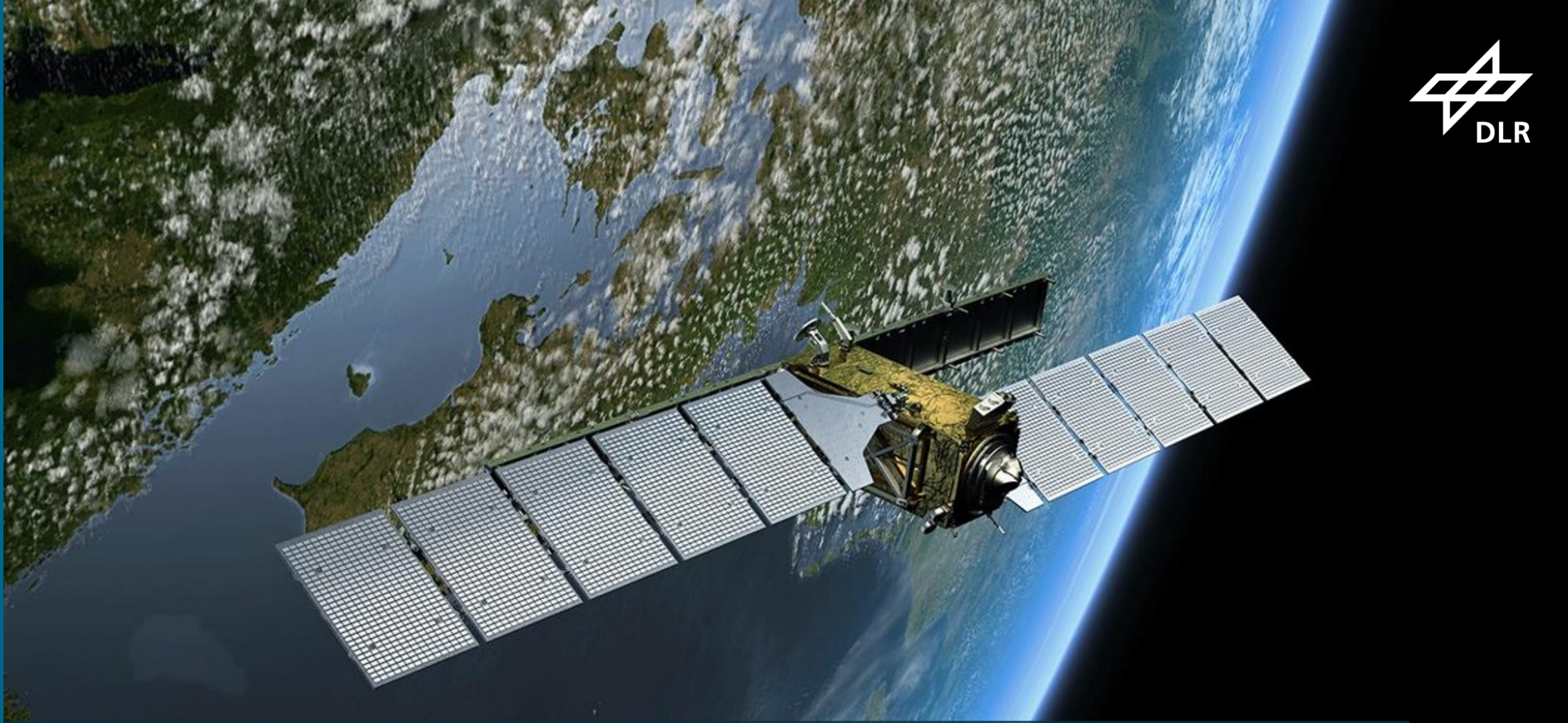
ZUSAMMENFASSUNG

Zusammenfassung

- Flugsoftware in der Raumfahrt hat besondere Anforderungen an Safety und Reliability
- Sichere Software lässt sich nur durch zuverlässige Software erreichen
- Standards sind vorhanden, müssen aber angepasst werden (Tailoring)
- Aspekte der agilen Software-Entwicklung sind auch in sicherheitskritischen Domänen möglich



© First Bose-Einstein condensation in space – Institut für Quantenoptik – Leibniz Universität Hannover (uni-hannover.de)



Daniel Lüdtke
Institute for Software Technology

daniel.luedtke@dlr.de
https://www.dlr.de/sc/

Impressum

Thema:	Qualitätskriterien Safety und Reliability in Raumfahrt-Flugsoftware
Datum:	2024-01-31
Autor:	Daniel Lüdtke
	mit Inhalten von Christan Prause, Zain Hammadeh, Peter Tomek, Andreas Lund
Institut:	DLR Institute für Softwaretechnologie
Bildcredits:	Alle Bilder „DLR (CC BY-NC-ND 3.0)“, sofern nicht anders angegeben