

# *Entwicklung der Forschungssoftware RCE*

*DLR Simulations- und Softwaretechnik (SC)*

*Abteilung Intelligente und Verteilte Systeme, Köln*

---

# RCE

Brigitte Boden  
Robert Mischke



Wissen für Morgen

A large, curved view of the Earth from space, showing the blue atmosphere, white clouds, and green and brown landmasses. The Earth is positioned in the lower right quadrant of the slide, curving upwards and to the left.

# Übersicht

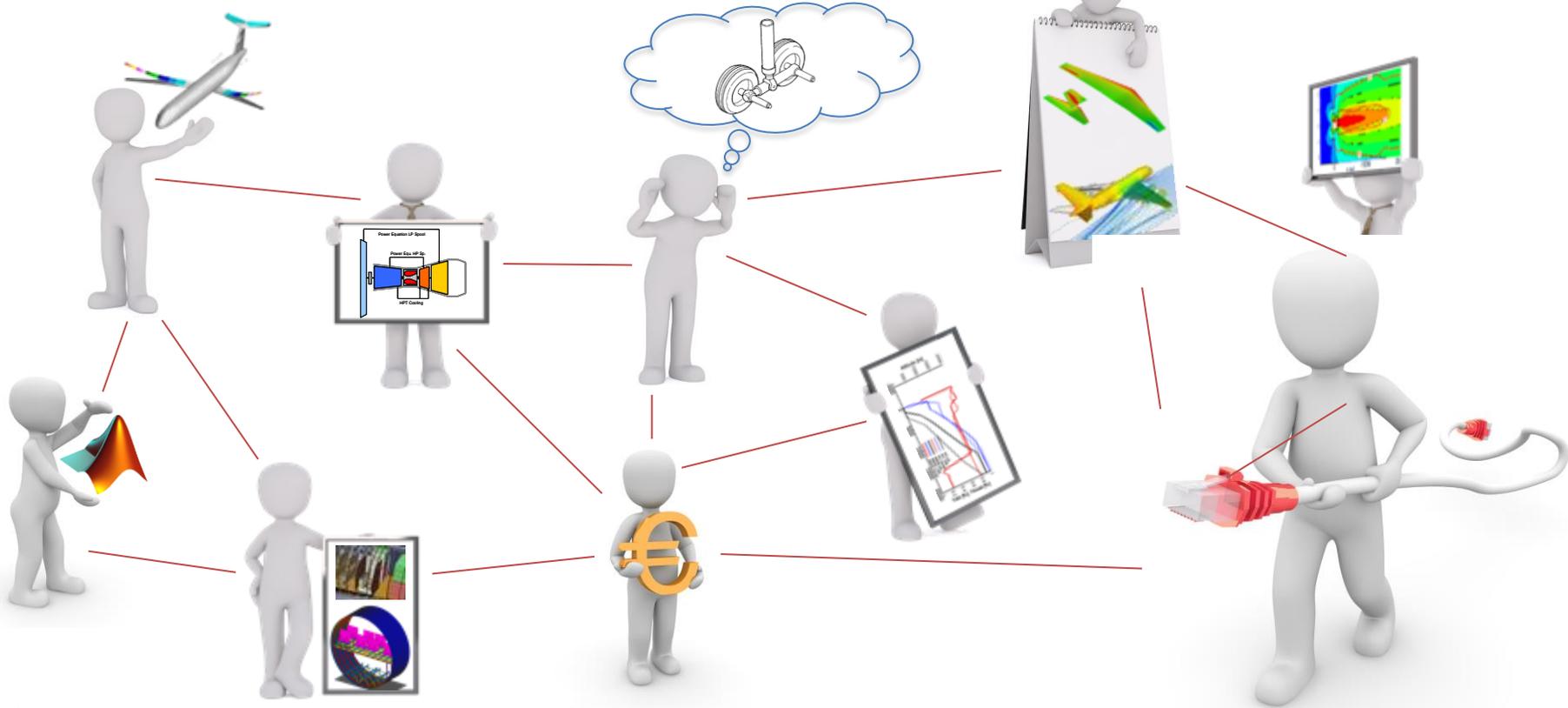
- Einführung RCE
- Forschungsprototyp vs. Produktivsoftware
- Finanzierung und Praxisprobleme
- Kompatibilität: pro und contra
- Feature-Breite vs. Projektmanagement



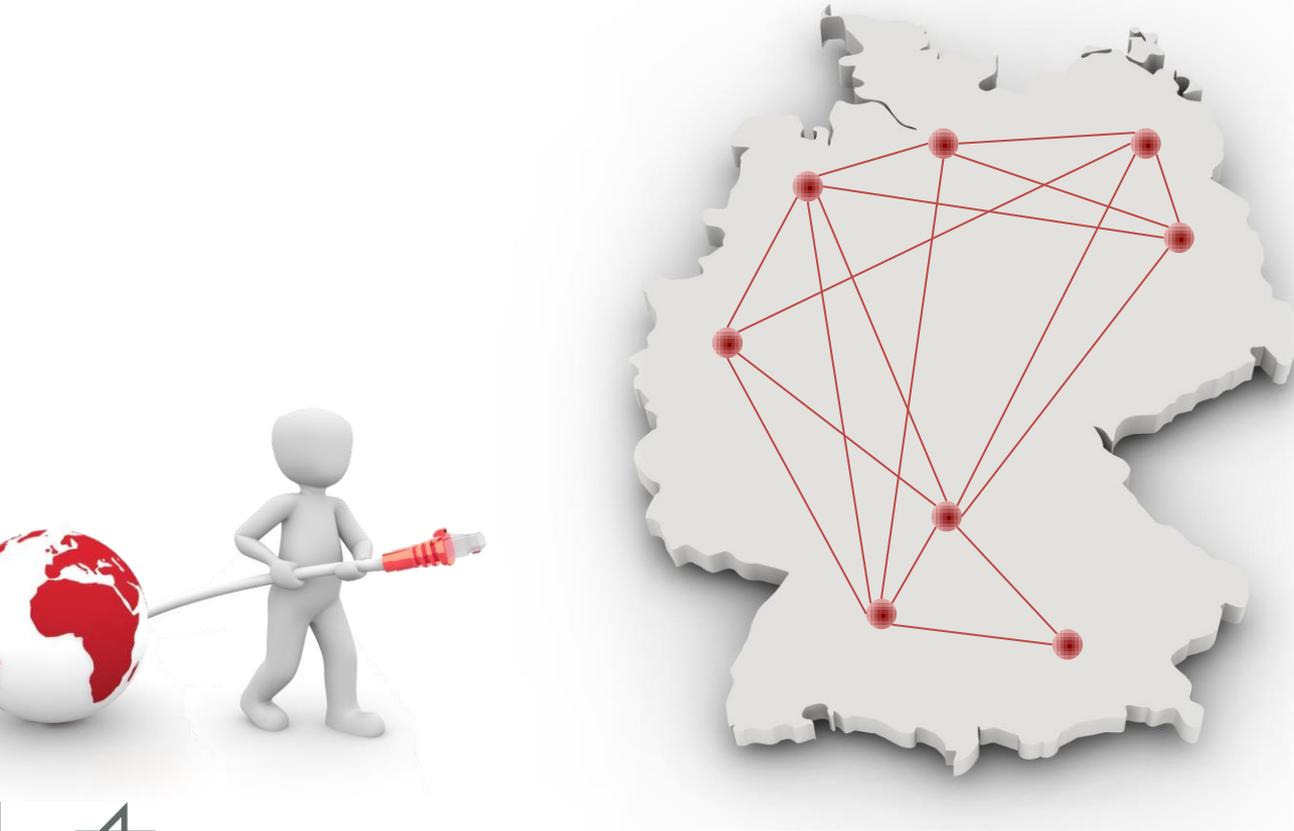
# Multidisziplinäre Entwurfswerkzeuge



# Kopplung multidisziplinärer Entwurfswerkzeuge



# *Kopplung verteilter, multidisziplinärer Entwurfswerkzeuge*



# Überblick RCE

## *Remote Component Environment*

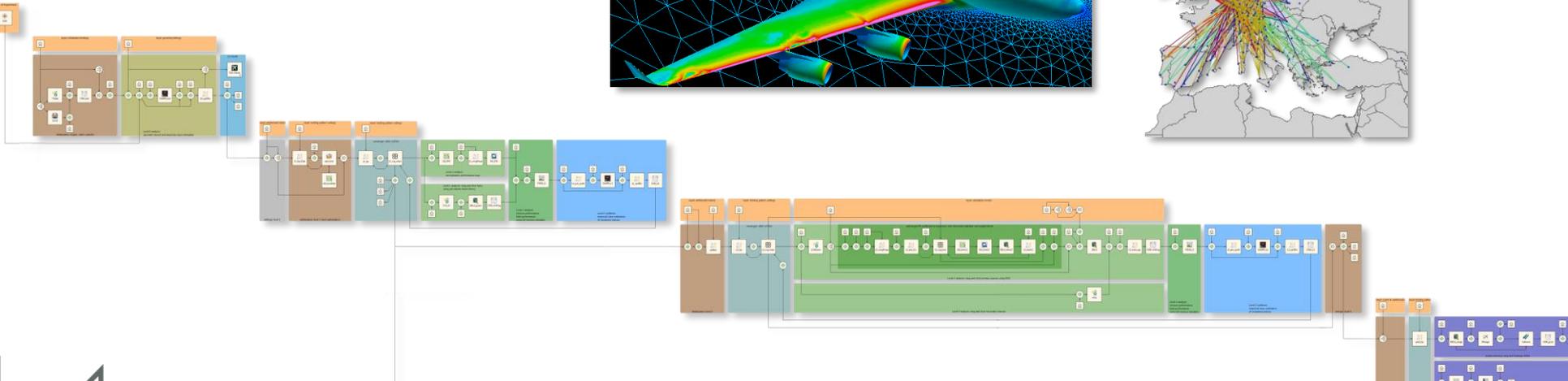
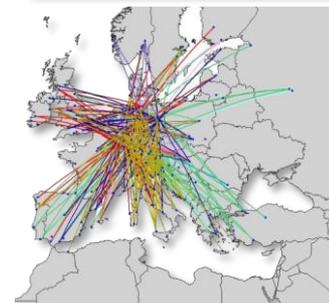
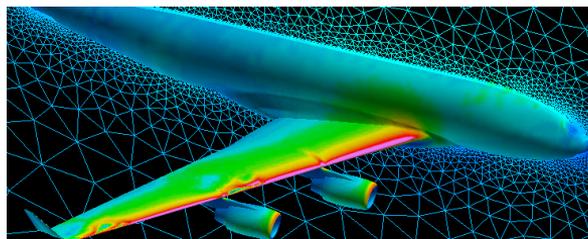
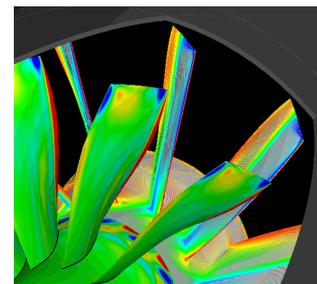
- Integrationsumgebung
- Verteilte, datengetriebene Simulationsworkflows
- Grafische Benutzerschnittstelle
- Batch-Modus
- Werkzeug-Integration
- Dezentrales Datenmanagement
- Workflow-Monitoring
- Remote Access (SSH)
- ...
  
- Java, Eclipse RCP
- Open Source



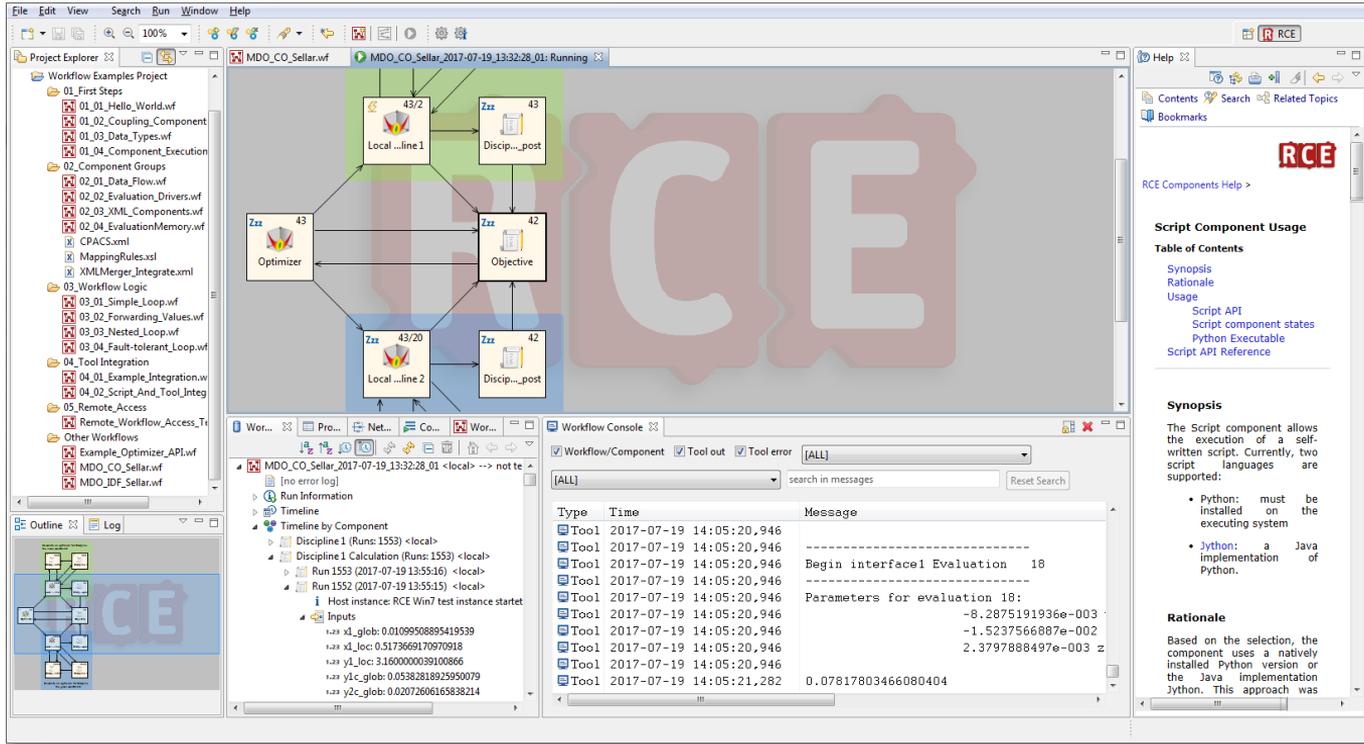
# Überblick RCE

## Einsatzbereiche von RCE - Beispiele

- Flugzeugentwurf
- Modellierung des Luftverkehrs in Deutschland
- Klimaoptimiertes Fliegen
- Design von Gas-Turbinen
- Schiffsentwurf



# Grafische Benutzungsschnittstelle



The screenshot displays the RCE graphical user interface (GUI) for workflow management. The main window shows a workflow diagram with components like 'Optimizer', 'Objective', and 'Discipline' connected by arrows. The interface includes a Project Explorer on the left, a Workflow Console at the bottom, and a Help panel on the right.

**Workflow Console Output:**

Type	Time	Message
Tool	2017-07-19 14:05:20,946	
Tool	2017-07-19 14:05:20,946	
Tool	2017-07-19 14:05:20,946	Begin interface1 Evaluation 18
Tool	2017-07-19 14:05:20,946	Parameters for evaluation 18:
Tool	2017-07-19 14:05:20,946	-8.2875191936e-003
Tool	2017-07-19 14:05:20,946	-1.5237566807e-002
Tool	2017-07-19 14:05:20,946	2.3797888497e-003 z
Tool	2017-07-19 14:05:21,282	0.07817803466080404

**Help Panel - Script Component Usage:**

**Synopsis**

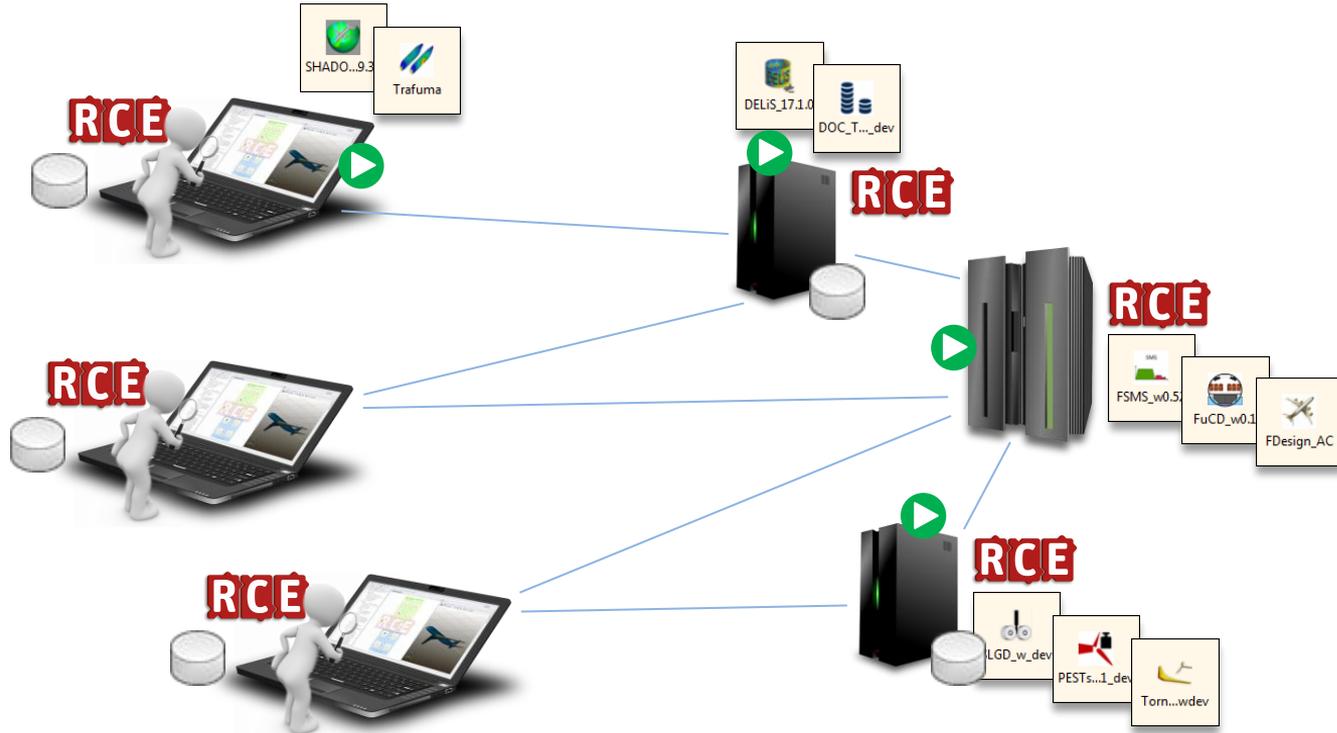
The Script component allows the execution of a self-written script. Currently, two script languages are supported:

- Python: must be installed on the executing system
- Jython: a Java implementation of Python.

**Rationale**

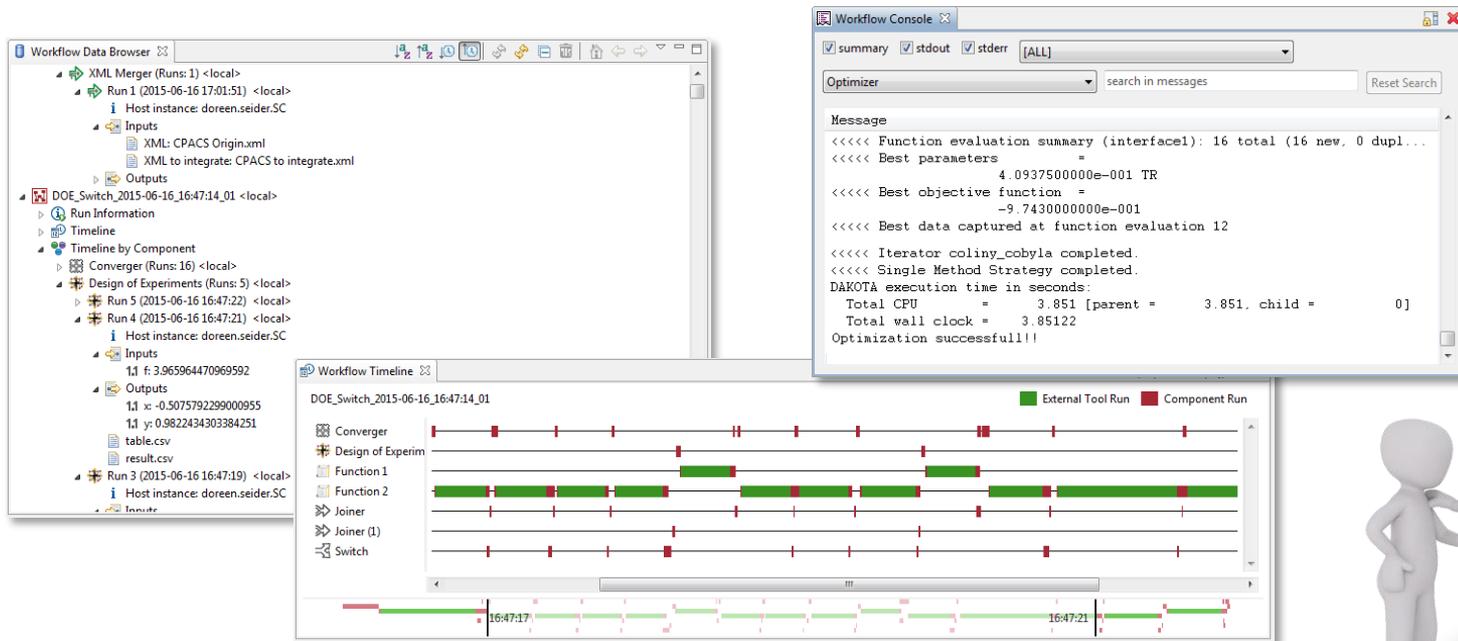
Based on the selection, the component uses a natively installed Python version or the Java implementation Jython. This approach was

# Mit RCE zusammenarbeiten



# Mit RCE zusammenarbeiten

## Workflowausführung gemeinsam monitoren



The image displays three windows from the RCE software interface:

- Workflow Data Browser:** Shows a hierarchical tree of workflow components. The selected component is 'DOE\_Switch\_2015-06-16\_16:47:14\_01', which includes a 'Converger' (16 runs), 'Design of Experiments' (5 runs), and two 'Function' runs (Run 4 and Run 3).
- Workflow Console:** Displays the output of an optimization process. The message indicates that the 'coliny\_cobyla' iterator completed successfully. Key statistics include:
  - Function evaluation summary (interface1): 16 total (16 new, 0 dupl...)
  - Best parameters = 4.0937500000e-001 TR
  - Best objective function = -9.7430000000e-001
  - Best data captured at function evaluation 12
  - Iteration coliny\_cobyla completed.
  - Single Method Strategy completed.
  - DAKOTA execution time in seconds:
    - Total CPU = 3.851 [parent = 3.851, child = 0]
    - Total wall clock = 3.85122
  - Optimization successful!!
- Workflow Timeline:** A Gantt chart showing the execution timeline of the workflow components. The timeline is divided into 'External Tool Run' (green bars) and 'Component Run' (red bars). The components shown include 'Converger', 'Design of Experim', 'Function 1', 'Function 2', 'Joiner', 'Joiner (1)', and 'Switch'. The timeline spans from approximately 16:47:17 to 16:47:21.



## *Zusammenarbeit im Entwickler-Team*

- Heterogenes Team
- Jedes Teammitglied hat eigene Themenschwerpunkte bei Entwicklung
- Keine festen Rollen wie Entwickler, Tester etc.
- Ein fester Ansprechpartner für jedes Forschungsprojekt
- Häufiges Betreuen von Studenten, Praktikanten...



## *Softwareentwicklung in einer wissenschaftlichen Einrichtung*

- Wissenschaftlicher Anteil vs. Produktivsoftware
- Eher wenige Gelegenheiten zu Publikationen
  - Teilweise Publikationen zu Projekten
  - Thema "Software publizierbar/zitierbar machen"
- Bewertung durch typische Kennzahlen schwierig



## *RCE als Open Source Software*

- RCE ist Open Source, bisher gibt es aber keine „Developer Community“ außerhalb des DLR

Herausforderungen auf dem Weg zu „offenerer“ Entwicklung:

- Wie funktioniert die Koordinierung verschiedener Features?
- Schaffung von Infrastrukturen nötig, z.B.
  - Plattform für Konzeptdiskussionen
  - Mailingliste, Forum etc. (teils erschwert durch IT-Richtlinien)
- Projekt bekannt machen/etablieren vs. Projekt-Breite handhabbar halten



## Interaktion mit Nutzern

- Benutzer-Workshops ca. alle 1-1,5 Jahre
  - Mischung aus Präsentationen und Gruppendiskussionen (Anforderungsanalyse, geplante Konzepte validieren)
  - Erfahrungsaustausch zwischen den Nutzern (Präsentationen)
  - Feedback an uns als Entwickler
- Bug-Reports, Vorschläge, Support-Anfragen von Nutzern
- Test-Snapshots für neue Features (oft für einzelne Projekte)
- Projekttreffen
- Hands-On-Workshops (Einführung in RCE)



# Forschungsprototyp vs. wissenschaftliche Produktivsoftware

- Forschungsprototyp:
  - Entwickler oft selbst die Anwender (oder im direkten Kontakt)
    - in der Regel wenige Personen
  - flexible Anpassung nach Bedarf; Ansätze „ausprobieren“
  - Software **ist** oft Forschung
  
- Wissenschaftliche Produktivsoftware:
  - Entwickler und Anwender sind oft im Kontakt, aber nicht unbedingt direkt
    - tendenziell größerer Kreis
  - mehr Fokus auf Robustheit, Rückwärtskompatibilität, Usability, Dokumentation, Administrierbarkeit, ...
  - Software **ermöglicht** oft Forschung



# Finanzierung wissenschaftlicher (OS-)Produktivsoftware

- Finanzierung über Forschungsprojekte
  - Verhandlungen mit potenziellen Projektpartnern
  - neue Features oder fachliche Unterstützung ↔ Finanzierungsumfang
- Projektziele oft offen/unscharf
  - nicht nur **wie** (normal in der Softwareentwicklung), sondern **was**; Interaktion mit Forschungsaktivitäten!
  - Lösung: agile Methoden?
    - *kann* funktionieren, aber auch kollidieren
    - Ressourcen und Roadmaps i.d.R. fixiert
  - Zielerreichung selten objektiv messbar / sanktionierbar
    - Korrektiv: der „gute Ruf“ als Projektpartner → Folgeprojekte!



# Finanzierung wissenschaftlicher Produktivsoftware: die Probleme

- Alles wunderbar?  
Mit Projektpartnern reden, neue Features finanziert bekommen, gute Arbeit abliefern?
- ...leider nicht ganz
- zentrales Problem: Produktivsoftware vs. Prototyp!
  - Maintenance (z.B. neue OS-Versionen, Library-Upgrades, Security, CI-Setup, ...)
  - Qualitätssicherung *bestehender* Features  
(Rückwärtskompatibilität, Konsistenz zu neuen Features, ...)
  - Support-Anfragen
  - größeres Team → mehr Overhead
  - mehr „Kathedrale“ als „Basar“



# *Finanzierung wissenschaftlicher Produktivsoftware: die Probleme*

- gleichzeitig immer weniger Neuerungsbedarf
  - reife Software oft „gut genug“!
  - ...aber woher kommt dann die Finanzierung?
- zum Teil klassisches Problem von Open-Source-Software
  - aber: typische OSS-Finanzierungsmodelle oft nicht möglich
  - gleichzeitig ist Open Source für Wissenschaft extrem sinnvoll!
    - Reproduzierbarkeit, Prüfbarkeit der Verfahren, ...
- wichtig: Projektplanung und Wartungsaufwand kommunizieren
  - User-Workshops, offener Issue-Tracker, ...



## *Kompatibilität: pro und contra*

- wichtig v.a. bei verteilter Software (Protokolle) und Dateiformaten
- pro:
  - Wissenschaftsanwender oft konservativ mit Updates
    - ...kooperieren aber über Institutionen hinweg
  - Kompatibilität macht Koordination in Projekten einfacher!
- contra:
  - sehr hoher Mehraufwand! (QA)
  - Einfügen neuer Features deutlich verzögert („breaking changes“)
    - auch Verbesserungen und teilweise Bugfixes
- Empfehlung: genau abwägen, wie viel Kompatibilität den Aufwand wert ist



## *Feature-Breite vs. Projektmanagement*

- wichtige Frage aus Projektmanagementsicht: welche Features einbauen/verfolgen?
- Problem der Projektfinanzierung: Anforderungen oft speziell
  - besonders bei vielen kleinen Projekten; teilweise <0,5 Stelle/Projekt
  - Gefahr: „Flickenteppich“ / „Zick-Zack-Kurs“
  - ideal: Projektpartner überzeugen, breit anwendbare Features zu finanzieren
- manche Features will jeder haben, aber sie passen in kein Projekt
- Priorisierung Projektwünsche ↔ Kernentwicklung



# *Feature-Breite vs. Projektmanagement*

- Features vs. Maintenance-Aufwand (v.a. Testen)
- kann man Features jemals wieder entfernen, und wenn ja, wann?
  - tatsächliche Nutzung oft schwer abschätzbar
  - was wird noch benutzt?
    - Telemetrie → Datenschutz



## Zusammenfassung

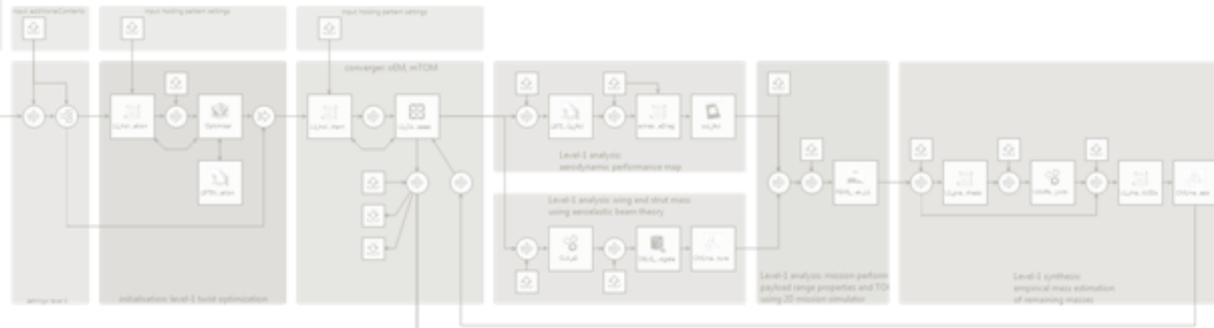
- Mit RCE werden Programme zu Workflows verbunden und verteilt ausgeführt
- Einsatz in Forschung und Industrie
- weitgehend über Forschungsprojekte finanziert
- Open Source
- Beteiligungen jeglicher Art willkommen

## Fazit/Emfehlungen

- Forschungsprototyp vs. Produktivsoftware → möglichst früh Ziel überlegen
- Feature-Strategie (Scope) überlegen und so gut wie möglich beibehalten
- Kompatibilität gut abwägen: wie viel wird wirklich gebraucht?
- Projektmanagement für Benutzer transparent machen



# Q&A



Download: <http://rcenvironment.de>

Twitter: <http://twitter.com/rcenvironment>

YouTube: <http://www.youtube.com/rcenvironment>

GitHub: <https://github.com/rcenvironment>

