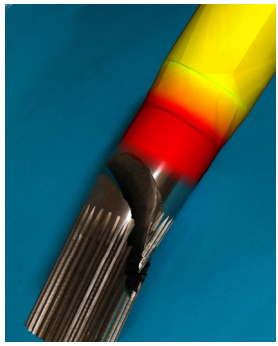
# Berechenbare Berechnungswerkzeuge



Langenbach, J.; Nagler, N.

*In Berechnungssoftware integrierte numerische Berechnungsmodelle stellen erhöhte Anforderungen an die Validierung des zugrundeliegenden Modells. Zum einen ist die Unkenntnis des späteren Anwenders zu berücksichtigen, zum anderen ist sicherzustellen, dass das Modell innerhalb der Grenzen verlässliche Ergebnisse liefert. Hierzu erfolgt mit Hilfe des Continuous Integration Konzeptes aus der Softwareentwicklung ein Lösungsvorschlag.*

*Numerical based simulation models, integrated in simulation software, make grate demands on the validation of the underlying simulation model. Thereby, on the one hand the user's abilities have to be considered. On the other hand it has to be ensured that the simulation model's results are reliable within the investigated limits. In order to achieve these goals, a solution, suggested here, consists of the Continuous Integration Concept which is widely-used in the software development.*

## 1 Einleitung

Numerische Berechnungsverfahren nehmen zuhnemend Ihren Platz in Regelwerken und Berechnungswerkzeugen ein. Mit der Änderung der Nutzung einhergehend, verändern sich auch die Anforderungen an die Modelle selbst. Stand bisher die manuelle Erstellung hoch spezialisierter Simulationsmodelle im Fordergrund, gewinnen automatisierte Modelle stark an Bedeutung. Die Attraktivität dieser Berechnungswerkzeuge liegt in der einfachen Bedienung für den Anwender und der gleichzeitig erwarteten Berechnungsgenauigkeit. Der Benutzer dieser Anwendungen erwartet, dass er ohne Kenntnisse der genutzten, komplexen Berechnungsverfahren ein Maschinenelement auslegen kann und dies jenseits der Grenzen der bekannten analytischen Verfahren. Damit einhergehend entstehen im Kontext des Maschienbaus neue Herausforderungen für die Ingenieure. Im Fall des Spezialmodells, ist der Ersteller gleichzeitig Anwender. Daher verfügt er zum einen über das entsprechende Fachwissen und zum anderen über die nötigen Kenntnisse über das Modell, um die Validierung der Ergebnisse selbst vorzunehmen. Der Nutzer des Berechnungswerk-

zeuges ist hierzu jedoch nicht in der Lage. Zum einen ist das zugrunde liegende Modell unbekannt, zum anderen sind die Fachkenntnisse über das Maschinenelement selbst und die genutzten Berechnungsverfahren in der Regel für eine Validierung der Ergebnisse nicht ausreichend. Der Nutzer muss also darauf vertrauen, dass der Entwickler den konkreten, berechneten Fall mit seinem Werkzeug sicher abdeckt oder die Software ihn deutlich sichtbar nicht unterstützt. Daraus folgt, dass der Entwickler das Modell detailliert validieren muss. Dabei muss nicht nur die gesamte Anwendungsbreite der Eingangsparameter geprüft, sondern auch die Konsistenz des Systems bei Änderungen sichergestellt werden. Nur so lassen sich Funktionsverluste durch Änderungen am Modell vermeiden.

## 2 Continuous Integration – Ein Ansatz aus der Softwareentwicklung schafft Abhilfe

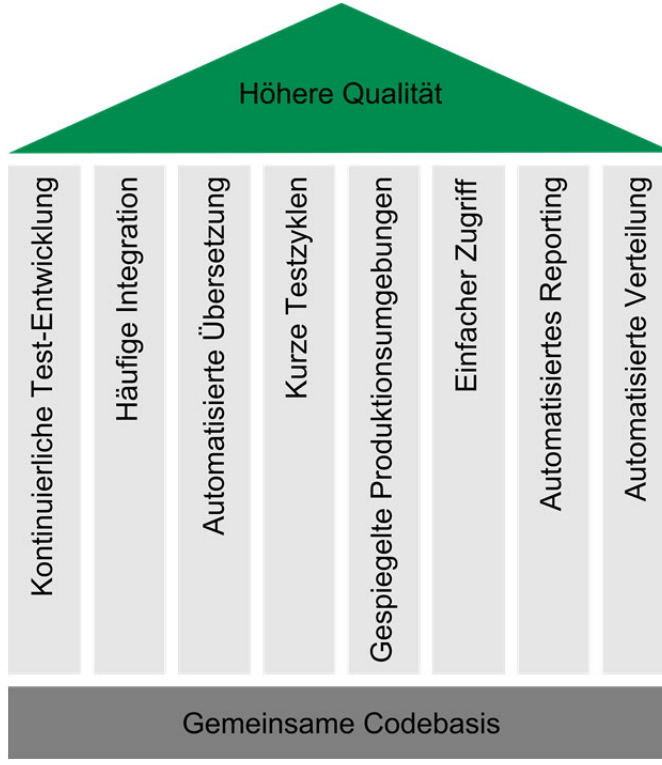



Abbildung 1: Prinzipien des Continuous Integration

In der Softwareentwicklung ist dieses Problem ebenfalls vorhanden und wird durch den Ansatz der Continuous Integration gelöst. Dieses Konzept besteht nach /1/ aus den in Abbildung 1 dargestellten Säulen, die eine erhöhte Codequalität zum Ziel haben. Die Basis für alle Maßnahmen bildet eine gemeinsame Codebasis. Dabei arbeiten alle Entwickler zwar mit einer lokalen Arbeitskopie des Quellcodes, pflegen jedoch die abgeschlossenen Änderungen in die gemeinsame Codebasis ein. Dies geschieht allerdings erst dann, wenn auch Tests für die geänderten Bestandteile vorliegen (1. Säule). Sobald die Tests vorhanden sind, erfolgt die Integration des geänderten Quellcodes in die gemeinsame Codebasis (2. Säule). Daraufhin werden die geänderten Teile übersetzt (Build-Vorgang, 3. Säule) und die Tests ausgeführt. Die geforderte, häufige Integration setzt allerdings eine kurze Laufzeit der ausgeführten Tests voraus (4. Säule). Damit die Tests schließlich unter realisitischen Bedingungen ablaufen, ist ein Testsystem anzustreben, welches dem System des Anwenders stark ähnelt (5. Säule). Eine weitere Forderung besteht im einfachen Zugriff (6. Säule) auf die Ergebnisse des Build-Vorganges sowie der Tests, damit alle interessierten Kreise Informationen erhalten können. Ebenfalls von großem Interesse ist ein gutes Berichtswesen, da nur so die Fehlerquellen identifiziert und Änderungen nachvollzogen werden können. Auch Qualitätskenngrößen lassen sich so ablesen und damit auch der Reifegrad der Software ermitteln (6. Säule). Dies ist zum Beispiel für eine automatische Verteilung (7. Säule) wichtig, da nur Software mit entsprechender Qualität freigegeben werden darf.

Dieser Ansatz kann auch im Falle der Entwicklung von Berechnungswerkzeugen, die numerische Simulationsmodelle enthalten, Anwendung finden. Hier stellen sich jedoch besondere Herausforderungen (s. Abbildung 2). Zunächst benötigt die Testentwicklung viel Erfahrung, da bei numerischen Simulationen die Rechenzeit schnell ansteigt. Da jedoch kurze Testlaufzeiten nötig sind, ist hier ein hohes Maß an Abstimmung notwendig. Ein weiteres Problem in diesem Zusammenhang stellt die Stabilität der Ergebnisse des Simulationslaufes dar. Wird beispielsweise bei FE-Berechnungen die Netzqualität verringert, um die Rechenzeit zu verkürzen, führt dies meist zu einer höheren Ergebnisstreuung. Daraus folgt eine sinkende Reproduzierbarkeit der Tests, da die Ergebnisse schwieriger zu vergleichen sind. Insbesondere vor dem Hintergrund, dass die Testauswertung automatisiert erfolgen soll. Eine weitere Herausforderung ist der Entwickler selbst. Während ein Softwareentwickler es gewöhnt ist, mit Versionskontrollsystemen und Testumgebungen zu arbeiten, ist dies für Maschinenbauer zunächst Neuland. Ähnlich wie bei der Einführung von Produktdatenmanagmentsystemen setzen auch diese Prozessänderungen eine entsprechende Unterstützung der Mitarbeiter und durch die Mitarbeiter voraus.


Abbildung 2: Herausforderungen zur Einführung von Continuous Integration für numerische Berechnungsmodelle

## 3 Ziele und erarbeitete Lösungskomponenten

Für ein Pilotprojekt wurde zunächst die in Abbildung 3 dargestellte Systemlandschaft als Ziel definiert. Die Grundlage bildet auch hier ein Versionskontrollsystem (VCS) als gemeinsame Codebasis. In diesem Fall kommt das System Subversion /2/ zum Einsatz. Die automatisierten Testläufe werden mit Hilfe von zwei Programmen umgesetzt. Zum einen die webbasierte Komponente Jenkins /3/ und zum anderen eine Eigenentwicklung, die im Folgenden als femtester bezeichnet wird. Jenkins hat die Aufgabe, den Buildprozess auf einem freien Testsystem zu starten, sobald neuer Code in das VCS eingecheckt wird. Hierzu bieten Subversion und Jenkins entsprechende Schnittstellen an. Darüber stößt Subversion nach einem Checkin automatisch den Testvorgang mit Hilfe von Jenkins an. Jenkins konfiguriert als erstes die Testumgebung, in dem es den aktuellen Stand aus dem VCS auf dem Testsystem auscheckt. Anschließend startet es den femtester, der die vorher definierten Testfälle ablaufen lässt und auswertet. Das Resultat der Tests meldet femtester daraufhin an das Dashboard. Dieses sammelt die Ergebnisse, bereitet sie grafisch auf und informiert interessierte Personen per E-Mail über die Resultate. Das Dashboard bildet

daher in diesem Szenario den zentralen Anlaufpunkt für das Thema Qualitätssicherung. Als Lösung dient in diesem Fall das Produkt CDash.

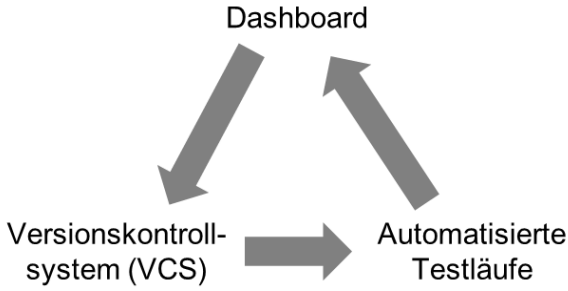



Abbildung 3: Komponenten des Pilotsystems

## 4 Funktionsweise und Aufbau des Programms femtester

Die Eigenentwicklung femtester ist für die Ausführung der Tests und deren Auswertung zuständig. Im Bereich der numerischen Berechnugswerkzeuge ergeben sich daraus verschiedene Anforderungen, die eine Eigenentwicklung notwendig machten. Zunächst müssen verschiedene Anwendungen von unterschiedlichen Herstellern nutzbar sein. Neben der FE-Software Ansys ist dies im Pilotprojekt auch die Software Simpack. Nach Beendigung eines Testes müssen zudem die Berechnungsergebnisse ausgewertet und auf Korrektheit geprüft werden. Dies ist ebenfalls Aufgabe des femtester-Programms. Um dies zu gewährleisten, basiert die Anwendung auf der Architektur aus Abbildung 4. Die Hauptanwendung besitzt zwei Schnittstellen. Die in Abbildung 4 links dargestellte dient zur Anbindung der verschiedenen Anwendungen, in denen die Tests ablaufen. Die zweite Schnittstelle ermöglicht die Auslagerung von Modulen für die automatische Ergebnisauswertung. Aktuell existiert ein Modul für die Auswertung von kommaseparierten Dateien (CSV) und ein weiteres Modul für die Anbindung von Ansys. Die Konfiguration der einzelnen Testfälle erfolgt in einer Konfigurationsdatei per Testfall.

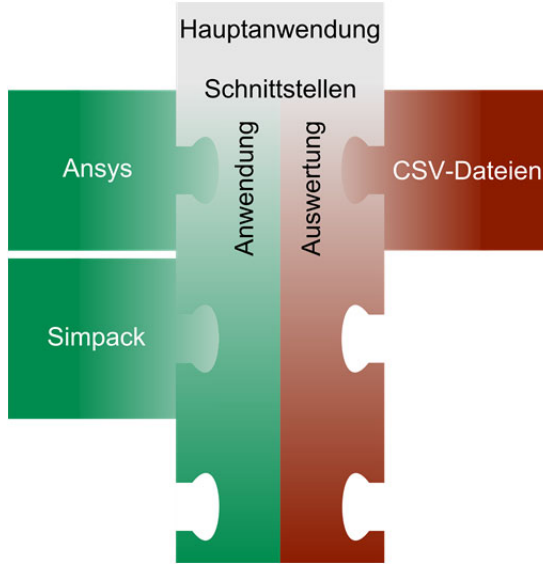



Abbildung 4: Architektur der Anwendung femtester, zur automatischen Testausführung und –auswertung

Jede Testkonfiguration wird dabei in einer Konfigurationsdatei im YAML-Format abgelegt. Das YAML-Format /4/ ist eine einfache Auszeichnungssprache, die in Textdateien genutzt werden kann. Das bedeutet, dass die Testkonfiguration in einem Texteditor erstellt und bearbeitet werden kann. In der Konfiguration ist zunächst hinterlegt, mit welcher Anwendung der Test ausgeführt wird. Außerdem ist dort das Dashboard angegeben, an welches femtester die Ergebnisse überträgt. Danach folgen spezielle Einstellungen, die das jeweilige Anwendungsmodul verarbeitet. Im Falle der Ansysintegration sind dies die verwendeten Ansys-Dateien mit Programmcode in der Ansys Parametric Design Language (APDL), für die Berechnung und optional für das Postprocessing. Ebenso muss der Pfad für die zu nutzende Ansysanwendung mit samt der zu verwendenden Lizenz angegeben werden. In einem zweiten Abschnitt der Konfigurationsdatei erfolgt die Angabe, welche Parameter für die Berechnung verfügbar sind und mit welchen Werten sie für den Berechnungslauf belegt werden. Hierbei ist die Eingabe von mehreren Testfällen mit unterschiedlichen Wertesätzen für die Parameter in einem Testfall möglich. Im letzten Abschnitt der Testkonfiguration erfolgt die Angabe des zu nutzenden Auswertemoduls mit entsprechenden Vorgaben. Im Falle der Auswertung von CSV-Dateien sind das Trennzeichen, eine absolute und relative maximale Abweichung sowie die Angabe einer Vergleichsdatei

nötig. Das Modul vergleicht nach Durchlauf eines Testfalles jeden Wert der Vorgabedatei mit dem Wert in der erzeugten Ergebnisdatei. Ist die Abweichung kleiner als die vorgegebene relative oder absolute Abweichung, ist der Vergleich erfolgreich. Andernfalls schlägt der Test fehl. Der Programmablauf folgt damit dem Prozess aus Abbildung 5.

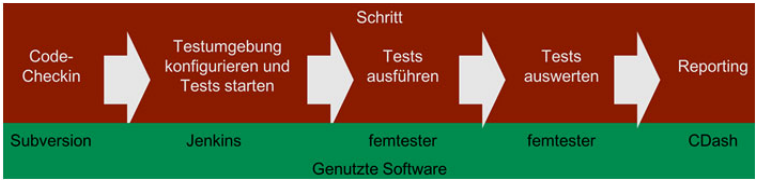



Abbildung 5: Ablauf der Continuous Integration im Pilotprojekt

## 5 Reporting

Die gesammelten Ergebnisse können im Dashboard abgerufen werden. Dort ist ein Test (entspricht einer YAML-Datei) pro Zeile aufgelistet. Der Name des Tests ist in der YAML-Datei hinterlegt und wird im Falle von Ansys um die verwendete Version erweitert. Ebenso ist angegeben, auf welchem Rechner und welchem Betriebssystem der Test erfolgte. Die in der YAML-Datei hinterlegten Testfälle finden sich schließlich in einer der Spalten im Bereich „Test“. Ist der Testfall fehlgeschlagen, erhöht sich die Anzahl in der Spalte „Fail“. War er erfolgreich, erhöht sich die Anzahl in der Spalte „Pass“. Über einen Klick auf die jeweilige Anzahl sind auch detaillierte Informationen zu den jeweiligen Testfällen abrufbar. So sind beispielsweise alle von Ansys erzeugten Hinweise, Warnungen und Fehlermeldungen dort hinterlegt. Ebenso finden sich an dieser Stelle Diagramme über die Testlaufzeit und die Historie der Testergebnisse. Durch erstere kann die Entwicklung der Rechenzeit nachvollzogen und durch letztere die Stabilität der Tests und des Berechnungsmodells nachvollzogen werden.

**Ansys**

| Site | Build Name | Update | Configure | | Build | | Test | | | Build Time |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Files | Error | Warn | Error | Warn | Not Run | Fail | Pass | |
| LANGENBACH | konfKontGauss130 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 10 hours ago |
| LANGENBACH | konfKontGauss145 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 14 hours ago |
| CIP-2009-25 | konfKontGauss145 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 hours ago |
| CIP-2009-26 | konfKontGauss145 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 hours ago |

Abbildung 6: Übersicht der Ergebnisse im Dashboard

## 6 Zusammenfassung und Ausblick

Im Rahmen des Pilotprojektes stand zunächst die Adaption des Continuous Integration Ansatzes für die Übertragung auf numerische Berechnungsmodelle im Fokus. Hierfür wurde ein Konzept erarbeitet, welches möglichst mit weit verbreiteten Standardanwendungen umsetzbar ist. Lediglich die eigentliche Testausführung und -auswertung ist mit vorhandenen Werkzeugen nicht realisierbar. Daher erfolgte die Neuentwicklung des Programmes femtester, das diese Aufgabe übernimmt. Das Programm besitzt eine Modulare Architektur, welche die Integration weiterer Berechnungswerkzeuge und Testauswertungsmethoden ermöglicht. Aktuell ist es mit diesem System möglich, Ansysmodelle zu testen, automatisiert auszuwerten und die Ergebnisse in einem Dashboard zu sammeln. Bei der Erstellung der Testfälle traten in der praktischen Anwendung die erwarteten Probleme auf. So fordert insbesondere die steigende Rechenzeit bei besserer Stabilität Optimierungsbedarf beim Testdesign. Im Pilotprojekt konnte der Zeitbedarf durch eine Berechnung mit gröberem Netz und geringeren Iterationszahlen letztendlich jedoch von etwa 20h auf 3h reduziert werden, ohne das die Ergebnisstabilität in mitleidenschaft gezogen wurde.

In Zukunft gilt es, Simpack als weiteres Berechnungswerkzeug zu integrieren und damit die Anwendungsbreite der femtester-Anwendung zu erweitern. Bei den anderen verwendeten Komponenten sind für die Integration von Simpack keine Änderungen notwendig.

## 7 Literatur

/1/ Webseite Kontinuierliche Integration: http://de.wikipedia.org/wiki/Kontinuierliche_Integration

/2/ Webseite Apache Subversion: http://subversion.apache.org

/3/ Webseite Jenkins: http://jenkins-ci.org

/4/ Webseite YAML: http://de.wikipedia.org/wiki/YAML