



Hallo und willkommen zur 14. Wissensspritze.

Mein Name ist Klaus Gieber und die heutige Wissensspritze, die erste in einer Reihe von Wissensspritzen rund um das Thema Softwareentwicklung in Teams, trägt den Titel „**Continuous Build über mehrere Programmiersprachen**“

Klaus Gieber hielt diesen Vortrag am 11.9.2009.

Die Aufnahme des Vortrags steht unter <http://wissensspritze.catalysts.cc> zur Verfügung.

Continuous Build?

Catalysts

- **Kontinuierliche Integration** (auch: fortlaufende oder permanente Integration, [en. Continuous Integration](#)) ist ein Begriff aus der [Software-Entwicklung](#), der den Prozess des regelmäßigen, vollständigen [Neubildens](#) und [Testens](#) einer [Anwendung](#) beschreibt. (aus wikipedia.de)

© www.catalysts.cc, 2009 Continuous Build über mehrere Programmiersprachen 2

Bei jedem Thema sollte man vorher erstmal die Definition abstecken, also schauen wir, was wikipedia zum Thema Continuous Build bzw. Continuous Integration sagt.

In vielen Software-Projekten ist ein funktionierender Continuous Build nicht mehr wegzudenken und wird von Anfang an eingerichtet, trotzdem gibt es nach wie vor Projekte, wo meistens nur ein Entwickler lokal auf seinem Rechner eine neue Version kompiliert, die dann ausgeliefert wird.

Das hat natürlich einige Nachteile, darum sehen wir uns einmal die Ziele eine Continuous Build's an:

Ziele

Catalysts

- Jeder Entwickler hat Zugriff auf ein zentrales Source-Code-Repository.
- Der Continuous Build wird entweder durch Repository-Commits automatisch angestoßen kann auf Knopfdruck ausgeführt werden und ist damit wiederholbar.
- Testfälle werden mit dem Build gemeinsam ausgeführt.
- Jeder Entwickler kann auch lokal auch seinem Rechner das gesamte System kompilieren, oder zumindest genau so viel, dass er ein lauffähiges System hat, mit dem er arbeiten kann.
- Lokale Builds benötigen oft andere Einstellungen als jener eines gemeinsamen Test-Systems oder des Produktions-Systems.

Ausgangssituation Catalysts

- Mehrere Mitarbeiter in verschiedenen Teams mit verschiedenen Rollen
 - Server-Entwickler
 - UI-Entwickler
 - Graphiker
 - Tester
- Mehrere IDEs
 - Eclipse, IntelliJ IDEA, Visual Studio, Flex Builder,
- Mehrere Programmiersprachen
 - Nicht alle Teammitglieder haben alle Build-Systeme zur Verfügung (z.B. Erstellung eines CAB-Files für Windows Mobile)

© www.catalysts.cc, 2009Continuous Build über mehrere Programmiersprachen4


In mittelgroßen bis großen Projekten werden die Teams immer größer und es gibt wenige bis gar keine Entwickler, die noch Detailwissen über den gesamten Quellcode haben. Bei Client-Server-Anwendungen wird das Team oft in Server- und Client-Entwickler getrennt, die natürlich sehr stark von einander abhängig sind.

Da zwischen Client und Server sehr oft nicht nur Netzwerk- sondern auch Technologiegrenzen liegen (z.B. eine andere Programmiersprache), kann die Kluft zwischen diesen beiden Entwicklergruppen sehr groß sein. Trotzdem müssen beide Teams in der Lage sein, auf eine lauffähige Version des Gesamtsystems zurückgreifen zu können.

Gibt es auch noch Test-Teams, die oft weniger von der Softwareentwicklung und den Technologien dahinter verstehen, so ist es noch wichtiger, dass das Gesamtsystem auf Knopfdruck neu kompiliert und deployed werden kann.

Natürlich gibt es dafür schon jede Menge Werkzeuge, die nur noch richtig eingesetzt werden müssen:

Werkzeuge



- Kommandozeilen-Compiler
 - javac, msbuild, groovyc, compc,...
- Wrapper über diese Tools
 - ANT, Maven
- Webbasierte Benutzeroberfläche
 - Quickbuild, Hudson, Teamcity,...
- IDE Unterstützung
 - eclipse2ant

© www.catalysts.cc, 2009

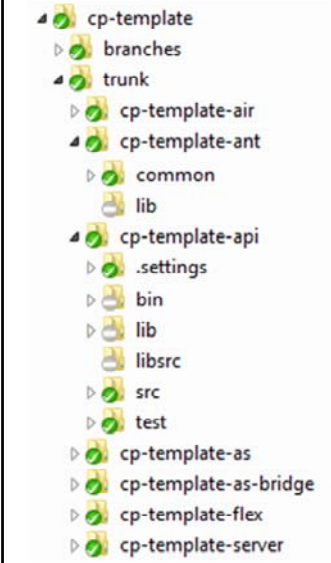
Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs

5

Jede Programmiersprache bringt meist ein Kommandozeilen-Tool mit, das Quellcode übersetzen kann.

Auf den weiteren Folien werden wir uns ansehen, wie wir bei Catalysts dafür gesorgt haben, dass wir alle unsere Projekte – viele davon aufbauend auf die Catalysts Components, auf Knopfdruck übersetzen können, alle Testfälle ausführen können, Statistiken und Dokumentationen genießen können; ganz unabhängig von der darunterliegenden Programmiersprache.

Mehrere Projekte Catalysts



- cp-template
 - ▷ branches
 - ▷ trunk
 - ▷ cp-template-air
 - ▷ cp-template-ant
 - ▷ common
 - lib
 - ▷ cp-template-api
 - ▷ .settings
 - ▷ bin
 - ▷ lib
 - ▷ libsrc
 - ▷ src
 - ▷ test
 - ▷ cp-template-as
 - ▷ cp-template-as-bridge
 - ▷ cp-template-flex
 - ▷ cp-template-server

- cp-template-ant
 - Build.xml
- Java-Projekte
 - cp-template-api
 - cp-template-server
- Flex/Air-Projekte
 - cp-template-as
 - cp-template-as-bridge
 - cp-template-air
 - cp-template-flex

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 6

Sehen wir uns die Projektstruktur eines typischen Projekts mit Client/Server-Architektur von Catalysts an:

Wir haben einen Server, der auf Java basiert und dazu einen Client in Flex (ActionScript). Dazu bietet der Server noch eine öffentliche Schnittstelle (ein API) an. Jedes Projekt ist gleich aufgebaut: es gibt IDE-Einstellungen, Bibliotheken mit deren Quellcode, den eigentlichen Quellcode (src) und Testfälle (test).

Für den Kompilervorgang gibt es dann ein eigenes Projekt (hier cp-template-ant), in dem sich ein „Master-Build-File“ befindet, das in der Lage ist, alle anderen Teilprojekte – unabhängig von der Programmiersprache – zu übersetzen, testen und deployen.

Ein großes Build-File Catalysts

- Bei uns: `cp-template-ant/common/common.xml`
- Kann verschiedene Programmiersprachen kompilieren
- Auch zuständig für
 - Tests
 - Reports
 - Dokumentationen
 - Statistiken
- Wird von den einzelnen Projekt-Build-Files referenziert

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 7

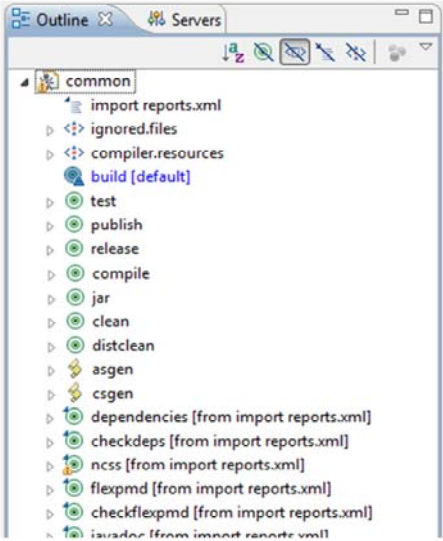
Bei uns heisst diese Datei `common.xml` und liegt in einem Unterverzeichnis von `cp-template-ant`, das wiederum über einen Repository-Link in dieses Projekt eingebunden ist; so können wir diese Datei in allen unseren Projekten verwenden.

Diese Datei muss nur ein einziges Mal erstellt werden und kann danach für alle Projekte eingesetzt werden. In dieser Datei ist ein einziges Mal definiert, wie Java kompiliert wird, wie in Actionscript Unit-Testfälle ausgeführt werden oder wie man in .NET die Quellcode-Dokumentation erzeugt.

Diese Datei weiss nichts von den eigentlichen Projekten, sondern wird lediglich von diesen referenziert.

common.xml als API Catalysts

- Ähnlich zu einem Interface im Quellcode
- Stellt alle wichtigen Funktionen rund ums Kompilieren zur Verfügung
- Wird von außen mit Parametern gefüttert und angestoßen



© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 8

Unser common.xml ist also nichts anderes als ein API mit vordefinierten Targets (das wären in der objektorientierten Programmierung die Methoden in einem Interface), die von außen aufgerufen werden können.

Wir stellen in dieser Datei neben dem Kompilervorgang und dem Ausführen von Testfällen auch verschiedene Targets für Architekturüberprüfungen, Statistiken oder Dokumentationen zur Verfügung; mehr dazu aber in den nächsten beiden Wissensspritzen.

Sehen wir uns einige Ausschnitte vom common.xml an:

common.xml (javac) Catalysts


```
<target name="compile" depends="resolve, version, init">
  <!-- java (jar/war) -->
  <contrib:if>
    <contrib:or>
      <contrib:and>
        <contrib:equals arg1="${module.kind}" arg2="jar"/>
        <contrib:equals arg1="${compile.jar}" arg2="true"/>
      </contrib:and>
      <contrib:and>
        <contrib:equals arg1="${module.kind}" arg2="war"/>
        <contrib:equals arg1="${compile.war}" arg2="true"/>
      </contrib:and>
    </contrib:or>
    <contrib:then>
      <!-- production -->
      <mkdir dir="${classes.dir}"/>
      <javac srcdir="${module.dir}/src" destdir="${classes.dir}" debu
        <compilerarg line="${compiler.args}"/>
        <classpath refid="compile.classpath"/>
        <patternset refid="ignored.files"/>
      </javac>
    </contrib:then>
  </contrib:if>
</target>
```

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 9

Hier sehen wir z.B. das target ‚compile‘:

Wird von einem Projekt dieses Target aufgerufen, so wird zuerst einmal überprüft, ob es sich bei diesem Projekt um ein Java-Projekt handelt. Wenn ja, wird mittels ANT der Java-Compiler javac aufgerufen.

common.xml (msbuild)



```
<!-- dotnet (dll, exe) -->
<contrib:if>
  <contrib:and>
    <contrib:or>
      <contrib:and>
        <contrib:equals arg1="$(module.kind)" arg2="dll"/>
        <contrib:equals arg1="$(compile.dll)" arg2="true"/>
      </contrib:and>
      <contrib:and>
        <contrib:equals arg1="$(module.kind)" arg2="exe"/>
        <contrib:equals arg1="$(compile.exe)" arg2="true"/>
      </contrib:and>
    </contrib:or>
  </contrib:and>
  <contrib:then>
    <contrib:if>
      <available file="$(module.dir)/$(module.name).csproj"/>
      <contrib:then>
        <dn:msbuild buildfile="$(module.dir)/$(module.name).csproj">
          <property name="Configuration" value="Release"/>
        </dn:msbuild>
      </contrib:then>
    </contrib:if>
  </contrib:then>
</contrib:if>
```

© www.catalysts.cc, 2009

Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs

10

Handelt es sich jedoch um ein .NET-Projekt (DLL oder EXE), so rufen wir msbuild auf, dem wir lediglich das VS-Projektfile mitgeben müssen.

Genauso verfahren wir dann für ActionScript, Flex, AIR, Outlook-Plugin oder Windows-Mobile-Projekte. Für alle Projekttypen haben wir ein einziges Mal festgelegt, wie dieser Code kompiliert werden muss.

Natürlich setzt das voraus, dass die Projektstruktur überall gleich ist; das ist aber aufgrund der viel besseren Wartbarkeit ohnehin wärmstens zu empfehlen, da sich so Entwickler von anderen Teams schneller in diesem Quellcode zurecht finden können.

common.xml (junit) Catalysts

```
<target name="test">
  <!-- java (jar, war) -->
  <contrib:if>
    <contrib:and>
      <contrib:equals arg1="${module.kind}" arg2="jar"/>
      <contrib:equals arg1="${compile.jar}" arg2="true"/>
    </contrib:and>
    <contrib:then>
      <contrib:if>
        <contrib:equals arg1="${cobertura.instrument}" arg2="true"/>
        <contrib:then>
          <cobertura-instrument todir="${testclasses.dir}" datafile="${output.dir}/cobertura.ser">
            <fileset dir="${classes.dir}">
              <include name="**/*.class"/>
            </fileset>
          </cobertura-instrument>
        </contrib:then>
      </contrib:if>
      <junit showoutput="false" printsummary="yes" haltonfailure="no" fork="true" dir="${module.dir}">
        <sysproperty key="net.sourceforge.cobertura.datafile" file="${output.dir}/cobertura.ser"/>
        <batchtest fork="yes">
          <fileset dir="${module.dir}/test">
            <include name="**/Test*.java"/>
          </fileset>
        </batchtest>
      </junit>
    </contrib:then>
  </contrib:if>
</target>
```

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 11

Sehen wir uns noch ein zweites Target an, nämlich ‚test‘.

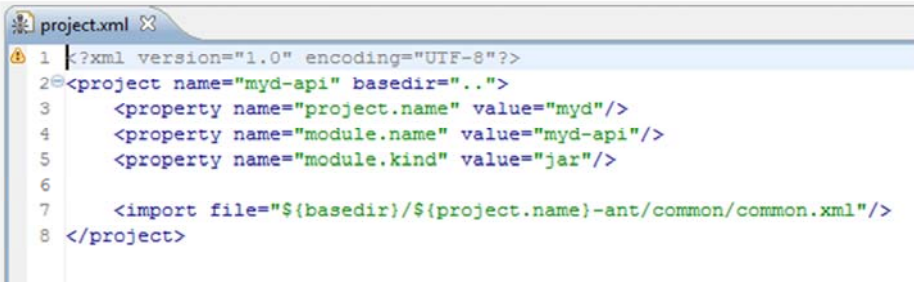
Hier wird wiederum überprüft, ob es sich um ein Java-Projekt handelt; falls ja, wird optional zuerst der Code durch Cobertura instrumentiert, um die Testfallabdeckung auf Sourcecode-Ebene zu überprüfen, und danach wird junit ausgeführt.

Analog dazu rufen wir flexunit für Flex/Actionscript und Nunit für .NET auf.

project.xml

Catalysts

- Liegen in jedem Projekt
- Geben Typ (Java, C#, AS,...) und Name an.
- Beispiel für myd-api:



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project name="myd-api" basedir=".">
3   <property name="project.name" value="myd"/>
4   <property name="module.name" value="myd-api"/>
5   <property name="module.kind" value="jar"/>
6
7   <import file="${basedir}/${project.name}-ant/common/common.xml"/>
8 </project>
```

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 12

Nun, da wir so etwas wie ein API zum Kompilieren und Ausführen unserer Tests haben, müssen wir dieses API nur noch aufrufen.

Dafür gibt es nun in jedem Projekt eine Datei mit dem Namen project.xml. Diese Datei ist von der Struktur her ein gültiges ANT-Buildscript und referenziert unser common.xml. Zusätzlich gibt es noch seinen Namen und den Typ an (also ob es sich hier um ein Java, C# oder Actionscript-Projekt handelt).

Nachdem wir das common.xml importieren, können wir dort nun alle Targets aufrufen, also auch unsere vorher definierten Targets ‚build‘ und ‚test‘. Der Entwickler dieses Projekts muss nun nicht mehr selbst die mühsamen javac und mxmhc Tasks schreiben, sondern verwendet einfach das fertige Skript, in das er nur mehr seine Parameter einsetzen muss.

Erstellen der Projektliste Catalysts

- Mit Hilfe von Ivy
 - Mehr Details zu Ivy in der übernächsten Wissensspritze
- Scant alle Verzeichnisse, sucht nach project.xml Dateien und erkennt dabei auch automatisch die Abhängigkeiten der Projekte

```
<target name="buildlist" depends="install">
  <ivy:buildlist reference="build-path">
    <fileset dir="${basedir}" includes="*/project*.xml"/>
  </ivy:buildlist>
</target>
```

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 13

Wir haben nun also mehrere project.xml-Dateien, die uns darüber Auskunft geben, in welcher Form dieses Teilprojekt kompiliert werden soll.

Unsere Projekte haben aber natürlich Abhängigkeiten untereinander, und diese Abhängigkeiten werden mit Apache Ivy beschrieben. Mehr zum Thema Ivy in 2 Wochen; jetzt sei nur so viel gesagt, dass es damit ähnlich wie mit Maven möglich ist, Abhängigkeiten zwischen Projekten zu beschreiben und Ivy ist dann in der Lage, daraus eine Reihenfolge zu bestimmen, in der die Projekte kompiliert werden müssen.

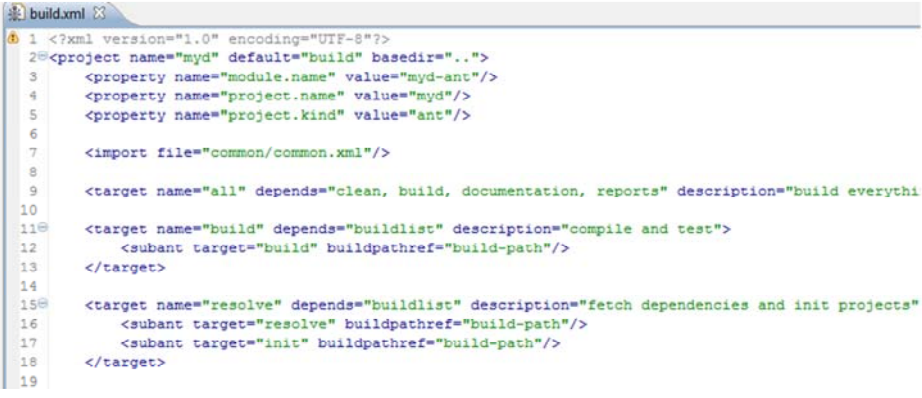
Dafür gibt es bei Ivy einen Befehl mit dem Namen buildlist. Diesem Befehl geben wir alle unsere project.xml Dateien mit, und Ivy erstellt dadurch automatisch die Reihenfolge, in der die Projekte kompiliert werden müssen.

Jetzt fehlt uns nur mehr ein build-File, das wir nehmen können um den Build für alle Projekte anstoßen zu können.

Bootstrapper

Catalysts

- myd-ant/build.xml
 - „Bootstrapper“ für alle Builds
 - Ruft zuerst buildlist und dann z.B. build auf
 - Mit Hilfe des ANT-Befehls subant



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project name="myd" default="build" basedir="..">
3   <property name="module.name" value="myd-ant"/>
4   <property name="project.name" value="myd"/>
5   <property name="project.kind" value="ant"/>
6
7   <import file="common/common.xml"/>
8
9   <target name="all" depends="clean, build, documentation, reports" description="build everythi
10
11 <target name="build" depends="buildlist" description="compile and test">
12   <subant target="build" buildpathref="build-path"/>
13 </target>
14
15 <target name="resolve" depends="buildlist" description="fetch dependencies and init projects">
16   <subant target="resolve" buildpathref="build-path"/>
17   <subant target="init" buildpathref="build-path"/>
18 </target>
19
```

© www.catalysts.cc, 2009

Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs

14

Dafür haben wir im ANT-Projekt ein sehr einfaches build.xml, das nur mehr unser common.xml referenziert und in dem mittels subant targets die build targets von unseren Projekten ausgeführt werden.

Die Targets in dieser Datei können wir dann z.B. von der IDE aus aufrufen.

Nun soll ja der Continuous Build in erster Linie nicht in der IDE, sondern auf einem zentralen Build-Server durchgeführt werden. Dafür gibt es wie oben schon erwähnt jede Menge Software, wie z.B. Quickbuild, Hudson, CruiseControl oder Teamcity.

Wir bei Catalysts verwenden Teamcity und auf den nächsten Folien möchte ich einige Screenshots zeigen, wie wir Teamcity konfiguriert haben.

The screenshot displays the Teamcity Build-Server interface. At the top, it says "Build-Server (Teamcity)" and "Catalysts". Below this, there is a list of build configurations, each with a "Run" button. The configurations are:

Build Configuration	Build Number	Status	Tests Passed	Artifacts	Changes	Build Time
tm-kernel	#1.6.86.15177	Success	393	No artifacts	No changes	09 Sep 09 10:21 (2m:45s)
tm-mobile	#1.6.86.15177	Success	-	Artifacts	No changes	09 Sep 09 10:27 (30s)
tm-mvc	#1.6.21.15177	Success	-	No artifacts	No changes	09 Sep 09 10:24 (36s)
tm-outlook	#1.6.103.15177	Success	-	Artifacts	No changes	09 Sep 09 10:28 (37s)
tm-plugin	#1.6.76.15177	Success	55	No artifacts	No changes	09 Sep 09 10:20 (40s)
tm-server	#1.6.125.15178	Success	27	Artifacts	No changes	09 Sep 09 11:08 (35s)

At the bottom of the interface, there is a footer with the text: "© www.catalysts.cc, 2009", "Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs", and the page number "15".

Nachdem die gesamte „Logik,, wie der Build laufen soll, in unseren ANT-Skripts verpackt ist, muss man am Buildserver nicht mehr viel einstellen. Wir empfehlen auch sehr stark, so wenig Konfiguration wie möglich in den Build-Server zu stecken, vor allem deswegen, weil Änderungen dort oft nicht mehr nachvollziehbar bzw. rückgängig gemacht werden können; ganz im Gegenteil zu einem build-Script, das unter Versionsverwaltung liegt.

Außerdem verhindert jede Einstellung, die nur am Buildserver liegt, dass ein adäquater Build auch lokal erstellt werden kann – dazu aber später.

Wir sehen hier einen Ausschnitt von Teamcity, hier jene Instanz, die wir zur Entwicklung von Taskmind verwenden.

Projekteinstellung tm-kernel Catalysts

Administration > Taskmind Build Project > tm-kernel Configuration

Version Control Settings

VCS root			Checkout rules
(svn) tm-ant	edit	detach	edit checkout rules (1)
(svn) tm-kernel	edit	detach	edit checkout rules (1)

Build Runner

Build runner:
Runner for Ant build.xml files

Path to a build.xml file:
Specified path should be relative to the checkout directory.

Targets:
Enter targets separated by space or comma.

© www.catalysts.cc, 2009 Continuous Build über mehrere Programmiersprachen 16

Für jedes Projekt muss man nun nur noch definieren, welche Teile des Repositories verwendet werden sollen. Teamcity checkt dann genau diese Teile aus und wir müssen jetzt nur noch auf das build.xml in myd-ant verweisen und dort das target ‚build‘ und ‚test‘ aufrufen. Das schaut nun für jedes Projekt völlig identisch aus.

Lokale Einstellungen Catalysts

myd-ant [mydesire/trunk/myd-ant]

- common
- lib
- .project 8744 02.12.08 21:54 kgieber
- air-config.template.xml 14956 26.08.09 14:10 hradi
- build.xml 15140 08.09.09 08:49 kgieber
- classycle.ddf 13619 23.06.09 16:42 hradi
- custom.properties
- default.properties 15135 08.09.09 00:00 hradi

custom.properties ist nicht unter Versions-Kontrolle, beide Dateien werden im build.xml geladen.

```

<property file="${common.dir}/custom.properties"/>
<property file="${common.dir}/default.properties"/>

```

In custom.properties können dann bestimmte Default-Einstellungen überschrieben werden.

```

2
3module.configurations:
4
5compile.dll=true
6compile.exe=true
7
8compile.jar=true
9compile.war=true
10
11compile.swc=true

```

```

1flex.home=C:/catalysts/f
2flex.tasks=ant/antlib/fl
3
4compile.dll=false
5compile.exe=false
6compile.msi=false
7compile.cab=false
8
9unzip.aslibs=true
10

```

© www.catalysts.cc, 2009
Continuous Build über mehrere Programmiersprachen
17

Zu Beginn hab ich erwähnt, dass es in größeren Teams sein kann, dass nicht jeder Entwickler das gesamte System übersetzen kann, weil ihm die dafür notwendigen Tools fehlen. Daher ist es von Vorteil, wenn man die Default-Einstellungen, die der Build-Server verwendet, lokal überschreiben kann.

Wir haben dafür in unserem ANT-Projekt eine Datei Namens default.properties. Dort werden unserem Build-Skript alle Parameter von außen mitgegeben, z.B. welche Module kompiliert werden sollen, wie das Projekt heisst usw.


Dann gibt es daneben noch eine zweite Datei mit dem Namen custom.properties. Diese Datei ist nicht unter Versionskontrolle und kann so von jedem Entwickler individuell angepasst werden.

Im build.xml werden beide Dateien geladen, das custom.properties zuerst.

Weiter oben beim common.xml haben wir z.B. den Parameter compile.dll gesehen, der abgefragt wird, wenn ein .NET-Projekt kompiliert werden soll. Wenn ich diesen Parameter nun bei mir auf false setze, so werden sämtliche .NET Projekte bei mir einfach nicht kompiliert.

Analog dazu kann man z.B. die Instrumentierung des Quellcodes durch Cobertura lokal ausschalten.

Zusammenfassung



- Build-Prozess sollte für alle Entwickler ersichtlich und lokal nachvollziehbar sein.
- Verwendung derselben Konzepte beim Erstellen von build-Files wie auch beim Erstellen von Code
 - API
 - Library
- Verwenden von bestehenden Tools
 - ANT, Maven
 - Teamcity, CruiseControl, Hudson, Luntbuild

© www.catalysts.cc, 2009 Continuous Build über mehrere Programmiersprachen 18

Hiermit sind wir schon so gut wie am Ende mit der heutigen Wissensspritze, abschließend noch eine kurze Zusammenfassung des heutigen Themas ...

Ausblick **Catalysts**

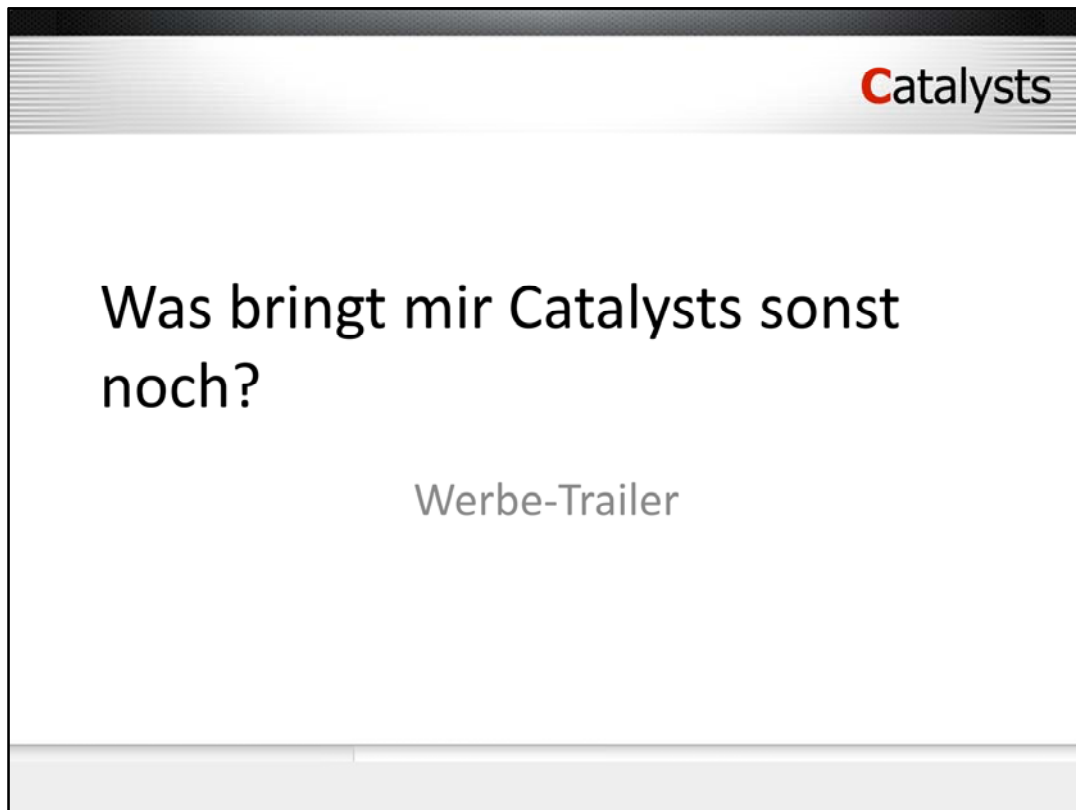
15. Wissensspritze: Automatische Prüfungen im Build-Prozess

16. Wissensspritze: Dependency Management mit Ivy

© www.catalysts.cc, 2009 Gemeinsame Softwareentwicklung im Team trotz unterschiedlicher IDEs 19

Weiter geht es in zwei Wochen mit dem Thema „Automatische Prüfungen im Build-Prozess“, wiederum zwei Wochen später gehen wir näher auf das Dependency Management mit Ivy ein.

Vielen Dank und bis zum nächsten Mal!



Effizient-Katalysator fürs Team

Catalysts

Nur perfekt funktionierende Teams arbeiten
hocheffizient.



hilft Teams bei der Planung und Organisation.

Registrieren Sie sich gratis unter
<http://www.taskmind.net>

Gesundheitscheck für Projekte



- „Wenn ich vorher gewusst hätte, dass all die Probleme auftreten, dann wären wir das Projekt anders angegangen...“
- Sind Ihnen zu Projektbeginn alle technischen und organisatorischen Risiken bewusst?
- Oder gibt's auch bei Ihnen immer wieder viel Unvorhergesehenes?
- Fordern Sie heute noch ein Experten-Team von Catalysts für eine Vorsorgeuntersuchung für Ihr Projekt an. 400 Euro, die sich auszahlen!

Wir setzen Ihre Ideen um

Catalysts

- Sie wissen was – wir wissen wie
- Mit gewohnter Catalysts-Qualität
- Zu vernünftigen Preisen
- Schnell
 - erster Prototyp nach wenigen Tagen
 - Wochenweise mehr Funktionalität
- Probemonat – Ausprobieren und nichts riskieren!

- Wir entwickeln **Software nach Ihren Bedürfnissen**
 - für den Büroarbeitsplatz,
 - für unterwegs am Notebook,
 - für Ihr Handy.
- Wir entwickeln **Software auf agile Art.**
- Dadurch gibt's
 - frühzeitige und regelmäßige Auslieferung,
 - rasches Feedback und
 - ausschließlich wertvolle Funktionen im Produkt, keine Schnörkel.

Karin S. über Catalysts



Karin S. (Geschäftsführerin eines kleinen Dienstleistungsunternehmens, Nicht-IT)

“Ich leite ein kleines Dienstleistungsunternehmen (16 Mitarbeiter). Wir brauchen zuverlässige Simulationssoftware nach Maß, um unsere Aufträge schneller abwickeln zu können. Wir haben selbst keine Software-Entwickler. Ich kenne mich mit Software nicht aus, verwende sie nur. Über unseren bisherigen Softwarelieferanten ärgere ich mich, weil er für jede kleine Änderung Länge mal Breite verrechnet.”

Erfahrungen von Karin S. mit Catalysts:

- [Günstiger](#) als andere Firmen in OÖ
- „Ausprobieren und nichts riskieren!“ ([Probemonat](#))
- [Wertvolles zuerst, Unwichtiges später, Schnörkel gar nicht](#)
- [Papier-Prototyp nach einer Woche, erste Version nach einem Monat](#)
- [Monatliche Auslieferung](#) und monatliche Kurz-[Retrospektiven](#)
- [Änderungen gratis](#)
- [Von Profis geführt, kritische Denker](#)

Hans M. über Catalysts



Hans M. (Abteilungsleiter IT in einer größeren Firma)

“Ich leite die Softwareentwicklung (25 Entwickler) in einer größeren Firma. Meine Leute sind mit der Wartung der alten Programme schon ausgelastet. Sie kommen bei den neuen Technologien allerdings nicht mehr mit. Aus den Fachabteilungen kommen immer mehr Anforderungen, die wir nicht erfüllen können. Wir suchen ständig nach guten Software-Entwicklern, finden die aber nicht.”

Erfahrungen von Hans M. mit Catalysts:

- [Misch-Stundensatz](#) unter unseren internen Stundensätzen
- [Kreativere und bessere Lösungen](#)
- [Scrum, XP, TDD, laufende Integration, Personas, User Stories](#)
- [Stabile Technologie-Plattform](#)
- Modulare und zyklenfreie Architektur, [ausführlich getestet](#)
- [Unternehmensberatung inbegriffen](#)
- Exzellentes Team