

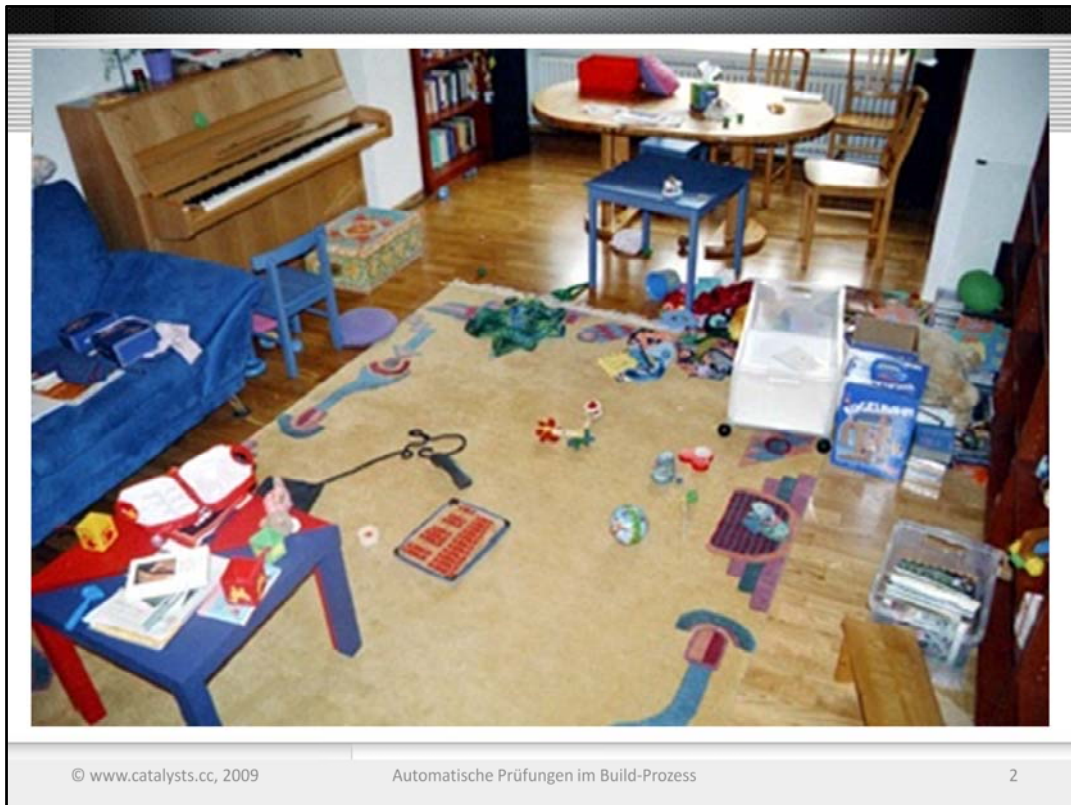


Hallo und willkommen zur 15. Wissensspritze.

Ich bin Christian Federspiel und der Titel der heutigen Wissensspritze ist „Automatische Prüfungen im Build-Prozess“.

Christian Federspiel hielt diesen Vortrag am 25.9.2009.

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



© www.catalysts.cc, 2009

Automatische Prüfungen im Build-Prozess

2

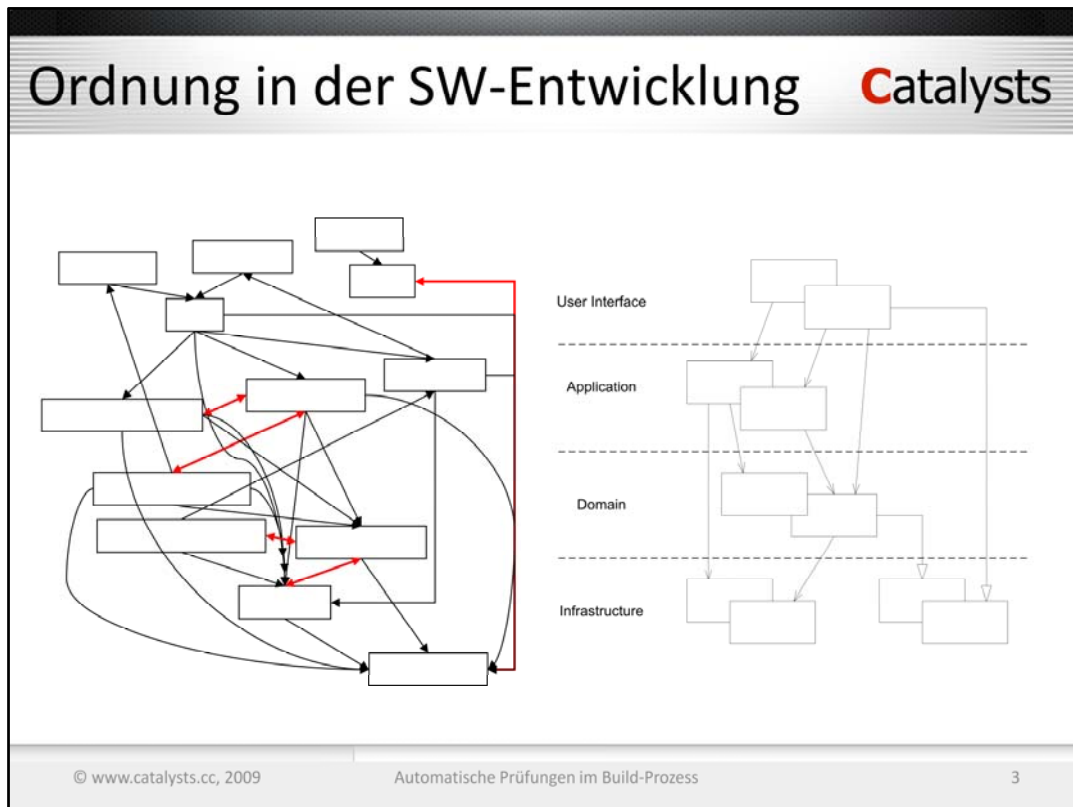
Sie sind Vater von 2 Kindern im Kindergartenalter. Es ist Freitag Vormittag, Ihre Kinder sind im Kindergarten, das Wetter ist eher herbstlich kalt und verregnet. Sie wagen – natürlich nur im Einverständnis mit Ihrer Partnerin - ein kühnes Experiment. Sie räumen ab heute Nachmittag, wenn die Kinder heimkommen, 48 Stunden Ihre Wohnung bewusst nicht auf und sagen Ihren Kindern auch nicht ständig, dass sie Hergeräumtes wieder wegräumen müssen.

Was wird passieren?

Hmm, was genau passieren wird, kann ich Ihnen auch nicht sagen. Nur soviel, wenn Sie das durchziehen können, dann ... ja dann sind Sie richtig mutig. Wir haben jedenfalls das Experiment nach 20 Stunden abgebrochen. Unter dem Kopfpolster war die Garage einer Lokomotive. Der Weg vom Schlafzimmer in die Küche war ein Slalomlauf zwischen diversen Spielsachen. Ständig lagen Legosteine, Holzschienen, Stifte etc. im Weg. Und im Kühlschrank saß der Eisbär.

Na ja, die Physik hat schon recht. Wir müssen einem System ständig Energie zuführen, um Ordnung zu halten. Die Energiezufuhr erfolgt in Form von Zusammenräumen (Eltern gemeinsam mit ihren Kindern) und einem ständigen Auffordern der Kinder zu etwas mehr Disziplin.

Aber was in der realen Welt so leicht ist – nämlich die Unordnung zu erkennen, ist in der SW-Entwicklung so schwer.



Als Software-Entwickler müssen wir nicht nur Termine und Kosten einhalten, sondern auch die gewünschte Funktionalität und Qualität liefern. Während die ersten 3 Faktoren in der Regel vom Kunden und/oder Management vorgegeben werden, geben wir uns meist selbst die Qualitätskriterien vor. Daheim in unserer Wohnung machen wir das auch. Was „aufgeräumt“ oder „Ordnung“ bedeutet, entscheiden wir selbst.

Seit wir objektorientiert zu programmieren begonnen haben, verlagern wir die Komplexität immer mehr weg vom Code in Richtung Struktur. Unter Struktur verstehen wir das Zusammenspiel von mehreren Objekten, um eine gewisse Funktionalität zu implementieren. Daher entstehen immer mehr Klassen in immer mehr Verzeichnissen.

Und in der Regeln entstehen auch viele Regeln.

- Welche Datei gehört in welches Verzeichnis und welcher Teil der Funktionalität wird in welcher Datei abgebildet.
- Welche Datei gehört zu welcher Schicht?

Alle Regeln zusammengenommen nennen wir Strukturregeln. Diese müssen eingehalten werden, um die Ordnung aufrecht zu erhalten. Im rechten Graphen sehen Sie z.B. die Regel, dass Objekte einer oberen Schicht nur von Objekten einer unteren Schicht abhängen dürfen, aber nicht umgekehrt.

10 Entwickler checken pro Tag ca. 100 Dateien in ein Versionsverwaltungssystem wie Subversion ein. Das birgt enormes Potential für Unordnung. Denn jedes Mal Einchecken, können natürlich eine oder mehrere unserer Qualitäts-Regel gebrochen werden. Weiters, wenn wir lange zuwarten, bevor wir einen Regelbruch beheben, kann es uns nachhaltig schädigen. Was rauskommt wenn wir unsere Qualitätsregeln nicht erhalten, das kennen die meisten nur allzu gut. Unwartbarer Code, unerwartet viele Fehler, ...

Typischerweise ein Graph wie Sie ihn hier auf der linken Seite sehen, wobei die Kästchen Objekte darstellen und die Pfeile Abhängigkeiten. Die roten Pfeile sind zyklische Abhängigkeiten.

Automatisch Überprüfungen

- Automatisch können folgende Qualitätskriterien überprüft werden
 - Struktur
 - Zyklen zwischen Klassen und Paketen
 - Abhängigkeiten zwischen Klassen und Paketen
 - Abhängigkeiten zwischen Schichten
 - Code
 - Richtlinien z.B: Einrückungen, if, while, switch, ...
 - Codeverdoppelung, Namenskonventionen, Code-Komplexität
 - Tests
 - Code Abdeckung auf Klassen- und Paket-Ebene

© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 4

Was Ordnung in der Software-Entwicklung bedeutet, legen wir im Team fest.

Die Qualitätskriterien beziehen sich in der Regeln meist auf Code, Struktur und Tests. Prozess-Regeln behandeln wir heute nicht. Ein Build-Server eignet sich für die automatische Überprüfung all dieser Regeln. Der gemeinsame Build macht Regelüberschreitungen im Team sichtbar.

Typischerweise überprüfen wir die Struktur:

- Wir wollen keine Zyklen zwischen Klassen und Paketen
- Wir wollen erlaubte Abhängigkeiten zwischen Klassen und Paketen definieren und die sollen dann auch eingehalten werden auch wenn neue Entwickler zum Team dazu stoßen.
- Wir wollen erlaubte Abhängigkeiten zwischen Schichten definieren und auch diese sollen ständig überprüft werden.

Im Code interessiert uns vor allem die Code-Komplexität und ob es Codeverdoppelung gibt.

Bei den Tests interessiert uns, wie viel Code unit-getestet ist.

Struktur-Überprüfungen Catalysts

Mit Build Server Plugins wie ClassCycle können Sie Beziehungen zwischen Klassen, Paketen und Schichten festlegen und automatisch überprüfen lassen.

Datei: classcycle.ddf

<p>Server Schichtung festlegen:</p> <pre>layer api = [api] layer dao = [dao] layer event = [event] layer exception = [exception] layer service = [service] layer util = [util]</pre> <p>Mit z.B:</p> <pre>[service] = #\${package}.*.*Service ... check layeringOf util exception dao service</pre>	<p>Client Schichtung festlegen:</p> <pre>layer command = [command] layer context = [context] layer mediator = [mediator] layer proxy = [proxy] layer repository = [repository]</pre> <p>Mit z.B:</p> <pre>[command] = #\${package}.*.*Command ... check layeringOf context repository service proxy command mediator</pre>
--	---

© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 5

Mit Build Server Plugins wie ClassCycle können Sie Beziehungen zwischen Klassen, Paketen und Schichten festlegen und automatisch überprüfen lassen. ClassCycle ist so eine Art DSL - Domain Specific Language - für Abhängigkeiten in Software Projekten.

Bei uns werden neue Projekte mit einem Skript auf immer gleiche Art und Weise angelegt. Im ant Projektverzeichnis werden die Komponenten oder Pakete, die Schichten in der Datei classcycle.ddf im ant Verzeichnis definiert.

Dann können Sie sofort beginnen erlaubte Abhängigkeiten zu definieren. Nachdem bei uns die meisten Projekte nach einem ähnlichen Muster am Server und am Client aufgebaut sind, haben wir auch hier schon wieder einiges vordefiniert.

layer service = [service] bedeutet:

- alle Klassen im set (so wird etwas in [] bezeichnet) service werden zu einem layer zusammengefasst.

[service] = #\${package}.*.*Service bedeutet:

- \${package} ... Gibt das root Verzeichnis unseres Software-Entwicklungsprojekts an.
- alle Klassen die mit "Service" enden können nun mit [service] angesprochen werden

Check layeringOf bedeutet, dass alle Klassen im package

- exception auf util zugreifen dürfen aber nicht umgekehrt
- dao auf exception und util zugreifen dürfen aber nicht umgekehrt
- service auf dao, exception und util zugreifen dürfen aber nicht umgekehrt

Wir verwenden in unseren Projekten serverseitig oft Java verwenden und clientseitig oft ActionScript oder C#. ClassCylce unterstützt mehrere Programmiersprachen.

Isolierungen

Catalysts

Wenn alle Implementierungen in Paketen sind, die per Namenskonvention "impl" beinhalten, dann können Sie überprüfen ob Interfaces (unerlaubt) auf Implementierungen zugreifen.

```
[impl] = ${package}.*.impl.*  
[interfaces] = ${package}.* excluding [impl]
```

```
#check sets [impl] [interfaces]
```

Die Mail-Implementierung darf auch wirklich nur von der Mail-Schnittstelle abhängen

```
[mailimpl] = ${package}.mail.impl.*  
[non-mailimpl] = [impl] excluding [mailimpl]
```

```
#check sets [mailimpl] [non-mailimpl]
```

Abhängigkeiten im Griff!

Catalysts

Die Api-Schicht muss unabhängig von Hibernate sein!

```
check [api] independentOf org.hibernate.*
```

Es darf keine Klassen- und Paketzuklen geben!

```
check absenceOfClassCycles > 1 in ${package}.*
```

Überprüfung nach jedem Build-Lauf! Catalysts

```
[classycle-check] check [impl] independentOf [interfaces] OK
[classycle-check] check [non-mailimpl] independentOf [mailimpl] OK

[classycle-check] check absenceOfClassCycles > 1 in net.taskmind.*
[classycle-check] net.taskmind.projects.ProjectItem et al. contains 10 classes:
[classycle-check] net.taskmind.notifications.Notification
[classycle-check] net.taskmind.projects.ProjectItem
...
```

© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 8

Code Komplexität

Catalysts

CCN – Die Cyclomatic Complexity Number, misst die Anzahl der möglichen Pfade durch eine Methode.

Je größer CCN, desto komplexer der Code

1 – 10 ... keine Code Komplexität code
 11 – 20 ... wenig Code Komplexität
 21 – 50 ... sehr komplex
 über 50 ... untestbarer Code

Blieben Sie unter 20!

Yes, we can!

CCN

Methods

[package] [object] [method] [explanation]

TOP 30 Methods containing the most NCSs.

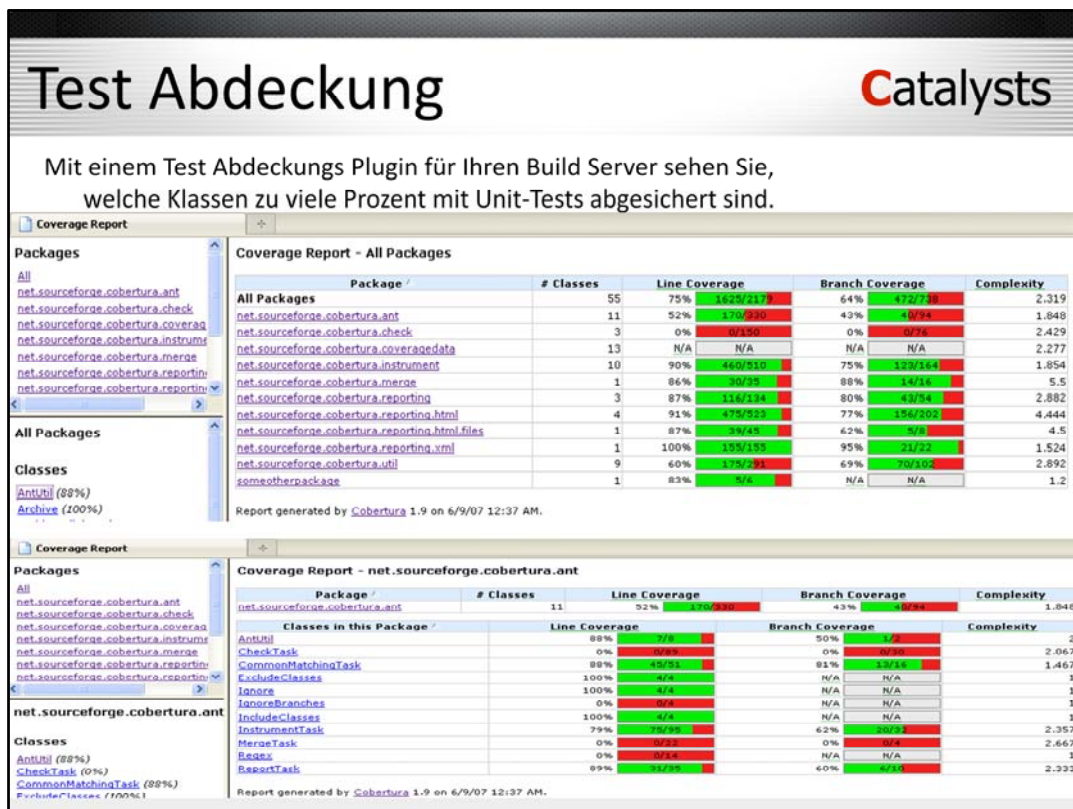
Method	CCN
catcoder.context.impl.DefaultContextManager.receiveScheduleContexts(List,Map)	54
catcoder.util.ExcalProcessor.generateCode(String,Code[])	48
catcoder.simulator.impl.Scanner.nextSy()	46
catcoder.server.CatCodeAssembler.assemble(boolean,String,boolean,String,String,String)	37
catcoder.server.CatCodeAssembler.assembleRecovery()	34
catcoder.simulator.impl.CommandParser.parse()	34
catcoder.context.impl.DefaultContextManager.selectGameInternal(String,String)	27
catcoder.context.impl.WirelessInstruction.getWirelessCode()	27
catcoder.server.ContextProviderForHttpContextProviderForHttp()	27
catcoder.context.impl.DefaultContextManager.assembleContexts(String,SimpleContext,String)	26
catcoder.context.impl.DefaultContextManager.receiveContext(String)	26
catcoder.context.impl.PasswordHandler.verifyCode(String,String)	26
catcoder.context.impl.SimpleContext.getCodeDetails(List,List)	26
catcoder.context.impl.SimpleContext.isAvailable(String,Map)	23
catcoder.util.Log.DecreaseLog(String,String)	23
catcoder.games.impl.DefaultGameSelector.getGameInfo(String,String)	22
catcoder.util.ZipCreator.zipCode(String,ZipOutputStream)	22
catcoder.context.impl.SimpleContext.getAcceptedSelfAccessCodes(String)	21
catcoder.simulator.generator.generator.generate()	21
catcoder.games.impl.GameInfoHandler.getGameInfoGameInfo()	19
catcoder.util.Log.impl.TimeFormatter.getTimeOutMinute(String)	19
catcoder.context.impl.SimpleContext.receiveProgress(String,int,List)	18
catcoder.games.impl.CommandHandler.getCommand()	18
catcoder.server.ContextProviderForHttp.getCodeArrayFromCodeArray(Code[])	18
catcoder.util.Log.impl.TimeFormatter.getTimeOutMinute(Calendar)	18
catcoder.util.ZipCreator.getZipCodeFile()	18
catcoder.context.impl.ContextProgressReceiver.receiveGameInfo(Object,String,List)	17
catcoder.context.impl.DefaultContextManager.scheduleContext(String,String,String,int)	17

© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 9

Die CCN – Die Cyclomatic Complexity Number, misst die Anzahl der möglichen Pfade durch eine Methode. Diese Metrik ist auch bekannt unter dem Namen „McCabe-Zahl“. Je größer die CCN ist, desto komplexer ist der Code in der Regel. Wenn Sie ein Tester sind, dann wissen Sie, dass Sie Code mit einer CCN von über 50 nicht zu testen brauchen. Der Code ist so komplex, dass er quasi „untestbar“ ist. D.h. Es zahlt sich nicht aus, diesen Code zu testen. Sie müssten zu viel Aufwand in die Tests stecken. Besser ist es, diesen Code neu zu schreiben.

Natürlich erzwingen gewisse Situationen eine höhere CCN. Beispiele wären ein herkömmlicher Scanner-Code für einen lexikalischen Analysator oder ein Dispatcher-Code. In der CCN Tabelle sehen Sie die Methode Scanner.nextSy() mit einem CCN von 29. Wenn sie den Code öffnen, dann sehen sie lediglich ein größeres Switch-Statement. Der Code ist trotz hoher CCN leicht zu warten und leicht zu testen.

Stimmen Sie sich als Tester also mit Ihren Entwicklern ab, bevor Sie loslegen. Der Build-Server ermittelt den CCN für alle Methoden. Jeder im Team kann sie sehen. Kaum ein Entwickler möchte seinen Code mit einem hohen CCN verbunden sehen.



Mit einem Test-Abdeckungs-Plugin – wie z.B. Cobertura - für Ihren Build Server sehen Sie, welche Pakete (im oberen Bild) und welche Klassen (im unteren Bild) zu wieviel Prozent mit Unit-Tests abgesichert sind. Sie sehen die Line Coverage, die Ihnen sagt wie viele Zeilen Code Sie mit Unit Tests testen und die Branch Coverage. Diese gibt an, ob Ihre Testfälle auch in die verschiedenen Zweige (z.B. bei einem if-statement der then und else-Zweig) abtesten.

Daneben schreibt Cobertura auch die CCN Zahl. Als (Black Box) Tester, wissen Sie welche Klassen bzw. Pakete Sie sich besonders vornehmen sollten.

White Box Tests

Catalysts

```

104 301
105
106 12
107 12
108 12
109 12
110 12
111 12
112 4
113
114
115
116
117
118
119
120
121 12
122
123 0
124
125 0
126
127 0
128 12
129
130 12
131
132 264

```

```

    if (java == null)
    {
        java = (Java)getProject().createTask("java");
        java.setTaskName(getTaskName());
        java.setClassname(getClassName());
        java.setFork(true);
        java.setDir(getProject().getBaseDir());
        if (maxMemory != null)
            java.setJvmargs("-Xmx" + maxMemory);

        /**
         * We replace %20 with a space character because, for some
         * reason, when we call Cobertura from within CruiseControl,
         * the classpath here contains %20's instead of spaces. I
         * don't know if this is our problem, or CruiseControl, or
         * ant, but this seems to fix it. --Mark
         */
        if (getClass().getClassLoader() instanceof AntClassLoader)
        {
            String classpath = ((AntClassLoader)getClass()
                .getClassLoader()).getClasspath();
            createClasspath().setPath(
                StringUtil.replaceAll(classpath, "%20", " "));
        }
        else if (getClass().getClassLoader() instanceof URLClassLoader)
        {
            URL[] earls = ((URLClassLoader)getClass().getClassLoader())
                .getURLs();
            for (int i = 0; i < earls.length; i++)

```

© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 11

Ein Test-Abdeckungs-Plugin erlaubt Ihnen auch meist, direkt getesteten bzw. ungetesteten Code zu sehen.

Ganz links im Bild sehen Sie die Zeilennummer. Danach sehen Sie, wie oft der Code in den Testfällen aufgerufen wurde.

Die Zeile 111 wurde 12 mal aufgerufen, die Zeile 112 dagegen nur 4 mal.

Die Zeile 121 12 mal, aber immer hat die Bedingung in der if-Anweisung false ergeben.

Die Zeile 123 schließlich wurde nie getestet.

Als Entwickler können Sie nun besser abschätzen, ob Sie hierfür jetzt einen Unit-Test schreiben oder nicht.

Würden Sie nun einen Testfall für die Zeilen 121 – 127 schreiben?

Also wenn der Code auch auf einen Mac laufen soll, dann würde ich es sicher machen.

Sie können auch toten Code entdecken.

Typischerweise sollte jede Methode einer Klasse von mehreren Unit Testfällen aufgerufen werden.

Wenn Sie trotz größerer Anstrengung nicht alle Codeteile testen können, dann riecht das nach totem Code.


Ich hoffe Sie wissen wir toter Code riecht ;-)

Unser Tool Stack

Catalysts

- Testautomatisierung
- Statische Code Analyse
- Struktur Checks
- Continuous Integration Server

Cobertura



© www.catalysts.cc, 2009 Automatische Prüfungen im Build-Prozess 12

Die Diskussion um die Wahl der geeigneten Tools gibt es nicht nur bei Entwicklungsumgebungen, sondern bei allen am Build-Prozess beteiligten Komponenten. Nachdem wir schon viele unterschiedliche Tools ausprobiert haben, und mit einigen davon auch längere Zeit produktiv gearbeitet haben, fiel unsere Wahl auf folgende Komponenten.

.....

Bis auf TeamCity sind das alles Open Source-Projekte.

TeamCity ist für kleine Teams auch gratis.

Grund: weitverbreitet, gut unterstützt durch andere Tools, pflegeleicht in der Wartung, geringer Initialaufwand beim Einrichten

Zusammenfassung Catalysts

1. Qualität muss auch automatisch überprüft werden.
 - rasches Feedback, schafft Transparenz

2. Der Build-Server ist dafür geeignet. Er liefert schnelles Feedback über
 - Testabdeckung, Struktur-Verletzungen, Code-Qualität
 - und wenn Sie wollen auch über vieles mehr

3. Noch schnelleres Feedback (während des Tippens in der IDE) wäre uns noch lieber.
 - Open Source Tools dafür sind schon in Entwicklung ...

© www.catalysts.cc, 2009Automatische Prüfungen im Build-Prozess13


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

The slide features a header with the word 'Ausblick' on the left and the 'Catalysts' logo on the right. The main content area contains the title '16. Wissensspritze: Dependency Management mit Ivy'. At the bottom, there is a footer with three items: '© www.catalysts.cc, 2009', 'Automatische Prüfungen im Build-Prozess', and the number '14'.

Die nächste Wissensspritze beschäftigt sich mit dem Thema „Abhängigkeit von 3rd Party Libraries mit Ivy managen“

Diese Wissensspritze befasst sich damit, wie man Abhängigkeiten zu externen Bibliotheken sauber verwaltet und wie man sicherstellt, dass jeder im Team immer die richtige Version dieser Bibliotheken verwendet. Wir zeigen das am Beispiel von Ivy.

Tschüss und bis zum nächsten Mal!



Was bringt mir Catalysts sonst noch?

Werbe-Trailer

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

Catalysts

- „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!

Schneller am Ziel mit Catalysts - 1A-Qualität



- 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 zyklensfreie Architektur, [ausführlich getestet](#)
- [Unternehmensberatung inbegriffen](#)
- Exzellentes Team