Die Revision schaut mit: Jira überwachtes Deployment mit SAS Promotion Wizard

Carlo van de Rijt
Basler Versicherungen AG
Aeschengraben 21
CH-4002 Basel
carlo.van_de_rijt@baloise.ch

Michael Fabritius
Basler Versicherungen AG
Aeschengraben 21
CH-4002 Basel
michael.fabritius@baloise.ch

Zusammenfassung

Im Projekt Kepler zum Aufbau eines Enterprise Datawarehouse für die Basler Versicherungen setzt die IT Schweiz auf einen von SAS entwickelten Promotion Wizard für das Deployment von SAS DI Studio Jobs in die Test- und Produktivumgebungen.
Für die Revision ist es sehr wichtig, dass die dafür gedachten Prozesse eingehalten werden. Dabei ist die Atlassian Software JIRA eine gute Hilfe. Der Deploymentprozess wird komplett von JIRA gesteuert und überwacht.
Es wird gezeigt, wie für das Deployment relevante Daten aus JIRA eingelesen und mit den SAS Metadaten verknüpft werden, um den Release-Status einzelner Metadatenpakete (SPKs) auszulesen. Hierbei werden auch sog. 'custom fields' in JIRA verwendet. Der Deploymentprozess selbst wird von der Paketierung auf DEV bis zur Bereitstellung in PROD über SAS Stored Processes ausgeführt und von Anfang bis Ende registriert. Somit wird eine maximale Transparenz sichergestellt.
Dank der Prozessüberwachung mittels JIRA werden alle Entwickler im Projekt in der Lage sein, den ganzen Promotionsprozess bis in die Produktion nach Vorgaben der Revision auszuführen.

Schlüsselwörter: Jira, Metadaten, Interoperabilität

1 Einleitung

Die externe Revision ist ein von der Hauptversammlung eines Unternehmens bestelltes Prüforgan, das im Rahmen des Aktionärsschutzes den Jahresabschluss dieses Unternehmens überprüft. Da viele Prozesse der IT einen direkten Einfluss haben auf das Ergebnis, fallen auch diese Prozesse unter diese Prüfung. Die Revision erwartet von der IT, dass beim Deployment von SAS-Jobs keine manuelle Eingriffe stattfinden, und dass die verschiedene Schritte von der Entwicklungsumgebung bis in die Produktion völlig transparent sind: welche Jobs wurden wann und von wem in die betreffende Umgebung (vor allem die Produktion) eingespielt und welche abhängigen Objekte wie Tabellen und externe Dateien wurden dabei wie angepasst. Der von SAS entwickelte Promotion Wizard stellt dieses sicher.

2 Promotion in SAS DI Studio

In SAS wird von Promotion gesprochen, wenn SAS Jobs und andere Metadaten von der Entwicklungsumgebung DEV über die Teststufen INT (für die Integration mit den Umsystemen) und ACC (für die fachliche Abnahme: Acceptance) in Produktion genommen werden. Wenn man in SAS DI Studio entwickelt, werden dazu Metadatenpakete gebildet über eine Exportfunktion:

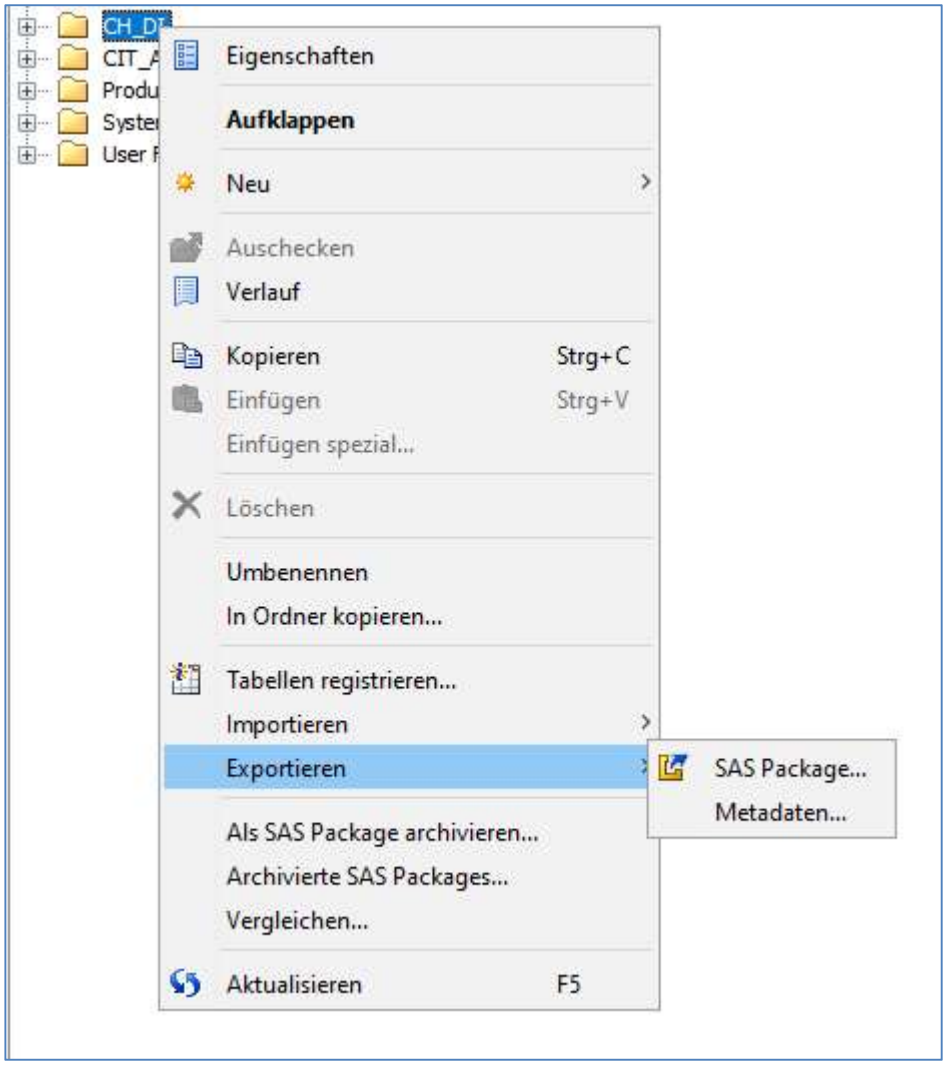


Abbildung 1: Exportfunktion für Metadaten in SAS DI Studio.

Die gewählten Jobs und - wenn gewünscht - dazugehörigen Objekte wie Tabellen, externe Dateien und Transformationen werden zusammen in eine Datei gepackt, eine sog. spk-Datei. Diese Datei kann in der nächsten Umgebung wieder importiert werden. Eventuell müssen Jobs danach erneut bereitgestellt werden.
Für diesen Prozess hat SAS einige Batch-Tools entwickelt, die die Basis bilden für den Promotion Wizard: ExportPackage, ImportPackage und DeployJobs.

SAS Promotion Wizard

Der Promotion Wizard ist ein SAS StoredProcess für die Verwaltung der Metadaten im Deployment Prozess, d.h. die Promotion der Metadaten von der Entwicklungsumgebung in die Produktion über die Teststufen. Mit Hilfe des Promotion Wizard können Metadatenpakete erstellt und in den höheren Umgebungen importiert werden. Für den Promotion Wizard gibt es auf jede der Umgebungen eine eigene Version. Nur in der Entwicklungsumgebung können Pakete erstellt werden, in der Integrationsumgebung können Pakete nur importiert werden. Auch auf der Acceptance können Metadaten nur importiert werden, aber nur wenn dafür ein Release vorhanden ist. Ein Release besteht aus ein oder mehrere Metadatenpakete. Das gleiche Release-Paket wird später auch in die Produktion importiert. Der Promotion Wizard auf DEV, INT und ACC ist für die SAS-Entwickler freigegeben. Diese können bis in die Acceptance-Umgebung ihre Jobs und andere Metadaten selbst einspielen. Für das Deployment in die Produktion ist ein dediziertes Betriebsteam mit Admin-Rechte zuständig.
Jobs die mittels des Promotion Wizard importiert werden, werden automatisch auch bereitgestellt.

3 Deployment Prozess

In Absprache mit der externen Revision wurde der folgende Deployment-Prozess bei der Basler Versicherung aufgesetzt:

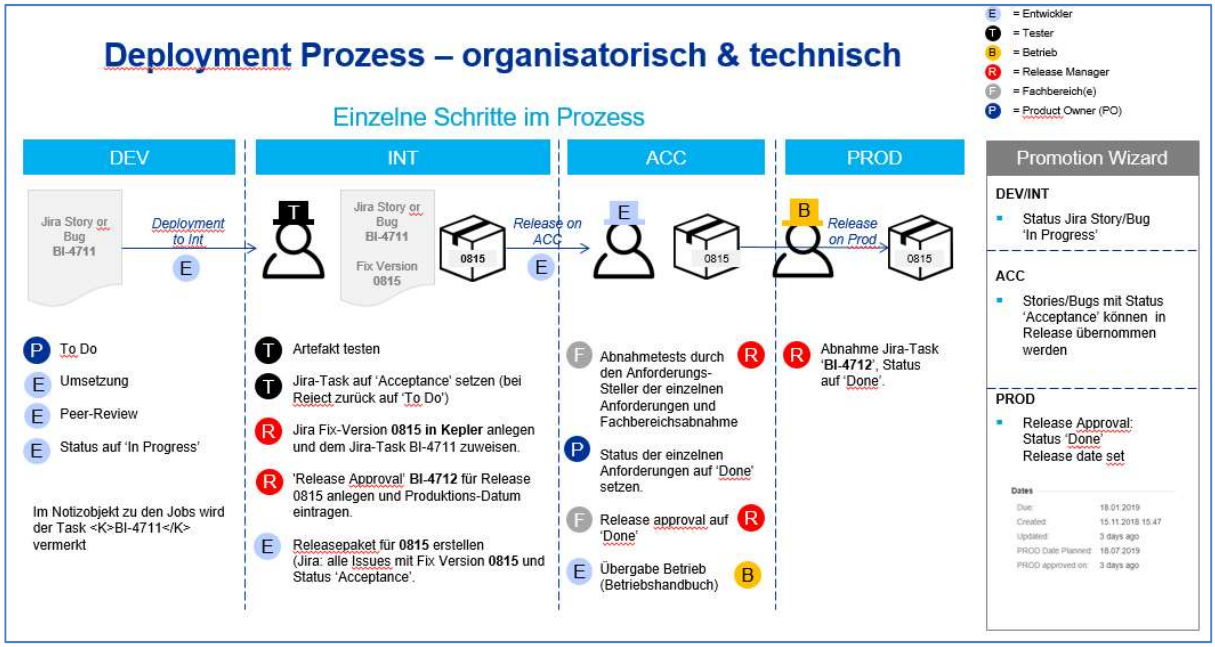


Abbildung 2: Organisatorische und technische Umsetzung des SAS-Deployment.

Wenn ein Bug gefixt oder ein Job deployed werden muss, setzt der Product Owner, der den Kunden vertritt, das betreffende Jira-Issue auf 'To Do'. Der EntwicklerIn darf diese Story an sich nehmen, setzt den Status 'In Progress' und nimmt die notwendigen Änderungen vor. Der Issue-key (Jira-Nr) der Jira-Story oder Bug wird in den jeweiligen Notiz-Objekten der Jobs eingetragen, damit das betreffende Notiz-Objekt mittels der

Klammern <K> und </K> in den SAS Metadaten identifiziert werden kann, hier im Beispiel: <K>BI-5680</K>:

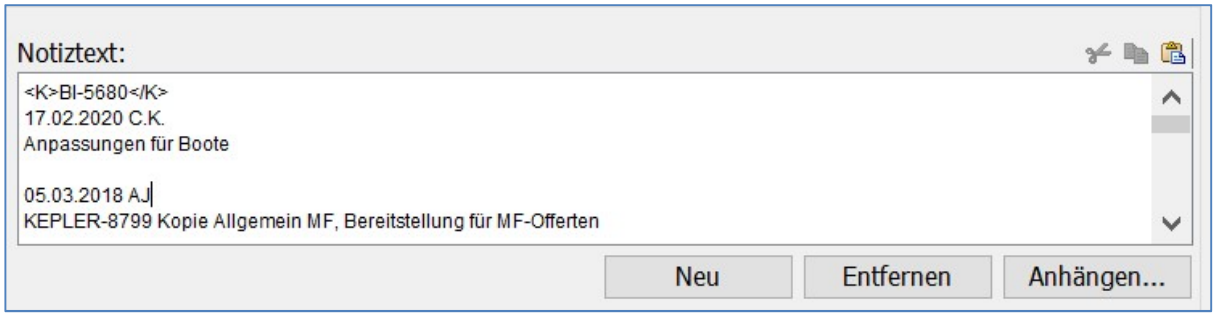


Abbildung 3: Jira-Nummer als Markierung im Notizobjekt eines DI Jobs.

Wenn der EntwicklerIn mit seiner Arbeit fertig ist, werden die geänderte Metadaten mit Hilfe des Promotion Wizard als Metadatenpaket auf ein zentrales Repository abgelegt.
Dieses Paket wird anschließend in die Integrationsumgebung importiert und die Jobs werden bereitgestellt. Das Jira-Issue wird jetzt einer TesterIn übergeben.
Wenn die Jobs erfolgreich getestet werden konnten, setzt der TesterIn den Jira-Status auf Acceptance, damit diese Jobs für die fachliche Prüfungen an die Fachbereiche übergeben werden können.

Für das Einspielen der Metadaten in die Acceptance-Umgebung wird ein Release gebraucht. Der Release-ManagerIn legt für alle Jira-Issues die gemeinsam in Produktion genommen werden müssen, eine Release-Nummer an (in Jira: Fix Version). Unter diese Nummer wird mit Hilfe des Promotion Wizards vom EntwicklerIn ein Release-Paket erstellt. Nur Jira-Issues, die erfolgreich auf der INT-Umgebung getestet wurden, können in ein Release-Paket aufgenommen und auf der Acceptance-Umgebung importiert werden.
Das Release-Paket gruppiert die zugehörigen bestehenden Metadatenpakete im Repository.
Auch legt der Release-ManagerIn ein sog. Release-Approval Issue in Jira an, das für die spätere Promotion in das Produktivsystem gebraucht wird.
Nachdem auch die fachlichen Prüfungen erfolgreich abgeschlossen sind, werden die betreffende Jira-Issues geschlossen (Status ‘Done’) und es kann das komplette Release-Paket auf Produktion importiert und bereitgestellt werden. Das Release-Approval Issue wird dazu auf ‘Done’ gesetzt und es wird ein Datum für das Produktions-Deployment eingetragen.

Der ganze Prozess funktioniert nach dem vier-Augen-Prinzip: Der Entwickler darf seine Arbeit nicht selbst abnehmen. Es wird immer eine zweite Person gebraucht, um den Status auf Acceptance setzen zu können. Nachdem der Fachbereich die Ergebnisse freigegeben hat, dürfen nur KollegInnen mit Admin-Rechte das Release-Paket in Produktion bringen. Die Entwickler liefern dazu ein Drehbuch mit Anweisungen, was zu tun ist: Datenbank-Aufträge, die gemacht werden müssen, Flows die gebraucht werden, Steuertabellen die aktualisiert werden müssen, etc.

4 Technische Umsetzung

4.1 Informationen aus Jira

Damit der Deployment-Prozess mittels Jira überwacht werden kann, müssen SAS und Jira ‘miteinander reden’. Der Deployment Wizard erstellt eine Verbindung zwischen beiden Systemen.
Die Jira-Issues können mittels eines Web-Interfaces abgerufen und mittels **Proc http** verarbeitet werden (siehe Appendix 1 für Details). Die gelieferte XML-Datei wird mit Hilfe des xmlv2-Engine als SAS Library abgebildet.

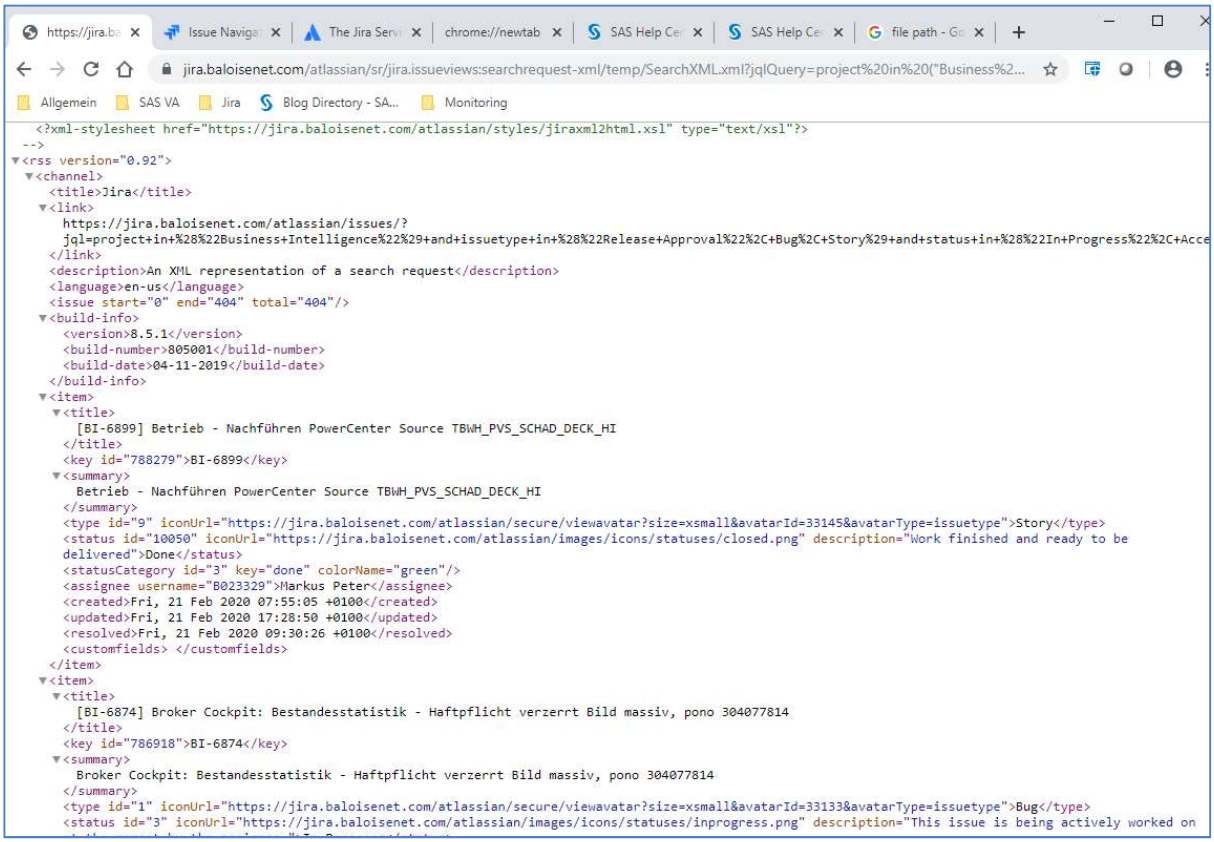


Abbildung 4: Jira REST API: XML-Ausgabe

Die SAS-Library enthält mehrere Tabellen, die über das Feld item_ordinal miteinander verknüpft werden können (Appendix 2).

4.2 SAS Metadaten

Mit den Data Step-Funktionen für SAS Metadaten werden die Informationen über Jobs, Tabellen und andere SAS Metadatenobjekte gesammelt. Zuerst werden alle Notizobjekte in eine Tabelle geschrieben (Appendix 3). Im Feld Notetext (in HTML) wird gesucht nach dem Text zwischen <K> und </K> und als Variable ‘kepler’ zusammen mit dem Metadaten-ID des Objects in eine Tabelle geschrieben:

```
kepler=substr(notetext,index(lowcase(notetext),'&lt;k&gt;')
+9,index(lowcase(notetext),'&lt;/k&gt;')-
index(lowcase(notetext),'&lt;k&gt;')-9);
```

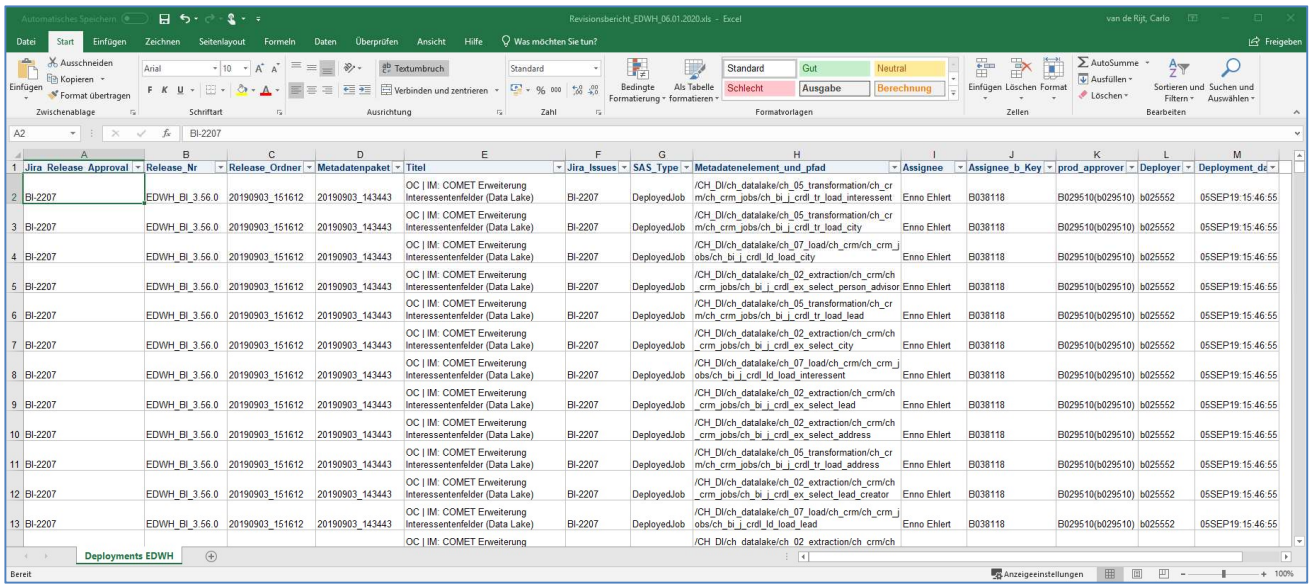
Anschließend werden die Jobs und User-written Transformations gelistet und deren Attribute wie Name, Pfad und verlinkte Objekte gesammelt. Zum Schluss werden diese Objekt mit den Notizobjekten verlinkt.

4.3 Zusammenführen SAS und Jira

Beim Exportieren der SAS Metadaten mittels des Promotion Wizard werden die Metadatenpakete in einem zentralen Repository abgelegt, das aus allen Entwicklungsumgebungen heraus erreichbar ist. Die Informationen der SAS Metadaten (welche Objekte?, wo liegen sie?) werden mit den Jira-Informationen (Status, wer arbeitet am Issue) über die Jira-Nr miteinander verknüpft und zusammen mit den Informationen des Deployment-Prozesses (wer hat das Paket erstellt?, wann?, wo liegt das Paket im Repository?) im Repository gespeichert.

Der Promotion Wizard für die Integrationsumgebung greift nur lesend auf dieses Repository zu, um die erstellten Metadatenpakete zu importieren. Dabei werden die Prozess-Informationen im Repository angereichert mit Informationen, wer das Paket importiert hat und wann.

In ACC werden die Release-Informationen angereichert: welche Metadatenpakete können für den Release verwendet werden. Erst danach können diese Metadatenpakete als Release-Paket in ACC importiert werden. Es wird aufgezeichnet, wer das Release-Paket in ACC importiert hat und wann.
Das gleiche Release-Paket kann in Produktion genommen werden, wenn das zum Release gehörende Release Approval dazu freigegeben wurde. Auch hier wird im Repository festgehalten, wer die Metadatenpakete abgenommen hat und wer den Release freigegeben hat für die Produktion.
Diese Informationen werden zwei Mal im Jahr automatisch aufbereitet und als Excel-Bericht der externen Revision zugestellt:



Jira_Release_Approval	Release_Nr	Release_Ordner	Metadatenpaket	Titel	Jira_Issues	SAS_Type	Metadatenelement_und_pfad	Assignee	Assignee_b_Key	prod_approver	Deployer	Deployment_da
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_05_transformation/ch_crm/ch_crm_jobs/ch_bi_j_crdl_tr_load_interessent	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_05_transformation/ch_crm/ch_crm_jobs/ch_bi_j_crdl_tr_load_city	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_07_load/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ld_load_city	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_02_extraction/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ex_select_person_advisor	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_05_transformation/ch_crm/ch_crm_jobs/ch_bi_j_crdl_tr_load_lead	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_02_extraction/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ex_select_city	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_07_load/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ld_load_interessent	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_02_extraction/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ex_select_lead	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_02_extraction/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ex_select_address	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_05_transformation/ch_crm/ch_crm_jobs/ch_bi_j_crdl_tr_load_address	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_02_extraction/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ex_select_lead_creator	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55
BI-2207	EDWH_BI_3.56.0	20190903_151612	20190903_143443	OC | IM: COMET Erweiterung Interessentenfelder (Data Lake)	BI-2207	DeployedJob	/CH_DI/ch_datalake/ch_07_load/ch_crm/ch_crm_jobs/ch_bi_j_crdl_ld_load_lead	Enno Ehlert	B038118	B029510(b029510)	b025552	05SEP19:15:46:55

Abbildung 5: Deploymentbericht für die externe Revision.

Appendix 1: Jira-Daten auslesen

Die folgende Jira-Abfrage:
project = "Business Intelligence" and issuetype in ("Release Approval" ,Bug ,Story) and status in ("In Progress", Acceptance, Done) wird wie folgt im Browser umgesetzt. Die Zeilenumbrüche dienen hier nur der Lesbarkeit. Der ganze String soll ohne Leerzeichen an Jira übergeben werden. XXXXX ist der lokale Jira-Server:

```
https://jira.xxxxx.com/atlassian/sr/jira.issueviews:searchrequest-xml/temp/SearchXML.xml?jqlQuery=
project%20in%20("BusinessIntelligence")%20and%20issuetype%20in%20
("Release%20Approval"%2C%20Bug%2C%20Story)%20and%20
status%20in%20("In Progress"%2C%20Acceptance%2C%20Done)
&tempMax=1000 /* eingrenzen auf 1000 Datensätze*/

&field=summary /* eingrenzen auf bestimmte Jira-felder */
&field=key_id
&field=assignee
&field=key
&field=title
&field=summary
&field=created
&field=updated
&field=resolved
&field=status
&field=type
&field=fixVersions
```

```
&field=customfield_12953
&field=customfield_13551
&field=customfield_12952
&field=customfield_13550
&field=customfield_24751
&field=customfield_24752
&field=customfield_14550
&field=customfield_12954
&expand=names
```

Das Ergebnis ist eine XML-Ausgabe:

Die Abfrage ‘https://jira.xxxx.com ……&expand=names’ kann von SAS mittels **Proc http** verarbeitet werden und liefert eine XML-Datei die mit Hilfe des xmlv2-Engine in eine SAS-Library mit mehreren Tabellen umgesetzt wird:

```
filename myxml TEMP;
filename mymap "Pfadname/jira.map"; /*XMLMap-Datei*/

%let url='https://jira.xxxx.com …… &expand=name';
proc http
url=&url
webusername=&e_metauser
webpassword=&e_metapass
method="get"
out=myxml;
run;
libname myxml xmlv2 xmlmap=mymap access=readonly;
```

Die XMLMap-Datei kann wie folgt mit Hilfe der xmlv2-Engine erstellt werden:

```
libname myxml xmlv2 xmlmap=mymap automap=replace;
```

Appendix 2: Jira-Daten weiterverarbeiten

Status-Informationen und wer diese gesetzt hat befinden sich in der Tabelle *custom_fields*

```
proc sql;
create table sub_status as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="SUB STATUS"
;

create table prod_appr_on as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="PROD APPROVED ON"
;

create table prod_appr_by as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
```

```
having upcase(b.customfieldname)="PROD APPROVED BY"
;

create table impl_appr_on as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="IMPL APPROVED ON"
;

create table impl_appr_by as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="IMPL APPROVED BY"
;

create table it_tested_on as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="IMPLEMENTED & IT TESTED
ON"
;
```

```
create table it_tested_by as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="IMPLEMENTED & IT TESTED
BY"
;

create table CCM_OK as
select a.item_ordinal
        ,b.customfieldname
        ,d.customfieldvalue
from myxml.customfields as a
left join myxml.customfield as b
on(a.customfields_ordinal=b.customfields_ordinal)
left join myxml.customfieldvalues as c
on(b.customfield_ordinal=c.customfield_ordinal)
left join myxml.customfieldvalue as d
on(c.customfieldvalues_ordinal=d.customfieldvalues_ordinal)
having upcase(b.customfieldname)="CCM"
;
quit;
```

Tabelle **jira** enthält für alle Jira-Issues die Standard- und Status-Informationen

```
PROC SQL;
CREATE TABLE jira AS
SELECT distinct
t1.key_id,
t1.key,
t2.title,
t2.summary,
t2.created,
t2.updated,
t2.resolved,
t3.status,
t4.customfieldvalue LABEL="sub_status" AS sub_status,
t5.fixVersion,
```

```
t6.type,
t7.customfieldvalue    LABEL="Prod    Approval    on"    AS
prod_appr_on,
t8.customfieldvalue    LABEL="Prod    Approval    by"    AS
prod_appr_by,
t9.customfieldvalue    LABEL="Impl    Approval    on"    AS
impl_appr_on,
t10.customfieldvalue    LABEL="Impl    Approval    by"    AS
impl_appr_by,
t11.customfieldvalue LABEL="IMPLEMENTED & IT TESTED ON" AS
it_tested_on,
t12.customfieldvalue LABEL="IMPLEMENTED & IT TESTED BY" AS
it_tested_by,
t13.customfieldvalue LABEL="CCM" AS CCM,
t14.assignee LABEL='Assignee' AS Assignee
FROM MYXML.key t1
LEFT   JOIN   MYXML.item   t2   ON   (t1.item_ORDINAL   =
t2.item_ORDINAL)
LEFT   JOIN   MYXML.status   t3   ON   (t1.item_ORDINAL   =
t3.item_ORDINAL)
LEFT   JOIN   SUB_STATUS   t4   ON   (t1.item_ORDINAL   =
t4.item_ORDINAL)
LEFT   JOIN   MYXML.fixVersion   t5   ON   (t1.item_ORDINAL   =
t5.item_ORDINAL)
LEFT   JOIN   MYXML.type   t6   ON   (t1.item_ORDINAL   =
t6.item_ORDINAL)
LEFT   JOIN   prod_appr_on   t7   ON   (t1.item_ORDINAL   =
t7.item_ORDINAL)
LEFT   JOIN   prod_appr_by   t8   ON   (t1.item_ORDINAL   =
t8.item_ORDINAL)
LEFT   JOIN   impl_appr_on   t9   ON   (t1.item_ORDINAL   =
t9.item_ORDINAL)
LEFT   JOIN   impl_appr_by   t10   ON   (t1.item_ORDINAL   =
t10.item_ORDINAL)
LEFT   JOIN   it_tested_on   t11   ON   (t1.item_ORDINAL   =
t11.item_ORDINAL)
LEFT   JOIN   it_tested_by   t12   ON   (t1.item_ORDINAL   =
t12.item_ORDINAL)
LEFT    JOIN    CCM_OK    t13    ON    (t1.item_ORDINAL    =
t13.item_ORDINAL)
LEFT   JOIN   MYXML.assignee   t14   ON   (t1.item_ORDINAL   =
t14.item_ORDINAL)
;
QUIT;
```

Appendix 3: SAS Metadaten auslesen

/* Notiztexte sammeln */

```
data notes (keep=docuri notetext);
    length docuri docname noteuri $256;
    length notetext $32000;
    rc=1;
    n=1;
    do while(rc>0);
/*Hole die Metadaten-IDs und speichere sie als docuri*/
        rc=metadata_getnobj("omsobj:Document?@Id   contains
'.'",n,docuri);
/* so lange es welche gibt: */
        if rc > 0 then do;
/* hole das Attribut Name als docname */
rcn=metadata_getattr(docuri,"Name",docname);
rc2=1;
n2=1;
do while(rc2>0);
/*  hole  alle  assoziierte  Objekte  des  Typs  "Notes"  als
noteuri */
rc2=metadata_getnasn(docuri,"Notes",n2,noteuri);
if rc2 > 0 then do;
/*  hole  von  diesen  notes  das  Attribut  "StoredText"  als
notetext */
rcn=metadata_getattr(noteuri,"StoredText",notetext);
output;
notetext="";
end;
n2=n2+1;
end;
end;
n=n+1;
end;
run;
```

/* Extract der Jira-Nr aus den Notizobjekten*/

```
data kepler (keep=docuri kepler order);
set notes;
length kepler $50;
order=0;
do while (index(lowcase(notetext),'&lt;k&gt;') and
index(lowcase(notetext),'&lt;/k&gt;'));
```

```
kepler=substr(notetext,index(lowcase(notetext),'&lt;k&gt;')
+9,index(lowcase(notetext),'&lt;/k&gt;')-
index(lowcase(notetext),'&lt;k&gt;')-9);

kepler=upcase(kepler);

order=order+1;
output;
notetext=substr(notetext,index(lowcase(notetext),'&lt;/k&gt
;')+100);
end;
run;
```

/* diverse Attribute aller Objekte sammeln */

```
data objects_with_notes (keep=docuri docname objname
objtype obj_folder);
    length docuri docname objname objtype objuri obj_folder
tree_uri tree_name $256;
    rc=1;
    n=1;
    do while(rc>0);
/* Hole die Metadaten-IDs und speichere sie unter docuri*/
     rc=metadata_getnobj("omsobj:Document?@Id      contains
'.'",n,docuri);
/* so lange es diese gibt: */
      if rc > 0 then do;
/* hole vom docuri das Attribut Name und speichere es unter
docname */
     rcn=metadata_getattr(docuri,"Name",docname);
     rc2=1;
     n2=1;
/* so lange es sie gibt: */
     do while(rc2>0);

/* hole alle assoziierte Objekte und speichere sie unter
objuri */
     rc2=metadata_getnasn(docuri,"Objects",n2,objuri);

     if rc2 > 0 then do;
/* hole von diesen Objekte das Attribut Name und speichere
sie unter objname */
     rcn=metadata_getattr(objuri,"Name",objname);
/* das Attribut PublicType als objtype */
     rcn=metadata_getattr(objuri,"PublicType",objtype);
```

```
     obj_folder="";
/* hole von diesen Objekte die assoziierten Objekte vom Typ
'trees' ((sub)-directories)*/
/* und speichere diese unter tree_uri */
     rc0=metadata_getnasn(objuri,"Trees",1,tree_uri);
     do while(rc0>0);
/* hole von jedem tree_uri den Namen */
    rc5=metadata_getattr(tree_uri,"Name",tree_name);
/* und lege den ganzen Datenpfad an unter obj_folder */
    obj_folder="/" || trim(tree_name) || obj_folder;
/* setze tree_uri zurück auf das Basisverzeichnis */
    rc0=metadata_getnasn(tree_uri,"ParentTree",1,tree_uri);
    end;
    output;
    end;
    n2=n2+1;
    objname="";
    end;
    end;
    n=n+1;
    end;
    run;
```

/* Job-Objekte mit Notiztext verheiraten */

```
proc sql;
    create table mydata.kepler_jobs as
    select  a.kepler as knr
            /*,cats("Kepler-",trim(left(a.kepler)))     as
knr_label*/
            ,a.kepler as knr_label
            ,a.order
            ,b.docname

    ,cats(b.obj_folder,"/",b.objname,"(",b.objtype,")")  as
object
    from objects_with_notes as b
    left join kepler as a
    on(a.docuri=b.docuri)
    where upcase(b.objtype) = "JOB"
    having          knr          ne          ""          and
index(upcase(object),"_IN_ARBEIT")=0
;
quit;
```

/* User-written Transformations mit Notiztext anreichern */

```
proc sql;
     create table mydata.kepler_trans as
     select   a.kepler as knr
              ,a.kepler as knr_label
              ,a.order
              ,b.docname
         ,cats(b.obj_folder,"/",b.objname,"(",b.objtype,")")
as object
     from objects_with_notes as b
     left join kepler as a
     on(a.docuri=b.docuri)
     where upcase(b.objtype) = "GENERATEDTRANSFORM"
     having knr ne "" ;
quit;
```