Inhaltsverzeichnis

1. Entwicklung

Die Realisierungsphase des ÖREB-Katasters besteht aus verschiedenen Teilprojekten. Dabei geht es um klassische Softwareentwicklung (z.B. ÖREB-Webservice) aber auch um «GIS-Entwicklungen» (Modellieren, Datenumbau, etc.) und automatisierten Deploymenents von Daten.

1.1. Code-Verwaltung

Sämtlicher Code - dazu gehört z.B. auch die Dokumentation - jedes Teilprojekts wird in einem einzelnen Github-Repository verwaltet. Ob und welche Repositories nach Abschluss der Realisierungsphase zusammengezogen werden, ist zum jetzigen Zeitpunkt nicht bekannt. Eigentümer der Repositiers ist die Organisations *sogis*. Die Namen der Respositories beginnen immer mit `oereb_`. Nach dem Unterstrich ist auf eine sinnvolle Namensgebung zu achten.

Es sind sinnvolle Commit-Messages zu verwenden.

1.2. Automatisierung

Jedes Teilprojekt resp. die daraus entstehenden (bei Datenumbauten und Modellierungen) Prozessen oder die klassichen Buildprozesse sind bereits während der Entwicklung zu automatisieren. D.h. die Prozesse sind nur mit einem einzelnen Befehl (o.ä.) zu starten.

Gleichzeitig ist darauf zu achten, dass keine Abhängigkeiten an die Entwicklungsumgebung gestellt werden, die nicht mit *Vagrant* und *Docker* erfüllt werden können. So soll nicht direkt mit den Datenbank der GDI gearbeitet werden, sondern die darin enthaltenen Daten müssen in einer Vagrant- oder Docker-Datenbank für die Entwicklung gespeichert werden. Idealerweise werden die Daten nicht von einer GDI-Datenbank in die Entwicklungsdatenbank kopiert, sondern mittels INTERLIS-Transferdateien, die für die Entwicklung ebenfalls als Teil des Codes betrachtet werden können. Denkbar ist auch, dass diese Transferdateien als HTTP-Ressource zur Verfügung stehen oder dass man mit kleinen Datenbankdumps arbeitet (nicht die ganze Datenbank).

Erklärtes Ziel ist, dass man unabhängig vom Entwicklungs-Device (Laptop, PC, …) und -Umgebung (Büroräume AGI, Home-Office, …) die Prozesse im des Teilprojektes ausführen kann.

Beispiel «Datenumbau, Export und Import Nutzungsplanung»: Die Daten der Nutzungsplanung müssen vom kantonalen Modell in das Rahmenmodell des ÖREB-Katasters umgebaut werden. Nach dem Umbau werden die Daten in eine INTERLIS-Transferdatei exportiert, validiert und in die ÖREB-Katasterdatenbank importiert (resp. ersetzt). Es muss eine Datenbank mit der Originaldatenstruktur vorhanden sein und ebenfalls eine Datebank mit der ÖREB-Struktur. Beides kann für die Entwicklung natürlich in einer Datenbank zusammengefasst werden. Die Daten können in die Originalstruktur mittels INTERLIS-Transferfile importiert werden. Ausgeführt wird der gesamte Prozess mit *Gradle* resp. *GRETL*.

Vor allem bei Softwareentwicklungen muss ab einem gewissen Zeitpunkt entschieden werden, ob die Software auch immer gleich in eine OpenShift-Test-Umgegung der GDI deployed wird.

1.3. Tests

Die entwickelten Prozesse und Artefakte müssten mit Tests kontrolliert werden. Falls immer möglich muss dies auch nach jedem Commit in *Travis* geschehen. Es dürfen keine nicht-funktionierenden Versionen in das Github-Repository committed werden.

1.4. Dokumentation

Ein wichtiger Bestandteil der Arbeiten ist die Dokumentation. Es muss insbesondere aus der Dokumentation hervorgehen wie man z.B. den Prozess startet oder die Software grundsätzlich betreibt. In den meisten Fällen wird es reichen die (technische) Dokumentation im Github-

Respository (mit Markdown geschrieben) zu verwalten. Informationen mit Auswirkungen auf andere Systeme oder mit Handlungsanweisungscharakter sind in diesem Dokument zu erfassen.

Das vorliegende Handbuch ist mit *Asciidoctor* geschrieben. Daraus lassen sich einfach andere Formate ableiten (HTML, PDF). Der HTML-Output wird zusätzlich auf Github-Pages nach jedem Commit deployed.

1.5. GRETL

In einem frühen Stadium soll für die Entwicklung der Prozesse, die *GRETL* benötigen nicht das heutige Docker-Image verwendet werden, sondern es muss mit einer aktuellen, «rohen» *Plugin-Version* gearbeitet werden, da mit grosser Wahrscheinlichkeit Features von darunterliegenden Bibliotheken (v.a. *ili2pg* und *oereb-iconizer*) verwendet werden müssen, die noch nicht im heutigen Produktionsimage vorhanden sind. Oder aber es fehlen sogar noch Features in diesen Bibliotheken, die während der Entwicklung des ÖREB-Webservices entstehen werden. Wie und wann das Docker Image ebenfalls mit den nachgeführten Basisbibliotheken publiziert wird, ist noch ungeklärt.

1.6. ili2pg

Ili2pg erhält in der Version 4.x praktische, neue Features:

- SQL-Queries zum Erzeugen des Schemas und der Tabellen (inkl. der Werte von Aufzähltypen) können ohne vorhandene und laufende Datenbank erzeugt werden.
- Der Primary Key («t_id») muss nicht mehr schemaweit eindeutig sein, sondern nur noch pro Tabelle. Dies sollte den Datenumbau erleichtern.

Für die Entwicklung des ÖREB-Katasters ist daher eine Version 4.x zu verwenden. Es ist angedacht, dass *GRETL* möglichst rasch ebenfalls auf der Version 4.x basiert.

1.7. Erzeugen der Symbole

Für jeden ÖREB (resp. Typ resp. Legendeneintrag, siehe «Transferstruktur») muss das Symbol (PNG-Datei) der Legende erzeugt werden. Das Erzeugen der Symbole kann mittels GRETL-Task gemacht werden, der wiederum einzelne WMS-GetLegendGraphic-Request absetzt und das Symbol in der Datenbanktabelle speichert. Da der WMS auf die Daten zugreift, die in diesem Schritt (wo auch das Erzeugen der Symbole geschieht) erstellt werden, handelt es sich um ein Huhn-Ei-Problem. Eine mögliche Lösung ist, dass man mit *Docker* und einem sehr kleinen Beispieldatensatz (GeoPackage-Datei) einen eigenen WMS baut und diesen bei Bedarf hochfährt. Ein passendes Docker-Image für QGIS-Server gibt auf hub.docker.com (Version 2.18 ist ausstehend).