

Projekt 1: TextureSync

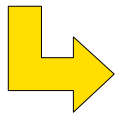
Texturen-Sammlungen einfach und sicher verwalten



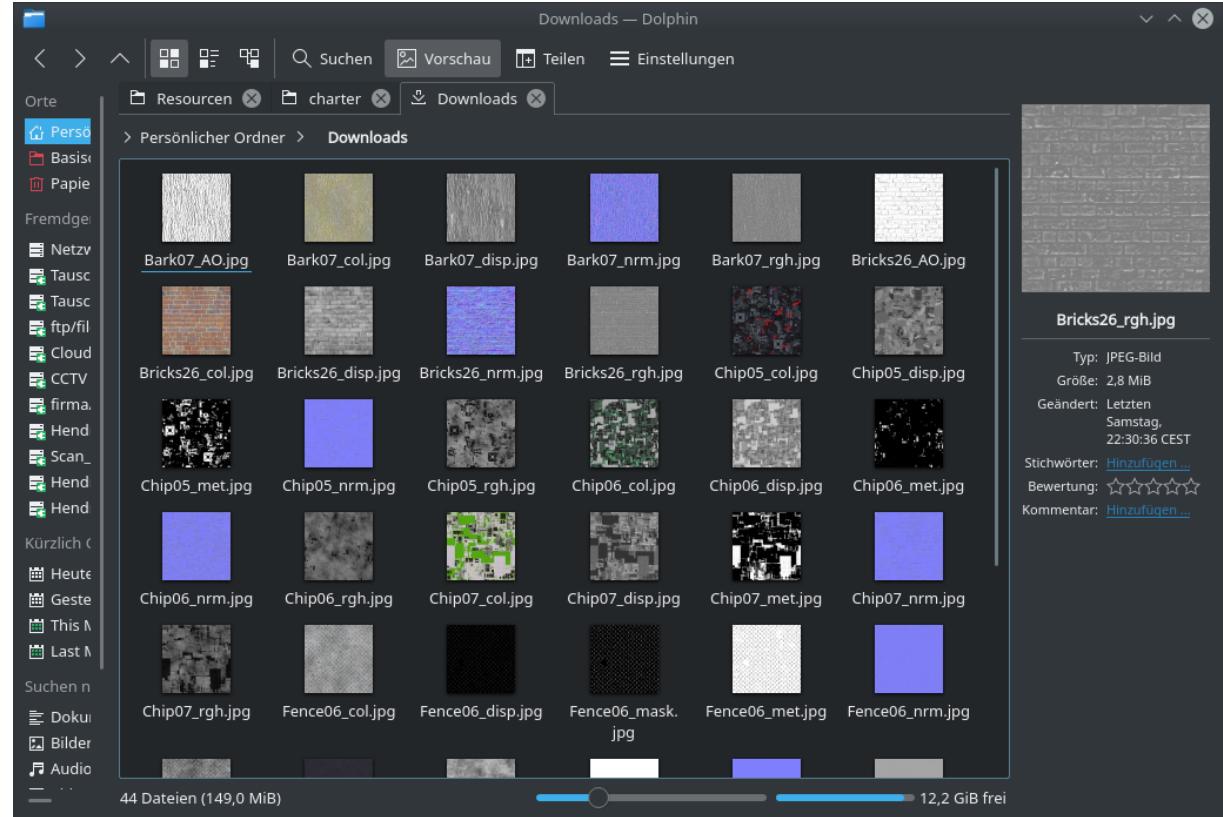
- Problem
- Lösung
- Artefakte
 - Lastenheft
 - Pflichtenheft
 - Grobdesign
 - Tests
- Technologien
- Fazit
- Demo

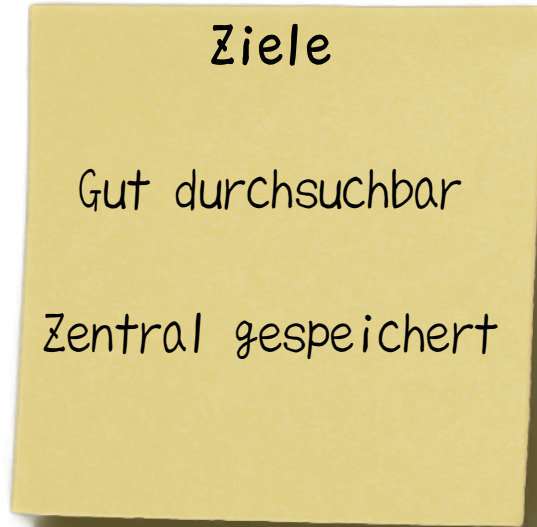


- Nicht durchsuchbar
- keine Sortierung
- Dezentral gespeichert
- keine Sicherung

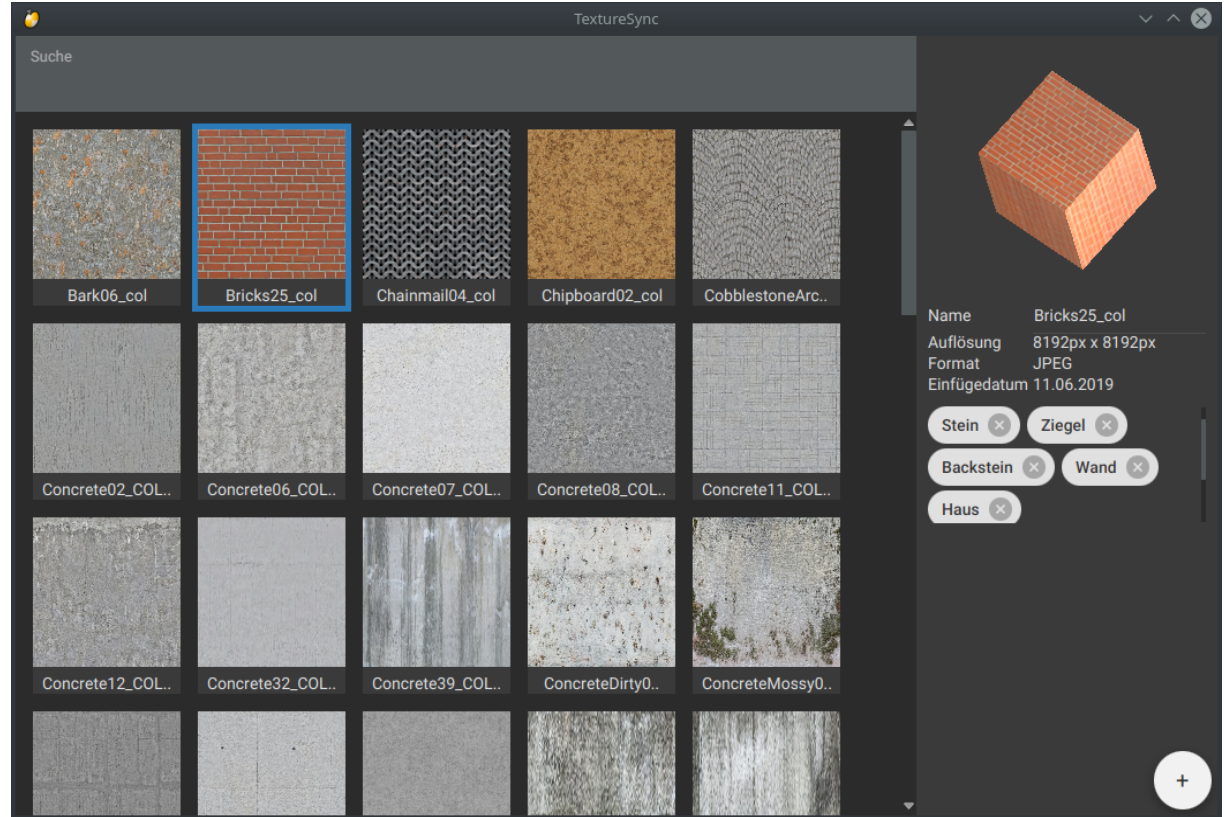


Verwaltungsaufwand



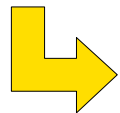


<https://upload.wikimedia.org/wikipedia/commons/e/e5/Post-it-note-transparent.png>

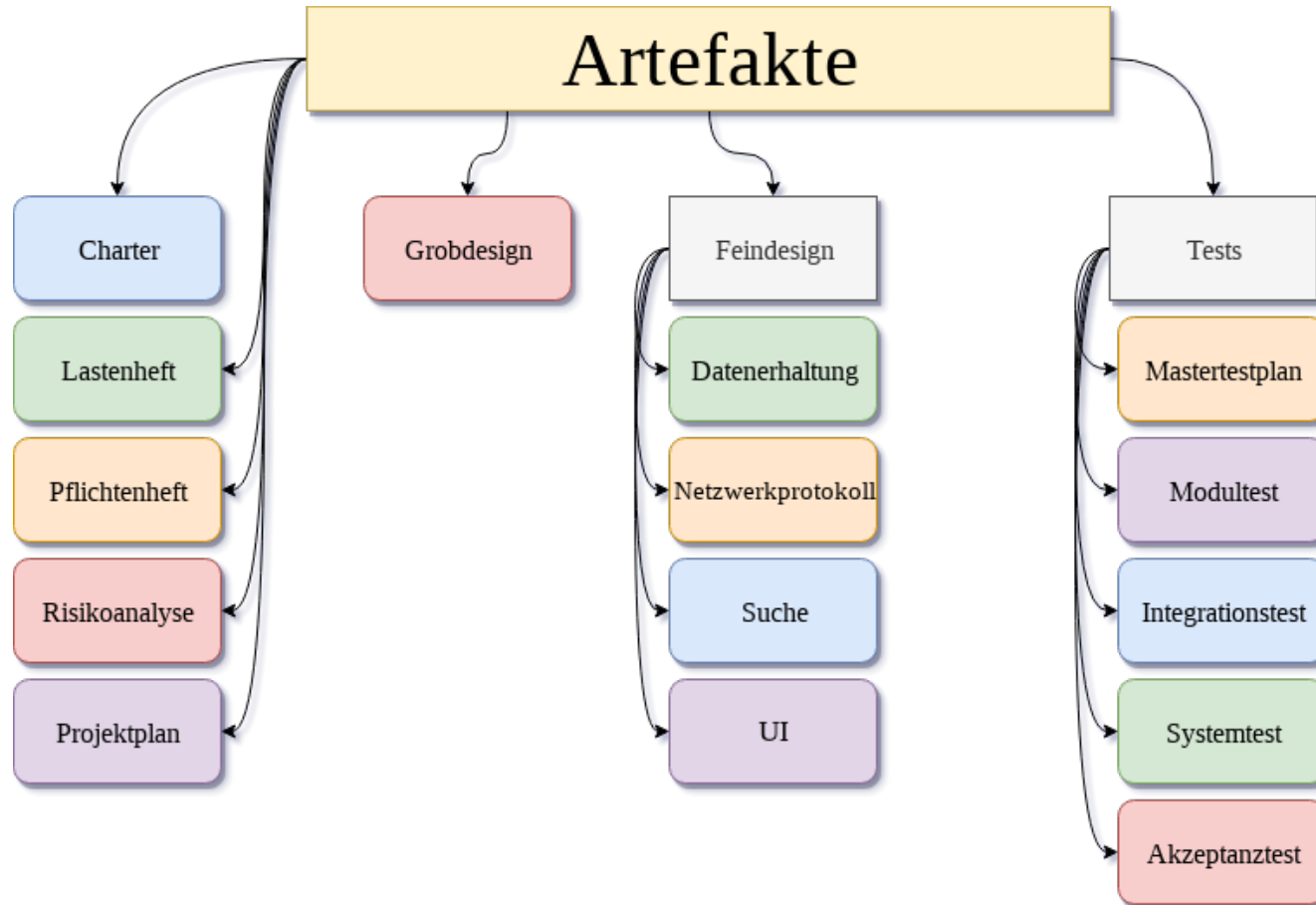


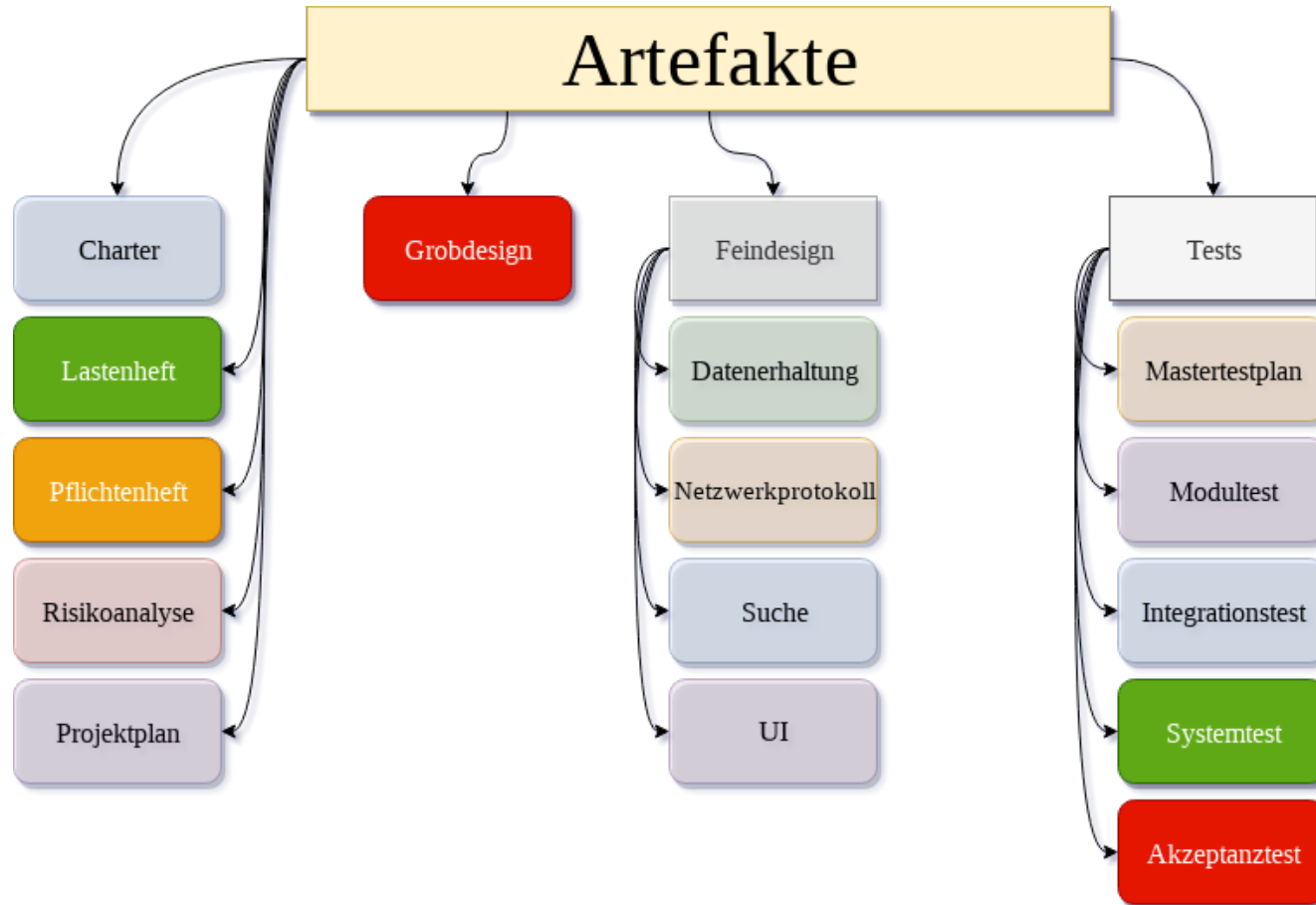
Auszug aus Projekt-Charta:

Kriterien	
Wichtigste Funktionen	<ul style="list-style-type: none">• Texturen anzeigen und verwalten• eine Preview beim Durchsuchen der Texturen• ein Tag-System, um Texturen zuzuordnen• Filter für Metadaten und Tags• Synchronisation mit zentralem Server
Akzeptanzkriterien	<ul style="list-style-type: none">• Das Durchsuchen darf nicht länger als 1 Sekunde bei 1000 Texturen dauern.• mindestens 10 Clients gleichzeitig aktiv



Kunden: Animationsstudios, 3D-Designer, Grafikagenturen



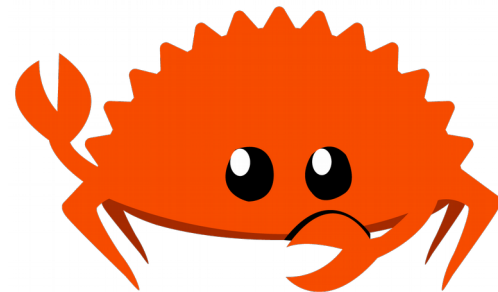


Server: Rust

- Compilersprache mit ähnlicher Performance wie C++
- Compiler verhindert gängige Fehler in Programmen
z.B. Seiteneffekte, use after free, race conditions, Pufferüberlauf
- cargo als Toolchain (ist Teil von Rust)
übernimmt Kompilierung, Testausführung, Abhängigkeitsverwaltung

Bibliotheken:

- image: Generierung von Vorschaubildern
- serde: Speicherung der Texturdaten in Json-Datei



<https://rustacean.net/assets/rustacean-flat-gesture.png>

Client Kotlin

- Kurze vorangestellt (Interpretersprache, benötigt .Net 2.1.4)
- Vorteile gegenüber herkömmlichen Sprachen (Java, C#, ...)
- Wie sieht toolchain aus (IDE, gradle)
- Verwendete Libs für das Projekt (tornadofx)

TODO (Hendrik)

- Planung ist alles
- Aufgabenpakete (abhängig, parallel)
- Fehler sofort dokumentieren und beheben
- Issue Tracker
- Versionsverwaltung
- Tests
- Wenn es geht nichts neues probieren, bewährtes beibehalten
- In Endnutzer hineinversetzen

DEMO