ABAP Product Development

Lars Hvam, Heliconia Labs, December 2022

Home	https://github.com/heliconialabs/abap-product-development
License	Creative Commons Attribution 4.0 International
Build	2022-11-23 09:30:56 UTC

1. Introduction

Additional functionality can be delivered to SAP systems via extensions. SAP partners typically develop these extensions and provide these to end customers.

ABAP extension development typically follows a release schedule with few releases each year. This whitepaper focuses on classic on-premise extension/product development.

Professional product development use central versioning like git, in the ABAP world the abapGit client can be used to connect the ABAP and git worlds.

It is possible to use the SAP Add-On Assembly Kit for product development, however its not a prerequisite.

2. Development

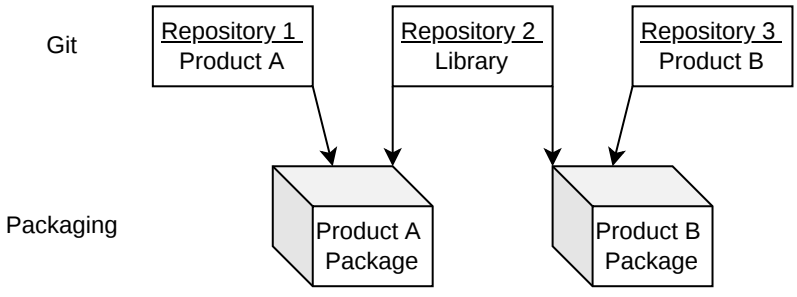
During product development, multiple choices must be made for how to organize and setup processes for the development, the below sections touch upon various development aspects.

2.1. Organizing repositories

Organize git repositories according to Conway's Law and which product options are provided to customers. Cross repository syntax checks are performed in the pull request process using abaplint.

2.2. Example 1

Two separately deployed products, one common library

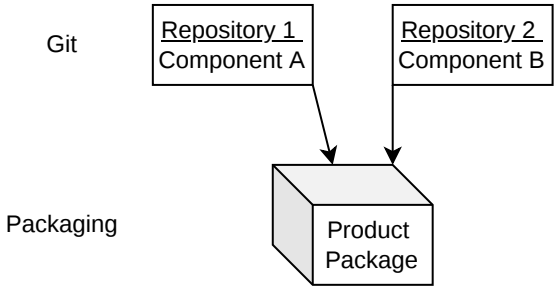


During packaging, two separately installable artifacts are generated.

Note that the library must be duplicated into each product package.

2.3. Example 2

One product, two teams with separate responsibilities



2.4. Quality assurance

All code is managed in git, all changes are reviewed by peers.

Check the ABAP code review guide for possible setups.

Static analysis is automatically triggered for each pull request, abaplint performs checks according to the team development guidelines.

If the product is developed in SAP BASIS systems higher than the target release, abaplint can help checking the syntax validity of the target language syntax.

Unit tests can be triggered in the pull request using Nuve Platform or using open-abap.

2.5. Development setups

The development landscape can be organized in various ways, a final setup will typically be a mashup of both central and de-central development.

2.5.1. Central Development

One working copy shared by all developers in one system, a single syntax error can impact all developers building and testing in the system

Automatic push to a branch in GitHub can be setup so developers does not have to dig into using git.

Note that a central system will always be required, to perform final testing and packaging.

2.5.2. De-central Development

Via Nuve Platform developers can instantiate additional systems, allowing for each branching and testing.

2.5.3. transpiler / open-abap

Alternatively, ABAP can be transpiled to JavaScript, allowing code to run on local developer machines or vanilla github actions.

This allows for easy branching, but is developer heavy, as all dependencies must be scaffolded.

Note that this does not replace real systems, which are required for final quality assurance.

2.5.4. Summary

Setup	Cons
Central development	Single working copy
De-central development	Extra infrastructure
transpiler / open-abap	Extra development effort

2.6. Duplicating code

If utilizing open or inner source libraries, duplicate objects to stay in control of the dependency. Automatic setups for updating the libraries is recommended.

- Vendor by default
- Automagic standalone renaming of ABAP objects

2.7. Reducing footprint into SAP

By reducing the footprint into SAP, eg. reusing of data elements. The product will become more stable, and easier to debug. If a dependency is the real semantic dependency, keep the SAP standard reference.

abaplint will list the unknown void types, plus provide observations in the pull requests when a developer adds new dependencies.

abapedia.org helps identifying which objects are marked as released in the latest Steampunk release.

2.8. Quality gate

Before packaging and distribution, perform a full system test for each target platform.

Nuve Platform allows easily spinning up different SAP releases for testing.

2.9. Packaging & Distribution

The product can be delivered to customers using classic transports or via abapGit ZIP files.

abapGit can be used from 7.02 and up, including Steampunk and S/4 HANA Cloud. The customer has full transparency into all changes before performing the import, code review can be done incrementally for each release by the customer. A developer can perform the upload via frontend PC.

Transport files are placed on the application server and imported via classic STMS, note the restrictions if generating transports on different releases than the customer system, 1090842 - Composite SAP Note: Cross-release transports. The customer cannot review the changes before import, and it requires access to the application server file system to save the transport files.

Note that product development can be done with abapGit, and still employ classic transports for distribution.

2.10. Versioning/tagging

Everything that is sent to a customer should be tagged in git, this way it is possible to track and reproduce errors locally instead of requiring access to customer systems.

For table contents, customizing and master data, use the abapGit data feature to push the database contents to git.

In case a hotfix is needed for a customer, a branch can be made from the tag, if the customer cannot upgrade to the latest version.

Use a versioning scheme like Semantic Versioning for the gags.

2.11. Customer Adjustments

After installation at a customer, the product might need additional adjustments and customizing by the vendor.

Put all all customer adjustments back to git, possibly a repository for each customer, depending on the actual content.

Do not adjust the product code in the customer system, instead provide the customer with a new release(triggering new versioning), or alternatively provide enhancement points in the product for customer-specific adjustments in customer namespace.