DSAG RECOMMENDATIONS

BEST PRACTICE GUIDELINES FOR DEVELOPMENT – USEFUL TIPS FOR ABAP DEVELOPMENT

Deutschsprachige
SAP® Anwendergruppe

DSAG

BEST PRACTICE GUIDELINES FOR DEVELOPMENT
USEFUL TIPS FOR ABAP DEVELOPMENT

Version:
2.0

Updated:
September 2016, translated February 2018

Authors:
Dr. Christian Drumm, Head of Application Development & Consulting,
FACTUR Billing Solutions GmbH

Martin Fischer, Portfolio Unit Manager SAP Database & Technology, BridgingIT GmbH

Judith Forner, Senior Consultant Finance & Controlling,
Mundipharma Deutschland GmbH & Co. KG

Edo von Glan, SAP Developer, Drägerwerk AG & Co. KGaA

Florian Henninger, Senior Consultant SAP Development, FIS GmbH

Martin Hoffmann, Head of Software Engineering, Miele & Cie. KG

Valentin Huber, Senior IT Consultant, msg systems ag

Jens Knappik, SAP System Architect, thyssenkrupp Materials Services GmbH

Dr. Christian Lechner, Principal IT Consultant, msg systems ag

Steffen Pietsch, Head of Backoffice, Haufe-Lexware GmbH & Co.KG

Daniel Rothmund, IT Business Analyst SAP, Geberit Verwaltungs GmbH

Holger Schäfer, Business Unit Manager, UNIORG Solutions GmbH

Denny Schreber, Senior Solution Architect, cbs Corporate Business Solutions
Unternehmensberatung GmbH

Andreas Wiegenstein, CEO, SERPENTEQ GmbH

Bärbel Winkler, System Analyst SAP Basis/Programming,
Alfred Kärcher GmbH & Co. KG

Further information on the authors can be found in Section 11 'The Authors'.

THANK YOU TO ALL THE PARTICIPANTS

Special thanks go to those SAP employees who actively supported compilation of the 2nd edition of the guide with their reviews, discussions and constructive ideas. In particular we would like to thank Jürgen Adolf, Carine Tchoutouo Djomo, Olga Dolinskaja, Thomas Fiedler, André Fischer, Dr. Thomas Gauweiler, Oliver Graeff, Michael Gutfleisch, Martin Huvar, Karl Kessler, Michael Schneider, Harald Stevens, Christoph Stöck, Dr. Wolfgang Weiss, Wolfgang Wöhrle and Margot Wollny.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

As standard software SAP is highly flexible and extensible. Almost every company that uses SAP software customizes the software, adds custom code or different kind of extensions. SAP software is consequently subject to a continuous process of modification and upgrading by both the manufacturer and the customer as a result of varying customer requirements.

This high level of flexibility and extensibility of SAP software has advantages and disadvantages. The software can be optimally adapted to meet customer-specific requirements and significantly increase value added as a result. At the same time extensibility exposes the risk of custom developments, which can be complex, difficult to maintain and prone to failure.

2012 saw the first edition of the DSAG Best Practice Guidelines published with the aim of providing practical tips for and impulses on the maintainable and efficient design of custom developments. The following years saw the guidelines translated into English. We received major readership feedback from the first version; not to mention the fact that much has changed in SAP development in recent years and a lot of innovations have been introduced. As a consequence, we now present you with the second, totally reworked and updated edition of the DSAG Best Practice Guidelines. The German version 2.0 was originally published at the DSAG Annual Conference in 2016. The English translation we present in February 2018. Hereby, we did not consider any updates but translated the content as of 2016.

1.1 MOTIVATION & YOUR COOPERATION

The work of the German-Speaking SAP User Group (DSAG) is based on three pillars – broadening the knowledge base, exerting influence and networking. The following document was drawn up by the DSAG working group SAP NetWeaver Development and addresses the first pillar, broadening the knowledge base for users and partners.

As the authors, our aim is to provide the implicit knowledge of development that is evident throughout companies to other DSAG members in the form of a compact document. Our goal is to see the document actively applied and the wealth of experience it encompasses continuously improved upon.

For this very reason we have established a community website that provides further information on the guide, author and other DSAG member contact information and also enables you to share your outlook with others:

www.dsag.de/leitfaden-abap

We look forward to your feedback!

1.2 POSITIONING

SAP and a whole range of specialist journals have produced excellent publications on application development, enhancing and upgrading the SAP platform. Throughout this guide we will be referring to what we regard as literature worth reading.

The added value of this document lies in the consolidation of established procedures, practical tips and well-proven rules applied in user companies. These guidelines provide users, developers and project, IT and development managers with recommendations and support so they can avoid 're-inventing the wheel' and instead build on the experiences of others. In the same breath, recommendations in this guide should not be interpreted as a set of hard and fast rules, but more as a selection of practical tips.

In our capacity as the authors, we have endeavoured to find the right balance between general knowledge and in-depth detail. As a consequence, we provide references to additional sources at relevant points to avoid unnecessarily reiterating topics that have already been thoroughly debated.

1.3 AMENDMENTS IN THE 2ND EDITION

The second edition of this document is structurally based on the first edition from 2012. Each section has been fundamentally checked for content and revised. As a result of DSAG member feedback, some recommendations from the first edition were updated, while others were comprehensively expanded on by the authors. The sections 'Development Environment' and 'User Interface' have also been newly added.

Even if you are well-acquainted with the first edition, we thoroughly recommend that you read and apply the recommendations in this comprehensively revised second edition.

2 PROGRAMMING GUIDELINES

This section defines firmly established and recommended programming guidelines for applications created using ABAP. It tells you how to achieve readable, maintainable, high-quality ABAP source code using standard SAP tools and discipline. This makes source code maintenance more straightforward and enables different internal and external individuals to work together efficiently on the (further) development and maintenance of programs.

Using official ABAP programming guidelines

Since the release of NetWeaver 7.31 SP5, official SAP ABAP programming guidelines have become a fixed component of ABAP keyword documentation. Developers can access these via the local system installation (transaction ABAPDOCU/F1 Help in ABAP Editor/ADT) or via the SAP Help Portal[1]. In addition to its comprehensive, high-quality content, the document has a further advantage over other development guidelines: instead of gathering dust in someone's desk drawer, it can be directly integrated into the development environment. Instructions for carrying out SAP Easy Access menu integration are available under SAP Note 1387086[2] or for SE80 integra-tion in the linked SAP Community Network (SCN) blog.[3]

SAP ABAP programming guidelines offer

- Extremely well-founded, practical recommendations
- Detailed information on each rule, including examples
- Availability in German and English
- Ongoing further development by SAP
- Interactive navigation to keyword documentation, release-dependent changes, performance and security themes, sample programs and transactions
- Use of all available ABAPDOCU functions (export as HTML or PDF, search, etc.)
- Well-known guidelines which have more or less established as a standard
- Can be extended by customer

1 Cf. *SAP Help Portal 'ABAP keyword documentation – NW7.50'*
2 Cf. *SAP Note 1387086 – HTML Viewer Control in SAP Easy Access screen*
3 Cf. SCN Article *'Enhancing the ABAP Workbench with a website containing dev guidelines!'*

Extension of ABAP programming guidelines

Extension the ABAP programming guidelines makes sense primarily if you are looking to incorporate areas not currently covered by the standard rules. These include object composition design principles (SOLID[4]/GRASP[5]) and architectural concepts such as SAP package concepts or the use of frameworks. Also conceivable is an additional listing of rules that are regarded as pivotal to your company. Refer to the appendix (Section B.1) for further details on how to expand ABAP keyword documentation.

BEST PRACTICE

- Make your development guidelines available within the the development environment to facilitate quick access.
- Use numerous transparent and verifiable examples within the development guidelines that can be re-used as code snippets.
- Use the official SAP ABAP programming guidelines as a benchmark. This extremely comprehensive docu-ment offers detailed recommendations for a broad range of activities.

4 Cf. *https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)*
5 Cf. *https://en.wikipedia.org/wiki/GRASP_(object-oriented_design)*

- If necessary consolidate the SAP document into a summarised version. Due to its scope and depth of detail, a comprehensive look at the official ABAP programming guidelines would be relatively time-consuming and partially encompass topics that are irrelevant for day-to-day development activity. If this document is to be used in your organisation, we recommend the assignment of a team to summarise the document and provide training on use of the document within your organisation. Especially in large organisations, it may be prudent to bring stakeholders from quality assurance and development into the team and possibly adapt certain points to meet the specific requirements of the organisation.
- If the document becomes an element of external service contracts, the external developer acceptance process should be incorporated into developer key activation and reference made to the document in the course of acceptance dialogue.

ADDITIONAL SOURCES:

1. Development Guidelines for Greenfield Implementation in sync with SAP Custom Code Management[6]

2.1 NAMING CONVENTIONS

Naming conventions specify uniform, binding guidelines for naming software objects (e.g. classes, function modules) and naming objects in the source code (e.g. variables).

We strongly recommend stipulating a naming convention as a guideline for any developments within the SAP system. The aim of using a standardised naming convention is to significantly enhance the maintainability of customer-specific modifications and upgrades. This consequently means lower maintenance requirements and costs as well as faster troubleshooting in the event of an error.

The explicitly formulated naming conventions should be a component of internal training to familiarise new employees with the basic rules and the specifics of the company. In addition, making the naming convention the subject of contracts with external developers and partner companies is also an established procedure. Automated checks ensure subsequent compliance with the convention (cf. Section 2.9 and Appendix A).

BEST PRACTICE
Appendix A contains an example of a naming convention.

6 *Cf. SCN article http://scn.sap.com/docs/DOC-56285*

2.2 NAMESPACE

The separation of customer objects and SAP objects can be facilitated using the prefixes Y and Z and also via a customer-specific namespace. The syntax is as follows:

Z…

Y…

/<customer-specific namespace>/…

An existing SAP customer can register a customer-specific namespace with SAP free of charge. Following confirmation, that namespace is exclusively registered worldwide for use by the respective company. This procedure reinforces conflict-free assignment of names for software objects.

The advantage of a customer-specific namespace is that there are no conflicts when importing external objects into the company SAP system (e.g. when using external applications that are imported by a transport request) and when combining SAP systems within the scope of a post-merger integration. With the reservation of the namespace, a software object with the same prefix cannot be created on any external system, i.e. a system that is not registered to the namespace in question.

One disadvantage of using the customer-specific namespace is that through consistent use of the prefix, object naming will inevitably 'use up' numerous characters. This can make assigning a name difficult, especially for objects that only have a few characters available for naming. Moreover, not all object types support the use of namespaces, e.g. authorisation objects. The same applies to some SAP tools und frameworks: although their use makes sense and is advisable, such utilisation can be problematic as the use of namespaces in partner and customer development can be inconsistent and even non-existent. We therefore recommend checking this situation prior to any utilisation of new tools or frameworks.

BEST PRACTICE
We recommend using a customer-specific namespace.

ADDITIONAL SOURCES

1. http://help.sap.com (Creating namespaces)
2. Note 105132 – Reserving namespaces
3. Note 84282 – Developing namespaces for customers and partners
4. SAP Support Portal: http://support.sap.com/namespaces

2.3 READABILITY AND MODULARISATION

Creating a clearly readable source code that is easy to understand is not simple and requires discipline and professionalism. Yet the effort invested is certainly worth it in the long-term, above all in the case of durable applications involving continuous maintenance and upgrading tasks. But what defines readable, easily comprehensible source code? If we look beyond ABAP[7], then it is all about the simplicity of the source code. In order to achieve this, we suggest the following approach:

- Use natural language ('meaningful' naming of variables, procedures etc. to reveal your intention)"
- Consistent use of Domain-specific vocabulary
- Avoidance of abstract, unclear terms and abbreviations

7 *Pertinent authors/books on this topic:*
- *Robert C. Martin: Clean Code: A Handbook of Agile Software Craftmanship. Prentice Hall, 2008*
- *Steve McConnell: Code Complete, 2nd Edition. Microsoft Press 2004*
- *Martin Fowler: Refactoring – Improving the Design of existing Code. Addison-Wesley, 1999*

- Uniform source code structure (indentation, writing style)
- Foregoing individual programming styles in favour of a common standard
- Modularisation
- No overly-long/complex procedures
- Avoidance of global variables

What are the specific characteristics of the ABAP environment?

- Avoiding abstract terminology is sometimes difficult. Older developments (classic SAP modules) in particular contain numerous German abbreviations and identifiers that make it harder to come up to speed for international project teams or junior developers with little SAP background knowledge.
- The development environment can sometimes significantly limit the use of natural language for naming repository objects and other identifiers. Furthermore, in contrast to other programming languages, ABAP uses a global as opposed to a package-based namespace. In combination with prefix namespaces (/*/), such conditions promote the use of abbreviations that are difficult to read and may be detrimental to developer productivity.

These specific characteristics aside, the above principles can be positively implemented. Refactoring tools from the ABAP Development Tools (ADT) for Eclipse (cf. Section 9) can be extremely helpful in this respect. For example, utilisation of ADT enables the renaming of individual variables or entire classes for all occurrences in one go. This minimises the effort required for correcting an unsuitably selected identifier and also reduces the risk of error.

BEST PRACTICE

- Consider the aforestated procedures during developer selection and training, project scheduling (time pressure during initial development can result in source code that is difficult to maintain with consequently higher follow-up costs) and the planning of quality measures (automatic and manual inspections).
- Use uniform, natural language with consistent terminology for source code, within documents and during discourse with colleagues and customers. Use the SAP terminology database as a guide (transaction SAPTERM or http://www.sapterm.com/).
- Define standards for the consistent handling of identifiers, taking into account the extent of integration with SAP standard software (customer/product supplier), quantity of customer source code and the structural and procedural organisation of your SAP development department (generalists/specialists in national/international teams). Depending on the constellation of the aforestated aspects, the naming conventions (e.g. variable identifier sales_organisation instead of vkorg[8] or index instead of i) can facilitate a lot maintenance and further development activities. The following table serves to support any necessary decision making.

8 *The five-character field identifier in the classic modules is a special case. When used in structures, conversion to descriptive identifiers can mean a greater workload (and error risk) for mapping in both directions. As such it may be prudent to retain them (despite the poor legibility).*

Small/ centralised team	Large/ decentralised team	Identical native language	Multilingual team	Module specialists	Generalists	Lots of custom code	Little custom code	Add-on development
•	• •	•	• • •	•	• •	• • •	•	• •

• • •	Strong impairment of maintenance/further development through problematic readability/ identifier selection
•	Little impairment of maintenance/further development through problematic readability/ identifier selection

Code formatting

Clear and readable source code helps any developer to (re-)familiarise with the code. The easiest and fastest way to make and retain readable source code is to use the Pretty Printer / Formatter in the ABAP development environment or ABAP Development Tools. At the push/click of a button/shortcut, the selected source code is formatted in a standard form and the tool offers various configuration options. These can be found in the workbench settings within the classic SAP GUI workbench, while in ABAP Development Tools, the settings need to be made in the project-specific formatting properties. We recommend to use source code indentation, with keywords in higher case and identifiers in lower case. This enables the source code to be easily understood even in printed form and without syntax colouring. Pretty Printer / Formatter is a straightforward way to create a standardised source code layout. We recommend deactivating the option 'Insert standard comments', as the generated comments are not automatically updated following later changes and include redundant information.

BEST PRACTICE
We recommend using Pretty Printer / Formatter and defining the settings as a uniform standard. In addition to the settings in Pretty Printer, we also recommend using functional syntax for method calls and not to use the CALL methode statement.

Software structuring with the SAP package concept

With the roll out of SAP NetWeaver Application Server 6.20, the development class concept was completely replaced by the SAP package concept and also expanded by numerous features to improve software structuring. In addition to high-level structuring of the software, the primary tasks of the package concept are to embed related objects in a common framework, regulate access to objects and control dependencies to other packages. Explaining the details of the package concept goes beyond the scope of these guidelines, which is why we provide reference to existing SAP Help documentation[9] and the extremely comprehensive SCN article series.[10] Generally speaking, package structures can be formed on the basis of functional, technical and organisational criteria, whereby the following lists the most common criteria:

- Dependency on software components
- Assignment to SAP standard application hierarchy
- Reusability of development objects
- Grouping of individual applications
- Stability of development objects
- Layer-specific/technical affiliation
- Organisational assignment of incorporated objects
- Translation relevance of development objects

9 *Cf. https://help.sap.com/viewer/search?q=package%20builder&language=en-US&state=PRODUCTION&format=standard,html,pdf,others*
10 *Cf. SCN article Tobias Trapp 'ABAP Package Concept – Part 1 – 4"*

BEST PRACTICE

- Use structure packages to highlight the dependency of various software components, meaningfully group your own comprehensive developments or when planning to use the package check.
- Within the scope of development, check if a main package for the commensurate top-level application components (SE81) already exists in the customer namespace and create one if not.
- Allocate packages to application components that are semantically most suitable.
- Use package hierarchies to organise your software system. Each software system should have at least one main package that consolidates its sub-systems and essentially defines its purpose.
- Use development packages to centrally bundle semantically and technically coherent repository objects.
- Avoid reciprocal dependencies between two or more packages. Transfer dependent components to separate packages so that only unilateral dependency exists.
- Arrange packages with unilateral dependency as far up the software system package hierarchy as possible. If functions are needed by multiple software systems, create a general foundation or basis package.
- Group repository objects using a package-specific namespace prefix. Use the application hierarchy or a product abbreviation in this respect (example: all objects in the package hierarchy for the main package Z_SALES_SUPPORT are given the prefix SAS: class ZCL_SAS_FIELD_WORKER, master data table ZSASFIELDWORKERS, etc.).

Package interface and package use access

Package access on all package levels is defined via the package interface and use access. Whereas the package interface serves to disclose reusable package components, use access helps to control access to components. If a developer does not pay attention to the package interface, and uses standard SAP components that are not released through the package interface, this can impact considerably on the customer as a result of necessary subsequent upgrading. The possible results of using unauthorised components are described by way of example in the SCN blog.[11]

Access to components of other packages can be optionally controlled via the package check concept. Further details on activating the package check are covered by SAP Note 648898[12]. SPAK_API package tools can be used and package access controlled following activation.

11 Cf. http://scn.sap.com/community/abap/blog/2013/01/28/is-sap-nw-ehp-3-really-non-disruptive
12 Cf. https://launchpad.support.sap.com/#/notes/648898

BEST PRACTICE

- Prior to reusing a repository object from a different package, first check if the object has been released in a package interface. Only objects released via a package interface can guarantee long-term stability. In SAP standard, using such objects reduces the risk of changes caused by a release change.
- Provide access to package objects through the definition of package interfaces. Only stringent package interface compliance can ensure that the defined package structure is maintained and no disruptive access to package objects occurs.
- If necessary provide more package interfaces and group the interfaces on the basis of semantic information, for instance data manipulation or reporting tools, or according to utilisation criteria.
- If required, move your package interfaces further up the package hierarchy where you have the possibility of further limiting access or enabling access to released elements from outside the system.
- Limit the use of critical functions by providing separate package interfaces for these objects. Interface usage rights can be determined through defining restriction on user packages.

Package namespaces

In addition to the generally known customer namespaces Y*, Z* and /namespace/, via the package concept SAP also offers the namespaces $* and T*. Every developer is no doubt familiar with $TMP; but did you know that you can also use those two namespaces for packages with local objects in your own package hierarchies? The difference between the namespaces is that $* cannot be transported and the developer has to carry out versioning manually. By contrast, T* package contents can be manually transported to all systems that are not designated production systems by way of a transport of copies. From a software structuring perspective this gives rise to the following benefits:

- It is easier to distinguish unfinished or prototypical components from productively used components.
- Production systems are not overloaded with unnecessary and potentially security-critical functions.
- Maintenance and further development costs can be reduced because necessary changes only have to be carried out on productively used packages.

BEST PRACTICE

- Use a suitable namespace for structuring your packages. Take into consideration the relevance of transporting and the purpose of the development (local test development/test development with deviating transport destination).
- Ensure that local developments do not enter the production system.

Using the package concept

Comprehensive utilisation of all the features of the package concept will not equally suit every company and if designed incorrectly can lead to unwarranted additional costs. We have created the following table to help you in the decision making process. The valuation criteria are based on the value range (+) = less important to (+++) = very important.

	Small cen-tralised team	Large/ decen-tralised teams	Regional special develop-ment	Transla-tion activities	A lot of custom code	Little custom code	Add-on-develop-ment
General package concept	++	+++	++	+++	+++	++	++
Package interfaces	++	+++	+++	+	+++	+	+++
Structure packages	+	++	+	++	++	+	+++
Package testing	+	++	++	+	+++	+	+++

Modularisation

Programs in which the logic units of work are overly long and not separated are consequently harder to read and therefore difficult to maintain and extend.

A modularisation unit (form routine, method or function module) consolidates logically cohesive statements that perform a clearly defined task. However, ensure that you do not end up with a lot of very small individual units that perform only trivial functions. Avoid modularisation units with only a few statements unless these improve readability of the source code and as such are part of the source code documentation.

Modularisation serves to clearly arrange the source code, despite the complexity of the task in hand. See also Section 2.12 'Programming model: object-oriented vs. procedural'.

ABAP Development Tools (see Section 9) provide comprehensive refactoring options, for example automatic method extraction, which facilitates subsequent modularisation of the source code (naturally there are limits, which means some manual editing may be required).

Avoiding global variables

In addition to structuring the program text, modularisation also involves restricting the visibility of variables.

Global variables destroy program modularisation. They make readability and maintainability considerably more difficult. Debugging and extension of code that uses mainly global variables is often only possible on a 'trial and error' basis.

BEST PRACTICE
Avoid all global variables.

Limitations of the rule

In smaller programs with only a few simple modularisation units, global variables are usually harmless (due to its small size, the program is normally fairly transparent and 'hidden' or less obvious side effects due to changes in global variables are unlikely). Unfortunately, global variables cannot always be avoided. In particular, they are still required for the elements of classic screens and selection parameters.

Further exceptions are the printing programs using the SAPscript technology, which explicitly works using global variables. For readability reasons, always work with returning parameters in this case, which are then correspondingly made available in global variables.

Avoiding code copies

Duplicate or multiple source code (clones) makes debugging and function modification or extension more difficult because all the copies have to be found and adapted accordingly. Moreover, because the copies are usually no longer totally identical, ascertaining whether differences exist and where these differences are takes a great deal of time and effort.

Copying SAP standard code is a special case. See Section 8.4 'Adaptation of SAP functionality'.

BEST PRACTICE
Adhere to the DRY principle: 'Don't repeat yourself'[13] and split off the repeated source code section and transfer it to a separate method.

Remarks

A more generous formulation of the DRY principle is the Rule of Three[14], which allows for a second copy.

When searching for copies of code[15], generated source code should be excluded as this constitutes a deliberate copy.

Multiple statements in one line, method chaining

To increase source code readability, we recommend avoiding multiple statements on one line.

One exception to this is so-called method chaining. Chaining method calls avoids variables introduced merely to hold intermediate results, thus creating a more readable source code. However, extensive method chaining, particularly in combination with less descriptive method names, can reduce readability. As such there is no all-encompassing guideline.

13 Cf. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
14 Cf. http://en.wikipedia.org/wiki/Rule_of_three_(programming)
15 For example with SAP Clone Finder, CCAPPS transaction code

2.4 SEPARATION OF PRESENTATION AND APPLICATION LOGIC

A separation of presentation and application logic should be carried out in all programs. This enables application logic results and functions to be displayed to the user via a variety of user interfaces (UIs) and also to provide these to other systems via a standardised interface. This rule applies to all standard UI technologies, but the degree of support to such logic separation can vary. Implementation in Floorplan Manager/Web Dynpro for ABAP already allows for separation of the model and UI logic within the framework. Separation is not facilitated in the same way in the case of classic Dynpros and BSPs, but separation can and must also be carried out within these environments. However, no technical checks are available in this case, unlike with Floorplan-Manager/Web Screens, where commensurate checks are undertaken in Code Inspector. The same principles naturally also apply for UI5 development (see Section 10), in which user interface technologies already support the model-view-controller pattern and the application logic is provided by the respective backend.[16]

Plausibility rules are a typical example of explicit separation of the application logic and UI. When a plausibility check of input is developed in a specific UI technology (in the presentation layer), if the UI technology is changed, these checks have to be re-developed. In order to avoid this, the functions for checking input or parameters should be created and maintained independently of the applied UI.

Most of the time it also makes sense to keep the source code for the data model and/or database access separate, be it via a separate local class (for small developments not targeted for reuse) or a framework such as BOPF (Business Object Processing Framework[17]). This complies with the classic model-view-controller pattern.

16 SAP roadmap for user interface strategies
17 http://scn.sap.com/community/abap/bopf

2.5 INTERNATIONALISATION

Language-dependent texts in programs should not be 'hard coded', but stored in text elements (program text, class text, online text repository [OTR]), standard text or message classes. As all independent developments should be of a standard that can be applied worldwide, all the texts should be translated into the commensurately principal languages.

Configurable language-dependent texts are stored in individual text tables. A text table possesses the same key attribute as the associated customising table and is referenced to this via a foreign key relationship. In addition, the first key attribute after the client field must be the language attribute (data element SPRSL or SPRAS).

BEST PRACTICE

- We recommend using Code Inspector or ATC to search for texts that cannot be translated.
- To create subsequent translations more easily, the length of the descriptions and text elements should be as long as possible. As a rule of thumb, the length of text elements should be 1.5 times the length of the native description.

To reduce the outlay for translations, the following aspects should be considered when establishing a translation strategy:

- Use English as the core source language
- Define the depth of translation work (e.g. only surface text, surface text and F1 Help functions, complete translation) using the translation levels defined by SAP[18]
- Assign packages to the SAP application hierarchy to enable use of domain-specific text recommendations for the distribution of top texts
- Label technical text tables that are not relevant to translation
- Use long text objects (e.g. via SO10) instead of dividing text into a number of lines
- Use uniform terminology and spelling supported by the terminology database
- Avoid wildcard characters (&) in message texts (instead use &1 &2 &3 &4, because the position of such within a message text may differ in the various languages)
- Work with supplementation language in systems different from the development system for filling translation gaps in the target language. Use pseudo language 2Q for technical user interface tests.[19]
- Use the SAP package concept for separating elements that are relevant and non-relevant to translation

ADDITIONAL SOURCES

1. Presentation 'Best Translation Practices in SAP Custom Development Projects' by Lindsay Russel (SAP SE)

18 Cf. https://websmp102.sap-ag.de/~form/handler?_APP=00200682500000002672&_EVENT=DISPLAY&_SCENARIO=01100035870000000122&_HIER_KEY=501100035870000008578&_HIER_KEY=601100035870000248115&
19 http://www.se63.info/pseudo-localization-sap-applications/

2.6 DYNAMIC PROGRAMMING AND AUDITABILITY

Dynamic programming

In 'traditional' static development, the development objects and source code are defined at design time and statically stored in the SAP system. The specified source code is then executed during runtime. In contrast, dynamic programming allows greater source code flexibility. Dynamic programming is illustrated by the following example:

The name of an ABAP class to be invoked is not stored statically in the source code; instead, an instance of a class whose name is provided during runtime (with the name stored in a variable, possibly coming from customizing), is created. The name and consequently the specifically executed implementation may well vary, for example, as a result of user input.

An advantage of this methodology is increased flexibility, while the disadvantage is greater complexity. Depending on the type of dynamic programming, security risks may also be an issue. Tools for automatically recognising security risks do not function as well with dynamic source code.

Advantages:

- Increased flexibility
- Better reusability
- Avoidance of copies and consequent minimisation of the amount of source code requiring maintenance

Examples of advantages:

Example 1 – Own user exit setup

The static definition of an abstract class incl. method signature forms the basis for a 'user exit'. Numerous specific implementations of the abstract class can then be subsequently created. At runtime, for example, the name of a specific class implementation is read from a customising table and invoked. Customising can consequently be used to activate/deactivate a variety of different implementation variants.

Example 2 – Dynamic WHERE clause

At runtime, the WHERE clause for a database operation, e.g. SELECT, is created within a string variable. This avoids complicated CASE queries, which execute various open SQL commands depending on the input.

Example 3 – Replacement of so-called boilerplate code[20]

Dynamic programming can also be used to avoid re-implementing similar logic in almost identical variants. One example is a database access layer: the tables maybe different, but the algorithm is always the same. In this case the problem can be solved using dynamic programming or generating the source code. This takes more work, but depending on the scenario pays off with less maintenance effort.

Disadvantages:

- When using dynamic calls, the where-used list within the ABAP development environment is not available. Changing the called code is then a problem.
- When using dynamic programming no syntactic checking is possible during design time. This can cause an abort in program execution where variable content is erroneously configured (e.g. erroneous compounding within a dynamic WHERE clause, wrong class name).
- Dynamic programming brings with it increased security risks if the dynamic content can be influenced due to unprotected access (e.g. where a WHERE clause can be influenced by user input; keyword: SQL injection).
- Increased source code complexity.

20 https://en.wikipedia.org/wiki/Boilerplate_code

BEST PRACTICE

- Dynamic programming should be used very conservatively and be well-controlled. Source code with dynamic content should be checked according to the dual control principle and documented as it poses a potential security risk. [21]
- When generating source code, a syntax check should be carried out after generation using the SYNTAX-CHECK command.
- For dynamic or generated source code, class CL_ABAP_DYN_PRG test mechanisms should be used, enabling whitelists for dynamic table access to be applied, for example.

Section 5.2 also addresses dynamic programming from a security perspective.

Auditability of ABAP source code

At all times it must be possible to check customer developed or generated ABAP source code for errors through manual inspection or by using static code analysis tools. All methods for concealing ABAP source code are impermissible as they impede such checking and can even be specifically exploited to place backdoors into a system. Hidden source code (e.g. macros) can no longer be inspected by the debugger, which makes auditability and error analysis all the more difficult. The techniques for hiding source code are explicitly not described within this document.

21 Code Inspector/ATC security checks can be used to find dynamic source code (pre-selection for dual control).

BEST PRACTICE

- Never apply methods for concealing source code and contractually agree this with any external developers.
- To avoid problems during maintenance activities carried out by different developers, we do not recommend to use the editor lock flag to prevent changes to source code. Instead, source code can be protected by assigning it to a separate package, for which access can be restricted using authorisation object S_DEVELOP with explicit values for field DEVCLASS.

2.7 NEW LANGUAGE ELEMENTS

ABAP syntax has been considerably expanded with NetWeaver 7.40 and 7.50.[22] New elements can be roughly divided into two areas:

1. Enablement of the expression orientation that is standard in many other programming languages

This replaces the former statement orientation arising from the ABAP origins as a macro language. ABAP developers should to be in a position to concentrate on the 'what' in terms of programming rather than the 'how' (see the many auxiliary structures in the legacy source code in the following example).

2. Support of the 'Code2Data' paradigm

In other words the transfer of data-intensive operations from the application server to the database, also called 'code pushdown'. The associated new language elements are partly available for all databases (extensions, Open SQL, Core

22 http://scn.sap.com/community/abap/blog/2013/07/22/abap-news-for-release-740
http://scn.sap.com/community/abap/blog/2015/11/27/abap-language-news-for-release-750

Data Service CDS views[23]) and partially only available for SAP HANA (AMDP – ABAP Managed Database Procedures). When considering the use of HANA-specific language elements, a balance has to be made between universal operability on all databases and performance optimisation for SAP HANA.

To illustrate the new potential offered by expression orientation in (1), here is an example featured in Horst Keller's blog (see below):

Source code in NetWeaver 7.0

```
DATA  itab TYPE TABLE OF scarr.
SELECT * FROM scarr INTO TABLE itab.

DATA wa LIKE LINE OF itab.
READ TABLE itab WITH KEY carrid = 'LH' INTO wa.

DATA output TYPE string.
CONCATENATE 'Carrier:' wa-carrname INTO output SEPARATED BY   space.

cl_demo_output=>display(  output  ).
```

Corresponding source code in 7.40 (SP08)

```
SELECT * FROM scarr INTO TABLE @DATA(itab).
cl_demo_output=>display( |Carrier: { itab[ carrid = 'LH']-carrname }| ).
```

The following language expansions in particular are fundamental:

Introduced in SAP NetWeaver 7.02:

- Inline calculations and inline method calls
- String templates (NW 7.40 example line 2: Parameter of the display method)
- String functions and regular expressions

In SAP NetWeaver 7.40:

- Inline declarations (in line 1: DATA(itab))
- Constructor operators VALUE and NEW
- Table expressions (in line 2: itab[...])
- New parser for Open SQL (in line 1: @ as a character for host variables)
- SQL expressions

BEST PRACTICE
We recommend learning and using the new language elements as they can be used to increase the development efficiency and improve source code readability. The above list is a good place to start.

23 *Still some restrictions in NetWeaver 7.40 (parameterised views only with SAP HANA), from NetWeaver 7.50 onwards, the CDS view functions are database-agnostic*

Also to be considered in this connection are team skills and the SAP NetWeaver release used within the system landscape. Older SAP versions do not always support the new language constructs or the Code2Data paradigm. If developers are frequently working on different systems with different SAP NetWeaver releases, then as much common ground as possible should be found to subsequently avoid even greater complications during the transfer to legacy systems.

Further information on the new language elements is available in the ABAP Keyword Documentation and Horst Keller's SCN blogs.[22]

2.8 OBSOLETE STATEMENTS

SAP has a strict backwards compatibility policy. Nevertheless, be aware that using obsolete statements such as headers in internal tables will cause problems if the code is transferred to classes. More up-to-date alternatives are always available for obsolete language elements. Habit aside, there is little reason to use these elements and the use of such should consequently be avoided.

BEST PRACTICE
We recommend the regular application of a static code analysis tool to discover obsolete statements. Suitable SAP built-in solutions include Code Inspector/ATC and using the extended program check. To minimise the amount of work required, obsolete statements should always be replaced when development objects are changed anyway in the course of other change requests. Usually, this requires only little additional test effort. In addition, excellent third-party analysis tools are available.

2.9 AUTOMATIC CHECKING OF DEVELOPMENT OBJECTS

SAP provides various tools for automatically checking development objects during design time:

- Simple syntax checking is carried out automatically upon activation and prevents the activation of faulty development objects.
- Extended Program Check can be executed for specific, already activated source code (programs, global classes …) and will report potential problems using three priority levels.
- Code Inspector provides a comprehensive catalogue of checks, some configurable, from which a check variant can be assembled. This includes Extended Program Check. Numerous performance and stability tests are also encompassed. In addition, internal company checks can also be programmed and incorporated.
- The latest tool is the ABAP Test Cockpit (ATC)[24], which uses Code Inspector check variants and also has a number of its own additional functions.

These tools can be used by developers during the development process and are all well integrated in the ABAP Workbench and ADT (exception: Extended Program Check is not directly accessible from ADT).

Code Inspector or the ATC can also be globally executed using transaction SE03 to enable the early recognition and rectification of problems and vulnerabilities during the release of transport requests. This saves spending unnecessary time and effort on transports, testing and error handling and improves program performance, stability, security and maintainability.

In SAP standard, Code Inspector or the ATC are only activated on the release of a transport request. The recommended process (and the only meaningful option for transport management with transports of copies – see Section 8.1.4) is to carry out checking upon release of the respective transport task or after completion of the development objects by the developer. This requires implementation of BAdI CTS_REQUEST_CHECK.[25]

24 *Available as of NetWeaver 7.4 or specific SPs in 7.0 and 7.3.*
25 *How this is implemented for the ATC is specified http://scn.sap.com/community/abap/blog/2016/04/20/how-to-perform-an-atc-check-automatically-during-transport-task-release*

BEST PRACTICE
Many of the rules and recommendations for development objects in these guidelines can be checked automatically with the tools. Developers should therefore be familiar with the tools and they should be applied early on in the development process. General testing with Code Inspector or ATC on the release of transports should be configured. Selection or configuration of a check variant is a prerequisite in this respect. Who may grant exemptions in the case of failed checks should also be established (e.g. quality assurance officer or development manager).

Various third party tools are also available to check source code for infringements of development guidelines even at design time and provide developers with direct feedback on certain code quality characteristics. In addition to design time checking, some of the tools also offer the option of automatic correction.

Section 7 and particularly subsection 7.2.2 'Automatic testing' contain comprehensive recommendations for this area.

ADDITIONAL SOURCES:

The procedure for implementing BAdI CTS_REQUEST_CHECK for Code Inspector is specified in the 'SAP Code Inspector User Manual' (SAP Press), as well as in this blog.

The book also describes the integration of a number of customer checks in Code Inspector/ATC. A blog is also available on this topic.

2.10 HARD CODING, MAGIC NUMBERS

Problematic hard coding is regarded as the coding of data such as texts, numbers, user names etc. directly into a program, which may then change; for example, because they actually comprise configuration information (interface paths…). Hard coding initially saves development time, but inevitably leads to higher costs over the entire lifecycle of the application because, for example, configuration changes would necessitate modification of the program. If the same value is hard coded in numerous programs or subroutines, then an additional problem arises when a change is required all the positions have to be modified (see also 'Avoiding code copies' in Section 2.3).

A related theme involves Magic Numbers, in other words numbers that are used in a program without explanation. The problem here is one of program readability and maintainability. Again, costs may also be saved initially in this case, but nevertheless are likely to be incurred on later modifications, perhaps at an even higher rate if the meaning of the magic number first needs to be determined.

A further aspect when using hard coding and magic numbers is that this may be a sign of implementation without any consideration for object-oriented concepts. One example being CASE statements on constants, which can be avoided either using the refactoring method 'Replace Code with Subclasses'[26] or via filter BAdIs[27].

BEST PRACTICE
Problematic hard coding should be avoided. Magic numbers should be replaced by distinctly named constants or furnished with a suitable comment.

The use of constant names that are identical to the value is to be expressly avoided as this will give rise to faulty information if the value is changed. (Paradoxically, changeability is the main reason for using constants.)

26 https://sourcemaking.com/refactoring/replace-type-code-with-subclasses
27 See also http://scn.sap.com/docs/DOC-10286

The definition of constants should be in the form of an attribute in a specifically dedicated interface or an abstract, final class. We do not recommend the use of constant includes. Within the scope of large scale developments, it may be a good idea not to store all the constants in one interface, but to distribute the constants to different interfaces according to business topic. Additionally, interfaces can be furnished with a tag interface (the same as BAdI Interface IF_BADI_INTERFACE). This ensures the discovery of interfaces that contain constants.

When introducing constants ensure that these can be found by other developers, otherwise identical constants may be potentially duplicated on the system. Unfortunately, there is no dedicated SAP tool that assists with this task. An alternative is to access the SAP ecosystem and use the openly available tool ConSea.[28] One option is to document the constants within an interface outside the system, but it is likely that maintenance of the external documents will be neglected. As of NetWeaver 7.50, ABAP Doc documentation can be exported like with JavaDoc. If the TREX search is installed and configured, this can be used to carry out a performant search.

2.11 AUTHORISATION CHECKS IN SOURCE CODE

The required authorisation objects have to be checked to access and present data. When standard objects are used, the check should utilise the corresponding SAP standard authorisation objects (simplifies maintenance of the required roles). SAP standard authorisation objects cannot generally be used to check customer data objects. Customer authorisation objects can be implemented and checked for this purpose. For further information see Section 5.1.1.

2.12 PROGRAMMING MODEL: OBJECT-ORIENTED VS. PROCEDURAL

It is advisable to move from a procedural programming model to object-oriented programming to promote future-proof development and encapsulate objects. Object-oriented development should be used exclusively in the case of new projects.

28 Cf. GitHub

Object-oriented development was designed so that logically coherent operations can be uniformly encapsulated within objects. Amongst other things, this increases the reusability of source code. In particular, such objects can be easily extended or changed by other developers to meet their needs without compromising the basic functions (open-closed principle). Additionally, core functionalities or individual variables can be specifically protected against undesired read and write accesses by invoking programs. An initial overview of the basic principles of object-oriented design can be found, for example, on Wikipedia.[29]

The introduction of object orientation into ABAP with ABAP Objects went hand-in-hand with a revision of the language and the unification of constructs. Utilisation of ABAP Objects therefore results in enhanced maintainability. For smaller programs[30], this enhanced maintainability can be achieved by replacing FORM routines with the class methods of local class lcl_main without the need to undertake a strictly object-oriented (re-)design.

SAP has now categorised procedural development in ABAP, for example using FORM routines, as obsolete. In recent years, procedural development has also proved to be extremely opaque, complex and error-prone due to global variables and includes. This is a further reason for switching to object-oriented programming.

BEST PRACTICE
We recommend that wherever possible new development projects are to be implemented using the principles of object orientation with ABAP Objects. Obsolete procedural ABAP language constructs such as FORM routines should be avoided and replaced with constructs from ABAP Objects. This recommendation is congruent with the specific rules 'ABAP Objects as a programming model' under the programming guidelines in the ABAP keyword documentation.

29 https://en.wikipedia.org/wiki/Object-oriented_design
30 *The upper limit of a 'small program' in this case is deemed as being between 200 and 500 lines, depending on the coding style and the complexity of the implemented functions.*

Possible counter-arguments:

- Available ABAP Objects team expertise and the planned and realistically expected development of expertise
- Systems with older Basis/NetWeaver release that include ABAP Objects that are less mature (prior to SAP Web Application Server 6.20) or in systems/modules in which procedural components also dominate in SAP standard.

ADDITIONAL SOURCES

1. Horst Keller and Gerd Kluger, Not Yet Using ABAP Objects? Eight Reasons Why Every ABAP Developer Should Give It a Second Look, Sap Professional Journal
2. Horst Keller and Gerd Kluger, NetWeaver Development Tools ABAP, SAP AG
3. Bertrand Meyer, Objektorientierte Softwareentwicklung, Hanser 1990, ISBN 3-446-15773-5
4. Bertrand Meyer, Object-Oriented Software Construction, Prentice-Hall 1994

2.13 DEVELOPMENT LANGUAGE

All developers should log on with the same language and maintain language-dependent texts in that language. Language-dependent texts should always be created in translatable form.

3 PERFORMANCE

In the following subsections we recommend a number of best practices that should be observed during day-to-day ABAP development work to ensure appropriate performance of the developed application. Any specifics of the SAP HANA as compared to other databases will be highlighted in the respective subsection.

3.1 THE PRINCIPLE OF AVOIDANCE

"The components of a computer system that are the most secure, fastest, most accurate, cheapest, reliable and easiest to maintain and document are those that are not there." (Gorden Bell)

BEST PRACTICE
Avoid all unnecessary source code, including unnecessarily executed source code.

3.2 PERFORMANCE OPTIMISATION ONLY IN THE APPROPRIATE AREAS

Performance optimisation (insofar as it increases overheads and complexity) should only be carried out in the appropriate areas. Performance must first be measured to ascertain where these 'hot spots' are.

BEST PRACTICE

- Focus first on clear and simple implementation of the essential logic. Then run performance tests on a system with sufficient data volume (usually a test system) in order to subsequently optimise performance at the relevant hot spots if the program is indeed slower than desired by the user. Exceptions to this involve programs where it is known in advance that they are performance critical. For these programs any decisions should be made at design time, e.g. use of parallelisation.
- We recommend starting the search for performance bottlenecks with a runtime analysis (transaction SE30/SAT) with full aggregation. This should clarify whether the runtime results from interaction with the database or from processing the data loaded into the main memory.[31] Important is that representative and realistic data is processed so as not to be led down the wrong track by unusual processing patterns. If more than half of the runtime is taken up with database processing, a closer analysis of the SQL commands should be carried out using transaction ST05. If more runtime is taken up by ABAP processing, a more in-depth analysis is undertaken using transactions SE30/SAT; whereby the aggregation levels are reduced in stages to obtain more precise feedback about critical points in the program. Results are compared and documented after each optimisation stage.

3.3 USE EXISTING TOOLS

The tools already available in the SAP system provide excellent support for creating performant applications and analysing performance issues.

Key transactions are

- ATC/SCI – ABAP Test Cockpit/Code Inspector
  Static analysis of performance aspects (amongst other things). Integrated within Workbench and ADT, administration through transactions ATC and SCI
- SAT – runtime analysis
  Full analysis at runtime
- ST05 – Performance Trace
  Analysis of SQL commands executed within a program (and other events)

In addition, a number of standard tools are available that can also be used for performance analysis

- SM50/SM66 – Work Process Overview
- Debugger – step-by-step execution of source code
- ST03 – Workload monitor
- ST22 – Analysis of runtime errors (lack of storage)
- STAD – Workload analysis (post-runtime)

The following tools are suitable for specific performance-related areas

- ST04 – DB Performance Monitor
- DB05 – Analysis of Table with respect to Index Fields

31 *An initial overview of runtime distribution between the database and application server is also available via the transaction STAD.*

- ST10 – Table call statistics to check table buffering
- ST12 – Single Transaction Analysis (end-to-end trace)
- S_MEMORY_INSPECTOR –
  Memory Inspector (also integrated within SAP GUI Debugger)
- SQLM/SQLMD – SQL Monitor
  Recording and analysis of all SQL requests within the production system. New from NetWeaver 7.0/7.4 onwards (For details see Note 1885926).
- SWLT – Performance Tuning Worklist
  Work list based on ATC and SQL Monitor results.
  New from NetWeaver 7.0/7.4 onwards (specific SPs, see SAP Documentation).

Within the context of the SAP HANA rollout, SAP provided further tools or extensions to existing tools to identify performance problems early on and establish a procedure to enable iterative optimisation of the performance of applications on SAP HANA.

Performance analysis in this context encompasses static source code analysis through ATC with the three testing variants FUNCTIONAL_DB, FUNCTIONAL_DB_ADDITIONAL and PERFORMANCE_DB. These variants carry out static code analysis within the scope of using SAP HANA as the DB. Further information is available in Note 1912445 – ABAP Custom Code Migration for SAP HANA – recommendations and Code Inspector variants for SAP HANA migration.

The ABAP SQL Monitor was also introduced. This tool enables system-wide tracing of all SQL requests over a long period with additional recording of the entry points (transactions, reports, etc. within the scope of which an SQL command was executed). More detailed information on SQL Monitor can be found on the SAP Community Network in the document 'Optimising Custom ABAP Code for SAP HANA – The New ABAP SQL Monitor'.

Consolidation of the SQL Monitor runtime screen and static analysis through ATC occurs via the SQL Performance Tuning Worklist. This enables setup of a work list for performance optimisation that can be processed following analysis and prioritisation. Further information on this procedure can be found on the SAP Community Network in the document 'Best Practice Guide – Considerations for Custom ABAP Code during a Migration to SAP HANA'.

3.4 DATA MODEL AND DATA ACCESS

3.4.1 DATA MODEL

The data model structure forms the basis for performant applications. A pragmatically normalised data model can work more efficiently with data and indices.[32] Independent of the programming language ABAP, normalisation rules are adhered to in this respect. Moreover, during the data modelling process, data elements in the data dictionary should be respectively assigned to domains. This results in greater transparency and improved maintenance.

3.4.2 DATABASE ACCESS

When accessing the database, five golden rules (source: SAP, see reference in Section 3.7) are to be complied with to ensure a performant execution of programs with database access. These five rules are:

1. Keep the result sets small – if the volume of selected data is kept small, loads are avoided both on the database system and on the network during transfer of the data to the application server.
2. Minimise the amount of transferred data – as the transfer of data between the database and application server is in blocks, it is best to keep the volume of data small to minimise the load on the network.
3. Minimise the number of database access operations – reducing the number of access operations to the database system minimises the load on both the system and the network as each access constitutes more work for the database system administration.
4. Minimise number of queries – using WHERE and HAVING clauses as recommend-ed further optimises performance if the volume of searching the database system is reduced by applying the appropriate indices.
5. Minimise (unnecessary) database loads – unnecessary database loads should generally be avoided; for example, making calculations via database query, the results of which are not required in the application. In addition, the application Server ABAP provides numerous options for minimising database loads through buffering.

32 *The S/4HANA model goes a step further and denormalises the data model. This is possible due to the type of data storage in the HANA in-memory database and provides a simpler data model and greater performance for data queries.*

These five rules apply to all database systems. Within the context of code push downs and code-to-data paradigms in SAP HANA, in contrast to non-HANA databases a modified weighting of the rules is to be observed. This is explained in Section 3.6.

Regardless of code push down, when using SAP HANA as a database, Rule 4 on the incorporation of indices has greater relevance. In SAP HANA, as few indices as possible should be applied. In principle, column-based storage enables each column to be used as an index, so theoretically no more indices are required: however, scenarios can arise, particularly in OLTP, where it makes sense to incorporate indices in SAP HANA, to avoid potential poor performance and/or high CPU consumption by SAP HANA. The relevant details can be found in SAP Note 1794297 – Secondary Indices for the Business Suite on HANA.[33]

BEST PRACTICES

Reduction of columns

- Avoid transfer from the database of unnecessary columns, in particular large objects (for example strings), to prospective performance-critical areas (see Section 3.2).
- Note: contrary to general opinion, SELECT * and INTO CORRESPONDING FIELDS themselves often do not pose a problem in these cases as the system intelligently analyses the source and target fields at compilation time.[34]

Optimised query

- When selecting data, try to use one of the existing indices in its entirety. If this is not possible, at least try and use the first elements of an index so the database can apply the index to a search.

Line reduction

- Use the WHERE clause to minimise the data to be transferred to the ABAP system. Use SELECT SINGLE/ UP TO n ROWS if only single lines are required.

Aggregates

- Aggregates (MIN, MAX...) are often (in the case of HANA always) practical as they have been previously analysed on the database server, so less data has to be transferred. However, be aware that any existing table buffering cannot be used by the system in this case, which could have a converse effect.

Updates

- Similar to column reduction when reading, the number of columns to be written can be reduced using the statement UPDATE SET. This statement should be used wherever possible.

33 *SAP Note 1794297 – Secondary Indexes for the Business Suite on HANA*
34 *Cf. SCN blog 'Why 'INTO CORRESPONDING' is much better than its reputation'*

Large volume operations

- Each execution of an Open SQL statement is associated with a certain overhead (parsing, check against the DBMS statement buffer etc.). Each statement should therefore transfer as much required data as possible at once. If the data for 50 orders is required, for example, the data should not be retrieved through 50 individual SELECTs, but rather just one single SELECT query. This is implemented using the additions INTO TABLE for reading or FROM TABLE for writing access (array operations).

Avoiding MODIFY

- Do not use MODIFY <DB table>. The statement is extremely critical from a performance perspective: even using array operations an UPDATE has to be executed initially per array line, and in case of an error a subsequent INSERT, all of which can unnecessarily hamper performance.

VIEWS/JOINS

- Nested and looping SELECT statements should be avoided. Use instead JOINS, views or the addition FOR ALL ENTRIES.
- Overly extensive JOINS (more than 5 tables) should be avoided as they could hamper the DB Optimiser.

Observe the following when using FOR ALL ENTRIES:

- If the internal table in the FOR ALL ENTRIES statement is empty, the WHERE clause is ignored and all DB table entries selected. Normally this is not desired and may result in performance problems. The problem can be solved by checking the size of the table prior to executing the query.[35]
- If the internal table in the FOR ALL ENTRIES statement contains duplicated entries, this may result in data sets loading twice from the database or unnecessarily bloating the SELECT statement in the DB interface. We recommend executing DELETE ADJACENT DUPLICATES for the internal tables prior to executing the query.
- As using FOR ALL ENTRIES activates an implicit DISTINCT select operation, all key fields should be selected as otherwise there is a possibility that not all relevant data sets will be read.

3.4.3 ABAP CORE DATA SERVICE (CDS) VIEWS

ABAP CDS Views is a DDIC artefact introduced with NetWeaver 7.40 SP05 that offers more extensive options for defining views than the classic SE11 Views. In addition to classic SQL functionalities such as outer joins and the Case statement, it also encompasses SQL functions such as currency conversion. Furthermore, CDS Views also offers the option of defining associations between tables and consequently defining business objects in a reusable manner in the form of views.

35 *SAP provides a Runtime Check Monitor with NetWeaver 7.40 that carries out a runtime check to address this problem. See also Note 1931870 – Downport of transaction SRTCM to SP02 / 03 / 04*

Further benefits include options to parameterise views (not supported for all databases in NetWeaver 7.40) and store annotations in a view or to view fields. From NetWeaver 7.50 onwards, the latter enables the straightforward exposure of views with OData Service and use of views and annotations to automatically develop UI5 interfaces (e.g. via Smart Templates). CDS table functions that enable access to SQL script are also available on SAP HANA. As such, all SAP HANA features (e.g. calculation views) are available via CDS Views. Further information on ABAP CDS Views can be found in ABAP keyword documentation for NetWeaver 7.40[36] and NetWeaver 7.50[37] as well as on the commensurate SCN landing page.[38] Please note that ABAP CDS Views can only be created using ABAP Development Tools in Eclipse.

3.5 INTERNAL TABLES AND REFERENCES

Internal tables are a core construct for developing applications with ABAP. In addition to enabling database access, they are also the secondary source of performance problems. Things like the selection of the right type of table and a suitable key are not really a problem with smaller data volumes; however, where larger data volumes are processed, areas previously deemed non-critical from a performance perspective can be the cause of considerably prolonged runtime.

36 http://help.sap.com/abapdocu_740/de/index.htm?file=abencds.htm
37 http://help.sap.com/abapdocu_750/de/index.htm?file=abencds.htm
38 http://scn.sap.com/docs/DOC-70385

BEST PRACTICE
We recommend adhering to the following guidelines for increasing application performance:

- Choose a suitable table type for subsequent application. Details on this are available in the keyword documentation under 'Selection of table type'.[39]
- If a Sorted or Hashed table type is accessed, this should always occur using the appropriate (partial) key.

As of AS ABAP 7.02 in addition to the primary key, additional secondary keys can be defined for internal tables that are rarely changed, but are read using more than one access pattern. The primary key is defined and used as previously. Secondary keys can be of a different type than the primary key (sorted, hashed).

For example, an additional sorted type key can be defined for a hashed table that has an unambiguous primary key. This additional key makes it possible to effectively access data in the table from another perspective (ambiguous or partial key possible) without the need to load the data a second time into the main memory and in the case of changes manually ensure consistency between the two tables.

39 *Access as described in Section 2 above. Path: ABAP – Reference → Processing internal data → Internal tables→ Internal table overview*

BEST PRACTICE

- Similar to DB access, there are single line and mass access operations for internal tables. Mass access operations should be used whenever possible because they perform better than multiple single access operations.
- When using the SORT statement avoid implicit sorting and for reasons of better verification always specify the desired sort fields.
- Before applying the DELETE ADJACENT DUPLICATES statement, ensure the table is sorted according to the same fields so that duplicate entries are actually deleted.
- Pure existence checks on internal tables should always be carried out using READ TABLE TRANSPORTING NO FIELDS.[40]

3.5.1 FIELD SYMBOLS

Field symbols offer the option of referencing existing data, e.g. internal table lines. Working with references is significantly more efficient than copying data. As such, field symbols should be used wherever possible. Notwithstanding, be aware of the fact that when using field symbols, any change to the value of a field symbol will also overwrite the value of the referenced data element.

BEST PRACTICE
Use standard field symbols for accessing internal tables.

3.5.2 PASSING PARAMETERS

Passing parameters by value should only be carried out where stipulated on technical grounds (e.g. RFC function modules, returning parameters in functional methods). This saves the unnecessary copying costs of passing parameters. This applies particularly for parameters with deep data types such as internal tables or strings. Moreover, as few parameters as possible should be defined as this increases source code readability. The Extended Program Check identifies superfluous variables and parameters, thereby supporting clean up.

BEST PRACTICE
Use as few parameters as possible. Generally use passing by reference and only use passing by value in exceptional cases deemed technically necessary.

40 *From NetWeaver Release 7.40 SP02 onwards the function LINE_EXISTS( ) is available and should be used for this type of check for reasons of enhanced readability.*

3.6 CODE PUSH DOWN

Using SAP HANA as a database system can greatly enhance performance, particularly of data-intensive applications, if data-intensive operations are delegated to the database layer. Delegation of application server logic in SAP HANA is called Code Push Down. The objective of this procedure is to no longer transport data to the application layer to process it there (data-to-code), but rather to execute the logic where the data resides (code-to-data).

In combination with the SAP HANA architecture, this objective consequently results in a different weighting of the rules defined in Section 3.4.1 in comparison to a non-HANA database. The first three rules (keep the result sets small, minimise the amount of transferred data and minimise the number of database transfers) become more significant in the wake of a Code Push Down, while the fourth rule (optimise the search overhead) becomes less important due to the column-based storage of data.

The fifth rule involves keeping unnecessary load away from the database, as the procedure consciously delegates data-intensive operations to the database and thus burdens it.

The extended language scope of ABAP and Open SQL in SAP NetWeaver 7.40 and 7.50 essentially enables Code Push Down in two ways:

- Using Open SQL expansions and ABAP CDS Views
- Using ABAP Managed Database Procedures (AMDP) and as a last option Native SQL

Which and to what extent the two options are used is highly dependent on the objective. In both cases existing code has to be modified. The complexity of such modification is lower using Open SQL and ABAP CDS Views than using AMDPs or Native SQL. If the primary aim is to maximise application performance, then there are limits to Open SQL and ABAP CDS Views that can only be overcome by AMDPs or Native SQL. A further aspect is that when using AMPDs or Native SQL, a more in-depth knowledge of SAP HANA specifics (e.g. SQLScript) is required to achieve optimum results. General consideration should be given to the fact that compatibility with other databases can only be ensured using Open SQL and ABAP CDS Views (with the exception of table functions). Database independence is lost if AMDPs and Native SQL are used.

BEST PRACTICE

- We recommend avoiding the use of external views and database procedure proxies where possible due to the various lifecycle problems in AS ABAP and SAP HANA stack.
- If carrying out application optimisation using Code Push Down, when selecting a code for optimising, we recommend the SQL Performance Tuning Worklist (transaction SWLT), which provides a prioritised list of source code positions on the basis of results from static code checks using ATC and runtime information from SQL Monitor. Recommendations specified in Section 3.3 are again to be observed.
- We do not recommend using Native SQL.

ADDITIONAL SOURCES

1. SAP course BC490 offers a good introduction to ABAP performance optimisation.
2. Blog posts by Olga Dolinskaja in the SAP Community Network provide an excellent introduction to the performance analysis tools SAT/SE30 and ATC
3. A description of profiling in Eclipse is available in the blog post by Thomas Fielder
4. The original 5 rules pertaining to SQL access operations are available now (7.40) in streamlined form in ABAPDOCU/F1 Help, under ABAP – Reference/Processing External Data/ABAP Database Accesses/Open SQL/Performance Notes. A link to comprehensive recommendations for secondary indices is also featured on this page.

5. A more extensive version of the 5 rules is still available for release 7.31 on the following page: Performance note 7.31

6. Siegfried Boes, 'Performance-Optimierung von ABAP-Programmen', dpunkt Verlag 2009, ISBN 3898646157

7. Hermann Gahm, 'ABAP Performance Tuning', SAP Press, 2009, ISBN 978-1-59229-555-5

8. Hermann Gahm, Thorsten Schneider, Eric Westenberger, Christiaan Swanepoel

9. 'ABAP-Entwicklung für SAP HANA', 2015, ISBN 978-3-8362-3661-4

4 ROBUSTNESS AND ACCURACY

Robustness is the ability of a program to deliver correct results and not to abort even in unfavourable circumstances. This principally means programs recognise errors and respond to them in a way that does not impair the desired functionality.

Conversely, sometimes errors cannot be meaningfully handled and in many cases implementation of genuine error handling with at least one exception or error message can prove uneconomical as the probability of occurrence is deemed exceptionally minimal. In such cases it is important to enable fast error location in the maintenance phase instead of concealing the cause of the error due to a ham-fisted error handling routine.

In this section we address best practices that foster the robustness of custom-developed applications.

Many aspects can be tested automatically using Code Inspector/ATC, see also Section 7.2.2 'Automatic tests'.

4.1 ERROR HANDLING

The ABAP runtime environment offers a variety of options for indicating an error to an application. These options are listed below together with best practices.

4.1.1 SY(ST)-SUBRC CHECKS

For a great many of ABAP commands, the global variable SY(ST)-SUBRC is set by the SAP kernel after the execution of a program. Generally a return value of zero indicates a successful execution.

Sy-subrc should be checked after all ABAP commands that set it. As an alternative to a dedicated evaluation of sy-subrc, the statement ASSERT sy-subrc = 0 can be used if the programmer thinks that no error exists or it cannot be meaningfully handled within the application. In the case of an error, the use of this statement will cause a program abort with the runtime error ASSERTION_FAILED.

This concise additional effort is well worth it as errors are recognised immediately and can be precisely located. It is a much better approach than having to endure 'puzzling' system behaviour that may well go unnoticed for a while and the root cause is frequently only discovered after extensive detective work. Handling of sy-subrc can be automatically checked using Code Inspector/ATC.

For programs involving mass processing, for example, the continuation of which must be ensured even in the event of an error in an individual object, genuine error handling is required (error recording, terminate this loop pass, etc.).

BEST PRACTICE
With function module calls always explicitly set the special value OTHERS, as the list of exceptions may change after the calling program was implemented.

4.1.2 MESSAGE STATEMENT

The MESSAGE statement is used to output status or error messages. MESSAGE statement behaviour is dependent on the type of message (status, information, error, exit) and the processing mode (dialog or background).

BEST PRACTICE
Avoid MESSAGE statements outside of modules that communicate directly with the user or are located in a defined dialog layer.

MESSAGE statements in specific message type and processing mode configurations may, as a result of a screen change, trigger implicit COMMITs in classic dynpros or interrupt connections in an RFC function call. Use class-based exceptions in the applications core to indicate errors. An exception to this rule is the use of the statement MESSAGE * INTO to set SY(ST) fields.

4.1.3 CLASS-BASED EXCEPTIONS

Class-based exceptions are an object-oriented error handling mechanism. A program identifies an error and throws an exception. If the caller does not catch the exception, it is propagated upwards through the call stack until caught at another point. If the exception is not handled in the call stack it terminates in a short dump. Exceptions can also be used for procedural source code.

Class-based exception handling has two essential advantages over classic error handling:

1. The error handling source code can be collected at a few points instead of having to be repeated after every method call and the like.

2. Exception objects may not only contain error messages, but also further attributes such as internal tables etc. This means more details can be displayed to the user for error analysis.

It is important to catch all exceptions that can be handled appropriately at the respective handling point. Exceptions that cannot be suitably processed locally must be declared within the procedure (methods, function modules, subroutines), so that to the caller it is clearly apparent that such exceptions can occur.[41] Important is coherent usage: consistent exception classes should be used for identical error situations.

Exceptions are caught using the ABAP commands TRY…CATCH…ENDTRY. Correspondingly, all exceptions of all called procedures that can raise class-based exceptions must be handled at some point in the call stack by a TRY…CATCH to guarantee a robust process.

41 *A detailed differentiation of exception categories and the commensurate user recommendations can be found in the ABAP keyword documentation. Path: ABAP – Programmer guidelines → Architecture → Error handling →Exception categories*

BEST PRACTICE:

- Do not leave exception handling blocks empty, as this makes it impossible for the caller to react properly to the exception situation. The only viable option in such cases is to propagate the exception. An exception to this rule is, for example, catching exceptions in a loop. However, an explanation is then required in a comment as to why neither handling nor propagation of the exception is necessary.
- Define an exception superclass that inherits from CX_STATIC_CHECK for new developments. Define for each application component one or more subclasses that relate to message classes with error messages (automatic implementation of the interface IF_T100_MESSAGE). In all new methods and function modules, declare the exception superclasses as possible exceptions. This ensures less effort is incurred later on, for example through additional checks.

In the case of enhancements to non-customer code, in some circumstances it may not be possible to add exception declarations to methods or function modules without modification. In such cases it is possible to use exception classes that inherit from CX_NO_CHECK. This enables error handling without explicitly declaring the exception class in method signatures and without aborting the program during runtime.

Declare subclasses of CX_DYNAMIC_CHECK only in method signatures in which the caller can prevent an exception occurring by transferring suitable parameters. As such, at the calling point no unnecessary TRY-CATCH block needs to be supplemented and the signature need not be complemented by an exception class.

BEST PRACTICE

Use CATCH blocks essentially to catch superclass exception objects as error handling routines normally work independently of the specific source of the exception. Only in certain cases is it meaningful to purposefully catch exception subclass objects in prefixed CATCH blocks to implement various error handling routines.

For chaining exceptions, the attribute PREVIOUS is applied when creating an exception.

The following source code pattern can be used to throw multiple exceptions:

Source code

```
DATA: lx_validaton_exception type zcx_validation_exception.
IF ….
  “ Check 1 unsuccessful
  lx_validation_exception = new ...
ENDIF.

IF ….
  “Check 2 unsuccessful
  lx_validation_exception = new … EXPORTING
PREVIOUS = lx_validation_exception. “ Important: chaining
ENDIF.

… “ other checks

IF  lx_validation_exception  IS BOUND.
RAISE EXCEPTION lx_validation_exception.
ENDIF.
```

BEST PRACTICE
Always write error handling routines in CATCH blocks in a way that ensures all chained exceptions are processed, save for where previous error messages are not relevant to error handling.

4.1.4 EXCEPTIONS THAT CANNOT BE HANDLED

Some exceptions cannot be handled in ABAP source code and therefore inevitably lead to an abort of the application and a short dump. In certain cases it is possible to proactively check the preconditions of a statement that raises an exception that cannot be handled prior to execution.

For example, the OPEN DATASET statement triggers a short dump if the user does not have sufficient authorisations to open a file. To prevent this, user authorisation needs to be proactively checked using the function module AUTHORITY_CHECK_DATASET.

4.2 CORRECT IMPLEMENTATION OF DATABASE CHANGES

4.2.1 LOCK OBJECTS

To prevent data inconsistencies, in applications using a pessimistic lock concept[42] the commensurate objects have to be locked prior to any change on the database. Related objects can only be changed after all related entities are successfully locked.

SAP locks are valid across multiple database LUWs (Logical Unit of Work), enabling consistent database changes to be performed across the entire business object being changed.

42 For more on pessimistic/optimistic locks see, for example, http://www.agiledata.org/essays/concurrencyControl.html

The locks should be set as specifically as possible in order to only lock the relevant objects within an LUW. Furthermore, locks should be retained for as short a time as possible but as long as necessary.

To change SAP standard data objects directly at database level use the respective SAP standard lock functions. This ensures consistent behaviour as these are also used for standard transactions.

For custom developments, commensurate lock objects and related lock functions should be implemented.

After an update has been performed, the object lock is released again by calling the appropriate dequeue function module. In this context, pay special attention to import-ant scope parameters if update function modules are used.

To ensure locked objects are unlocked even in exceptional circumstances, class-based exception handling also encompasses CLEANUP blocks.

In addition to pessimistic locking with exclusive locks, optimistic locking can also be used, which is based on the comparison of time stamps from the point of time of the change. Which locking concept is selected for new development projects is determined on a case to case basis.

ADDITIONAL SOURCES

1. ABAP keyword documentation for data consistency

4.2.2 DATABASE ACCESS FRAMEWORKS

BEST PRACTICE
Prior to manually developing a database access layer, evaluate existing frameworks provided by SAP.

For implementing database access, amongst other frameworks, SAP offers its Business Object Processing Framework (BOPF).[43] Induction in this or any alternative framework can reduce outlay in the medium-term, also as frameworks normally provide additional infrastructure components such as external interface tools.

4.2.3 UPDATE CONCEPT

Using classic dynpro user interfaces or remote calls from function modules may cause the program logic from numerous work processes to be executed one after another on an ABAP application server. Each change of work process is linked to an implicit database commit. If no framework is applied for the database access, it may be necessary in custom developed programs to initiate measures to bundle database changes. For further information on this subject, we refer you to the data consistency section of the ABAP keyword documentation.[44]

4.3 LOGGING

Errors, exceptions and log messages should principally be stored in the business application log to enable centralised checking of such (via transaction SLG1).

Access is via the function group SBAL, which is well documented within the system.

The main advantages of using business application logs are:

1. Central repository: the business application log enables messages to be managed centrally, which provides a clear overview and facilitates application administration.

2. Reusability: the business application log and/or the related function modules and classes provide comprehensive functionalities for logging, avoiding the need for custom development. Examples of the functionalities are:

   a Persistent storage of messages within the log
   b Hierarchical display of messages and aggregation into problem clusters
   c Integration of additional custom fields into log objects
   d Interactive reading of information upon message display and augmented information
   e Integration of log display into custom applications/UIs via sub-screens, controls or pop ups

BEST PRACTICE
It is a good idea to wrap SBAL functions once with a class to enable short and simple application through

```
MESSAGE i002(zmessage) WITH lv_value_1 INTO
DATA(lv_dummy).
  lo_logger->add_msg_from_sy(  ).
```

and at the same time maintain function of the where-used list.

43 http://scn.sap.com/docs/DOC-45425
44 ABAP keyword documentation on data consistency

The more complex calling of SBAL functions is then hidden within the class.

BEST PRACTICE
When creating business application logs provide an expiry date (BAL_S_LOG-ALDATE_DEL after calling BAL_LOG_CREATE) and plan a regular clean up using transaction SLG2.

A custom log table (DB Tables) can be a meaningful replacement for or supplement to a business application log when comparing cross-run values. This could be practical, for example, for daily batch runs, which then log 'n data sets of type x processed '.

A further supplement involves the activatable log points (LOG-POINT statement in combination with transaction SAAB, see reference 3). However, these must be first activated and they only log the respective number of events and field information for the last entry.

ADDITIONAL SOURCES

1. SAP Application Log – User Guidelines
2. Examples for using BAL functions in: Thorsten Franz, Tobias Trapp: ABAP Objects: Application Development from Scratch. SAP Press, 2008. ISBN 9781592292110
3. ABAP keyword documentation on checkpoints and checkpoint groups

5 ABAP SECURITY AND COMPLIANCE

This section describes how a programming error in ABAP can negatively impact on security within a company and what countermeasures exist. The subject of application security is principally relevant to all programming languages, but in the case of ABAP has not been given sufficient attention to date. Within this context, potential risks that may arise within a company as a result of corrupt ABAP programs are:

- Statutory violations (e.g. infringement of German federal data protection laws if the programming error leads to the theft of personal data)
- Non-compliance with compliance requirements (e.g. SOX or PCI/DSS)
- Cyber attacks aimed at carrying out industrial espionage, sabotage or extortion
- Backdoors, which enable extensive insider access to data
- Press reports if vulnerabilities or data theft become known

Custom developed ABAP code in particular, must not pose a risk to the confidentiality, integrity and availability of business data.

As this is an extremely complex subject, we can only broach a few selected core themes within the scope of this document. For broader queries, we refer to the commensurate literature listed at the end of the section.

5.1 SECURITY ISSUES RELEVANT TO TESTING IN SAP STANDARD

Our recommendations are based primarily on a document published as a guide in German by the German Federal Office for Information Security (BSI): the 'Top 20 Security Risks in ABAP Applications'.[45]

The document itself is based on a comprehensive statistical analysis of security defects in custom ABAP development projects (for further details see the published BSI document). The most frequent security defects were identified on the basis of these statistics. These were complemented by a set of less frequent, but particularly critical security risks. All in all, this provides valuable support for all companies that enter this sector and initially only wish to prescribe a limited number of rules.

45 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Hilfsmittel/Extern/TOP-20_Sicherheitsrisiken-in-ABAP-Anwendungen.pdf

The following initially divides the risks into areas followed by the commensurate recommendations.

5.1.1 AUTHORISATION CHECKS

Roles and authorisations are a key security topic in the SAP environment. It is therefore important to understand that ABAP uses a so-called explicit authorisation model.[46] Principally, authorisation checks should be explicitly programmed in the ABAP code to make sure that the checks are executed.

Security issues frequently arise because of the following:

- The developer forgets to program the authorisation check.
- The developer fails to check the transaction start authorisation when calling a transaction.
- The developer fails to check business authorisations in RFC-capable function modules.
- The developer fails to secure critical PAI events that can be triggered by the direct input of function codes, although the corresponding UI element was deactivated during PBO.
- The developer uses the wrong authorisation object.
- The developer uses a proprietary authorisation check.
- The developer does not handle the return value of the check.

5.1.2 AUDITABILITY

In addition to using internal teams, more and more companies are also having their custom developments tested by independent third parties. These either test how the program behaves during runtime (dynamic tests) or the test is based on the analysis of the source code (static tests). Two things must be guaranteed for a program to be testable: it must be fully executable and all of its source code must be readable.

Security issues can arise in the following cases:

- The developer writes code that behaves differently on different systems, which can hamper dynamic testing of potentially dangerous functionality on a quality assurance system.
- The developer writes code that behaves differently on different clients, which can likewise hamper dynamic testing on a quality assurance system.
- The developer hides code and consequently prevents analysis of the source code.

Whereas hidden code is fairly rare, hard-coded testing of clients or system IDs occurs with a likelihood of more than 70% per system.

5.1.3 DATA PRIVACY

The primary purpose of SAP systems is the management of business data. These include business secrets, personal data and financial data. The protection of such data is of fundamental importance to a company, not least as a result of statutory requirements. As a consequence, custom written code too must guarantee that such data cannot be extracted from the system without authorisation.

Security issues can arise in the following cases:

- Custom code programs display sensitive data without a sufficient authorisation check, e.g. on SAP GUI or in HTML pages.
- Custom code programs export sensitive data without a sufficient authorisation check.
- Custom code programs transfer sensitive data without a sufficient authorisation check.

46 Although implicit authorisation can be provided via S_TCODE in many cases, a plethora of gaps exist (external access via RFC or services, branching off to other programs, various display and change modes within a program)

5.1.4 INJECTION VULNERABILITIES

ABAP source code frequently uses dynamic statements, structures and identifiers that are assembled from user input. If no input validation/output encoding exists to prevent control characters in the input from changing dynamic command semantics, then the dynamic command can be maliciously manipulated at runtime.

Security issues can arise in the following cases:

- The developer combines input with dynamic ABAP commands or risky commands; for example execution of operating system commands or native SQL.
- The developer carries out inadequate input validation.
- The developer overlooks output encoding to avoid the harmful effects of control characters in input.

Some injection vulnerabilities occur frequently within custom coding. Accordingly, for example, statistically each customer system has 294 directory traversal vulnerabilities; albeit they only cause limited damage. But there are also injection vulnerabilities, which when exploited can grant an attacker complete access to the entire SAP system. Statistically, 16 of these severe vulnerabilities exist per SAP system.

5.1.5 STANDARD PROTECTION

SAP standard delivers various security mechanisms that protect SAP systems against attacks. These security mechanisms include client separation, logging and authorisation management. Companies consequently trust in these mechanisms when they have securely configured their systems.

Custom development projects must therefore avoid undermining the functionality of these mechanisms.

Security issues can arise, for example, in the following cases:

- The developer bypasses client separation.
- The developer deactivates database logging.
- Custom programs change business data without creating change documents.

Errors in this area are critical because they can potentially invalidate comprehensive security measures within the company.

5.2 SECURITY RECOMMENDATIONS

It takes many years of experience to write truly secure software and even then specialists may overlook something. Within this guideline, we focus our recommendations on a selected list that can in many cases at least significantly reduce the risk of a successful attack.

5.2.1 SEVEN UNIVERSAL RULES FOR SECURE ABAP PROGRAMMING

BEST PRACTICE

We recommend observing the following rules when programming with ABAP:

- Restrict the number of users that can execute any given business logic to the necessary minimum by means of authority checks.
  - In case a program has security vulnerabilities, then at least the number of potential attackers is reduced to the minimum.
- If a check is relevant, never assume that this check has already been carried out elsewhere.
  - If specific checks are required to secure program logic, carry these out.
- Avoid generic programming.
  - The more generic a program logic is, the greater the probability of an attacker finding a way to maliciously manipulate it.
- Do not bypass any SAP standard security mechanisms.
  - If SAP offers or prescribes a method for a particular functionality that is relevant to security, use it and do not try and write a 'simpler' alternative.
- Check the validity of all input parameter values.
  - Injection attacks become specifically more difficult if the range of input values is as limited as possible. When processing a phone number, for example, only allow digits to be used and necessary special characters such as *( ) + -*.
- Use the shortest possible variable types for input parameters.
  - Even where validity checks are insufficient, attacks are still considerably more difficult if the attacker only has a few characters available.
- Where possible, only use central library functions for security checks. Never write your own.
  - Creating functions that execute security checks is a complex matter and therefore open to errors. Creation and maintenance of such functions should be carried out by a centralised team within the company. This avoids potential errors and simplifies any subsequent rectifications.

5.2.2 THE MOST CRITICAL AND FREQUENT RISKS IN ABAP

BEST PRACTICE
Focus on preventing the following frequent and critical risks when programming. This way you assure a sound basic security level in your custom code.

The most critical ABAP risks and most important recommendations can be found in the German Federal Office for Information Security (BSI) guide 'Top 20 security risks in ABAP applications'.

A more compact list of critical risks is provided by the BIZEC APP/11 standard. However, the descriptions are not as comprehensive as in the BSI guide.

The following is a list of SAP messages that contain countermeasures for some of the aforementioned problems.

SAP Note	Schwachstelle
1520356	SQL injection
887168, 944279, 822881	Cross-site scripting
1497003	Directory traversal

SAP Notes that specify countermeasures

Naturally, we recommend that SAP security notes are checked and applied sooner rather later. Albeit these do only solve security issues in SAP standard source code. Regular checking of custom developments is therefore also imperative.

5.3 ABAP COMPLIANCE PROBLEMS

The subject of application security was rarely linked to compliance in the past, but is specifically relevant for auditing.[47]

Most companies have an internal control system (ICS) that counteracts compliance risks. This is specified, for example, in the internationally recognised reference models COSO & COBIT. Within a typical ICS structure, IT General Controls (ITGC) are a prerequisite for achieving all ICS objectives within an IT dominated environment.

A fundamental component of ITGC is change management, which in turn encompasses custom development. Any security defects in custom-written ABAP source code consequently infringe IT General Controls. So if change management and /or custom-written ABAP code is insufficiently covered in the ICS, then all the other measures will likewise become inadequate as they rely on a functioning change management system. This effectively results in a breach of compliance.

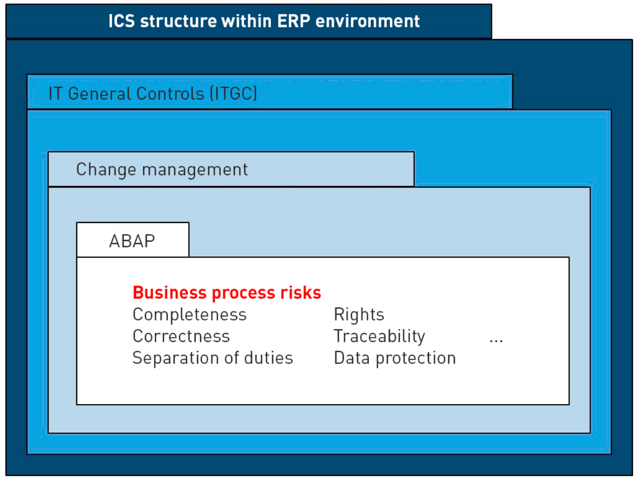


Figure 1: ICS risks resulting from insecure ABAP source code

47 For further information see: Maxim Chuprunov , Handbuch SAP-Revision, SAP Press 2011

In essence, this means that security defects in ABAP source code not only have a potential impact on compliance standards, but could also infringe statutory requirements.

All the security risks illustrated in the German Federal Office for Information Security guide 'Top 20 security risks in ABAP applications' are therefore also of relevance to compliance.

5.4 TESTING TOOLS

So-called static code analysis tools are particularly suitable for carrying out ABAP security checks. SAP has also expanded the Code Inspector/ATC with more complex security checks in the form of the SAP NetWeaver Application Server add-on for code vulnerability analysis, which is available under licence.

In addition, there are also a number of other commercial providers. The following features of such tools are of interest:

- Analysis also of SAP standard source code and third-party code, either in its entirety or limited to functions called by the customer code.
- Continuous monitoring, enabled through extremely fast scan speeds.
- Global data and control flow analysis, as these are fundamental to most security checks.
- Comprehensive description of the respective problem with proposed solutions.
- Adequate test coverage:
    - 'APP/11' standard from BIZEC
    - 'Top 20 security risks in ABAP applications' published in German by the German Federal Office for Information Security

Effective integration of external tools into the development environment (SE80, Eclipse) and development process (ChaRM, TMS) furthers the acceptance and use of such.

ADDITIONAL SOURCES

1. Andreas Wiegenstein, Markus Schumacher, Sebastian Schinzel, Frederik Weidemann, Sichere ABAP-Programmierung, SAP Press 2009
2. Maxim Chuprunov, Handbuch SAP-Revision, SAP Press 2011
3. BIZEC – The Business Application Security Initiative
4. German Federal Office for Information Security guide (in German) – Top 20 security risks in ABAP applications from 16.10.2014
5. The Business Code Quality Benchmark 2016
6. PCI / DSS (Payment Card Industry Data Security Standard)

6 DOCUMENTATION

In many cases software documentation is as important as the development itself. A lack of or limited documentation can cause severe complications at the very latest upon further development or following a developer change. This section describes the different options for documenting development within an SAP system.

It is important to use the level of documentation which is appropriate for the task at hand. Unfortunately, you often encounter just two extremes: either extremely comprehensive or no documentation. Comprehensive documentation, however, is often found to be lacking from a contextual perspective due to the extensive redundant/duplicate information it contains and the fact that it is not kept up to date. Also, when official requirements pertaining to documentation are too strict or abstract, the documentation in question is simply not created in some cases.

Documentation should be created during development, but at the very latest prior to go live or deployment. If the documentation is not ready, there should be no go live. Otherwise this usually results in additional cost or ultimately in missing documentation.

6.1 DOCUMENTATION INDEPENDENT OF DEVELOPMENT OBJECTS

In addition to detailing the many development objects that perform individual and very specific functionalities within the ABAP system, there is also the need to document the major relationships within a module and across modules. In this respect, questions such as the following need to be answered:

- What dependencies exist between modules?
- What applications are used with which business processes?
- Which background jobs run in a day/month/year and which development objects are impacted?

In our opinion there is no suitable repository available within the SAP development environment which provides answers to these questions and which also works well with integrated graphics. Because of this, we recommend using other options for the documentation of these cross-module relationships. Examples include:

- SAP Solution Manager
- Internal (product) wikis
- Documents in well-maintained public directories (storage portals, SharePoint, files sharing ...)

Experience tells us that the primary challenge is a question of discipline. This challenge cannot be solved with a tool – it is up to the development team and its management.

Templates such as arc42 are available for the documentation of the system architecture, including design decisions. These can prevent essential aspects of the documentation being forgotten and accelerate – when using a template for numerous projects – the search for specific information. Establishing templates also facilitates the creation of documentation parallel to development and compliance with an appropriate level of abstraction.

Document templates such as arc42 do not have to be completely filled in. Instead, the relevant parts are identified and the rest gets deleted dependent on the type and scope of the development project in question.

Furthermore, an out-of-date document can be misleading. As a consequence, all documents should display an update status and version number so that the latest version can be identified.

Within an SAP system landscape, SAP Solution Manager provides project documentation options. The following links provide more detailed information.

ADDITIONAL SOURCES

1. SAP Solution Manager 7.1 documentation
2. SCN blog 'Business process documentation with SAP Solution Manager 7.1'
3. arc42 template for architecture documentation
4. Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. Carl Hanser Verlag GmbH Co KG, 2015. ISBN 9783446429246

6.2 DOCUMENTATION OF DEVELOPMENT OBJECTS

In addition to methods, function modules and reports, where documentation can directly be included, there are other development objects within the ABAP workbench which do not have source code and which therefore have to be documented differently.

Examples are:

- DDIC objects
- Transactions

BEST PRACTICE
Our recommendation is to make use of the ABAP Workbench documentation options to document the tasks and meaning of these objects in the SAP system independently from the source code. Only the current status is to be documented, if necessary complemented by short references to the change documents (transport documents, defect numbers).

As Workbench documentation is also linked to the transport system, it is available in all the individual systems on the system landscape. Moreover, this documentation can be viewed by all users and is automatically integrated into the user interface for ABAP system reports. An additional benefit is that the documentation can be maintained in multiple languages.

ABAP Doc comments can be used in the source code on SAP systems with SAP_BASIS >= 7.40. This can be used as an alternative to documentation with ABAP Workbench. However, the full functional scope of ABAP Doc comments can only be currently exploited using ABAP Development Tools for Eclipse. When Core Data Services are used to define DDIC objects, considerably more development objects can be documented in the source code and external documentation is no longer required.

From SAP NetWeaver 7.50 onwards, class and interface ABAP Doc comments can be exported as HTML files.[48].

48 http://scn.sap.com/community/abap/eclipse/blog/2015/10/21/new-abap-doc-features-with-netweaver-75

6.3 DOCUMENTATION IN THE SOURCE CODE

6.3.1 DOCUMENTATION LANGUAGE

BEST PRACTICE
All program comments should be in English.

Today, development teams collaborate primarily on an international basis. Even if they are currently developing purely in German, their project may well be distributed on an international level at a later date. The costs incurred at that point due to coordination problems or even subsequent translation will be disproportionate compared to the perhaps greater effort of documentation in English.

Experience has also shown that the readability of source code and comments is enhanced when comments are in English. This is because the ABAP statements are themselves in English and have a sentence-like structure. Consequently, English documentation means the source code reader does not have to constantly switch languages.

6.3.2 CHANGE DOCUMENTATION

From the point in time when a program goes live make sure that all program changes are properly documented. An essential rule of thumb is that a full version history of all changes and comprehensively commented code reduces the readability of the source code. Despite this disadvantage, some developer teams consciously document all source code changes in order to simplify troubleshooting in productive or test systems where no version history exists.

BEST PRACTICE
Subsequent source code changes – except header comments – should only be documented directly within the source code in exceptional cases.

6.3.3 PROGRAM HEADER

Program changes can, for example, be documented in the program header with the name or initials of the developer, the date or month, the change document number (change document, incident ticket ...) and an abbreviated description of the change.

Example:

```
*/ Change Log
*/ VN/Date                   ChangeDoc Description
*/ MZ/2012-08-06 CD4712      Add MMSTA in Output
*/ MZ/2012-02-01 CD4711      Import Material number
*/ MM/2009-01-01 CD0815      Added Field ABC in method signature and source
                             code in order to support quick search
```

This information makes it easier to understand which addition or error correction was the reason for a change. These header comments also help recognise how often a program has been changed, who most likely knows most about it and how long ago the last change was made. This information is invaluable even before the next changes to a program are planned or installed.

6.3.4 SOURCE CODE COMMENTS

Source code comments help developers to understand the source code whenever this is not achievable solely via proper source code design (modularisation, method and variable name selection).

Comments are primarily meant for other developers, but can also serve the original developer as time passes. The essential question here is why was code written and not what code was written. Ultimately, the latter can be gathered from the source code itself, while the underlying reason for why it was coded like this may often be unclear. Developers themselves can significantly help in this respect by following the principle 'as few comments as possible, but as many comments as necessary'.

Comments starting with an asterix should only be used in program headers or for the temporary commenting out of old source code.

SAP recommends using inline comments for all other comments. These should be placed before the source code that is being documented and indented the same as the commented on line(s) in the source code. Pretty Printer correctly executes the latter (only) for inline comments.

ADDITIONAL SOURCES

1. Horst Keller, Wolf Hagen Thümmel: ABAP-Programmierrichtlinien. SAP Press, 2009. ISBN 9783836212861

7 FEASIBILITY AND ENFORCEABILITY

This section describes how the best practices contained within these guidelines can be implemented in practice. To this end, a distinction is made between feasibility and enforceability of the guidelines.

The 'Feasibility' section explains the aspects companies wishing to introduce programming guidelines should be aware of. A description is given of what a commensurate process might look like, how it can be implemented and, above all, how it should be maintained. The 'Enforceability' section provides details of how a company can check the specifications of the process, including organisational aspects, methods of checking and tools. The limits of enforceability are also examined.

The conclusion presents a number of practical tips gained by the authors while working on various projects within the SAP development environment.

7.1 FEASIBILITY

Those wishing to successfully introduce programming guidelines within a company must first convince the management to get behind the initiative, given that improving source code quality requires initial investment in processes, tools and training of the personnel involved. In particular, the management must feel confident that the respective processes will save the company costs in the long-term.

7.1.1 MOTIVATION FOR A PROCESS

The following provides information on quality aspects addressed during implementation of quality assurance processes and the respective advantages of such for companies:

Security

- Advantage: companies are able to prevent users gaining access or being able to change critical data without authorisation.
- Quality deficiency risks: sabotage, industrial espionage, undesired press reports resulting from data leaks, production system stoppages.

Compliance

- Advantage: the company is able to effectively demonstrate at any time that developed software meets the requirements of applicable compliance standards and statutory regulations.
- Quality deficiency risks: audit failures, infringement of compliance requirements or statutory regulations (e.g. data privacy).

Performance

- Advantage: the company is able to ensure optimum use of the available hardware, in turn protecting previous investment in hardware. Employee satisfaction is also increased as use of the application is more productive.
- Quality deficiency risks: reduced user acceptance and additional costs for faster hardware to compensate for software deficiencies.

Robustness

- Advantage: the company safeguards continuous operation of business applications and avoids loss of productivity owing to system failures.
- Quality deficiency risks: reduced user acceptance and increased operating costs due to loss of user productivity, fault analysis and technical maintenance.

Maintainability

- Advantage: the company is able to ensure sustainable, cost-efficient maintenance of the application because the program structure is easy to understand and well documented.
- Quality deficiency risks: high maintenance costs and generally increased application susceptibility to errors.

Extensibility

- Advantage: high-quality source code and suitable documentation ensures that developments can be extended and updated throughout the entire lifecycle at reasonable expense.
- Quality deficiency risks: additional expense for analysis and subsequent documentation during extension of existing functionality, triggering of side effects owing to incorrect interpretation of available source codes.

7.1.2 PROCESS DESIGN AND MAINTENANCE

A worthwhile step in terms of practical implementation of the process in question has proved to be the formalised description of the process, including clear procedural instructions and responsibilities. The precise details of the process will be unique to the individual company concerned and so cannot be specified here; however, the reference to its necessity is universal.

BEST PRACTICE
Define and document the applicable process in a form that is accessible to all. Also define how changes and improvements to the process should be performed and how feedback and critique can be incorporated. Comprehensively document all the verified rules, including sections on background/motivation, bad examples, good examples, notes on the procedure for remedying quality problems, relevant literature and commensurate contact person(s) within the company.

Motivation

A best practice process during development helps to proactively and efficiently improve software quality and therefore reduce costs within the company in the long-term. The sooner an error is detected during development, the easier and more cost-effectively it can be rectified. The fewer bugs in an application, the more its use will meet the expectations of the company. Specifically, the application will run without adverse side effects that impact negatively on the business.

Which aspects are relevant for the process?

Internal development

Guidelines are needed for internal development as a reference for day-to-day work and regular training on current risks.

External development

Clear quality specifications are needed for the external development bidding process. Before approval, the commensurate requirements must also be checked.

BEST PRACTICE
Process specifications should be monitored with appropriate tools to the greatest extent possible. A manual verification process, including random sampling if necessary, will need to be introduced for all specifications that cannot be validated using tools. Suitable mechanisms in this respect would be, for example, pair programming[49] or iterative code audits.[50] Manual validation essentially relates to software structuring/architecture, maintainability and extensibility, which could be high cost drivers in the case of long-lasting applications. As a result, in addition to automated inspection, random manual iterative validation is also recommended at an early stage. This process lends itself particularly to larger projects as defective developments can be recognised in good time and minimised through commensurate counter action.

Each rule used for subsequent quality assurance must be defined in terms of how it can be verified manually or with tool support. Where tool-supported inspection is not possible, ensuring consistent compliance of the rule in practice will involve a significant amount of organisational effort.

Formal code reviews require considerable effort, both in terms of implementation and also preparation and review, including the control of corrections. As a result, they should generally be limited to fewer non-tool-supported, verifiable aspects of critical development objects. If, for example, the compliance of performance requirements has high priority, commensurate code reviews should be restricted solely to development objects with access to databases or extensive calculations.

7.2 ENFORCEABILITY

7.2.1 MANUAL TESTING

Many tests can be automated. However, a number of areas are not suitable for automatic testing, such as documentation, architecture as well as numerous functional requirements. Language is highly complex and consequently, as an example, the content of documents and documentation needs to be tested manually. Only the human eye can judge whether a text is meaningful, complete, comprehensible and correct. An automatic test can only analyse the existence of the specified languages; however, an automatic test of the non-functional aspects is nonetheless recommended.

For manual testing, a complete test by analysis of transport lists is preferable, affording consideration to the various prevailing internal company guidelines. Depending on the number of objects, a full or at least random test should be carried out, with the test result sent to the person responsible for improvement or completion of the documents/documentation.

Whether or not a process meets all specifications can only be ascertained by way of a periodic manual check. If the specifications change or defects are discovered in the process, it is to be adjusted accordingly or redefined if necessary.

In practice, a regularly scheduled cyclical review of the process has proved worthwhile.

49 Cf. Dami Meyer blog 'Pair Programming'
50 Cf. Wikipedia 'Code Audit'

When and how should testing be performed?

The concepts upon which the tests are based must be regularly checked for relevance and conformity in terms of specifications. Relevance also needs to be ascertained by means of upgrades to new releases (enhancement package). Consultation with auditors tasked with auditing the company is also a worthwhile move with regard to the specifications.

In the case of externally developed source code, these tests must be performed prior to acceptance.

With regard to the acceptance of tests or in case of complaints within the course of manual testing, it is useful, within the course of manual testing and in the scope of developer and quality assurance checks (dual control principle), to have the manual tests be conducted by other developers.

The same applies for (security) penetration and load tests. Given that penetration tests are also a critical security issue, it may be necessary to consult external partners for this purpose at regular intervals.

7.2.2 AUTOMATIC TESTING

Automatic tests using static code analysis (with Code Inspector/ATC or third-party tools) quickly cover a large part of the necessary checks and examinations. Scheduled as a background task, regular repetition without any extra effort is possible. As a result, these regularly performed and uniform quality checks enable developers to improve their programming style.

When and how should testing be performed?

Developers should receive feedback regarding the conformity of developments with the respective guidelines at the earliest possible stage. This can be achieved by scheduled daily checks in the development system, with the developer furnished with the commensurate results.

Ensuring that the same checks and metrics are applied during testing by individual developers, central QA units and for each transport approval is essential. If different tools or settings are used for the checks, development team acceptance will decrease significantly.

As a core level of protection, the checks should be integrated within the transport management system and implemented upon transport request release at the latest (ideally upon release of the individual transports). This ensures that no untested or non-guideline-compliant developments are transported into other systems or the production system itself.

As a 'final security net', a regular test run (full scan) should be carried out in the production system. This can be planned as a background task to be performed during a low-load period. The respective result is to be submitted to the responsible QA representative, who is then able to arrange the additional steps with higher priority (including correction if necessary).

Prior to introducing automatic testing, the procedure for dealing with old source code should be defined. A useful step in this respect is to create a timetable detailing when and how the new rules are to be applied to the respective old source code.

The internal costs and benefits for the company should first be considered in terms of the systematic revision of old source code. Costs should also take into consideration the test effort required by the pertinent specialist department and the risk of new errors.

BEST PRACTICE
In dealing with old source code, we recommend that check results for old source code are documented at the start of regular test runs and monitored at each further test run to ensure no deterioration of test results has emerged in comparison to the initial test.

SAP offers various tools for performing automatic testing; an overview of these tools is provided in Section 2.9.

7.3 PRACTICAL EXPERIENCE AND TIPS

7.3.1 SOURCE CODE QUALITY ASSURANCE

Safeguarding successful implementation of quality assurance necessarily requires a carefully staged process for which a dual approach is advisable. Firstly, new source code should be created 'error-free' and checked to ensure this is indeed the case. Only when this process has been stabilised, should existing source code be progressively incorporated within the checks. Otherwise the task becomes overwhelming for the developer, in turn resulting in a sharp drop in motivation.

Where automatic code testing is not introduced from scratch with a new development, prior clarification of how to deal with existing code needs to be undertaken. Even in the case of new developments, changes to existing objects cannot be avoided and may lead to problems in the event of transport checks. In such cases, clarification of the responsibilities for development objects is beneficial, with the results then, for example, subsequently documented in the commensurate package definition field. Those responsible must decide if errors in existing source code need to be corrected immediately or whether an exception is possible.

To avoid the problems described, an alternative concept is conceivable within which no deterioration of old coding is accepted, meaning that no new findings should occur in automatic quality testing. SAP is currently working on implementing such a concept for the ATC. In the meantime, an SCN blog is available with details of customised implementation.[51] The advantage is, for example, that new methods in existing classes are also checked.

In many cases, working with standard on-board tools will generally suffice, such as the ABAP test cockpit that can be extended with custom tests and therefore adapted to individual needs.

An additional strategy for new developments involves incorporating quality requirements in the requirement analysis process and linking these directly to the object being created. This enables a subdivision into requirement-specific quality criteria and those driven by IT governance. While quality requirements subject to IT governance (e.g. security criteria) are a mandatory obligation, the ABAP product manager is responsible for evaluating and approving the exception of other criteria. Technically, the concept can be realised by incorporating the product manager within the package and providing he package or source code object it contains with manually entered classification attributes from the Classification Toolset.[52]These classification attributes should be determined based on factors such as anticipated lifecycle, frequency of use, criticality, extent etc. The result is a scenario which eases the burden on central quality representatives and delegates partial responsibility to the development/project team.

51 http://scn.sap.com/community/abap/blog/2016/05/23/how-to-filter-atc-findings-to-detect-only-new-findings
52 SAP-Help-Search „Classification Toolset"

BEST PRACTICE
Quality assurance and mandatory tests such as ATC/Code Inspector tests upon transport release are an effective means of ensuring quality assurance. Nonetheless, this does create a certain level of inefficiency given that at this stage any previously completed source code has to be revised. A more practical approach is for developers to deliver high quality source code from the outset. As an example, this can occur through the application of clean code criteria.[53] Where a team essentially pursues development on the basis of these criteria, the subsequent adaptation effort will be reduced.

7.3.2 TIME AND BUDGET QUALITY ASSURANCE (QA)

To keep the time and effort required for QA activities to a minimum, developers must be able to independently examine the source code for errors during its development.

If this is omitted, the developer must be notified of errors or opportunities for improvement automatically. Daily inspections with an appropriate tool and distribution of the results ensure that errors are detected early on, meaning the developer is still able to remember activities from the previous day, which significantly simplifies troubleshooting. As a result, even developers who shy away from manual work or are under time pressure nonetheless have an opportunity to fix their coding errors. Ultimately, the later QA takes place, the higher the costs of debugging. This extra cost and effort arise, for example, from additional transports after the original transport had already been released.

53 See also http://clean-code-developer.de/

It is therefore important to consider quality assurance when planning and estimating a project, not only at the end, but during the course of the project and subsequently across the entire software lifecycle.

Not be underestimated is the requisite training needed to communicate the necessity of the process to developers.

Where third-party developers are employed, the programming guidelines and naming conventions must be an element of the contract.

7.3.3 PROBLEMS

The introduction of source code QA gives rise to a series of issues that are addressed in brief here.

One point of contention is the question of who is responsible for QA, creator or changer. For new source code this is not a problem; however, for existing source code the following issue repeatedly arises:

"Why should I check source code in which I have only changed one line?"

vs.

"Why should I check source code that I haven't touched for years?"

Both positions are naturally understandable; therefore, a clear decision must be taken and implemented regarding the handling of existing source code based on other or no conventions. If the creator is still available, the incorporation of this creator within QA is recommended.

Also problematic is a scenario where a small change is made to an include program within the scope of maintenance. An automatic quality check then finds quality errors within the main program (possibly even in several main programs in the case of multiple-use include programs). Depending on the environment, the respective rectifications will require extensive additional testing of these main programs by the pertinent specialist department.

BEST PRACTICE
A central authority should be created to respond to developers' questions in the event of problems. In addition, a process must also be in place to enable the release of even faulty transports in emergencies. The absence of such options will reduce acceptance of the measures implemented. One approach is to install an approval process that also helps facilitate the release or transport of faulty source code. Most tools on the market (including SAP Code Inspector and ABAP Test Cockpit) offer this possibility as standard.

7.3.4 DECISION MAKING REGARDING MODIFICATIONS

The threshold for modifications should be set as high as possible. This is particularly important if SAP enhancement packages are to be promptly implemented (see Section 8.4 re. SPAU problem). The starting point in this respect is the modification key. The number of developers with authorisation to create a modification key must be kept to a minimum as this enables modifications to be controlled and handled using change requests and via a commensurate process. Given that SAP coding cannot be changed through modification-free enhancements and that this reduces the potential for problems with upgrades, modifications are essentially preferable.

The question of whether a modification is justifiable is an individual matter for the respective company and must be consistently implemented. There is no general answer to this question; in each particular case the decision needs to be based on equivalent criteria that are defined and communicated in advance.

For alternatives to modifications see Section 8.4.

BEST PRACTICE
If SAP coding requires enhancement or change, four options are available: these being explicit and implicit enhancements, modifications and copying the SAP object into the customer's namespace. The individual options should be considered in the following order: explicit enhancements prior to implicit enhancements, implicit enhancements prior to modifications and modifications ahead of copies. Copies are particularly critical given that possible errors will also be copied, while the copied source code will no longer be subject to SAP maintenance through notes and updates.

7.3.5 PRACTICAL FIELD REPORT: COMGROUP GMBH

The following example regarding Comgroup GmbH, IT service provider to the Würth Group, shows how programming guidelines and naming conventions can be implemented and enforced within software development.

Comgroup GmbH began source code QA within the global development system of a multistage development landscape. Code Inspector was employed to provide automated support. Since the introduction of source code QA coincided with a new namespace, only new namespace objects were checked at the start of the project and existing source code was not considered. This simplified selection in Code Inspector given that Code Inspector does not consider change or creation data. In addition, Code Inspector performance and security checks were not initially activated in order to keep efforts within a manageable scope.

Using this tool, developers are able to independently check their source code during development. In addition, a nightly run was also scheduled to analyse all source code being scanned, with commensurate emails sent to the respective developer in the event of errors being established.

Email addresses missing from user masters initially caused problems because the emails could not reach their recipient. Such emails are now sent to a central address, thereby allowing incomplete data to be easily identified. In addition, development users can no longer be created within the system without an email address.

The emails were not sent by an anonymous batch user, but rather from the email address of the person responsible for QA in order to give the developer a straightforward option to ask questions. Initially this resulted in a lot of work, although the number of questions decreased during the course of the project. Ultimately this was achieved through training that enabled developers to correct errors more efficiently or avoid them in the first place in the course of development.

The development language in the development landscape is English, checked by an additional job and with errors reported to the developer. SAP standard does not offer the option of setting a default language for a development object. Therefore, the only options were to either implement the commensurate test logic at an appropriate place using a modification, or live with the fact that objects were created in the wrong language and would subsequently need to be changed.

Moreover, when implementing objects naming conventions were also checked using a modification to ensure that they could only be named according to the conventions.

To perform custom checks at transport release (e.g. custom naming conventions/German and English translation as a minimum) the business add-in CTS_REQUEST_CHECK was implemented and the CHECK_BEFORE_RELEASE method employed.

Once the process had stabilised within the global development system, it was rolled out to the subsequent development systems and namespaces. Existing source code has not yet been checked. In addition, the use of an external tool to simplify quality assurance is also planned.

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

This section focuses on infrastructure and explores the lifecycle of a software component. In addition to methodical recommendations and tools for software development within the SAP system, these are principal framework conditions for delivering successful performance.

8.1 INFRASTRUCTURE

As a rule, an SAP system landscape has a multi-level structure. Depending on the release strategy, two practical alternatives should be considered. Where small changes are regularly transported at frequent intervals in the shape of short-term releases, a classic three-system landscape is preferable. By contrast, in the case of a release strategy with long release cycles, a five or six-system landscape would be beneficial. The individual systems and their relevance are presented below.

8.1.1 CLASSIC THREE-SYSTEM LANDSCAPE

8.1.1.1 Development

The development system is used to implement developments, change customising data and perform initial developer tests. Given the instability created through ongoing developments and customising activities, third-party testing (pertinent specialist department) is not practical for this system.

Developers and module administrators have extensive authorisations within the development system. Generally there is very little test data.

8.1.1.2 Quality assurance

Customising and development is essentially prohibited within the quality assurance system (system settings 'not modifiable'). Customising and workbench objects are exclusively imported via transports.[54] Imports to production are effectively carried out via the quality assurance system to ensure a system environment is compatible with production.

54 Copies can also be transported within the three-system landscape, as described in Section 8.1.4 'Best Practice'

The quality assurance system should be regularly copied from the production system, for example, for more comprehensive validation actions and so ensure compliance with the operative (data) environment. Within the quality assurance system, test plans are processed following changes and new developments.

Developers and module administrators have extensive authorisations within the quality assurance system. Restrictions need to be applied individually, for example, for particularly sensitive data (HR etc.). Data privacy is to be considered in relation to personal data.[55] Use of additional users with identical production authorisations is recommended in order to also test the issue of authorisation.

8.1.1.3 Production

A prohibition on customising and development also applies within the production system. Customising and workbench objects are exclusively imported into this system via transport request.

Developers and module administrators have limited authorisations within the production system. Urgent changes to tables (&SAP_EDIT) are to be documented with the date, time and reason for such. The use of a commensurate tool is also recommended to ensure report standardisation. The procedure for emergency users with more extensive authorisations must be determined both technically and organisationally.

8.1.2 FIVE- AND SIX-SYSTEM LANDSCAPES

This landscape is practical if long release cycles are used. The development of new releases occurs within the development system, with maintenance and error processing in relation to the production release carried out in two separate systems.

8.1.2.1 Development

In this landscape, the development system is in the same role like in the three-system landscape.

55 Cf. DSAG Data Protection Guidelines

8.1.2.2 Test

An absolute prohibition on customising and development applies for the test system. Customising and workbench objects are exclusively imported through transport requests.

Generally, imports to this system are performed through the transport of copies (see Section 8.1.4 'Best Practice'). Accordingly, this ensures that development objects processed within the development system remain locked throughout the entire release period. The original transport request is first released at go-live. The transport of copies allows developments and customising to be tested up front within the test system. As a result, numerous developments and customising within the development system can be bundled to minimise the number of release transports for quality assurance and the production system. Should this advantage be deemed unnecessary, the test system can be dispensed with.

The test system is copied from the production system where necessary. Consequently, this allows comprehensive testing to be carried out in advance.

Developers and module administrators have extensive authorisations within the test system. Restrictions must be applied individually, for example, for particularly sensitive data (HR etc.). However, the deployment of test users with production-related authorisations is also recommended.

8.1.2.3 Quality assurance

The quality assurance system of this landscape corresponds with that of the three-system landscape.

Provision to the quality assurance system is facilitated:

- in a six-system landscape solely with release transports,
- in a five-system landscape through normal ongoing transport requests.

8.1.2.4 Maintenance

Within the maintenance system, maintenance and administration of the production software is carried out over the period the new release is developed within the development system. All corrections in the new release from the maintenance system must be incorporated.

After a Go-Live, meaning the transport of a release to the production, the maintenance system has to become consistent again with the production system. This can be achieved in a number of ways, one typical approach being to import the release transports into the maintenance system.

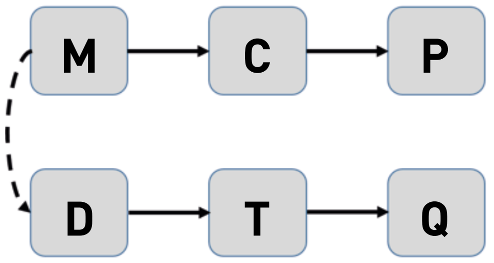
8.1.2.5 Consolidation

Changes from the maintenance system are tested in this system.

8.1.2.6 Production

In this landscape, the production system is commensurate with that of the three-system landscape.

8.1.2.7 Schematic illustration of six-system landscape



$$\begin{tikzcd}
M \arrow[r] \arrow[d, dashed, bend right] & C \arrow[r] & P \\
D \arrow[r] & T \arrow[r] & Q
\end{tikzcd}$$

System	Application
D	Development
T	Test
Q	Quality assurance
P	Production
M	Maintenance
C	Consolidation

Figure 2: Schematic illustration of six-system landscape

8.1.3 SANDBOX

The sandbox is purely a testing and 'exploratory system'. There are no authorisation restrictions in the sandbox or with regard to customising and workbench developments. No transports are carried out from the sandbox to other systems, meaning that complex changes can be tested prior to their implementation within the development system. As a result, this avoids having to roll back complex developments within the development environment if, for example, a new development is subsequently discarded. To clean up prototype developments within the sandbox system, it should be regularly refreshed, for example, as a copy from the production system.

8.1.4 TRANSPORT SYSTEM

The transport route is defined as follows:

- Three-system landscape:

  Development → Quality assurance system → Production
  (possibly with transport of copies after quality assurance system)

- Five-/six-system landscape:

  Development → Test (in this case, always with transport of copies)

  Development → Test → Quality assurance system → Production
  (release transports)

  Maintenance → Consolidation → Production (maintenance/
  administration during release development)

The sandbox should be isolated from the transport route. Imports from the development system to the sandbox occur solely following individual request.

Transports within the development system are generally released by the developer. If multiple developers are working on transport tasks within a project, one developer in the role of coordinator will release the transport request. Automatic quality checks, for example with the ABAP Test Cockpit (see Section 2.9) or the Code Inspector (see Section 7.2.2 'Automatic checks'), can be configured to prevent transport release and require subsequent remedial action or special approval.

Imports into the test and/or quality assurance systems must be organised based on the system landscape. Frequently, imports into these systems are performed automatically within a cycle in order to alleviate employee workload in the basis department and reduce waiting times for developers and testers.

Where multiple developers make changes to a development object, problems can occur in certain situations if the transport requests are not transported into production systems in the right sequence or if objects from other transport requests are missing.

BEST PRACTICE
To avoid problems with the import sequence of released transports relating to the same development object, the transport of copies should be used for delivery to test and quality assurance systems. The modified development objects remain blocked by the actual transport request within the development system until the point of going live. The system will notify a developer if another developer is already working on the object so that they can coordinate their actions accordingly. Following completion of the project or change, only the actual transport request will be transported into the production system via the quality assurance system.

The Change and Request Management system (ChaRM) contained within the SAP Solution Manager works with this process internally.

As a general rule, imports into the production system should be carried out by internal employees with commensurate authority; Prerequisite is the formal release (within the meaning of a documented decision, see Section 8.2 'Change Management'). Use of a suitable tool (e.g. Solution Manager ChaRM) is also recommended in this case to ensure standardisation/formalisation.

8.1.5 SAFEGUARDING CONSISTENCY OF NEW DEVELOPMENTS AND EXTENSIONS

If projects are run in parallel a risk of overlapping occurs. This may result in overlapping use of objects that are not (yet) available in the target system, ultimately leading to import errors. Consequently, there is an obligation to check the use of objects created by (non-SAP) third parties.

New developments and extensions must be encapsulated in suitable packages or transport requests. It is recommended that transport requests for a given project be limited to a single transport request for workbench, customising and authorisation roles respectively. 'Preliminary transports' should only be permitted using the transport of copies.

Final release and transport only first occurs at the end of the project. All project participants only apply tasks relating to a given transport request within a project. There should be no 'personal' transport requests for individual project members.

Generally, an import will only be performed following formal release (see change process) by a process owner, quality assurance or similar. The sequence of performance and participating areas must be specifically determined in relation to the company.

8.1.6 ROLL BACK OF NEW DEVELOPMENTS

Should developments be created in the development system that prove to be unusable or inaccessible, a number of options are available for remedy:

- Complete roll-back of the changes, insofar as the changes are not also required in later releases.
- Deactivation of the development to prevent the source code running in the production system. This can be achieved using various strategies:
    - Commenting out the source code (in the case of small code changes).
    - Incorporation of a switch within the source code that prevents the unreleased source code from running. The nature of the switch will depend on the specific circumstances.
    - Complete removal or roll back of customising.

Where TMS Quality Assurance (QA)[56] is used as the approval workflow, the following applies:

- Rejection by the quality manager merely prohibits the transport of functionality into the production system in a useable or accessible form. The source code can be supplied, but should not be run under any circumstances.
- A process must be in place to ensure that the rejected development is transferred to all relevant systems in an inaccessible form. As stated above, this can be achieved through complete roll back or deactivation via switches in the source code.
- The responsible developer must be advised of the rejected transports. The obligation to forward commensurate information lies with the quality manager.

A small Z-program is available in the SCN regarding rolling back 'abandoned' developments (Report for Rolling Back Abandoned Developments)

56 https://help.sap.com/saphelp_nw70/helpdata/de/9c/a544c6c57111d2b438006094b9ea64/content.htm

8.2 CHANGE MANAGEMENT

Creating a maintainable and controllable SAP system landscape requires a formal change process for each system modification. The basic procedure also similarly applies for changes to SAP standard and customer developments.

The framework for introduction of a change and release management system is the Information Technology Infrastructure Library (ITIL).[57] This is a reference guide that lists the comprehensive and generally accepted best practices.

In this section we present a specific example that can be adapted to a particular company or sector accordingly.

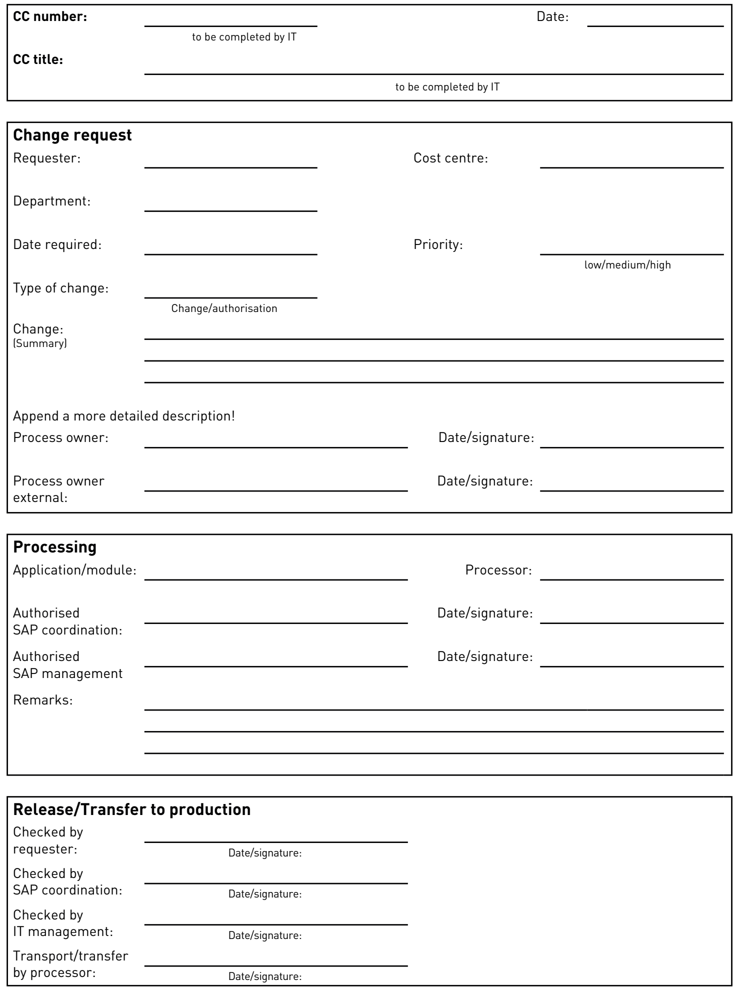
Fundamentally, we recommend considering the following aspects when introducing a change control process (also: change request process):

- Functional requirement
- Motivation
- Evaluation (estimation of cost/effort)
- Approval
- Release of the change to the production system

The approval/release group (process owner, QA ...) depends on the nature of the change (area affected, application affected, possibly also effort involved).

Use of a suitable tool is strongly recommended (e.g. Solution Manager ChaRM). However: a paper-based solution is better than no solution!

Example of a change control form (CC):



CC number:	to be completed by IT	Date:	
CC title:	to be completed by IT		

Change request

Requester:		Cost centre:	
Department:			
Date required:		Priority:	low/medium/high
Type of change:	Change/authorisation		
Change: (Summary)			

Append a more detailed description!

Process owner:		Date/signature:	
Process owner external:		Date/signature:	

Processing

Application/module:		Processor:	
Authorised SAP coordination:		Date/signature:	
Authorised SAP management		Date/signature:	
Remarks:			

Release/Transfer to production

Checked by requester:	Date/signature:
Checked by SAP coordination:	Date/signature:
Checked by IT management:	Date/signature:
Transport/transfer by processor:	Date/signature:

Figure 3: Change control form (CC)

57 see https://de.wikipedia.org/wiki/IT_Infrastructure_Library

The illustrated CC form includes, for example, the essential data that must accompany a system change process.

Basic process and roles:

- The requester fills in the 'Requesting department' part and obtains the signature of the process owner and/or external process owner.
- The process owner is usually the senior manager of the requester and is responsible for a certain part of the data and use of certain SAP software elements e.g. the purchasing manager is responsible for SAP purchasing data and programs.
- The external process owner should be included whenever a change affects data or programs that lie outside the process owner's responsibility.
  Example: the purchaser requires authorisation from asset accounting; in this case the process owner with responsibility for this part of the system must also furnish approval (e.g. asset accounting manager).
- A detailed description of the change is to be attached as an appendix to the CC in all cases; CCs without a detailed description will be rejected. The requester passes the CC on to IT.
- The SAP coordinator is the person who coordinates the respective activities relating to SAP or a part of SAP and allocates tasks to the individual module administrators. Depending on the size of the organisation, this responsibility can also be undertaken by a group of individuals or a department manager within IT. The respective person enters the application or module on the form and allocates the CC to a processor. They may also reject the CC on the basis of formal errors (e.g. absent or insufficient description, lack of process owner or external process owner signatures). The SAP coordinator will issue a specific CC number and CC title; this CC number can also be provided by a project management tool.
- The IT manager or SAP manager approves/rejects/defers the change (with suitable justification) accordingly, following approval by the coordinator.
- Where approval is issued, the processor receives the CC for further processing; any change intended for further transport may be carried out by a processor solely on the basis of a fully approved CC.
- After completion, the processor requires the requester to check the change.
- Where implementation corresponds with the respective requirements, the requester releases the change for transport; the requester confirms correct implementation by way of signature; a transport may only be carried out after release by the requester.
- The SAP coordinator and IT manager confirm correct implementation of the change. Correct implementation should be verified using a check list, supported by testing tools (see following best practice recommendations). A transport request may not be imported into the production system without prior release by the SAP coordinator and the IT manager.
- The processor transfers the transport to the production system and forwards the CC form to the SAP coordinator and IT manager.

The approval and release process and therefore also the content of the form may vary greatly depending on the specific sector. In pharmaceutical companies, for example, QA is essentially incorporated within the CC process. Moreover, all companies will generally require additional implementation approval in the event of certain (estimated) project cost threshold being exceeded. As such, the approval structure also depends on the respective company organisation.

Consequently, the form provided merely contains the minimum requirements of a CC without taking into account any requirements in terms of sector or organisation.

Additional approval and/or release steps or additional fields referring to other documents (e.g. validation documents) should be individually supplemented as required and the process expanded accordingly.

BEST PRACTICE

Check list prior to release of a workbench transport by the SAP coordinator and IT manager for transport into the production system:

- Have automatic source code tests been performed? Has the code inspector and ATC or extended program check been carried out for all transport request programs? The ensuing results list should not contain any errors or warnings, however information reports are permitted. Optimally, tests are carried out automatically for every transport release (see Section 2.9).
- Third-party tools? If additional test tools for areas such as security, performance, etc. exist, have these been executed?
- Manual testing as necessary, possibly random sampling.
- Manual preliminary work or reworking? Verify whether there is a fully executed check list for manual preliminary work and reworking of transports.
- Multiple languages? If the transport request contains translation-relevant objects, a check should be carried out into whether translations are provided as per the translation strategy.
- Transport dependencies? A check of transport dependencies must be carried out.
- System-internal documentation? See Section 6.2

ADDITIONAL SOURCES

1. Mathias Friedrich, Torsten Sternberg, Change Request Management mit dem SAP Solution Manager, SAP Press, 2009

8.3 SOFTWARE MAINTAINABILITY

Software maintainability is a criterion that applies in software development and indicates the effort and cost required to carry out changes within an overall system environment.[58]

From a technical perspective, a modular construction is required (see Section 2.3 'Readability and modularisation'). Reusable source codes need to be organised in global classes or function modules. Package interfaces can be used to identify reusable objects.

In system environments consisting of different development and production systems (transport streams) the basic principle applied is that an identical object name (transaction code, program, include, table) will also have identical coding and identical object attributes.

All developments, changes and bug fixes are to be documented accordingly (Cf. Section 6 'Documentation').

8.4 ADAPTATION OF SAP FUNCTIONALITY

Various options are available for adapting the functionality of an SAP system to individual requirements, each having both advantages and disadvantages:

- Enhancements (user exits, customer exits, BTE, BAdIs, enhancement points and sections, CDS extensions)
- Implicit enhancements
- Modification
- Z-copy, copy in customer namespace
  Use of this option is expressly not recommended!

User exits, customer exits, BTEs and BAdIs are all usable techniques that can be employed without problem. They should therefore be used when available in suitable positions and with suitable interfaces.

The various techniques are briefly described in the following.

58 *Wikipedia 'Maintainability'*

User exits

User exits are subroutines contained in includes in the SAP namespace that are only delivered once by SAP and so can be 'modified' without problem.

Customer exits

Customer exits are function modules that can be activated and deactivated; they can be implemented by the customer in order to enhance the standard functionality.

Business transaction events (BTE)

In the FI environment, BTEs provide an additional option for enhancement. BTEs are comparable to customer exits, but are essentially restricted to the FI module and provide a predefined interface to which the developer can add enhancements. Further information is available in the SAP standard documentation.

Business add-ins (BAdI)

BAdIs are used by SAP in an attempt to remedy the disadvantages of previous enhancement technologies such as:

- access to all global variables (user exits)
- only single usage (customer exits)
- no dynpro enhancements (BTEs)
- no menu enhancements (BTEs)
- no maintenance tools (BTEs)

Hence BAdIs can be multiple-use and offer all enhancement types (program, menu and dynpro exit). If multiple enhancement techniques are available for a desired enhancement, the use of BAdIs is recommended.

Enhancement framework

With the new enhancement framework SAP is attempting to remedy the disadvantages of previous enhancement technologies. The enhancement framework includes:

- explicit enhancements (enhancement points and enhancement sections)
- 'new' BAdIs (whereby implementation of the old BAdl technology can be automatically migrated)
- implicit enhancements

Enhancement points

These enable source code to be included at fixed points. To this end:

- multiple active implementations are possible in parallel and
- all active implementations are executed.

Enhancement sections

These provide the option of replacing a defined section of a program with custom source text. To this end:

- multiple active implementations are possible in parallel, however
- only one active implementation is executed;
- it is unclear which active implementation is executed.

Note: implementations of enhancement sections can be substituted by the provision of SAP enhancement packages or the activation of business functions through new active or newly activated implementations. In such cases it is very difficult to identify substituted or no longer executed enhancement implementations. As such, a change in SAP standard may change the behaviour of the enhancement. This increases the requisite test effort (TCO) significantly and could easily cause disruption during an SAP release upgrade or EHP. Consequently, the use of enhancement solutions should be examined very carefully.

ABAP CDS extensions

These enable modification-free enhancement of CDS views. The enhancement facilitates the addition of view fields and associations and is subject to restrictions specified in the ABAP key word documentation for the respective NetWeaver releases.[59]

If the desired result cannot be achieved with the previously stated enhancement techniques, a number of other options are available, the use of which should be considered on a case to case basis.

Implicit enhancements

Code can be added or the complete code (methods) substituted.at the start and at the end of procedures.

There is a major difference between implicit and explicit enhancements: implicit enhancements are similar to modifications with some of the same disadvantages. Explicit enhancements are similar to BAdIs.

If implicit enhancements are used SPAU_ENH should be executed as these enhancements are not displayed in the regular SPAU transaction.

The decision as whether to use implicit enhancement options not only depends on the implementation cost, but must also consider possible subsequent costs.

59 CDS View SAP NetWeaver 740
CDS View SAP NetWeaver 750

Modification

Basically, modification of source code supplied by SAP is problematic because:

- modified development objects are not subject to SAP maintenance and therefore need to be restored by the customer within the scope of SAP support ticket processing.
- modifications must be checked for compatibility with the SAP source code and adjusted if necessary during each upgrade of the SAP system at the latest and, in some cases, also during the installation of SAP notes or service packs.

Therefore modifications should only essentially be applied if:

- customising or personalisation cannot meet the requisite requirements and
- no suitable enhancement options are planned.

Within the change process, as an exclusive case modifications should be mapped separately in the interests of traceability.

Copying into own namespace/Z namespace

Copying SAP standard source code into the customer namespace requires a great deal of maintenance. Executing copies is not recommended. There is no automatic process or manual provision regarding how to effect subsequent alignment (e.g. following installation of support packages or notes) between the original and Z-copy.

BEST PRACTICE

- Priority should be afforded to the use of enhancement options provided for by SAP (BAdIs, user exits, customer exits, BTEs, enhancement points or sections).
- Generally, workbench modifications are only permitted using the modification assistant!
- In certain situations, the use of a Z-copy will involve extremely high realisation and subsequent costs. Moreover, further developments within SAP standard bypass the Z-copy without consideration, in turn generating costs for the adaptation or creation of a new Z-copy. Problems may occur with the standard includes applied following the importation of enhancement packages.
- The decision regarding modification vs. Z-copy vs. implicit enhancement not only depends on the realisation cost, but is also determined by potential subsequent costs.
- None of the options (modification/Z-copy/implicit enhancement) are purely advantageous or purely disadvantageous. A check should be carried out in each individual case to determine which technique offers the least disadvantages in the specific case.
- Centralised, formal and technical documentation should be created for each type of enhancement, For this purpose, suitable templates should be provided and a duty to use them should exist.

ADDITIONAL SOURCES

1. SAP Training BC425 and BC427

8.5 AUDITABILITY OF APPLICATIONS

8.5.1 TEST PROCESS BASICS FOR THE CREATION OF SOFTWARE PRODUCTS

Application testing is a software quality assessment[60] and improvement tool. Amongst other things, such testing serves to check functional and non-functional product requirements, reveal anomalies in terms of system resource use and ensure precise orchestration within the scope of temporally dependent process steps. In this respect, the earlier a defect is identified and can be remedied, the lower the resulting cost.

Error Correction costs

The correlation between debugging and respective subsequent costs is ably illustrated in the elder, but nonetheless still basically valid study by Barry W. Boehm on the ´relative cost of debugging the development cycle´ from 1981. According to Boehm, a bug found in the maintenance or operational phase will be up to 100 times more expensive to resolve than if it had been corrected in the analysis phase. Drawing on Balzert[61] the following table provides an example orientation aid and illustrates the phases of the development cycle in which certain error frequencies are likely to occur and the likelihood of the error being discovered.

Phase	Analyse	Design	Implementation	System test	Acceptance test	Maintenance/ operation
Relative expense	0,2	0,5	1	2	5	20
Notional corrective costs	€1	€2,5	€5	€10	€25	€100
Errors introduced	55%	30%	15%			
Errors found	5%	10%	40%	45%		

60 Cf. https://de.wikipedia.org/wiki/Softwarequalit%C3%A4t#QS-Schwerpunkt_Softwaretest
61 Cf. Balzert, Software management, Spektrum 2008 p. 484 et seq

Error avoidance measures

The discovery and handling of defects can be supported with the use of various aids. From the definition of a process model (e.g. I2M, ITIL, V-model) to establish test levels and quality gates, the alignment of a quality standard (e.g. SQuaRE, ISO/IEC 9126) to define test criteria and the use of various product development methods (e.g. SAP Agile SE, KANBAN, Scrum) for handling production and acceptance criteria, right through to the introduction of an effective risk management system and establishment of protection classes for application categories, the available spectrum in terms of testing is extremely broad and dependent on a number of factors. To name but a few, these include the sector in which the company operates, statutory framework conditions and the potential impact of damage the application can cause to the business process.

Test levels

In essence, three different test procedures can be distinguished in relation to the production of software products:

- White box
- Grey box
- Black box

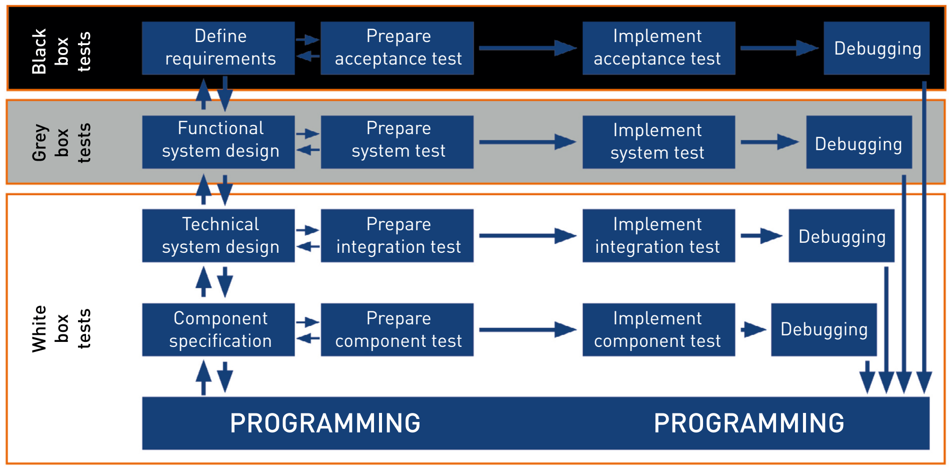


Figure 4: W-model with test levels [61]

Typical examples of white box tests are component and integration tests. A common denominator in these tests is that the structure, implementation and sequence of tested software artefacts are known. The tests are generally created, performed and further developed within the scope of development directly by the developer team within the development system.

The grey box test process precedes the acceptance test and is carried out following production of the product or a product increment in the form of system tests. Generally, these tests are performed in an environment similar to that of the production system by a separate test team. The organisational division between grey and white box tests has the advantage of allowing the development team to focus on its core tasks. The test team validates the application from an alternative perspective and seeks to identify possible weak spots or defects and the commensurate causes.

The black box test process examines the finished product without knowledge of the internal software structure. Test scenarios are derived from the specifications and the requisite functionality contained therein and serves as an acceptance test in relation to acceptance of the developed product. Black box tests are carried out by the customer in an environment that mirrors that of the production system as closely as possible.

62 In reference to http://www.informatik.hs-bremen.de/spillner/ForschungSpillnerWmo.pdf

BEST PRACTICE

- Ensure that the defined test strategy is suitable for your organisational and operational structure, role responsibilities are clearly defined and development team productivity is not hampered by unnecessary organisational interfaces.
- Base the intensity and extent of testing on the anticipated level of damage in the event of error and its likelihood of occurrence. Ensure that only absolutely necessary test cases are carried out and avoid redundancies.
- Involve the customer in the test case definition process at an early stage. An overall test scenario to validate the requirement should be established with the customer during the requirement definition phase. An improved product specification frequently results as a positive ancillary effect. An initially general description of test cases in the requirement definition phase can be honed during subsequent development phases.

8.5.2 TEST AUTOMATION

Test automation is suitable for all areas in which a sequence of manual work steps regularly has to be performed to check the correct behaviour of a functionality. The decision as to which actions should be automated will depend on the risk category, the level of complexity, frequency of execution and the ratio of manual costs to automation costs. Repeatable automated tests can be extremely efficient.

Two approaches apply:

- at source code level: unit tests with ABAP Unit
- on a higher level: automated function tests using eCATT or third-party tools

Advantages of automated tests:

- Software functionality can be tested following changes without the additional investment of time.
- Unit tests are useful for finding errors as early as the development phase and help developers create a good design (modularisation, simplicity).

Disadvantages of automated tests:

- following changes existing tests and test data need to be adapted to some extent
    - This should be a somewhat seldom requirement for unit tests with good modularisation.
    - In the case of automated function tests, this will depend on whether the changes impact down to 'the very core' of the tested level. Interface changes are generally more problematic because the tests are usually controlled from this level, meaning controlling has to be adapted.
- greater initial expense
- possibly more complex implementation, e.g. for unit tests, if database calls are not encapsulated

The use of unit tests is particularly favourable in the following scenarios:

- Frequent software modifications are to be expected
- Software parts will presumably be reused,
- Complex functionality
- SAP NetWeaver software version

- Test Double framework available (SAP NetWeaver 7.40 SP9 upwards)
  - Alternatively, the open-source tool MockA[63] is available for release SAP NetWeaver 7.01 upwards
- „Test Seam" ABAP statement available for SAP NetWeaver 7.50 upwards

The use of unit tests is less advisable in these scenarios:

- Classes designed to provide database access ('database layer', 'model' in model view controller model) that contain no complex logic.
- Classes designed for control and communication with the interface ('view' in the Model View Controller design pattern) that contain no complex logic.
- Existing software with poor modularisation (above all if test seams are not yet available, see above).

BEST PRACTICE

- Consider the use of automated tests.
- Develop competence in this area.

In the ABAP environment, database or interface integration can frequently hamper the creation of repeatable, automated tests. For tests, the modularization is the already useful solution.

BEST PRACTICE

- Within your applications, separate direct interaction from the database, user interface and remove systems from the actual application core.

63 see https://github.com/uweku/mockA

Where the application core, for example, no longer communicates directly with the database, data constellation simulation is an option for the implementation of unit tests. In this respect, the test doubles referred to above are particularly practical.

Separation of the layers can sometimes also be helpful for automated function tests; however the focus with eCATT centres on control via SAP GUI.

BEST PRACTICE

- In deciding which processes to use for automated regression tests, use the call statistics available in the SAP system (transaction statistics via ST03N/ usage procedure logging).
- When introducing unit tests or automated function tests start with a small committed team. The experience and success of this team can be used as a success story for further introduction of the subject within your organisational unit. It is important that the team is committed to its task and can substantiate the benefits of this activity with examples.

ADDITIONAL SOURCES

1. http://www.testbarkeit.de
2. http://de.wikipedia.org/wiki/Testbarkeit
3. http://www.testbarkeit.de/Publikationen/TAE05_Artikel_jungmayr.pdf
4. ABAP Unit Tests
5. ABAP Test Double Framework
6. ABAP Test Double Framework versus mockA
7. Test Seams and Injections

9 ECLIPSE DEVELOPMENT ENVIRONMENT

In the past, developments within the SAP environment exclusively addressed ABAP Workbench tools (SE80). Within the scope of new technologies, recent years have seen the launch of other development tools such as SAP Web IDE (SAP UI5 browser-based development) or SAP HANA Studio (Eclipse-based environment for administration and native development on/with HANA).[64] As of AS ABAP 7.31 SP4, the successor to ABAP Workbench, ABAP Development Tools (ADT) for Eclipse, has been available for ABAP development. ADT is based on the Eclipse development environment, expanded by the commensurate plug-ins for enabling development in ABAP.

9.1 REQUIREMENTS AND INSTALLATION

As a minimum, the aforementioned release AS ABAP 7.31 SP4 and Kernel 7.21 are required to use ADT for Eclipse. However, the range of available ADT functions is not only dependent on the ADT version used, but also the version of AS ABAP. A continuously updated overview of the functions available with the various AS ABAP versions is obtainable under [1] in SCN.

In contrast to the ABAP Workbench, an additional installation at the developer workstations is required for ADT. An updated installation manual for ADT is available under [2].

As an alternative to the manual and depending on the size of the enterprise, an alternative basic package (Eclipse IDE + ADT) is to be made available via the standard distribution mechanism, which only needs to be updated locally to the latest version (via the SAP update site or in-house).

9.2 NECESSITY

ADT for Eclipse has been designated by SAP as the successor to ABAP Workbench. New object types such as ABAP Core Data Services (CDS) can no longer be processed with ABAP Workbench (SE80 or other SAP GUI transactions) as this is under constant maintenance. Also the range of ADT functions is growing with every release. SAP GUI transactions seamlessly integrated into ADT are called for those functions not yet implemented in ADT (for example creating extensions). A changeover to ADT will effectively become necessary in the mid-term.

64 *This is not a recommendation for HANA Studio for developing native HANA applications.*

9.3 ADVANTAGES

The advantages and disadvantages of using ADT are principally dependent on the particular procedure and specific project environment. In the following subsections, we outline what we believe are the most important advantages and disadvantages of working with ADT. [1] provides an excellent overview of all the functions available with ADT as does the ADT FAQ document [7].

ADT workspace

Working with ADT means when editing ABAP source code, developers are no longer bound by SAP GUI modes. In other words, numerous sources can be open simultaneously and these will be maintained in the exact same form even after logging out/in. ADT links enable the creation of links with line-based precision to points in the ABAP source code and also the sharing of such (see [3]). It also enables organisation (see [4]) of the Eclipse tool Mylyn, the ADT workspace and opened ABAP source code using tasks (e.g. from a ticket system).

Configurability of the ADT workspace

Eclipse and consequently ADT enable the workstation to be configured more flexibly. Views can be enlarged or the position of certain views changed simply by drag and drop. Furthermore, the linking of views enables, for example, the ABAP Documentation view to always display the documentation relating to the currently selected keyword.

Editor functions

The editor itself offers a range of additional functions to enhance developer efficiency.

- Ongoing syntax check (not first after a manual call as in SE80 through pressing a button in the editor)
- Quickfix functions: the system suggests options for resolving current syntax errors (e.g. semi-automatic creation of a new method with the parameters used)
- Syntax or ATC check results are marked in the editor window and any associated messages are revealed per mouseover
- Code element information: pop-ups with information on development objects used in the code
- ABAP Doc as an integrated documentation option

Refactoring functions

ADT offers a plethora of automatic refactoring functions, which allow, for example, the renaming of a variable and all its applications or automatic deletion of unused variable declarations. More complex refactoring functions enable automatic extraction of a method or attribute.

Improved runtime analysis

The runtime analysis (see [5]) integrated within ADT allows the graphic visualisation of ABAP traces. This visualisation enables source code critical to performance to be easily identified and optimised.

Enhanced version history

Version history integrated within ADT can be remotely used across systems and also has a local (workstation-based) history, thereby providing optimum flexibility.

SQL tools

Classic SQL developer tools have also been reinvented in ADT, examples being an SQL console and an integrated data preview function.

Extensibility

As Eclipse is an open platform it is extremely flexible in terms of expansion, with the commensurate SAP tools consequently available. ADT can also be expanded using existing tools in the Eclipse ecosystem. Also available is an SDK (Software Development Toolkit) for ADT (see [6]) that can be used to expand ADT with customised functions. There is also an SAP code jam on the subject.

9.4 CONSIDERATIONS

In addition to the advantages mentioned, working with ADT for Eclipse also has a few disadvantages.

Difficulties getting started and with changeover

Starting out using any new tool involves an initial phase of readjustment and familiarisation. Specifically, the changeover from ABAP Workbench to ADT also means getting used to a new set of development paradigms. While form-based editors largely prevail (e.g. for classes and methods) in ABAP Workbench, ADT primarily uses source-code-based editors. So in addition to learning how to use a new tool, developers need to familiarise themselves with a new form of processing source code. In our opinion, this is a preliminary hurdle that should not be underestimated.

All things considered though, after changeover it ultimately becomes clear that development based on source code is significantly more efficient.

Omitted special transactions

ADT does not support all special transactions that are available on SAP GUI. Although the range of ADT functions is constantly being expanded (e.g. to include modelling functions for BOPF), all the special functions will still not be definitively available in ADT in future. These can however be called within ADT through SAP GUI integration. That said, application using SAP GUI integration is not always the ideal solution. Work with SAP CRM WebUI tools in ADT, for example, is sometimes inconsistent. Certain objects within the SAP CRM WebUI tools are opened for processing in ABAP Workbench, while other objects are opened in the ADT editor.

9.5 PROBLEMS AND SUPPORT WITH CHANGEOVER

Experience shows that the first few days working with ADT may be a bit bewildering. Particularly the free configurability of the development environment and development paradigms based on source code can prove difficult for experienced SAP developers. These may not initially be perceived as being advantageous and may even be regarded as an unnecessary modification devoid of added value. This will change after overcoming the initial hurdles and development efficiency will increase accordingly.

Despite having the option of changing all the shortcut keys, we recommend that developers familiarise themselves with the standard shortcuts. This will avoid any unnecessary setting up work for different workstations and remote working environments.

It is also a problem if ADT development is carried out on several systems that have a different release status. In this case, certain functions may not be available on all the systems (compare [1]).

If a company has a large number of developers, starting the introduction of ADT with a training seminar or presentation of the tool is an established way forward (terminology and basic settings).

Alternatively, integrated into ADT is the so-called Feature Explorer, which is a kind of self-study tutorial that explains the basic work process with ADT. For example, it shows how to extract a development system in ADT, or parts of a method using the refactoring tool.

A further induction option is to participate in an SAP code jam for ABAP in Eclipse.[65]

9.6 CONCLUSION

As of release 7.40 at the latest, the advantages of ADT will outweigh the disadvantages. The level of effort required for changeover is reasonable (albeit different, depending on the motivation and ability to learn). Changeover will likely be much more straightforward for younger employees who have worked in development environments outside the ABAP Workbench.

BEST PRACTICE
Based on our experience, we advise using ADT as of AS ABAP Release 7.31 SP6.

9.7 ADDITIONAL SOURCES

[1] ABAP in Eclipse Feature Matrix

[2] ABAP in Eclipse Installation Guide

[3] How ADT links change the way you work

[4] Use mylyn tasks to organize your ABAP in Eclipse workspace

[5] ABAP Profiling in Eclipse

[6] First version: SDK for ABAP Development Tools

[7] ADT FAQs

[8] BOPF Modelling in ADT

[9] ADT Feature Explorer

[10] Get Started with the ABAP Development Tools for SAP NetWeaver

65 Cf. http://scn.sap.com/community/events/codejam

10 USER INTERFACE (UI)

10.1 UI TECHNOLOGIES IN PRACTICE

UI technologies are playing an increasingly important role in development projects. As such, we would like to address this in this section of the guidelines.

The "new" SAP UI technologies form the focal point of the UI section given that; with SAP products, we are observing a clear trend towards SAPUI5 as a UI framework and SAP Fiori as the leading way to access the SAP environment.

Notwithstanding, we still rate web dynpro and classic dynpro, for example, as important UI components in the SAP environment. Many customers are availed of recommendations and also have a broad basic knowledge base regarding these long established technologies.

SAP EA Explorer[66] is recommended to gain an initial overview of the various UI technologies. This platform offers a good introduction to the various aspects of user interfaces. This guideline presents the key UI technologies listed in the following table. The table itself is limited to an overview and brief assessment for use in with (new) developments.

UI Technology	SAP Roadmap	Comment	Recommendation for new developments
Dynpro (classic)	Only support	SAP advises against new development projects. Small development projects, principally simple reports with generated selection screen. Favoured by power users	Still useful in many cases for small development projects
Business Server Pages (BSP)	Only support	Replaced by Web Dynpro	No longer practical
WebClient UIF	Only support	Developed in CRM on the basis of BSP technology and operational	Still relevant for classic CRM apps. SAP Hybris C4C uses SAPUI5/SAP Fiori in this case
Web Dynpro Java	Only support	Should no longer be used	No longer practical
Web Dynpro ABAP incl. Floorplan Manager	Minor expansions	In combination with Floorplan Manager less effort involved as standalone	Practical for large new development projects. Also consider SAPUI5
SAP Screen Personas	Minor expansions	Configuration and scripting (JavaScript), to make existing applications more appealing and operable based on classic dynpros	Practical for UI revision of existing dynpro programs
SAPUI5	Strategic		Practical. See advantages / disadvantages below

SAPUI5 is currently the favoured UI technology for state-of-the-art SAP applications. Nevertheless, we recommend that consideration is given to the UI technology to be applied at the beginning of a development project. The following contrast of advantages and disadvantages should help in the decision about the application of SAPUI5.

66 SAP EA Explorer

Advantages SAPUI5	Disadvantages SAPUI5
• Comprehensive collection of standard GUI elements that significantly simplify implementation • State-of-the-art look • Theoretically everything possible that the HTML5/JavaScript combination allows • Can be used on tablets and smartphones • No client to install • Responsive UI (automatically adapts to the respective screen size) • Utilisation of client device capabilities e.g. cameras • Native SAP Fiori Launchpad integration • Relatively new technology; as such, optimum integration in current web browsers	• Requires JavaScript (ABAP only in backend). Hence skill development necessary • Requires SAP Gateway (additional cost in the case of recommended separate installation) • Missing features and reduced performance in specific cases and circumstances in comparison to SAP GUI/ALV • Relatively new technology, hence problems possible when using tools and as end product • Complex apps require more effort (stateless apps)

SAP Fiori is the new user experience (UX) in terms of current SAP solutions established based on modern design principles. The subject of SAP Fiori is only briefly addressed in this SAPUI5 section, as the underlying technologies in the on-premise environment are currently SAPUI5 and SAP Gateway. The subject of the HANA Cloud Platform (HCP) is not explicitly broached in the UI section as the SAPUI5 part is not dissimilar to the on-premise environment and the on-premise best practices can almost be applied one-to-one.

Design thinking has become increasingly important for UX in recent years and should also be considered for comprehensive end-to-end processes.

Closely related to design thinking is the subject of mock-up. Established tools are available that encompass ready-to-use SAP Fiori Design Stencils[67] that visualise SAP Fiori UX patterns. A new SAP product in this area is Splash/BUILD, but the position in summer 2016 was that the product was still at the beta stage, so not a great deal of experience has been garnered.

67 SAP Fiori Design Stencils

10.2 SAPUI5

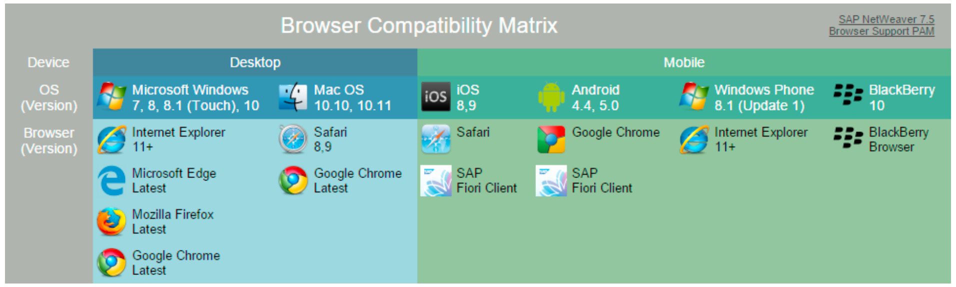
Within the framework of the SAP user experience strategy[68], SAPUI5 represents a modern toolkit for HTML5-based development projects. Applications developed with SAPUI5 are characterised by a responsive web interface on desktop browsers as well as mobile client devices.

SAPUI5 is also distributed through an open-source licence (Apache 2.0) under the name OpenUI5.[69] However, this distribution does not encompass certain components such as diagrams and smart controls.

Currently developed applications are being brought to fruition in accordance with SAP Fiori design guidelines[70] to ensure an optimised multi-device user experience and to assimilate the look and feel of SAP applications. From the beginnings of a library SAPUI5 now only contains out-and-out desktop-oriented components (sap.ui.commons), which should no longer be used (deprecated).

10.2.1 REQUIREMENTS

To use SAPUI5, a supported client device is required in the frontend that is supported by SAP within the SAP NetWeaver 7.5 PAM browser support.[71]



Browser Compatibility Matrix

SAP NetWeaver 7.5 Browser Support PAM

Device	Desktop		Mobile			
OS (Version)	Microsoft Windows 7, 8, 8.1 (Touch), 10	Mac OS 10.10, 10.11	iOS 8,9	Android 4.4, 5.0	Windows Phone 8.1 (Update 1)	BlackBerry 10
Browser (Version)	Internet Explorer 11+	Safari 8,9	Safari	Google Chrome	Internet Explorer 11+	BlackBerry Browser
	Microsoft Edge Latest	Google Chrome Latest	SAP Fiori Client	SAP Fiori Client		
	Mozilla Firefox Latest					
	Google Chrome Latest					

Figure 5: SAP NetWeaver 7.5 PAM browser support

68 SAP user experience strategy (UX)
69 OpenUI5 homepage
70 SAP Fiori Design Guidelines
71 SAP NetWeaver 7.5 PAM browser support

SAPUI5 is highly significant from a strategic perspective and is a supplied component with the following SAP systems:

- SAP NetWeaver AS ABAP (e.g. SAP Business Suite or SAP Gateway)
  As of NetWeaver Version 7.40 UI and SAP Gateway components are constituents of SAP NetWeaver. In previous versions they can be partially retrospectively applied as add-ons.
- SAP HANA
  SAP HANA delivers a version of SAPUI5 SDKs appropriate for HANA SP (XS Classic). As of SPS 11 and XS Advanced, SDK must be integrated via CDN.
- SAP HANA Cloud Platform (HCP)
  SAP HCP offers the option of developing SAPUI5 and Fiori apps and in the form of SAP Web IDE provides a development environment for SAPUI5.
- SAP Enterprise Portal (EP)
  SAP Web IDE enables direct deployment of an SAPUI5 application within the portal as an iView.

All systems also provide an SAP Fiori Launchpad (FLP), albeit in various stages of development.

BEST PRACTICE

- For SAPUI5 development, we recommend using the Google Chrome browser (position in 2016). In the form of UI5 Inspector[72] (Google Chrome extension) SAP provides a tool that is indispensable for developers. In the SAP Web IDE, the layout editor only supports Google Chrome (position 2016). End users can use the developed applications with all standard browsers and devices, insofar as supported by PAM.
- In addition to SAPUI5 SDK, all the above-mentioned systems also support SAP Fiori Launchpad (FLP). Role-based SAPUI5 applications can be started via FLP, if these were designed in compliance with SAP Fiori design guidelines and implement independent components.

10.2.2 DEVELOPMENT

SAP follows and supports the design thinking[73] approach with regard to modern UX development, which is then used to develop new processes and user interfaces. Design thinking encompasses three phases: discover, design and deliver.

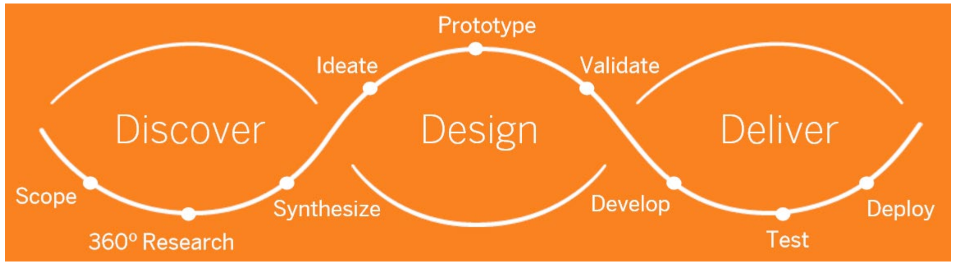


Figure 6: Phases of design thinking

72 UI5-Inspector (Google Chrome Extension)
73 Wikipedia definition of design thinking

10.2.2.1 Discover

The discovery phase involves endeavouring to understand a current customer problem in order to creatively work out a solution process within a team. This resulting solution stipulates a specific work process for a defined target group.

10.2.2.2 Design

The design phase involves outlining a prototype, with the target group, that represents the best possible work process for meeting the requirements from the analysis.

Depending on the know-how and target group, the following tools can be used:

- Paper and pencil
  Still the simplest way to record ideas.
- Whiteboards/whitewalls
  A more exclusive version of the paper and pencil.
- SAP Splash and BUILD
  With Splash and BUILD[74], SAP provides a browser-based tool with which prototypes can be created by drag & drop that comply with SAPUI5/ SAP Fiori standards and contain the commensurate widgets. A BUILD prototype can be imported into SAP Web IDE for further development.[75]
- SAP Web IDE
  SAP Web IDE[76] is actually a tool for developing and deploying SAPUI5 applications. However, the position since 2016 has been that an option is available in the layout editor to create UIs via drag & drop. Mock-up data can also be stored to enable a functionally expanded prototype incl. test data to be made available online without the need for prior development of data services.

BEST PRACTICE

- The user target group should become involved in the prototyping process as early as possible given that an application is designed to precisely meet the requirements and align with the work methods of the user. Subsequent acceptance is also increased because the target group was brought in during the design phase and assisted with the design.
- In mutual workshops, the fastest way forward to achieve results is through using the whiteboard. SAPUI5 Explorer[77] and the SAP Fiori Demo Cloud[78] are extremely helpful tools for teaching new users about SAPUI5.
- In 2016, Splash and BUILD still had a few bugs that partially hindered productive use incl. code acceptance.
- The openSAP course 'Build Your Own SAP Fiori App in the Cloud – 2016 Edition[79]' provides a comprehensive introduction and overview of the subject SAP Fiori UX.

74 *SAP Splash and BUILD (Design Great UX in the Cloud)*
75 *Kickstart your Fiori Web IDE project with Splash and BUILD*
76 *SCN – SAP Web IDE – Quickstart*

77 *SAPUI5 Explored (UI Explorer)*
78 *SAP Fiori Demo Cloud*
79 *openSAP course 'Build Your Own SAP Fiori App in the Cloud – 2016 Edition'*

10.2.2.3 Deliver

If a commensurate prototype is adopted within the scope of prototype design, this can then be transferred to an SAPUI5 application.

Tools

SAP supports SAPUI5 development with the following tools:

- SAP Web IDE (on the cloud platform)
  SAP Web IDE is a favoured SAP tool supplied within the scope of SAP HCP. In addition to the free trial version (recommended for non-productive purposes), a compact entry-level version[80] for 5 named developers was introduced in 2016. The IDE supports the entire prototyping process from graphic layout editor to deployment in the cloud, onto the SAP system (when using SAP HANA Cloud Connector), or the SAP portal (when using the commensurate SAP Web IDE plugins).

- SAP development tools for Eclipse
  As an alternative to SAP Web IDE, an Eclipse-based environment can also be used and the required SAPUI5 functionality retrofitted using SAP development tools for Eclipse.[81] Development and testing is then carried out locally within Eclipse. The finished SAPUI5 application/components can be directly deployed onto the SAP system (as of AS ABAP 7.31 with Team Provider add-on). Currently, the plug-in is under maintenance at SAP, in other words, no further development is being carried out. Compatibility updates will still be delivered.

BEST PRACTICE

- The recommendation in the SAP Fiori 2.0 guide is to develop SAPUI5 user interfaces with the SAP Web IDE and to use Eclipse ADT for SAP backend services; the reason being that Core Data Services (CDS) are not supported on SAP GUI (position in 2016).
- The SAP Web IDE offers the most comprehensive range of tools and options for SAPUI5 development on the cloud platform. Testing against an OData service in SAP Gateway (http Destination) and deployment also requires SAP HANA Cloud Connector, which opens a tunnel via Internet proxy and consequently builds a trust position to the SAP Cloud. SAP systems and services that are still released can then be authorised within SAP Cloud Connector.
- If use of the Cloud Connector is not possible, the app developed using SAP Web IDE can be exported (ZIP file) and subsequently deployed to the ABAP system via Eclipse ADT or the report /UI5/UI5_REPOSITORY_LOAD.
- Alternatively, Eclipse can be used with the commensurate tools. However, the graphic editors (WYSIWYG layout editor), templates and SAP Fiori extension support[82] will not be available in this case.

80 SAP Web IDE Entry-level version for 5 developers
81 SAP development tools for Eclipse
82 SAPUI5 extension using component configuration

Test

The SAPUI5 SDK provides functionalities for unit and component testing, with which tests can be largely carried out automatically.

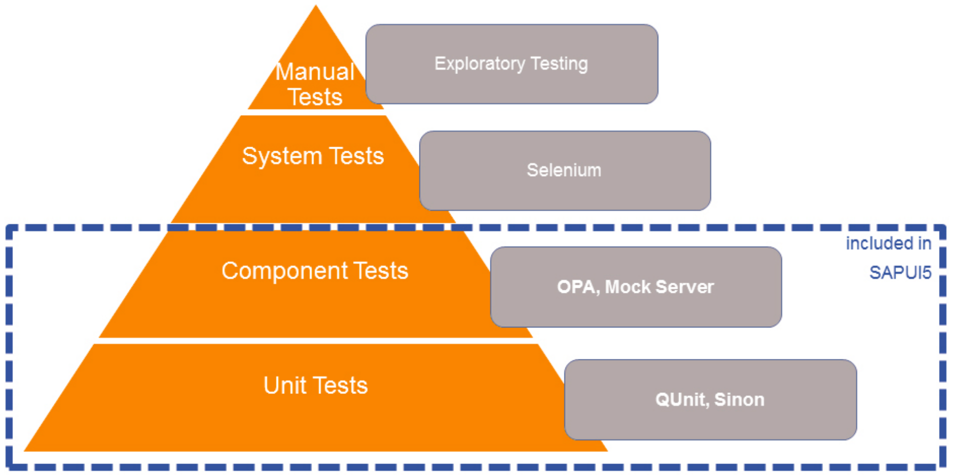


Figure 7: Supported test phases in SAPUI5

SDK

The SAPUI5 Demo Kit[83] encompasses all the relevant information pertaining to SAPUI5 development. The kit is effectively an SAPUI5 knowledge base provided by SAP.

BEST PRACTICE

- Prior to a project, every SAPUI5 developer should work through the developer guide (a component of the SAPUI5 Demo Kit). The walkthrough tutorial in particular demonstrates the basic use of core functionalities.
- Frequently during development, code samples from the Explorer app can be directly incorporated into custom coding and modified.
- Development based on SAP Fiori guidelines uses the library sap.m. sap.ui.commons components which are deprecated, i.e. where possible, thus they should not be used. The latest versions of the controls listed here are broadly available in redundant form within sap.m components.

Interfaces

To support communication between the SAPUI5 application and an SAP backend system (AS ABAP), a tool is available for SAP Gateway that as a service provider offers OData Rest services as well as the SAPUI5 SDK. Service implementation is carried out with the aid of ABAP Objects.

Within the SAPUI5 framework, service communication generally occurs via the commensurate data models. This can involve the following models depending on the scenario:

- OData[84]
  The OData model defines the data type and structures and enables REST-based access to backend systems. SAP Gateway supports OData via the commensurate services and from NW 7.40 onwards via Core Data Services (CDS). From NW 7.50 onwards, CDS can be provided as an OData service directly via an annotation. This must first be manually created in SEGW.

83 SAPUI5 Demo Kit

84 OData REST specification

- JSON
  Data exchange via standard JSON structures via http(s) calls. Within the AS ABAP system, either the JSON RPC service can be used or an HTTP handler.

- XML
  Data exchange via XML structures. The AS ABAP server accordingly provides the requisite XML transformations.

In addition to the models that enable request-based access to backend systems, a further technology is also available in the form of the WebSocket API:

- WebSocket
  Push service technology with which a backend system can actively push data to a client. The SAP-specific SAP Push Channel Protocol (PCP) can be used in this connection. SAP Gateway provides the commensurate APC/AMC services.

BEST PRACTICE

- Version 2 of the OData model is the data exchange model favoured by SAP for accessing backend systems (position in Q2/2016). Version 2 and 4 are implemented in SAP Gateway.
- Where possible all communication should be implemented on the basis of OData services, whereby one component should communicate with only one OData service (recommendation SAP Fiori design and simultaneous restriction of the SAP Web IDE).
- In certain scenarios, additionally including WebSocket communication in the application scenario is an option. This enables a real-time response to asynchronous processes on the server.

10.2.3 GENERAL RECOMMENDATIONS

- New implementations should always be based on the SAP Fiori design guidelines[85] and use the sap.m library. This is the only way to ensure that developers do not apply deprecated components, which are no longer further developed and could disappear from the library at some point.

- UI design should exclusively use XML views, as only these are supported by tools such as SAP Web IDE. The SAPUI5 extension framework only supports XML views within the scope of extending SAP Fiori apps.

- The library sap.ui.commons contains components that are not covered by the SAP Fiori design guidelines and which were designated as deprecated at UI5con. Insofar as possible these libraries should not be used. Many customers have built large-scale application scenarios on this basis; which then have to be migrated to sap.m in the medium term to remain upgradeable.

- One option to start with is to use an SAP Web IDE template. Unfortunately the quality of the templates differs significantly as they were based on the SAPUI5 options available at the time they were created. The best option is to use a suitable template based on the latest respective version.

- In the SAP Web IDE projects can be created on the basis of SAP Fiori Reference Apps[86], which adopt a best practice approach for the apps Shop, Approve Purchase Order and Manage Products.

- The option of a UI prototype with mock-up data is available prior to developing relevant services as SAP annotations[87] in the SAP Web IDE are easier to write and test than in the SAP Gateway designer.

- Business data should be accessed via OData entities provided by the SAP backend system (e.g. Gateway). Consequently, the entire business logic is controlled and managed in the backend. The SAPUI5 interface only maps the relevant field information and controls workflows und visibility.

85 SAP Fiori Design Guidelines
86 SAP Fiori Reference Apps
87 SAP Annotations for OData Version 2.0

- Developers should use smart controls[88] (or at least smart fields) where possible to enable data type control by the OData service. If OData services based on annotated CDS views cannot be used (only as of NW 7.50), then the required SAP annotations can be applied manually and declaratively in SAP Gateway.
- SAPUI5 does not currently support customer-specific namespaces (e.g. '/SAP/FLIGHT'). When using smart controls the relevant Gateway services should be in the Z namespace.
- Use the UI5 extensibility concept[89] for adapting and extending existing applications and components.
- The UI theme designer available on all supported systems is usually used for theming. A custom theme is normally based on the SAP BlueCrystal theme. As of SAPUI5 1.38, work is being carried out on two versions of the new Belize theme, which will showcase the new SAP Fiori 2.0 design. Belize theme version 1.40.x will become the new standard theme and replace BlueCrystal.

10.2.4 ADDITIONAL SOURCES

- SAP Fiori Design Guidelines
- SAP User Experience Community
- SAP Fiori Cloud Demo
- SAPUI5 SDK
- SCN SAPUI5 Developer Center
- SAP Web IDE
- OpenUI5
- openSAP courses
  a. Developing Web Apps with SAPUI5
  b. Build Your Own SAP Fiori App in the Cloud – 2016 Edition

88 SAPUI5 smart controls
89 Extensibility concept

10.3 SAP GATEWAY

SAP Gateway serves as a mediator between web technologies and classic SAP solutions. SAP Gateway exposes data from ABAP-based backend systems.

10.3.1 USING SAP GATEWAY

The functionality of backend systems, which may be subject to extensive change, should not be made directly available externally. Using APIs, the aim is to provide a variety of consumers with stable access points on the systems (see API Economy[90]).

A primary application area for the SAP Gateway is user interface integration. Typical consumers are websites, native mobile apps, web apps based on JavaScript (see Fiori and UI5) and cloud platform applications. Whereas previously RFC-based technologies have mainly been used for SAP development, RESTful services dominate for the above application scenarios, usually in combination with the OData protocol.

OData

The OData protocol is a data exchange standard for the web that has full CRUDQ functionality (Create, Read, Update, Delete, Query), which is why it is also known as ODBC for the web. OData is based on the Entity Data Model, which describes each modelled entity (object) and its associations.

JSON (JavaScript Object Notation) and ATOM/XML data formats are supported.

90 CW: API Economy article

Deployment options

SAP Gateway can be deployed in three different variants:

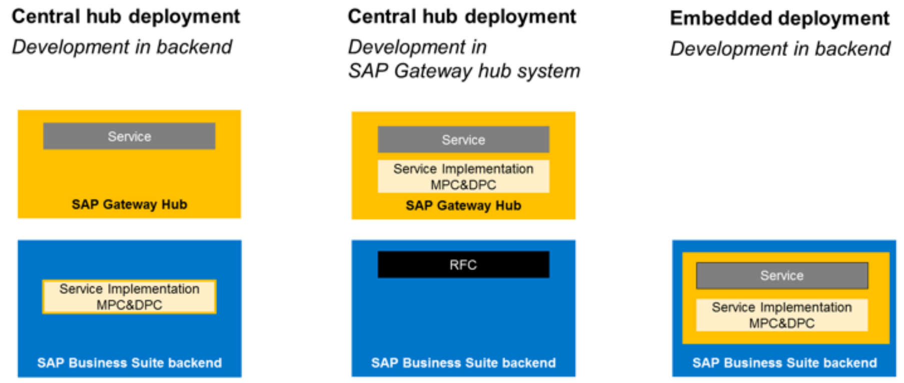

Central hub deployment
Development in backend
Central hub deployment
Development in SAP Gateway hub system
Embedded deployment
Development in backend
Service
SAP Gateway Hub
Service Implementation MPC&DPC
SAP Business Suite backend
Service
Service Implementation MPC&DPC
SAP Gateway Hub
RFC
SAP Business Suite backend
Service
Service Implementation MPC&DPC
SAP Business Suite backend

Figure 8: SAP Gateway deployment options [91]

BEST PRACTICE
We recommend using the first variant (central hub deployment, backend development). The main reasons for this are better scalability and better integration into the network infrastructure (e.g. Gateway in the DMZ). Service implementation (business logic) takes place in the backend. The second variant is only used if the backend system does not have the necessary release status or there is no way to deploy the requirements necessary (IWBEP components, NW Basis Release < 7.40). The embedded deployment variant offers a streamlined start, but should not be operated as long-term solution.

A fourth deployment option is now available with the SAP Fiori Cloud Edition, whereby the SAP Gateway hub operates within the HANA Cloud Platform.[92]

10.3.2 DEVELOPING WITH SAP GATEWAY

RESTful services are an important component for realising stateless web apps, i.e. SAPUI5-based apps. No app state or user session is stored in the backend system, only the state of the last communication is known. This is in complete contrast to ABAP programming with SAP GUI, where the backend always manages the condition of the client (stateful apps). This has consequences for programming because pessimistic lock logic can no longer be used for data changes within an environment of high-frequency, parallel access operations. Moreover, there is also no session handling in the classic sense, i.e. after closing a browser session only the last state following the last OData service call in the backend is known.

If service calls are idempotent, i.e. always evoke the same result in the backend (create, delete), then no problems arise. Difficulties emerge in the case of changes, as locks should only be set for as long as the OData call is being executed in the

91 SAP Help: SAP Gateway Deployment Options

92 SCN SAP Gateway deployment options in a nutshell

backend. As such, when a new query is initiated, it is possible that the data may have been changed by another party. In such a case, the decision has to be made during development whether such a change is relevant or not. A change that is not relevant could be, for example, an approval step, whereby in accordance with the FCFS rule the first changing party wins, and for subsequent callers the web app merely refreshes with the change in question. For critical interfaces in which data synchronisation between the backend and API consumers plays a major role, we recommend the Gateway eTag-functionality.[93] This enables data currency to be checked within the application and provision of an appropriate response. Notwithstanding, options also exist for continued use of classic lock concepts;[94] although greater development effort should be planned for in this case. Furthermore, concepts known from Idoc Inbound Processing, for example, can also be used.

Developing with SAP Gateway generally consists of three phases, in which proposals for naming conventions and best practices are presented. For the development of OData services, SAP provides the Gateway Service Builder (transaction SEGW).

OData modelling

API design plays a significant role in development projects. This triggers a classic conflict of objectives between API reusability and fast data availability that is as efficient as possible. Under no circumstances should the (standard) function module interface simply be made available in the original form. More suitable is, for example, encapsulation of the functionality using wrapper function modules. API design should be driven by API consumer (e.g. a SAPUI5 web app) requirements.

- We consequently recommend observing the following for OData modelling:
- Entity type: entities should be tailored to the respective application and not driven by the underlying database model. The application should be reflected in the entity name.
- Complex types: serve entity structuring.
- Entity sets: entity type group, SAP recommends suffix sets, alternatively the plural of an entity.
- Associations: associations should be formed where possible. The relationships between data are highlighted and the number of calls minimised.
- Navigation properties: use entity type for relationship 1 and entity set[95] for relationship N

Service implementation

For implementation use the standard ABAP Objects guidelines and naming conventions.

When using the Gateway Service Builder ensure that the generated class only uses the first 20 characters of the Gateway project name. Ideally the project name should be selected so that either the first 20 characters enable accurate service identification or the name must be adapted to the generated class. Otherwise a counter is automatically added to the generated class (Model Provider Class, Data Provider Class), which makes identifying the class during development difficult.

Service configuration

The external service name should draw on the functionality. Suggestions for the naming convention:

- Technical service name:
- Z + <external service name>
- The _SRV suffix is automatically appended by SAP.
- External service name:
- Namespace: /<customer namespace>/ (if customer namespace is used, otherwise Z*)
- Optional: GW_* as a prefix or infix enables simpler allocation as a Gateway service
- System alias: using the alias name as the landscape name is recommended as system aliases are transportable.

93 *SAP Help: ETag Handling*
94 *SCN blog: Stateful*
95 *SCN-Blog: OData Templates*

10.3.3 GENERAL RECOMMENDATIONS

The following subsection contains a number of guidelines that should be observed when developing with SAP Gateway. Some points refer to additional sources.

OData/functionality

- SAP Help provides an excellent overview of OData best practices.[96]
- Observe the 'separation of concerns' design principle when developing OData services. Business logic belongs in the backend, service implementation should be limited to 'glue code'.
- Complete OData syntax is not offered. SAP Note 1574568 encompasses an overview of what is supported, e.g. restrictions exist for $filter options.
- The use of OData annotations[97] simplifies the UI development process considerably. Consequently, the use of annotations in OData services is recommended especially from release status 7.50 onwards.
- When using multiple backend systems within an SAPUI5 web app, use of the Multi Origin Composition (MOC) functionality is recommended.

Performance

- Use $batch and $expand[98] instead of multiple parallel calls.
- With the $select option, OData gives the option of limiting the volume of transferred data. In principle, as little data as possible should be transferred.

96 *SAP-Help: OData do's and dont'*
97 *SCN blog: Annotations in ABAP*
98 *SCN blog: Performance do's and dont's*

Security

- The use of TLS is mandatory (HTTPS)
- When using SAP Gateway as a hub, a trust relationship[99] (trusted RFC link) must be established between the Gateway and called backend system.
- The SAP NetWeaver ABAP application server supports a variety of authentication methods that can be used in the SAP Gateway. Various methods are recommended (Kerberos, SAML 2.0 ...) depending on the scenario (desktop applications, web applications...).[100]

10.3.4 ADDITIONAL SOURCES

- SAP Community Network
- SAP Online Help

99 *SAP Help: Trusted System*
100 *SAP NetWeaver Gateway Authentication and Single Sign-On*

11 THE AUTHORS

The following authors contributed significantly to the creation of the 2nd edition of these guidelines:

Dr. Christian Drumm
Head of Application Development & Consulting, FACTUR Billing Solutions GmbH
Dr. Christian Drumm has worked within the SAP environment since 2004. Following a period with SAP research, he has held various roles (developer, project manager, architect) as a consultant focusing on SAP CRM and SAP IS-U. Since 2013 he has headed the application development and consulting department at FACTUR Billing Solutions GmbH.

Martin Fischer
Portfolio Unit Manager SAP Database & Technology, BridgingIT GmbH
Martin Fischer has been involved with SAP since 2001. After starting out as an SAP FI/CO module functional consultant, his work since 2007 has centred on software development and architecture with an emphasis on ABAP. Following positions at various consulting companies, in 2011 he joined bridgingIT, where he has technical responsibility for the SAP database & technology portfolio.

Judith Forner
Senior Consultant Finance & Controlling, Mundipharma Deutschland GmbH & Co. KG
Judith Forner has worked as an SAP consultant and ABAP developer since 1999. In addition to classical process consulting in relation to FI, CO and PM modules, her focus centres on user-friendly enhancement of the standard SAP program and external system integration.

Edo von Glan
SAP Developer, Drägerwerk AG & Co. KGaA
Edo von Glan has worked as a software developer for IBM and Commerzbank and also spent seven years in product development at SAP. Since 2008 he has been employed in the SAP development department at Dräger, six years of which have been in a managerial position. His sphere of responsibility encompasses application and interface development, custom code lifecycle management and software quality for the SAP systems administrated centrally within the company.

Florian Henninger
Senior Consultant SAP Development, FIS GmbH
Florian Henninger is a certified ABAP developer and has worked in SAP-ABAP development since 2010. His professional focus centres on core development/enhancements and output management in relation to ERP implementation projects and existing systems. In addition, he is also a quality management team member, coach and SAP mentor.

Martin Hoffmann
Head of Software Engineering, Miele & Cie. KG
Martin Hoffmann has worked in a development-related environment since 1987, gaining experience both as a developer and in various managerial positions within the development sphere. Since 2007 he has been responsible for software engineering at Miele.

Valentin Huber
Senior IT Consultant, msg systems ag
After previously gaining experience in C++ and Java development, Valentin Huber has worked as an ABAP developer and software architect for custom developments since 2012. As an iSAQB certified professional for software architecture (foundation level) and advocate of clean code development, he has a passion for the extensibility and maintainability of software systems.

Jens Knappik
SAP System Architect, thyssenkrupp Materials Services GmbH
Jens Knappik has worked as an ABAP developer within an international project environment since 2004. Up to 2012 he steered product development of the SD-CAS and CS modules in his capacity as lead developer. Since then he has been a consultant for the strategic alignment and definition of standards in the ABAP environment for the central business area template.

Dr. Christian Lechner
Principal IT Consultant, msg systems ag
Dr. Christian Lechner has worked in SAP software development (standard and custom development) in various functions (developer, project manager, software architect) since 2005. Since 2015 he has headed the architecture department of the development business unit at msg systems ag.

Steffen Pietsch
Head of Backoffice, Haufe-Lexware GmbH & Co.KG
Steffen Pietsch has worked in SAP development-related fields since 2003, gaining extensive practical experience as a developer and in various managerial positions in the consulting and development environment. Since 2009, as spokesperson of the DSAG working group on development, he has championed the interests of customers and partners in cooperation with SAP.

Daniel Rothmund
IT Business Analyst SAP, Geberit Verwaltungs GmbH
Daniel Rothmund has worked in various roles within the SAP environment since 2008 and has been responsible for the central SAP development team at Geberit since 2016. As spokesperson of the DSAG working group on UI technology he has championed the interests of customers and partners in cooperation with SAP since 2015.

Holger Schäfer
Business Unit Manager, UNIORG Solutions GmbH
Holger Schäfer has worked within the SAP environment since 1999, focusing on user interfaces, integration, HANA/HCP and e-commerce. He is also a member of the working group on UI technologies and the OpenUI5 community. Since 2010 he has been responsible for the strategic business development of new SAP technologies

Denny Schreber
Senior Solution Architect, cbs Corporate Business Solutions Unternehmensberatung GmbH
Denny Schreber has worked with SAP-related technologies since 2007, focusing on mobile and UI technologies and integration. Previously he was involved in the development of Java software. Since 2015 he has been deputy spokesperson for DSAG-AG UI technologies.

Andreas Wiegenstein
CEO and Co-founder of SERPENTEQ GmbH
Andreas Wiegenstein works in the field of SAP security since 2002, focusing on cyber risks. He frequently lectures on the subject of SAP security at international conferences such as Troopers, IT Defense, RSA, Black Hat, Deep Sec, Hack In The Box, hackERP, SAP TechEd and at DSAG events.

Bärbel Winkler
System Analyst SAP Basis/Programming, Alfred Kärcher GmbH & Co. KG
Bärbel Winkler has been involved with ABAP programming and project work in the SAP environment since 2000 and has worked on custom developments and their coordination at Alfred Kärcher GmbH & Co. KG. since 2011. Her responsibilities include the regular updating of development guidelines, evaluation and appraisal of detailed development requirements and overseeing the implementation of external programming.

APPENDIX A: NAMING CONVENTIONS

Two contrary approaches exist for ABAP naming conventions. The main distinction is whether the data type (structure, table, reference, primitive...) should be encoded with a prefix letter in the object name or not.

Naming conventions that prescribe the data type as a prefix to object names normally also call for further technical, functional or organisational categories to be included in the prefix. As such, these could be described as 'long' naming conventions. There is a connection to the Hungarian Notation invented by Microsoft.[101]

A closer look reveals that the long naming convention is only half-heartedly applied, as with encapsulated data types, where the convention is only applied at the first level, (LS_CUSTOMER-PARTNER-HISTORY instead of LS_CUSTOMER-S_ PARTNER-T_HIS-TORY).

In contrast, there is the 'short' naming convention, where previously only prefixes were stipulated to differentiate the parameters (I_ for IMPORTING, E_ for EXPORTING, etc.) and the focus instead currently lies on meaningful 'definitive' names. The authors of the 'ABAP Programming Guidelines' (now part of F1 Help) are advocating this type of naming convention.

Advantages of long naming conventions

- The object type is immediately clear upon reading the source code, without, for example, the need to navigate in the DDIC.
- Unintentional obscurity (e.g. types in methods vis-à-vis types in the class) is more easily avoided.
- Object names look consistent and organised through the uniform prefix (even if the definitive part of the name is not well selected).

Disadvantages

- A complete and unambiguous system is comprehensive and complex, see the discussion in the 'ABAP Programming Guidelines' manual.
- The reading speed of source code is reduced as technical and semantic information is mixed.
- The amount of effort required increases if the technical type and/or visibility changes. This is also the case where, for example, allocation to the application hierarchy/organisation is part of the name and the objects are to migrate to another area.
- Semantic information can be lost due to lack of space.
- Mastering, controlling and updating conventions involve more effort.

The advantages of short naming conventions

- More space and greater awareness of distinctive names when reading and writing.
- No redundant information (the data type is determined through forward navigation/F2 (ADT), original system is in the object catalogue, allocation to the application hierarchy can be derived via the package, etc.).

The first version of these guidelines supported a long naming convention variant, which on the basis of our own experiences and the current position of debate in trade literature and the SCN (see Appendix A.3) we no longer advocate.

Below is an example of what a more streamlined naming convention could look like.

General note:

- Objects in ABAP Dictionary are subject to various restrictions on the number of characters available, which have to be observed when naming an object.
- The customer namespace Y... is used in the following. As specified in Section 2, the alternative namespace Z... or a custom namespace /.../ can also be applied.

101 https://de.wikipedia.org/wiki/Hungarian_Notation

A.1 REPOSITORY OBJECT NAMING CONVENTIONS

Naming conventions in the ABAP repository serve to avoid name conflicts through duplicate names and by using a standard approach make object names more comprehensible. They also take developers out of the decision-making process, which enables them to focus on implementing requirements and means they do not have to invest effort in individually avoiding name duplication. Furthermore, conventions in the source code support the application of object-related operators (for example: INSERT instead of APPEND for adding another line to a sorted table). Repository objects are largely differentiated[102] by their object catalogue entries, which means from a technical perspective very few conflicts will arise during naming. Also, naming conventions are enforced by the development environment for various objects such as exception classes (prefix = CX) and lock objects (prefix = E).[103]

The following conventions expand on the principles defined in the information above. A primary aim of the guidelines is to provide a set of practical and pragmatic rules that largely avoid any overregulation. In this respect conventions are only established for competing objects based on SAP standard (e.g. prefix conventions for type information from the BOPF framework).

Naming conventions comprise the following elements and are sometimes separated by an underscore:

Element	Meaning	Example
[Namespace]	Customer namespace or reserved customer namespace.	Z = Development without transport layer Y = Transport-relevant development
[Type information]	Technical object abbreviation for a specific type. Not required for every object.	CL = Global class
[Context]	Consolidating work area, a product or the allocation to the application hierarchy or software components. Normally the context is formed via a structure or main package.	Application hierarchy SD: Sales & Distribution
[Semantic information]	Information on a domain object, whose purpose or known application pattern is based on a ubiquitous and comprehensible language (see Section 2.3).	OPEN_ITEMS_LIST
[Pattern]	A pattern is an optional part of semantic information and specifies an established procedure within the scope of design/programming.	Application: Worklist/Chart/Overview Design: Master/Detail implementation[104] Factory/Data Access Object/Gateway

102 Except tables and structures. These both have the TADIR object type TABL.
103 Further details on the commensurate objects are provided in SAP Note 16466

104 As multiple patterns are often used simultaneously within the scope of implementation, details of the patterns used are better commented in the source code than in the object name.

A.1.1 PACKAGE HIERARCHY

Type	Structure/main/development package: Packages are ideal for grouping multiple development objects into a semantically coherent unit.
Name formation	[Single noun|compound noun]
Examples	Application Hierarchy / Software Component: - Y_DEVELOPMENT_FOUNDATION (Structure package) - Y_ERP_CENTRAL_APPLICATIONS (Structure package) - Y_ACCOUNTING (Structure package) - Y_... - Y_LOGISTICS (Structure package) - Y_LOGISTICS_EXECUTION (Main package) - Y_PROCUREMENT (Main package) - Y_SALES (Main package) - Y_COMPUTER_AIDED_SELLINGS (Main package) - Y_CAS_COMMON (Development package) - Y_NETWEAVER_PLATFORM (Structure package) - Y_SAP_DELIVERABLES (Structure package) - Y_RAPID_DEPLOYMENT_SOLUTIONS (Main package) - Y_SAP_BEST_PRACTICES (Main package) - Y_SAP_NOTES (Main package) - Y_SNOTE_2294645 (Development package)

Type	Subpackages: Subpackages divide superior packages into specialised work areas. To highlight object association we recommend using the same prefix namespace for all objects in the package/subpackage hierarchy.
Name formation	[Abbreviation of superior package] [semantic information]
Examples	- Y_COMPUTER_AIDED_SELLINGS (Main package) - Y_CAS_BUSINESS_OBJECTS (Development package) - Y_CAS_COMMON (Development package) - Y_CAS_CORE (Development package) - Y_CAS_CONFIGURATION (Development package)

Type	Package interfaces: Naming package interfaces serves to declare an intention of how specific objects propagated via the package interface should be handled.
Name formation	[Package name] [[Visibility] | [Access type] | [Consumer]]
Examples	- Y_CAS_SALES_ACTIVITY_PUBLIC - Y_CAS_CONFIGURATION_READ - Y_CAS_CONFIGURATION_WRITE

A.1.2 DICTIONARY OBJECTS

Type	Elementary data types: Data elements and domains specify the technical and semantic characteristics of a data or reference type. As they can be technically differentiated via their object directory entry, identical names may be used.
Name formation	[Single noun|compound noun]
Examples	Y_CAS_ACTION_IDENTIFIER (Domains) Y_CAS_ACTION_IDENTIFIER (Data elements)

Type	Structure types: Multi-dimensional data types encompass one or more data types as components. They are divided into single line and multiple line structure types.
Name formation	[Single noun|compound noun]

Structures:
Structures historically share the same object directory entry with transparent tables. As such, it is imperative to differentiate the two types using a technical prefix.

Example:

YS_CAS_ACTION
YS_CAS_MASTER_LINE

Transparent tables:
Generally, the prefix D should suffice for transparent tables. If further division is necessary, due to the length restriction to 16 characters we recommend extending the prefix with the technical table delivery class or T for tagging text tables.

Example:

YD_CAS_CONFIG

YDC_CAS_ACTIONS (Configuration table)

YDT_CAS_ACTIONS (Text table)

YDA_CAS_MASTER (Master and transaction data)

YDL_CAS_FILEINPT (Temporary data)

Views:
If the prefix V is not sufficient for views, we recommend further division on the basis of the view type.

Example:

YV_CAS_CONFIG

YVC_CAS_ACTIONS (Maintenance view)

YVH_CAS_ACTIONS (Help view)

YVD_CAS_MASTER (Database view)

YVP_CAS_MASTER (Projection view)

Table types:
If multiple table types have to be created for the same line type, we recommend including the access type in the prefix and supplementing the semantic identifier with information on key definition. Otherwise, the prefix T should suffice.

Example:
YT_CAS_ACTIONS

YTH_CAS_ACTIONS (Internal table with hash access)

YTS_CAS_ACTIONS_BY_ACTIVITY (Internal table sorted by the ACTIVITY field)

Core Data Services:
Newly created CDS views create three repository objects. The DDL source, a CDS view and a CDS database view. To differentiate between CDS database views and normal database views, we recommend following the SAP standard examples[105] and marking the CDS database views with the prefix SQL.

Example:
YSQL_CAS_MASTER (CDS database view)

A.1.3 CONTAINERS FOR SOURCE CODE OBJECTS

Type	Executable applications The application name should immediately elucidate what the application does, which business object it focuses on and, if necessary, which process operates the application (worklist, wizard…). During naming we recommend adapting recognised operating procedures from Floorplan Manager[106] or the Fiori Design Guidelines.[107]
Name formation	[Single noun|compound noun] [Pattern]
Examples	Y_CAS_CUSTOMER_VISIT_PLANNER Y_CAS_SALES_ACTIVITY_WORKLIST Y_CAS_DATA_MIGRATION_WIZARD

105 SAP Help: 'Implement the CDS View as Data Model'
106 SAP Help: 'Floorplans Concept'
107 Fiori Design Guidelines: 'Floorplan Overview'

Type	Where possible, includes for framework programs and function group includes should no longer be used for reusable modularisation techniques and should be replaced with ABAP Objects for new developments. If their use is absolutely necessary, modularisation should be aligned to the dynpro or current business object.
Name formation	[Framework program | Abbreviation] [Container type] [Dyn-pro* | Counter]
Examples	Y_CAS_SALES_ACTIVITY_WORKLIST (Report) Y_CAS_SALES_ACTIVITY_WL_TOP (Global declaration section) Y_CAS_SALES_ACTIVITY_WL_O0100 (PBO dynpro 0100) Y_CAS_SALES_ACTIVITY_WL_I0100 (PAI | POH | POV dynpro 0100) Y_CAS_SALES_ACTIVITY_WL_F01 (Form routines) Y_CAS_SALES_ACTIVITY_WL_C01 (Local classes) Y_CAS_SALES_ACTIVITY_WL_T01 (Unit tests)

Type	Function groups for table maintenance views Generated function groups for table maintenance dialogue should have the same name as the associated maintenance view.
Name formation	[Name of maintenance view]
Examples	YVC_CAS_ACTIONS (Maintenance view) YVC_CAS_ACTIONS (Function group for table maintenance view)

Type	Enhancements Enhancement spots serve as containers for enhancements of objects that are specified in defined places with BAdI definitions. Configuration can contain specific enhancement implementations with diverse BAdI implementations, filters etc.
Name formation	[Name of the object to be enhanced] [[Filter information] | [Variants]]
Examples	Y_CAS_SALES_ACTIVITY_WORKLIST (Enhancement spot) - Y_CAS_SAWL_DEFAULT_EXTENSION (Enhancement implementation) - /ABCDEF/CAS_SAWL_EXTENSION (Enhancement implementation) - /ZYXWVU/CAS_SAWL_EXTENSION (Enhancement implementation) - Y_CAS_SAWL_ADD_VALIDATION (BAdI definition) - Y_CAS_SAWL_DEFAULT_VALIDATION (BAdI implementation) - /ABCDEF/CAS_SAWL_DV_D0100 (BAdI implementation) - /ZYXWVU/CAS_SAWL_DV_D0200 (BAdI implementation)

ABAP Objects

We recommend following SAP guidelines for naming ABAP object components.[108] Exceptions are the 'recommendation' type guidelines and local conventions within methods for parameters. For parameters, observe the recommendations in Appendix A.2.3 Signatures. Instead of the prefix recommendation for e.g. constants and local classes, it is better to use semantically appropriate and easily readable identifiers.

108 SAP Help: 'Naming Conventions in ABAP Objects'

Example:

```
“ Local class in the local types context of a global class
CLASS sales_activity_constants DEFINITION
  CREATE PRIVATE ABSTRACT FINAL.
  PUBLIC SECTION.
    CONSTANTS:
      BEGIN OF partner_function,
        key_account_manager TYPE tpar-parvw VALUE ’KA‘,
        ...
      END OF partner_function.
ENDCLASS.
```

A.2 NAMING CONVENTIONS FOR ABAP SOURCE CODE

Comprehension of naming conventions for source code is enhanced by using a prefix for particularly important type information. Otherwise, the focus is on clear and understandable naming (see Section 2.3).

A.2.1 CLASSIC USER DIALOGUES (SELECTION SCREENS/DYNPROS)

Due to the restriction in the length of components in classic user dialogue, we recommend using the following prefixes:

Type	Prefix
Parameters	p_
Select options	s_
Table dialogue structures for dynpro binding	none

A.2.2 VISIBILITY

Unintentional obscuring of local declarations can be avoided by defining prefixes for signatures and global variables/types. The introduction of further prefix conventions is not necessary from a technical perspective.

Type	Prefix
Global data objects	g_

Exception:

For local declarations within methods, the prefix convention l_ can be used in the following circumstances:

- The local declaration obscures a static class attribute.
- The method has to access the static class attribute.
- The class name is extremely long.
- Access via the component selector (=>) results in poor readability.

A.2.3 SIGNATURES

We recommend standardising use of signatures within the various procedure types (subroutines/function modules/methods) and applying the following prefixes.

Type	Präfix
Importing/using/tables	i_
Exporting/tables	e_
Changing/tables	c_
Returning	r_

A.3 FURTHER INFORMATION ON NAMING CONVENTIONS

For fans and advocates of a more comprehensive naming convention, for reasons of consistency we refer you to the appended de-facto standard for comprehensive naming conventions from the SCN community:
http://scn.sap.com/people/uwe.schieferstein/blog/2009/08/30/nomen-est-omen--abap-naming-conventions

Another somewhat reduced version is available here:
http://scn.sap.com/community/abap/blog/2016/02/05/fanning-the-flames-prefixing-variable attribute-names

Naturally we would not want to deprive you of the commensurate countergroup on this subject:
http://scn.sap.com/community/abap/blog/2013/05/23/abap-code-naming-conven-tions--a-rant

http://scn.sap.com/community/abap/blog/2015/09/22/hungarian-beginners-course--a-polemic-scripture-against-hungarian-notation

Highly recommended are the comment passages associated with the above blogs. Take the opportunity to consider things from the perspective of the respective faction and familiarise yourself with the commensurate positive and negative aspects.

Ultimately, you need to establish the conventions that offer you, your internal team and where applicable your external team, the greatest benefits in relation to productivity, desired quality and lifecycle costs. We highly recommend focusing on an existing standard and avoiding reinventing the wheel. This increases the chance that new employees will already be familiar with the standard and hence are able to generate added value for the company without the need for a long induction period.

To conclude we refer you to the official ABAP programming guidelines that served as orientation for defining the naming conventions listed here:
http://help.sap.com/abapdocu_750/de/abennaming_guidl.htm

A.4 MISCELLANEOUS/LESSONS LEARNED

A.4.1 CUSTOMER NAMESPACES

Where multiple system landscapes are employed between which developments are transported back and forth, incorporation of the (hopefully transparent) system ID in the prefix namespace is recommended. This will avoid any potential naming conflicts between the respective systems.

Examples of system ID = D01:

- YD01*
- /ZYXD01/*

A.4.2 AVOID SUPERFLUOUS IDENTIFIER INFORMATION

Avoid oversized conventions for organisational components or fast-moving information in short object names. Instead use Meta information that can, for example, be defined using the Classification Toolset.[109]

Otherwise, cryptic identifiers are created that are difficult and costly to interpret as this requires extended analysis of the conventions.

Examples:

- Name: /NAME/CL_T1_ERP_BC_SYS_DPC_607
- Meaning: data provider class that provides system information in corporate template T1 on the basis of ECC 6 EHP 7.

Examples of Meta information to avoid in object names:

- [RICEF-ID] / [Change Request ID] / [Ticket ID]
- [Corporate template] / [System type] / [Version]
- [Release status]*

109 *SAP Help Search 'Classification Toolset'*

The use of release status details only makes sense for product development that is actually developed for a range of releases and differs in terms of ABAP syntax or data model. Such utilisation should be the exception rather than the rule.

A.5 FORMS

Type	Adobe Interactive Form Interfaces should stand out from standard forms through the addition of a suffix.
Name formation	[Form name] [Interface abbreviation]
Examples	Y_CAS_SALES_ACTIVITY_IF

A.6 PROTECTION OF NAMING CONVENTIONS IN ABAP WORKBENCH

Naming conventions for repository objects can be managed using on-board means in the Change and Transport System (CTS). CTS mechanisms provide so-called prefix conventions to facilitate definition. Configuration of prefix conventions is implemented using the table maintenance views V_TRESN and CTSRESNAME for reserved customer namespaces. These views enable the specification of naming conventions for repository objects per package.[110]

Maintenance of individual object types per package is extremely time-consuming and involves a high level of change management. Generic maintenance for all objects of a single package only works for reserved customer namespaces and is therefore not relevant for all customers. An additional shortcoming is that the conventions do not allow the overlapping of prefixes for various packages and the elements they contain. For example it is not possible to apply prefix /ZYX/123 for the package /ZYX/SOME_PACKAGE and at the same time the prefix /ZYX/1234 for the package /ZYX/SOME_OTHER_PACKAGE, as the second prefix is a subset of the first. There is no guarantee that all elements in a package should begin with the defined prefix, as objects without a defined V_TRESN prefix can be allocated to the package without problem.

An innovation for contemporary package hierarchies was provided by SAP Note 2297645 shortly before the editorial deadline and is encompassed within the latest SAP_BASIS 700 component service pack available since May 2016. Further details are available in the SCN blog.[111] Although this innovation ensures that elements with the defined prefix may only be stored in a package within the package hierarchy, it still retains the aforementioned weaknesses.

BEST PRACTICE
Due to the restricted functionality, we recommend that the table maintenance views V_TRESN and CTSRESNAME should only be used after careful consideration and preferably not at all.

110 Cf. *Definition of Naming Conventions (SAP Library – Software logistics)*

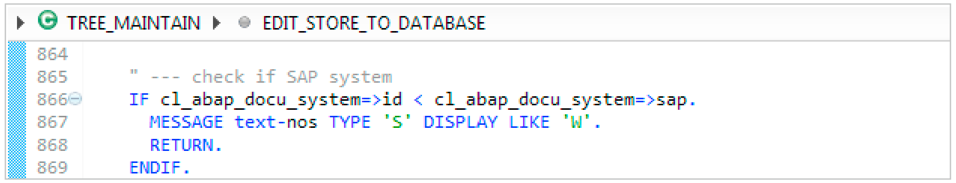
111 SCN blog: *'News about ABAP Package Concept: Naming Conventions for Package Hierarchies'*

APPENDIX B: FURTHER EXAMPLES

B.1 ADDITIONAL ABAP KEYWORD DOCUMENTATION

The individual expansion options for ABAP keyword documentation are not an officially supported feature of the current version. At present, only limited information is available relating to expansion of the program guidelines. The following information is therefore used solely at the user's own risk:

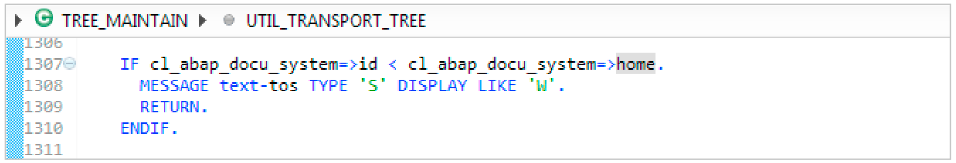
Tools for maintenance of transaction ABAPDOCU are provided in the package SABAP-DOCU. This contains the program ABAP_DOCU_TREE_MAINTAIN, which provides help in relation to adding and editing the individual items in the navigation tree. Upon saving the navigation tree the program runs the function module of the same name ABAP_DOCU_TREE_MAINTAIN; within its local class TREE_MAINTAIN this starts the method EDIT_STORE_TO_DATABASE, which initiates a comparison check via a constant with a central SAP documentation system.



```

  " --- check if SAP system
  IF cl_abap_docu_system=>id < cl_abap_docu_system=>sap.
    MESSAGE text-nos TYPE 'S' DISPLAY LIKE 'W'.
    RETURN.
  ENDIF.
```

At the end of the method, storage of the modified navigation tree is delegated in a transport request to the method UTIL_TRANSPORT_TREE, which initiates renewed validation of the underlying SAP system.



```

  IF cl_abap_docu_system=>id < cl_abap_docu_system=>home.
    MESSAGE text-tos TYPE 'S' DISPLAY LIKE 'W'.
    RETURN.
  ENDIF.

```

After working around the both checks, the changed navigation tree can be saved and the changes transported to other systems.

LEGAL NOTICE

NOTE:

Deutschsprachige SAP® Anwendergruppe e.V.

Altrottstrasse 34 a

69190 Walldorf | Germany
Telephone +49 6227 35809-58

Fax +49 6227 35809-59

Email info@dsag.de

www.dsag.de

FURTHER INFORMATION IS AVAILABLE FROM:
Working group on development, www.dsag.de/ak-development