Affine Guidelines to Make Your BI Implementations Easier

These guidelines represent the most common ideas that come up during QAs and project reviews, and are mostly "easy wins" that can benefit your project with little effort, if considered up front. These are not intended to be comprehensive, nor are they a set of design guidelines, rather they are a collection of useful hints and tips for a more successful BI implementation.

	Guideline	Details and Benefits
Standards	**Use a BW naming convention** for all the main BW objects	Naming conventions provide many benefits in making objects appear in a logical order when configuring, checking, sorting and for easy identification in for e.g. query builder tools. As a minimum the following is suggested, and there are many variations on these themes: **MultiProviders** can be named MSSFFTNN where M is the Module, SS is the Sub Module, FF are free characters, T is InfoProvider Type (M for MultiProvider) and NN are consecutive numbers. Example FARDEM01 for Finance, AR, DE for Germany, MultiProvider 1. **Basic InfoProviders** can be named MSSFFTNN where M is Module, SS is Sub Module, FF are free characters, T is the InfoProvider Type (**C**ube, **O**DS, **V**irtual Cube, **I**nfoSet) and NN represents the year when using logical partitioning by year, or 00 to represent no logical partitioning (so cube contains all years). Example: FARDEC00 for Finance, AR, DE for Germany, Cube contains All Years of data. **InfoObjects** are often best left as all free characters, making them as meaningful as possible, but this depends on your specific environment.
	Use ABAP carefully in Includes, in a modular way and with a naming convention	If you use Includes to hold your ABAP routines then you get version history, and any code changes you make can be transported without transporting and reactivating the associated Update Rules or Transformations. Using ABAP in a modular way means to try to reduce risk in shared exits like ZXRSRU01 by placing each section of variable code in a separate Form, Include, Function Module or Program. The last two options reduce the risk of a single compilation error affecting the whole user exit. Even simple ABAP can appear complicated once it has been "enhanced" with Change Requests a few times. A consistent ABAP naming convention for global/local variables and internal tables is recommended.
Design	**Keep it simple** and don't introduce unnecessary complexity	Over time, BW designs often get more complex as more functionality is added. When starting from scratch, then, the data model should be kept as clean and simple as possible. For example, don't use time dependency if its not needed and don't keep more granularity of data if its not needed. In particular, try to push all data manipulation as far "down" the data flow as possible, ideally to the source systems, so that BW is not forced to do lots of transformations. Each manipulation is a new test script that needs written later, and is a potential point of failure for the future.
	Keep a **Data Flow Diagram** that shows the whole design on one page	It is always useful to have a single Data Flow Diagram that shows a physical representation of the Cubes and ODSs of the whole system. This is useful for on-boarding new team members, for support teams, and for using when examining new design additions.
	Data should be correct at point of entry, not validated and rejected later	A key principle for a successful implementation is to have data correct either before it enters BW, or in a staging/cleansing layer within BW. This has considerable benefits in reducing complexity.
Performance	**Performance tune as you go**, but not so much that it slows development	It is important to consider performance and scalability right from the start of a project. Design and build with this in mind as you go along and you will save much work later on. Take care, however, not to over-engineer as the true bottlenecks won't be known until later on in your performance test phase. Note also that aggregates are great, but have drawbacks in terms of rollup time. They should not be used as the default response to performance problems. Consider logical and database partitioning, pre-loading the OLAP cache and reducing the query result set.
	Use database and logical partitioning on all appropriate InfoCubes	Database partitioning is always worth doing on any InfoCube that you intend to compress. Logical partitioning (say be Year, or Business Unit) is worth doing for InfoCubes you know will get large before archiving.
Transports	**Keep systems aligned** so that Dev and QA and Production are reliably identical	Although BW offers the flexibility to edit many objects directly in each system, take care when doing this. It is standard to create packages and perhaps amend Process Chain start variants in Production, but do not forget the golden rule that all systems should be identical. Without this rule, testing in QA is compromised.
	Use the BW CTS carefully and avoid editing transported objects in target systems	The BW Correction and Transport System has been stable for some time, and editing objects in target systems should be avoided. There are ABAP programs to activate most common objects if necessary (search for ABAPs named RS*ACTIVATE*), or you can use the ABAP RSDG_AFTER_IMPORT_FOR_CORR to just run the process after import step. If you need to create InfoObjects and Process Chains locally, perhaps directly in Production, then set Object Changeability to "Changeable Original" in the Transport Connection to allow new objects to be created, but to preserve the ones sourced from Development.
Reporting	**Use a MultiProvider** above all Basic InfoCubes and ODSs, and only report from the MultiProvider	This gives two benefits. Firstly, if all queries and authorisations are written from the MultiProvider, then it is very easy to "plug" and "unplug" new Cubes underneath the MultiProvider. This has huge flexibility benefits, allowing various sorts of logical partitioning to be done in future. Secondly, this approach allows two sets of dimension definitions to be defined. On the Basic InfoCubes the dimensions can be structured to suit performance, and on the MultiProvider they can be structured to suit end users creating queries.
	Allow users to create queries in production, and **often used user created queries should become standard**	BW has powerful end user tools and super users should be allowed to create queries in production. One approach is to allow users to create, but not share queries. Then, if they like the query, they request the support team that it is created "properly" in Development and transported. The benefits here are the query can be used in QA for performance testing, and regression testing after upgrades.

Affine are a specialist Performance Management Consultancy based in Surrey, England. Affine provide consultancy and resources for the evaluation, planning and delivery of projects that have SAP Business Intelligence at their core. Contact us at info@affine.co.uk.