HP ALM Best Practices Series

For ALM Practitioners

Versioning and Baselining Best Practices

Document Release Date: December 2012
Software Release Date: December 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered trademark of Oracle and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
  - The number before the period identifies the major release number.
  - The first number after the period identifies the minor release number.
  - The second number after the period represents the minor-minor release number.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition, visit the following URL:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the New users - please register link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

You can visit the HP Software support web site at:

www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require an active support contract. To find more information about support access levels, go to the following URL:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to the following URL:

http://h20229.www2.hp.com/passport-registration.html

Contents

Welcome To This Guide

Welcome to the HP Versioning and Baselining Best Practices guide.

This guide provides concepts, guidelines, and practical examples for the best implementation of version control and for the creation and maintenance of baselines in various organizations.

This guide applies to HP ALM 11.00 and later.

About Versioning and Baselining

As organizations look to reduce costs across a wide range of operational disciplines, IT comes under more and more pressure. The impact of project slippage, poor quality delivery and inadequate solutions is not tolerated anymore. This may prove a pain-point for many organizations who have traditionally failed in these critical areas. Application software plays a dominant role in today's business, regardless of the vertical market or core competency. Every organization must be able to guarantee high quality, working software to properly position and deliver its products to the market. Now, more than ever, software is a critical component for winning the competition.

The HP ALM suite successfully serves various organizations in their quest to deliver quality software that drives the business. HP's unified management and automation capabilities offer customers modern solutions for modern delivery. The result is improved predictability, repeatability, quality, and change readiness across the application lifecycle.

One of the characteristics that should be addressed when supporting application life cycle, is the ability to keep multiple versions of the main entities participating in Software Development Life Cycle (SDLC), such as requirements and test cases. Since many industries are heavily regulated and must pass a variety of compliance-based tests such as Sarbanes-Oxley and HIPAA, every step in the process must be audited and presented to the authorities. Even if your project is not regulated, it is important to keep previous versions for mission-critical assets. Enabling the version control feature of HP ALM helps achieve this goal.

Coupled with baselining, the ability to take a snapshot of current project activities at important milestones, versioning enables the tracking of mission critical business projects, the ability to compare saved entities with their current state, the retrieval of older versions, and more.

The purpose of this document is to assist HP ALM customers to assess their current testing practices and successfully build and maintain testing methodology using advanced features provided by HP ALM. All aspects of this process have been researched using best practice data and expertise from various sources, including HP's operating system administrators, HP's professional services organization, technical documentation, books from industry experts and the personal experience of many customer testing organizations. These guidelines help reduce the initial creation time and achieve maximum value in operating HP ALM.

Audience

This guide is intended for:

- Business Analysts
- Testing CoE Managers
- Testing Automation Engineers
- Development Managers
- HP ALM Administrators

Prerequisites

To use this book, you should have a good acquaintance with major phases of Software Development Life Cycle (SDLC). You should also be familiar with the business processes in actual IT organizations.

Operational knowledge and administrative privileges of HP ALM are essential in implementing these best practices.

Note: all features discussed in this document are available only in HP Quality Center Enterprise Edition and Application Lifecycle Management Edition. In HP Quality Center Starter Edition these features are limited.

Structure

This guide is organized as follows:

- Introduction to Versioning and Baselining
- Making Versioning Work
- **Error! Reference source not found.**
- **Error! Reference source not found.**
- Conclusions

Feedback

If you have questions, comments, or valuable best practice information you want to share, send a message to the following email address:

alm_cust_feedback@hp.com

1 Introduction to Versioning and Baselining

Importance of Keeping Track

For quite some time, it has been an IT industry motto to be “aligned with the business”. This notion implies that IT serves an important but secondary role in the life of the company and therefore coordinating its activities with the broader business agenda is sufficient to make operations go smoothly. Lately, however, there are voices saying we need to take this paradigm one step up to become “IT is the business”. This is mainly due to the fact that many companies work in the information industry, where ideas, technology and data are the products of trade. Another reason is that the data accumulated by the firms has immense business value, helps to better serve customers, predict their needs, and communicate with them, and produces added value.

Having piles of data presents challenges to the companies. Not only can it affect performance, day-to-day operation, and backup and restore procedures, but it also tests their abilities to keep track of information.

Many organizations, especially those in the finance, healthcare, and government sectors, are required to comply with specific government regulations, such as HIPAA, Sarbanes-Oxley, and US Section 508. It is therefore obligatory for their IT divisions to demonstrate adherence to the highest level of regulatory compliance. The following steps are usually required by these regulations:

- Sign-off of certain processes and documents after passing necessary reviews and approvals. In a regulated environment, businesses must provide proof and reasoning when they make decisions that may potentially affect compliance with the law and standards.
- Generation of reports in predefined formats that provide sufficient proof that the organization meets the required level of compliance with specific government or industry regulations and requirements.
- Auditing of changes that impact regulatory requirements throughout the application lifecycle to show application consistency.

In many cases, companies want to follow the changes being made by various persons working on the same entity. They want to instill revision control, especially in complex and mission-critical projects.

The growing push for automated testing puts pressure on the testing resources. With agile proliferation and its quest for continuous delivery, it is not an easy task to navigate a web of test components, some of them shared, that are in use in multiple projects. We have to be able to monitor modifications made in the automated tools to react to environment changes, user mistakes and so on.

HP ALM Version Control answers these challenges and enables you to keep track of changes made to entities in your project, including requirements, tests, test assets, and business components. You can "check out" an entity to make changes, and "check in" the entity to store the changes, making a new version of the entity available to other users. You can view and compare previous versions of an entity, or "check out" a previous version.

When there is a need to oversee changes over time, HP ALM Baselining provides the ability to take snapshots of a library, a set of entities in the project and the relationships between them, at a specific point in time and compare these snapshots at all stages of the application development lifecycle. You can use a baseline to mark significant milestones in the application development lifecycle, such as the signoff of a functional specification between the business and IT. Baselines enable you to monitor changes made to your project over time by creating detailed comparisons of any two baselines in a library. You can also compare a baseline's entities to their current state in the library. For example, if you create a baseline at the start of a new release and changes are made to the requirements over time, you can compare the requirements in the initial baseline to the current requirements and determine whether the project is proceeding as planned.

Versioning and Baselining can be implemented independently according to the business needs of the company.

When to Implement Versioning

Versioning enables organizations to maintain control over changing business assets by storing a history of entities that can be viewed, compared, and restored. In a version control enabled project, you can create and manage HP ALM entities while maintaining previous versions of these entities. This includes requirements, tests, test resources, business process models, and business components.

However, implementing versioning requires a different methodology to be applied in your day-to-day operations, as it adds some steps to both ALM users and administrators.

So when is it worthwhile to implement versioning? **HP recommends** checking numerous aspects that can affect the decision to implement versioning.

One of the first checks must be the mission criticality of the project to the business. To check if your project is *mission critical:*

- Verify whether this project is expected to be up and running 24x7.
- Understand the nature of the applications maintained in the project. Frameworks and business applications like billing and ERP are examples of core projects.
- Calculate the cost of the application or service for which the project was created. High cost, high revenue, and high visibility applications demand thorough control over the assets.

If any of the classifications are true then your project is considered mission critical and is **recommended** to be versioned.

Another aspect weighing in the decision over versioning of the project is its *longevity*. There are sometimes long or virtually endless development projects containing various product versions, patches, private fixes and so on. Due to the accumulated number of assets, such projects can be **recommended** as candidates for versioning to allow better traceability of the changes.

Another input in the decision making process is the *number of users* in the project. If there are many project users accessing and changing the same entity, this entity can become corrupted or overwritten with wrong data. Implementing versioning brings some clear benefits to the process:

- Users can access their private version and make changes without affecting others.
- Users can detect problems early in the cycle by using the compare function.
- Users can revert to the older version in case of the failure of the new version.

Such projects should be **recommended** as candidates for versioning.

Similar logic is applied if the project in question has a *large number of entities* and these entities have *complex structures*, such as multiple tree levels in requirements or tests.

Last but not least is the impact *automated testing* can make on the project. Automated tests run unattended and as such do not require user

intervention. However, this also means that no user judgment is applied. Therefore all test changes influence every user. By implementing versioning, the test change is kept private and does not affect other users. The modified test sets are checked and tuned until completed and only then released to general usage. Furthermore, if a problem occurs and a test fails, it is possible to trace when the change happened, who initiated it, and what was the nature of the change in order to fix the problem.

To summarize, **HP recommends** implementing versioning when your project has one or more of the following characteristics:

- Mission critical
- Project longevity
- Number of users
- Number of entities
- Complexity of the entities
- Level of test automation

When to Implement Baselining

Baselines enable you to keep track of changes made to your project over time. A baseline is a snapshot of your library at a specific point in time. A library represents a set of entities in a project and the relationships between them, such as coverage and requirements traceability. The entities in a library can include requirements, tests, test assets, and business components. A baseline can also include any related entities outside of the library that the tests in the library need in order to run, such as called tests and test resources.

So when is it worthwhile to implement baselining? If one of the scenarios below describes the situation in your organization, **HP recommends** implementing the baseline feature.

Baselines are indispensable when you prepare the content of the *new version* of the application. As the new version is usually a continuation of the existing application release, business analysts compile a list of requirements that trigger the development and testing process. These requirements are normally reviewed and approved by various stakeholders. After approval, the content of the version can be signed off and a baseline created to hold this important information for comparison with current application activities at different points of time.

Many times you need to *monitor the changes* being done to the application entities during the lifecycle. For example, if during the application development the product manager discovers that some features are being implemented differently from the initial intent, the product manager can compare the current requirements with their agreed upon content in the baseline taken at the beginning of the release.

Another reason to use baselines is the need to measure the *impact of change*. If some requirements have been changed as a result of the review process, it usually means that underlying tests must be updated to reflect these changes. To do that, a testing manager can compare the current requirements with the requirements in the baseline created at the beginning of the release. When changes that affect tests are identified, you can update these tests to reflect the changes.

One of the most important drivers for using baselines is the functionality of *pinning a test set to a baseline* to associate a test set with the versions of the tests stored in the baseline you select. When you run the tests in a pinned test set, the versions of the tests stored in the specified baseline are run. This allows developing tests in one cycle and running them in the next, while working on those same tests in parallel without fear of changing the test results. To avoid confusion when getting status reports from different runs, you should run pinned tests only in the context of a test cycle. By running your execution status and coverage reports in the context of a test cycle as well, you gain a clear picture of your status even when using pinned test sets. Note that in each run cycle you should run test sets pinned to a single baseline.

Baselines are also a foundation of the entities sharing between projects. There are quite a few possible models of sharing based on the project topology and development process. See the *HP ALM Entities Sharing Best Practices* guide for more details on using baselines in sharing.

2 Making Versioning Work

This chapter describes the basic steps for the best implementation of Versioning.

Sometimes customers confuse Versioning and Auditing, not fully understanding the differences between these two approaches. Versioning is a more sophisticated means of auditing. The audit mechanism provides detailed “before and after” data for fields that have changed and that have been selected for audit. The Versioning mechanism enables a much larger set of functionality, including checking out entities to a private version, checking in entities, viewing historic versions of entities, rolling back to previous versions of entities, comparing two historic versions of entities, and so on. Versioning also typically covers more fields than Auditing.

How Versioning Works

Using version control, you can create and manage multiple versions of HP ALM entities, including those associated with HP Business Process Testing (BPT) and HP QuickTest Professional (QTP), encompassing requirements, tests, test assets, and business components.

For quality control, consistency, compliance and other purposes, many organizations enforce version control when working with testing assets. In many cases, BPT and QTP tests use reusable components shared by many different test sets. If a problem occurs during the automatic test execution, it is difficult to pinpoint the cause. If version control is in place, it is possible to revert to the last stable version of the test and then analyze the changes using the version compare feature. This is the best and fastest way to identify points of failure without breaking the testing process.

Version Control Process

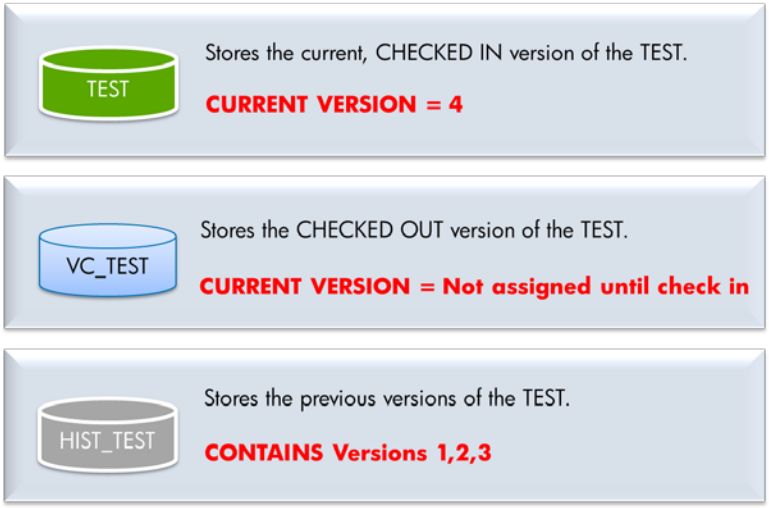
In a version-controlled project, if the user wants to make changes to an entity, the user must first “check out” that entity. While “checked-out”, the entity is exclusively locked. This prevents other users from editing it at the same time and overwriting the changes. The entity can be “checked-out” only

by a single user at a time. From this moment, all modifications are visible only to that user.

When the user has finished making changes, the user “checks in” the entity and a new version of the entity is now available to all users. This check in removes the exclusive lock on the entity. Generally speaking, the lock is removed in two cases:

- The user “checks in” the changes.
- The user undoes the changes, rolling back all the changes the user has made.

The following example illustrates a technical perspective of the three versions of the Test entity managed by version control mechanism. The Test entity (TEST table) itself stores the current “checked in” version of the test case. The VC_TEST table contains a row for every test that is “checked out” and the HIST_TEST table has a row for every previous version of every test.



Preventing Inconsistency

In a regular flow a user can be asked to free a certain entity either by “checking in” the entity or by cancelling the “check out”, thereby undoing the changes. But what happens when the user who “checked out” the entity is unavailable for a long period of time (for example, on vacation or leave)? The project administrator can “undo the check out”, freeing the entity for editing by other users. In doing so, the pending changes are cancelled.

Some customers want to perform a forced “check in” to commit changes done by the person who is unavailable. For example, there might be testing scripts “checked out” by several contractors who do not work for the company anymore. The testing manager wants to retain the changes made by the departed contractors. If the checkout is cancelled, the changes are lost.

There must be a clear understanding that performing a check in by a user who is different from the “check out” user, is not the same as undoing a “check out”. Undoing a “check out” just deletes the changes already made by a certain user and rolls back to the last version found on the server side. “Check in” creates a new version containing all changes made by a user that are found both on the server side (in the database) and on the user’s local machine. Collecting the client side data in this scenario is impossible. This is why HP, like other version control tools, does not allow you to perform “check in” for other users.

When to Check In

The issue of timing in the “check in” process has a profound effect on the entities lifecycle. If version control is in place, the user is prompted to “check out” every time the user tries to modify the entity. Ideally, “check in” of the modified entity should be performed at the end of the logical change cycle.

HP recommends following these guidelines to perform “check in”:

- If the change to the entity is minor and no impact on other entities is expected, then it is recommended to “check in” the entity as quickly as possible to free it for other users.
- It is always a good practice to break large changes into smaller pieces. Once your modification is finished, “check in” to minimize the disruptions to the team’s work.
- In complex projects with interconnected entities, the project administrator can define a time interval during which the changes must be “checked in”. If this is the case, use the technique above to break large changes into smaller pieces to fit the “check in” windows.
- Regularly review the list of “checked out” entities, such as Requirements, Test Sets, and Tests, in the module you are working with. Think of this list as the task list that has to be completed to let other people see the changes.

- Frequently provide comments when “checking in” entities, especially in regulated environments. The comments help to understand the possible cause of test failures or other problems and offer a means of journaling that can be presented to the compliance bodies.
- Prior to executing any maintenance action on the project with version control, consider performing “check in” on all “checked out” entities. If some users, who “checked out” assets are not available any more, then use project administrator’s power to undo their changes (see the *Preventing Inconsistency* section). For example, entities that were “checked-out” by any user remain “checked-out” by that user, even if that user is not an authorized user for the newly restored or imported project.
- Before you run a test set, make sure all its tests were “checked in”. By not doing this, you risk working with a version of the test that is not completely finished. The results of the test run can be inaccurate or un-reproducible.
- Sometimes there is a need for a massive update of the entity. This can be done by either selecting multiple entities in the user interface or performing a bulk “find & replace” operation. If the field is marked as a “non-versioned field” in the customization module, you can change its value in one shot instead of executing “check in” and “check out” for every entity. See details in the *Non-versioned Fields* section.
- If you are about to create a baseline, remember to “check in” all entities. Otherwise, all the changes you made in any “checked out” entities are not included in the baseline.

Entities with Version Control

The majority of modules in HP ALM participate in versioning if it is enabled. The list of the modules includes:

- Requirements
- Business Models
- Test Plan
- Test Resources
- Business Components

To maintain usability and data integrity, HP ALM stores previous versions of an entity without most data related to the relationships between entities. The following data is not stored for previous versions:

- Requirements and tests coverage
- Requirements traceability
- Defect linkage

In addition, risk data is also not stored for previous versions of an entity.

For example, when “checking out” an old version of a test, its coverage data remains the same as the current version, and only its content is rolled back (the link between entities is not stored in the version data, only in the entity itself).

In addition to that, there are some fields that are not stored under version control:

- Requirements
  - Reviewed
  - Direct Cover Status
  - Target Release
  - Target Cycle
  - All Risk Based Quality Management (RBQM) fields
- Tests
  - Execution Status

Non-versioned Fields

While the system fields above are purposely absent from the version control process, there are cases when marking some additional fields as non-versioned proves to be beneficial. For example, a project administrator may want to assign a category, such as Web 2.0 or DB, to a large portion of the tests. This user-defined field does not need to participate in version control as it holds no business criticality and its value change does not need to be recorded. Another reason for declaring a field as non-versioned in a version controlled entity is to ease the overhead imposed on the user who is required to perform the “check out” and ”check in” procedures even if the field itself does not affect the main business processes. This allows for refined usability as well as the ability to perform massive updates.

When working with non-versioned fields, their special behavior should be taken into the consideration:

- When undoing a checkout, any changes you made to a non-versioned field while the entity was “checked out” are lost and only the new value remains.
- When you “check out” a previous version, the value of a non-versioned field is the value in the currently “checked in” version.
- When you update a single entity’s non-versioned field, the regular “check out” dialog still appears, even though no version of the entity is created. When you update a set of entities, the update of the non-versioned field does not trigger the “check out” dialog appearance.
- During comparison with the previous version, non-versioned fields are displayed with the value Non-versioned Field, indicating that they do not participate in the comparison.

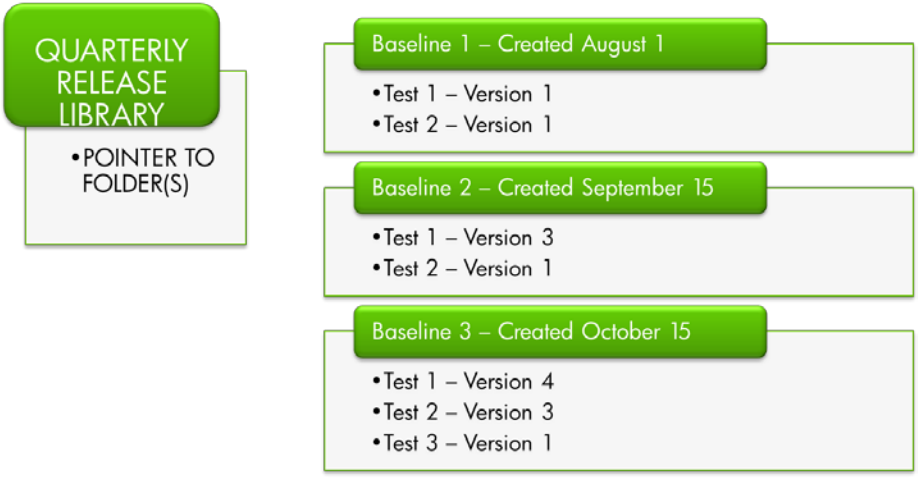
3 Making Baselining Work

This chapter describes the best practices for capturing baselines in HP ALM.

Baseline is a physical copy of the entities in a library at a moment in time. Once you create a baseline you cannot change the contents, like taking a picture and printing it. Viewing and comparing baseline histories enables you to track changes made to individual entities in your library over time. As development continues, you can view and compare all states of an entity that are stored in a baseline.

As opposed to versioning, baselines capture both the data for the actual entities, such as requirements, tests, test assets, and business components, and the links between those entities, such as coverage, traceability, and so on. Baselines can be captured in projects **not** using Version Control, because baselines and all associated features are available even if the version control capabilities are not turned on for the project. However, "checked-out" versions of entities are not captured as part of the baseline - the baseline captures the last "checked-in" version of each entity.

The example below shows the progress of the release accompanied by baselines. The baseline captured on August, 1st stores the latest "checked in" version of the entity, which in this example is version 1 for both Test 1 and Test 2. On September, 15th, when another baseline is captured, Test 1 is now up to version 3, which is the version that will be stored with Baseline 2. The last baseline again shows both Test 1 and Test 2 in their even more advanced "checked in" versions as well as a new Test 3.



HP **recommends** synchronizing the baseline creation time with a major step in the development process, such as the end of a cycle, iteration, or release. Ideally, the new baseline should include the date of the change in the title and some relevant comment. If a source code control system is in place, even if not automatically connected to HP ALM, we **recommend** that every time a baseline is created a notation of the source code tool baseline label or numbering pattern is written in the baseline comments. For example, the baseline comments in ALM can contain "Subversion Release 4.3 beta" or at least "Checkpoint 1.7". This allows for an easy correlation of the two related entities. Also consider using a naming convention or custom attributes for baselines to enable the correct identification of their type, such as after synchronization, after reconciliation, before synchronization, and so on. It is possible to enforce naming conventions by using workflow code.

HP also **recommends** examining the granularity of the libraries before creating them. As libraries are only collections of pointers to actual entities, they can result in actual baselines ranging from small to large.

Smaller libraries give greater flexibility in assembling multiple combinations of assets for sharing. On the other hand, too many libraries can cause management overhead and confusion. When creating libraries, you can use filters to select only the relevant informational resources instead of selecting generic roots. This model gives the user more control over library content, and helps to define libraries that are not based solely on the hierarchical structure of the project.

Another approach is to define “initial roots” for the libraries and let HP ALM automatically gather all the relevant entities based on predefined links (for example, coverage and requirement traceability). Be aware that this option can result in a large population of linked entities even if you selected a small number of records.

To avoid performance problems, the number of entities **recommended** for a single library is calculated based on two site configuration parameters:

- REQUIREMENTS_LIBRARY_FUSE, with default value of 3500
  - The maximum number of requirements in a library must not exceed this parameter value.
- LIBRARY_FUSE, with default value of 2500
  - The maximum number of tests in a library must not exceed this parameter value.
  - There is a ratio of 1:4 between tests and resources. Therefore, the maximum number of resources must not exceed one quarter of LIBRARY_FUSE (625 entities).

- The ratio of 1:4 is also true for business components. Therefore, the maximum number of business components must not exceed one quarter of LIBRARY_FUSE (625 entities).

HP strongly **recommends** limiting the number of entities in a library to the sum of the various records according to the above rules.

4 Estimating Storage

In addition to their functional effects, both Versioning and Baselining have an impact on storage allocation. As an HP ALM project is composed of a database schema and a file system repository, this impact is seen on both sides in different ways. As HP ALM projects tend to grow over time, the site administrator must take into consideration the additional storage requirements of these two features to ensure adequate storage space.

Understanding Data Growth

The storage consumption of the Versioning and Baselines features is attributed to their method of operation:

- Each new version of an entity creates a copy of that entity in the HP ALM database and file repository.
- Each baseline creates a copy of all entities in the library.

Note: The "Enable versioning" action by itself has **no** effect on storage consumption. The data is duplicated **only** when an entity is changed.

However, enabling versioning and/or baselining does **not duplicate all** HP ALM data over and over again:

— Not all modules are covered by these features, but rather specific ones such as Requirements, Tests, QTP Resources and BPT Components (see more details in the *Entities with Version Control* section).

— All other HP ALM entities do not support these features by design. This means that only the storage space used by the covered entities is expected to grow. The majority of storage used by a Quality Center project is used by other entities (defects, test runs, and so on), and it is not expected to change.

— Only a relatively small percentage of the entities are duplicated, and a duplication rate relates to usage patterns.

— Since introduction of SmartRepository in HP ALM 11, the data stored in the repository is no longer duplicated when a new version is created. The actual attachment, script or resource file location is registered in the database but no operating system file copy is

executed. Hence the file is physically kept once unless it has been changed. See an expanded explanation in the *How SmartRepository Decreases the Storage* section.

While these features' impact on storage is less significant than previously thought, a larger database and repository affect the overall application performance. Therefore the storage issues need to be evaluated and a remedy provided for the problem.

Factors Affecting Growth

What are the parameters to be taken into account when estimating the storage? Following is the list of factors affecting storage allocation:

Version Control

- **Current Storage Size Used By Versioned Entities**

  The percentage of the database and file repository used by versioned entities. The Test entity is the most substantial out of all versioned entities. The percentage of storage it uses out of the total project size determines the impact of Versioning and Baselines growth.

  Note: Automatic tests, such as HP QTP Tests and HP Performance Center/LoadRunner Tests usually use a lot of the file repository for storing their results. However, baselines and version control **do not copy** those test results. Only the test scripts are logically duplicated but this has no effect on SmartRepository and only adds one record per test in the database. Sites with custom test types can also configure Baselines and Versioning not to copy their result files. See the *HP ALM Custom Test Type* guide for further details.

  To calculate the database portion of this value, use database specific queries on the data dictionary. File repository size can be obtained by connecting to SmartRepository via an FTP client and using its system commands.

- **Normal Growth Rate**

  HP ALM projects tend to grow over time, as new entities are constantly being added. When measuring the impact of Versioning and Baselines, it is important to consider this parameter.

  As we are mostly interested in Test entity growth, issue a database specific query against the TS_CREATION_DATE column to get this measurement.

- **Average Change Rate**

  Sometimes also called Additional Growth Rate in a Versioned Project, its meaning can be explained as the average number of times an entity is modified by users per year. This is the main parameter affecting Versioning storage growth, as in versioned projects the user must "check out" the entity in order to edit it. When editing is complete, the changes are committed by "checking in" the entity, thus creating a new version.

  To get this measurement, issue a database specific query against the AUDIT_LOG table.

The formula that calculates the impact of versioning is:

Versioning impact = (*the current storage size used by version-controlled entities) * (average change rate)*

Baseline

- **Live Data Percent**

  Known also as Baseline Content, this parameter tells the percentage of entities actually accessed in the last year. This is a good estimate for what percentage of entities are included in libraries, as they normally consist of only live, recently accessed data. The customer data shows that a constant number of entities is used every year, while older entities are not used at all as time passes by. This data would later be included in baselines. Therefore, it is also the main parameter affecting baseline storage growth.

  Issue a database specific query against the VER_STAMP column to get the value of this metric.

- **Baseline Creation Frequency Policy**

  The organization's policy on how often the HP ALM project administrator should create a baseline of the libraries definitely affects overall storage estimates of the entire project. See the *baseline timing* recommendations discussed in the previous chapter.

The formula that calculates the impact of baselining is:

Baselining impact = (*the current storage size used by version-controlled entities) * (live data percent) * (baseline creation frequency as times/year)*

Examples

Let's evaluate the impact of the parameters above on the real life projects in HP Software R&D lab. Each example calculates storage impact separately as each of the features can be used independently. To get the overall impact when both Versioning and Baselining are in place, simply combine the results below.

Version Control

Let's take a typical HP ALM project that uses 20 GB of storage size (database and file repository):

- It counts 30,000 tests, with an average storage size of 20 KB per test in the HP ALM database and file repository. In this case, the current storage size used by test entities is 30,000*20KB = **0.6GB**
- By executing SQL queries, it has been determined that each year 5,000 additional new tests are created, bringing the project's normal growth rate to 5,000 * 20KB = **0.1GB**
- When the project administrator enables version control, it initially has no impact on storage consumption, as was noted above.
- By analyzing the current project, it was found that the average change rate stands at 2. That is, each test is changed twice per year on average.

To plan the storage the following calculations must be performed:

- According to the *versioning growth formula*, this HP ALM project will grow each year an additional 30,000 * 20KB * 2~= **1.2GB**
- This growth represents 6% of the current 20GB, on top of its normal growth of 0.1GB (0.5% of 20GB).

Baseline Capture

Let's assume we want to create a baseline in the same HP ALM project:

- By executing SQL queries, it has been determined that the live data percent is **50%.**
- If the HP ALM project administrator creates a single library that contains all the live data, it will cover 50% of the tests.

- It has already been stated that the project counts 30,000 tests, with an average storage size of 20 KB per test in the HP ALM database and file repository. In this case, the current storage size used by test entities is 30,000*20KB = **0.6GB**
- The storage used by tests in the baseline is 50% of the above amount, or **0.3GB**
- The baseline creation frequency policy is to capture a baseline for each library, once a month.

To plan the storage the following calculations should be performed:

- According to the *baselining growth formula*, this HP ALM project will grow each year an additional 30,000 * 20KB * 50% * 12 = **3.6GB**
- This growth represents 18% of the current 20GB, on top of its normal growth of 0.1GB (0.5% of 20GB).

How SmartRepository Decreases the Storage

HP ALM version 11 introduced an exciting feature, SmartRepository, which has a profound effect on the way Versioning and Baselining use storage, among other positive results.

While the database schema holds most of the information of the project, the repository holds different types of files such as attachments, automated test results, workflow scripts and more. Versioning actions of “check out” and “check in” on entities that use repository assets, such as attachments and automated scripts and results, add their heavy footprint. Baselining takes a lot of space when simply creating a baseline with a considerable number of entities.

Until HP ALM 11, when Versioning or Baselining were implemented, entities were copied ‘as is’, including their repository assets, which caused massive duplications. This was one of the reasons why the repository was hard to maintain and back up. Its increased size degraded the performance of many day-to-day operations in the HP ALM user interface.

SmartRepository is based on an abstraction layer between the logical file system structure and the physical file system structure that reflects the actual structure of the files on the disk. This approach allows creating a balanced tree on the disk as well as saving only a single copy of each file. This virtually eliminates the endless duplicates on the physical storage device as every file is only saved once but has multiple references in the logical table.

With SmartRepository in place, the physical file system shrinks, both in the number of nodes as well as in volume on the disk. The reduction ratio can vary dramatically, and best results are expected on projects with large numbers of nodes. Also, the database is expected to grow slightly because it holds occurrences of each physical file and its corresponding entry in the logical file system.

As this is the default HP ALM behavior since version 11, customers report significant storage reduction, especially when Versioning and Baselining are in use. It is safe to estimate that with SmartRepository, the impact of Versioning and Baselining on the file system is decreased at least **40%** on average. Larger file systems experience even more meaningful gains.

Useful Recommendations

While the impact of Versioning and Baselining on storage is difficult to calculate without proper testing for a specific site, HP **recommends** the following guidelines:

- Before activating Versioning and Baselines features in production, test common usage scenarios in a staging environment on typical HP ALM projects with simulated user activity.

  Try to emulate daily behavior as similar as possible to the real world in order to assess the capacity growth needs. Consider using HP automated testing tools such as QTP and LoadRunner to create such an environment.

- Perform a gradual implementation of Versioning and Baselines features, initially using a few projects a month, and monitor the change in storage requirements.

- Define a clear policy for managing libraries and baselines that best serves the user needs without overloading the system.

- Instantiate a purging policy to delete baselines that are no longer needed.

  However, HP ALM provides limited purging abilities. If the project administrator decides that data in a certain baseline is obsolete and no longer needed, the baseline can be deleted. Deleting the baseline releases all the storage space it uses, both in the database and in the file repository. Note that this action is irreversible, and must be used with caution.

- Separate file system from other parts of HP ALM

  SmartRepository shows some amazing results in storage reduction but is also I/O intensive. If all parts of a typical HP ALM installation are on the same machine, its I/O becomes a bottleneck. This is why it is strongly recommended to designate a separate file system on another server or network storage device. Best results can be achieved by having all machines on the same network, with wide bandwidth and low latency.

5 Conclusions

The purpose of the IT organization is to enable the business to reach its strategic objectives. This is accomplished in a variety of ways, such as providing technical functionality, streamlining processes, and supporting new requests or initiatives, all while ensuring that what is delivered works. This sounds deceptively simple, but in actuality is quite difficult considering how complex the IT environment has become. Many of today's IT organizations are struggling with expanding infrastructures, sophisticated emerging technology, distributed computing, expanding supply chains, increasing regulatory pressures, and mergers and acquisitions.

HP ALM provides answers to some of the challenges through a business centric and quality driven suite of tools that focuses on aligning the business and IT across the lifecycle. Version control and baselining, described in detail in this document, enable you to manage multiple versions of entities and creates an audit trail of change history, allowing distributed teams to collaborate on joint development of requirements and tests without overriding each other's changes. Baselining protects data by enabling the rollback of versioned entities to specific points in the application lifecycle.

We believe that the best practices listed in this document help in the adoption of HP ALM Versioning and Baselining in your organization.