Test Data Management in Practice

Problems, Concepts, and the Swisscom Test Data Organizer

Do you have issues with your legal and compliance department because test environments contain sensitive data outsourcing partners must not see? Do your testers have idle time because test data are missing or test environments have inconsistent data? If any of these challenges applies to your situation, the concepts of this paper can help. The first concept is database-application-aware test cases. They enforce that test cases for business applications provide all information needed for repeatable execution. Second, a type concept eases the test case maintenance and preparation of test data for a test start without delays. Third, a test data catalogue lists database objects for the various types. Finally, architectural patterns describe various ways to set up test environments with production and/or synthetic test data. This paper focuses on integrating these concepts into the daily test process. This includes the tool aspect, which we illustrate with our Swisscom Test Data Organizer.

Motivation

Over the last decades, IT systems have become increasingly complex. The same holds true for testing them. Today, testers need an in-depth understanding of IT architectures and testing methodologies. A sample mobile app illustrates the phenomenon. The app enables bank customers to do intra-day trading in equities. The app runs on various smartphones. It sends trades to the bank's core banking system, which forwards the trades to a stock exchange. End-to-end tests for such architectures are challenging. First, testing must consider various mobile platforms and smartphone types. Second, many interfaces must be tested. Third and often overlooked, testers need consistent test data on the smartphone, the core-banking system, and the trading server. How to manage such test data efficiently is the focus of this paper.

Test data management covers all concepts and tools for ensuring that the data in the databases of a system-under-test fulfill the preconditions of test cases. Is there a test user in the core-banking system? Is the tester allowed to trade certain financial instruments? Is the account balance of his account sufficient? Getting the precondition for a test case right can be more challenging than its execution.

This paper provides answers for the following questions:

- Why and when does test data management matter?
- What are the benefits?
- What are the main test data management concepts?
- How can the concepts improve processes and make the daily work more efficient?

The answers are based on three pillars. Pillar one is our conceptual work on testing database applications throughout their life cycle [1][2][3][4]. Pillar two is a vendor's perspective on developing and testing database applications [5]. Pillar three is made of our insights from various consulting projects [6]. The three pillars together allow us to formulate today's best practices in test data management.
The structure of the paper is as follows: after a short discussion of related work (page 33), the paper explains aims and approaches of test data management projects (page 34f). This is followed by the various concepts for test data management (page 35ff) and the system architecture of Swisscom's Test Data Organizer (page 39f). After discussing the test data management related tasks and roles in the testing process (page 40ff), the paper concludes with a short summary (page 42f).

Related Work

Test data management and testing databases are highly relevant for many companies. However, it is only a niche topic in research and on conferences. Thus, only limited related work exists. The related work falls mainly into three groups:

1. **Testing in general.** Much literature exists on how to test. Perry [7] is a profound example. His book spans from the various testing stages or process aspects to testing data warehousing and web applications. Other examples are books preparing for certifications such as from ITSQB [8]. They introduce a common terminology, discuss testing techniques, and describe a general process

for testing. Test data management or even testing database applications is, if it exists at all, a side aspect.

2. **Papers on test automation of database applications.** These focus (mostly) on unit or unit integration tests and describe test frameworks. Examples are the AGENDA prototype (Deng, et al. [9]) and the work of Dai and Chen [10]. More theoretical work comes from Willmor and Embury [11]. There are also commercial tools for testing business logic in the database, such as the Quest Code Tester for Oracle [12]. Obviously, testing Oracle PL/SQL code demands dealing with (test) data in the database.

3. **Provisioning techniques for test data.** A straightforward concept for populating a database is analyzing its schema. A random generator then populates the tables with suitable data (Houkjær et al. [13]). This might help for unit tests or load and performance tests for non-complex data constellations. Binning et al. [14] suggest a more sophisticated approach. They create a database state based on one or more constraints respectively queries. Suárez-Cabal and Tuya provide a concept helpful for data warehouse tests. It remove rows from large data sets that do not contribute to test coverage [15]. Finally, there are techniques to anonymize (test) data [16][17].

All techniques (besides anonymization) are especially helpful in early test stages (unit tests, unit integration tests), which is common for academic papers. This paper complements the existing work. It combines industry experience and organizational success factors (e.g., processes) while focusing on late test stages.

Writing a Business Case: Aims & Approaches

Managers invest in test data management projects only if there is a business case. The business case defines, first, the aim of the project. This is the need of the business the project wants to address. Second, a business case sketches the technical approach used to achieve the aim. Finally, the business case provides a high level project plan and cost estimations. Costs and project plans are project-specific, while the aims and approaches are generic. This section explains the important aims and approaches.

A test data management business case aims to optimize the testing or/and to improve data privacy. Business cases built around **data privacy** do not argue with cost savings. They focus on regulation, reputation, and trade secrets. *Regulation* means that e.g. laws (e.g., data protection acts, bank secrecy) demand the protection of certain data. Noncompliance can result in law suits and interventions of regulatory bodies. *Reputation* focuses on how customers, employees, society, etc., perceive the company. If, for example, customer data become public and the media pick it up, current and potential customers question the reliability of the company. This happened with Lufthansa when frequent flyer data of German politicians found their way to the press [18]. Finally, trade secrets can be a reason to invest into test data management, too. Examples are customer lists or production process details. If competitors get copies of such data, it can ruin the company.

Today, IT departments are aware that production databases store sensitive data, so access control mechanisms are in place. Business users see only data they need for their work. In contrast, IT departments often ignore the risk of test servers. Many contain production or production near data. The data are copied periodically or on request from production to the test servers. Such data eases (or might even be a prerequisite for) system tests, system integration tests, and user acceptance tests. As a result, many developers and testers can potentially access production data on test servers. They might even have access rights and tools to extract large data sets (e.g., by running an SQL-queries for all customers). This is a high risk for data loss. An aim of a test data management project can be to mitigate this risk.

Optimizing the testing by ensuring repeatable tests, relevant test execution, and efficient data provisioning is the second potential aim for test data management projects. *Relevance* means that a test execution reflects the purpose of the test case. A test case might be that a US citizen who lives in Singapore buys bonds at the London Stock Exchange. This test case must not be executed with a US citizen living in the US. Executing tests with wrong data is useless and a waste of time and money. *Repeatability* is closely related. Testing and bug-fixing make a chain of actions. When a tester finds a bug, he reports the bug to the developers, who fix the bug. Then the tester re-tests to see whether the bug-fix solves the bug. Repeatability requires that the re-test after the bug-fix can be done with data/databases semantically

equivalent to the one being processed when the bug occurred.

Efficient test data *provisioning* appears on the agenda when test organizations have mastered the basics. One challenge can be the "logical distance" between test data creation and consumption. It is common for system integration tests that testers need data created in applications, modules, or by batch jobs they do not know. Then the data flow via other modules and applications to get finally to the system under test. Test data management projects can optimize how testers get data from "logical far away" applications. When test centers already use anonymous or synthetic test data, they might want to improve the efficiency. Such test data provisioning might require tool knowhow or that only dedicated persons are to be allowed to do tasks such as anonymizing production data. This demands an optimal organization to keep costs low and prevent delays in test projects.

Optimization and data privacy are the two main aims for test data management projects. There are four main approaches a business case can propose to reach them.

- Anonymization promises to take production data as input and, by masking or "shredding" data or swapping values, to produce "sanitized" data. In reality, it is more obfuscation or veiling than true anonymization (see [3] for more details).
- Synthetic test data. This data are not derived (neither anonymized nor non-anonymized) from production data. They are defined and "hand-made" for concrete test cases.
- Database application test tools. Test case management tools must reflect the specifics of testing database applications and IT landscapes. This might require new tools or, as is often easier to achieve, customizing tools already in use.
- Processes & Organization. Database-application-specific tools and anonymization tools bring only benefit to the organization when aligned with the daily work (i.e., the testing process and the organization).

Figure 1 summarizes the suitable approaches for concrete projects. If the aim is "data privacy," the approaches of "anonymization" and "synthetic test data" help. Certainly "processes & organization" must be addressed too. The aims "relevance" and "repeatability" require improved "tools" and adjusted "processes & organizations." In contrast, improving "test data provision" requires working on "processes & organization" first. Certainly, better tools might bring benefits, too.

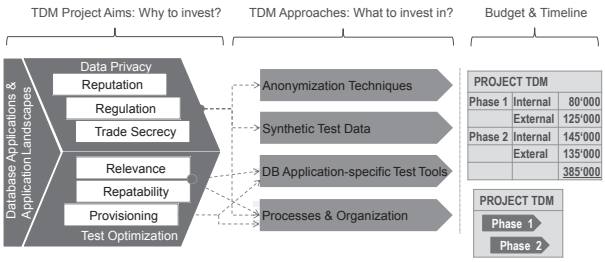

Figure 1: Business Cases for Test Data Management Projects

Test Data Management Concepts

A successful test data management project implements three concepts: database-application-aware test cases, test object types, and a test catalogue. This section describes them and brings them together in one UML class diagram. Furthermore, it discusses architectural patterns. They describe how to set up test environments focusing on what test data are used. This is a need if the business case aims for data privacy.

Database-Application-aware Test Cases

Database applications incorporate one or more databases. The state of the database (i.e., its data) impacts the test result. So a test case must specify the database state precisely. Then the test case execution is repeatable. Such test cases are named **database-application-aware test cases** (Figure 2; see [3] for a detailed discussion).

Many test cases for database applications "do something" with *one* object, the **database test object.** The one object can be a US citizen living in Singapore when testing tax reports. The one object can be a debit card when testing ATM withdrawals. However, the one object alone is not sufficient to run the test case. ERP systems, for example, have hundreds or thousands of tables. They must contain clients, accounts, branches, etc. The data must be "consistent enough" that the system runs and does not cause false positives. Only few data items really matter, e.g., the system date for testing end-of-year batch jobs. All tables together with their data form the **database test system state.** To formulate it in a more prosaic way, the database test system state sets the stage for a play. The play is the test case. The database test object is the main actor.

Certainly, test cases must define the "normal" test case attributes as well. These are the test steps, the GUI input, and expected GUI-output. If the test cases provide all this information, database-application-aware test cases ensure *repeatability*.

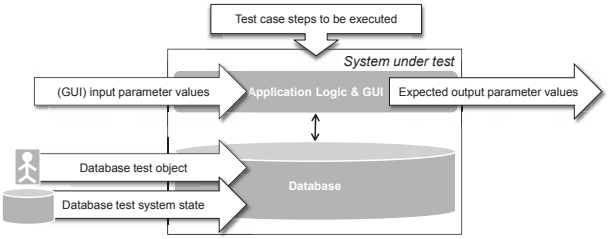

Figure 2: Database-application-aware test cases

Test Object Types

Database test objects are often complex and have many attributes. **Test object *types*** make clear which attributes really matter. A type definition consists of:

- **Name and Description.** They ease using and maintaining a type for testers.

- **Semantically mandatory attributes.** When a database test object of this type is created, all semantically mandatory attributes must have the values as specified in the database object type definition. This has implications for writing test cases too. If a test case needs an US client living in Singapore, the test case must use a database test object type with these attributes. If none is available, testers must define a new type.

- **Consistency mandatory attributes.** These attributes ease the test data creation but must not be relied upon while defining and executing test cases. GUIs or databases often demand attributes to be filled out that are not relevant for the test case. An address might require a city and a ZIP code, even if the test case only needs the country. Consistency mandatory attributes are a way to write down suitable values and value combinations work (e.g., ZIP code "8001" and city "Zurich") to smooth the test data creation. Testers can never rely on any of these values.

- **Irrelevant attributes.** They are not relevant for the test case and are not required by the system. They can have an arbitrary value or remain empty/NULL.
- **Reusability flag.** The flag defines how objects of this type can be used. When the flag is set to "reusable," test cases must not modify the values of semantically mandatory attributes.

Test centers or projects must manage their test object types. A straightforward way is with an Excel sheet, as in Figure 3. The Excel sheet lists five types plus the relevance of and values for certain attributes. When a tester writes a test case, he might need a customer with the country of residence "CH." During the test execution, the country changes to "AU." Then a suitable test object type must consider two requirements. First, the type must have a semantically mandatory attribute "Country of residence" with the value "CH." In Figure 3, three test object types fulfill the requirement: "Credit Rating Tests 5", "Credit Rating Tests 6." and "Swiss Wealth Management." The second requirement is that the test object type must not be flagged as reusable, because the test case changes the semantically mandatory attribute "Country of residence." After the execution of the test case, the used object is "destroyed." The object cannot be used for rerunning the test case, since the country is now "AU." The second requirement, "non-reusable," narrows down the list of suitable test object types for the sample test case to "Swiss Wealth Management." Before the execution of the test case, an object of the type "Swiss Wealth Management" must be created or found in the existing data. Both the country of residence and the customer type are fixed ("CH"/"Wealth Management"). All other attributes can get any value. A suggestion provided by the definition is to use "CH" as nationality.

A simple Excel sheet, as in Figure 3, looks like a solution for one tester. However, all testers of a project (or even complete test centers) can collaborate if the Excel document is put on a Sharepoint server.

The Excel sheet comes to its limit when database test object types are complex. Sketching a diagram as a specification is one solution (Figure 4). The figure exemplifies a type with a US client living in Austria having one safekeeping account with two specific equities.

Certainly, a tool-based solution is possible instead of Excel sheets. The Swisscom Test Data Organizer described later is an example. Tools can ease the integration of the type management into the test process.

	A	B	C	D	E	F	G
1	**Test Object Types**	**Reusable**	**Country of residence**	**Nationality**	**Customer Type**	**Customer Segment**	**Credit Rating**
2	***UK Corporation in London***	***YES***	UK	UK	Corporation		
3	***SME US non US Citizen***	***NO***	US	PL	SME		
4	***Swiss Wealth Management***	***NO***	CH	CH	Wealth Management		
5	***Credit Rating Tests 5***	***YES***	CH	CH	Corporation	F3	BBB-
6	***Credit Rating Tests 6***	***YES***	CH	CH	SME	C1	B

Figure 3: Simple Test Object Types for Clients (grey cells: semantically mandatory, white cells: consistency mandatory, empty cells: irrelevant)

The **benefits of test object types** are improved maintenance, clear communication, and delay-free starts of testing. [1]

Improved maintenance addresses that objects in a database change, whereas regression test cases are often stable for years. A test case might define that it should be tested using the client with ID #466119. The reason is that this client lives in Switzerland. Some months later, the client moves to Australia. The test case still points to the client though he is not useful any more. Any tests run with this client are worthless; nobody might notice that. Test object types solve this. A test object type defines the kind of object needed when writing the test case. It defers the selection of a concrete object until the test execution.

Clear communication is a side-effect of test object types. A test case stating, "Use suitable US customer," is not clear for anyone besides perhaps the person who writes the test case. A test case specified as in Figure 3 is clear. This improves collaboration and know-how transfer. This helps in two scenarios: if testers in a project change or if one tester writes the test case and a second tester executes the test case, potentially in a country time zones away.

Delay-free starts of testing means that the start of a test is not blocked by missing test data. A test manager derives from the test cases what test data he needs. He can do this well before the start of testing. He plans which testers provide the data, or he contacts other teams from which he needs data. This is important if the data cannot be created ad-hoc but requires, for instance, one or two overnight batch jobs to run beforehand.

1 While test object types are helpful, there is no need for a type concept for database test system states. A state should be described in detail to understand its purpose (e.g., end of year test 2013). However, one file with the database export is sufficient. It can be imported into as many test environments as needed.

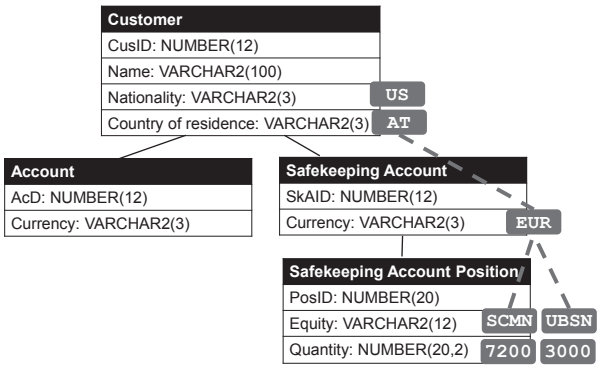

Figure 4: Complex Test Object Type Specification

Test Data Catalogue

When a tester is asked to execute a test case, she opens the test case definition, e.g., in Jira or HP Quality Center. She sees the test steps she has to execute. She sees the database test system state and the database test object type. Next, she must find a concrete object in the database that fits to the database object type, e.g., a US citizen. To find a suitable client, she can query the database and its tables. This works if all testers can access the databases on an SQL level and the queries are not too complex. Also, tests should not interfere if, by chance, various tests use the same objects.

The mentioned criteria can be met, but probably only seldom. Then a **database test object catalogue** helps which...

- provides a list of suitable objects for each database test object type; and
- allows reserving database test objects for testers.

Figure 5 contains a sample test data catalogue realized in an Excel sheet. Again, if put on a Sharepoint server, even complete teams can work with this one catalogue. A more advanced concept is a dynamic catalogue on the level of a test center. It is updated based on querying the database for "fitting" objects. These queries can be submitted in an ad-hoc mode (the database is queried when a tester looks for an object) or run as overnight batch job. Later, this paper sketches an implementation.

	A	B	C	D	E
1	DB Test Object Type	Reusable	ID	Assigned to ...	till...
2	UK Corporation in London	YES	#455464	---	
3			#573895	---	
4			#872424	---	
5	SME US non US Citizen	NO	#172885	Jane	2-Mar-2013
6			#366251	Bob	4-Mar-2013
7			#379115	---	
8			#418225	---	
9			#477521	Urs	31-Dec-2012
10			#826234	---	
11	Swiss Wealth Management	NO	#297218	Aline	22-Apr-2013
12	Credit Rating Tests 5	YES	#634321	---	
13			#634322	---	
14	Credit Rating Tests 6	YES	#634328	---	

Figure 5: Excel-Based Test Data Catalogue

UML Class Model for Test Data Management

The UML class diagram for test data management (Figure 6) unites the three concepts of database-application-aware test cases, test object types, and a test data catalogue. It is the base for our Swisscom Test Data Organizer (described later) and can also guide other test centers when incorporating test data management into their tool chain.

The class diagram has three areas. The upper area represents typical functionality of test management tools:

- **Test Case Definition.** This class represents a test case/test case definition.
- **Test Case Execution.** This class represents one execution/result of a test case execution. The only extension compared to "normal" test case executions is a reference to the database object that was used when running the test.

The lower area represents the database of the system under test.

- **Database Object.** This represents a data item (or a combination of data items) in the database of the system under test.

The middle area contains all classes specific for test data management. Today's test management systems usually do not provide this functionality.

- **Database Test System State.** This class models a state as a database export, e.g., an Oracle database export file. The class stores the name and the location of the export file plus links to the database from which the export comes. Thus, one repository can manage all exports from various databases of a test center.
- **Test States Repository.** This class is responsible for managing the database test system states of the various applications.
- **Test Data Catalogue.** This class manages a list of entries and provides the Update() method that update the entries.
- **Entry.** An entry links to an actual object in the database and states for which database test object type it could be used. The database object can be locked for exclusive use by a tester. This prevents various persons' using the same object and interfering.
- **Database test object type.** Test cases can have one database test object type. It has common attributes such as a name and a description plus the reusable flag (see page 36). As an abstract class, only its two subclasses can be initialized:
  - *Static DB test object type.* A static type is a user-defined list that links to suitable database objects.
  - *Dynamic DB test object type.* The class has two attributes, a database connection string and an SQL query. The SQL query is used by the Update() method of the class Test Data Catalogue to query the database and to update its entries. The connection string identifies the concrete database from which to pull the data.

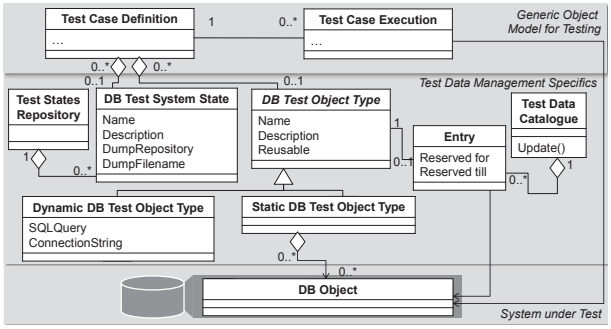

Figure 6: Object Model – Test Data Management

Data-privacy Aware Test Environments

When data privacy issues are a reason for a test data management project, it raises the question: should test environments have production data? Various patterns exist that differ when it comes to using production and/or synthetic data for testing. Figure 7 illustrates the main patterns. The classic pattern copies all data from production to the test environment. There are no restrictions in place to prevent testers and developers from seeing production data. The on-top pattern protects production data to a certain degree. The test environment is a copy from production. It is enriched with synthetic test data. There is, for example, one tenant of the application working with the synthetic data. So (certain) testers test on the tenant with synthetic data. If they have no database access, these testers do not see production data.

The *castrate & inject* pattern sanitizes the database from critical data by deleting critical data ("-") or masking/anonymizing it ("*M*"). This is can be done on a table, column, or row level. So names, such as IBAN account numbers and booking texts, or rows of customers from a specific country are deleted. If testers need some kind of data that is completely eliminated (e.g., IBAN numbers), synthetic data are inserted into the tables ("Synth"). Finally, the isolated towers pattern provides maximum data protection. The production and the test environments do not exchange any data. The test environment contains only synthetic data. Software vendors rely on this pattern, if their customers are not willing to provide database copies for them or if they want to build up a controlled data set for testing and demonstration purposes. This pattern is expensive for late-testing stages, such as system integration test environments or even system test environments for complex applications. However, it might be the only option for vendors or in the case of cross-border regulation issues.

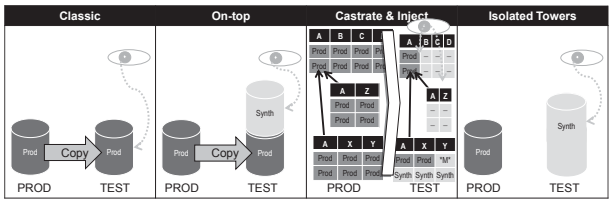

Figure 7: Production and Synthetic Test Data in Test Environments - Four Patterns (The eye symbolizes data testers can see.)

Swisscom Test Data Organizer: The System Architecture

Currently, most test management tools do not support test data management well. Thus, test centers can either invent or invest in additional tools or try to integrate the missing aspects in their existing tool chain. Obviously, the latter is less expensive and less risky. It allows the user to focus on solving the test data management challenges and to rely on industry best practices for everything else. Consequently, the Swisscom Test Data Organizer builds on top of HP Quality Center. It was chosen for three reasons:

1. **Process-focus.** Quality Center supports the complete testing process from test cases and test execution to test reports. [2]
2. **Openness.** HP is a vendor that supports tool extensions. This is a technical decision, but it has a strong foundation in HP's partner-friendly business model.
3. **Relevance.** Quality Center has a relevant penetration in the financial industries in Switzerland.

The Swisscom Test Data Organizer extends two Quality Center modules, *TestPlan* and *TestLab*. Test engineers, analysts, and managers can specify and manage test cases in *TestPlan*. Test managers can define test sets in *TestLab*. A high level understanding of a test set is that it contains a set of test cases. They might belong to one software release and, thus, are tested together. TestLab documents also the test results.

The Swisscom Test Data Organizer uses three options for parameterizing Quality Center:

- Adding new attributes and buttons to existing GUI masks.
- Implementing new functionality in Visual Basic Script. The code is triggered e.g., when pushing buttons
- Accessing Quality Center data via the Quality Center API Open Test Architecture (OTA) with a web application

The driving architectural principle is to implement as much as possible in Quality Center. Thus, the Test Data Organizer has three components (Figure 8), TDO-EXT, TDO-CUS, and TDO-SCHEMA. **TDO-EXT** is a web application. It implements the test data type management and the test data catalogue in Visual Basic. These features are too complex for Quality Center internal extensions. TDO-EXT runs outside Quality Center in a Microsoft IIS. **TDO-CUS** contains the parameterization of Quality Center: new buttons, lists and Visual Basic Script code running within Quality Center. This covers the test case extensions for database test system states, test object types, and recording the concrete database object used in tests. It also implements links to TDO-EXT. **TDO-SCHEMA** is a database schema. Its tables store data that cannot be "pressed" into the Quality Center data structure. One example is of the database test object types. The types are not a simple list of type names (this would be easy to implement in Quality Center). The types also have additional attributes such as SQL queries (see Figure 6), which do not fit to the Quality Center schema. All tables, Quality Center tables and TDO-EXT and TDO-CUS tables, reside in the same Oracle database on which Quality Center

2 Many potentially better known tools, such as Selenium, JUnit, and HP Quick Test Professional, do not meet this requirement. They are test automation and not test management tools.

runs. This prevents access problems.
The following section provides some screenshots of the tool and explains the roles of various GUIs in the test process.

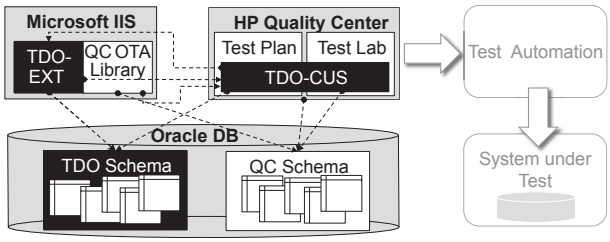

Figure 8: System Architecture

Test Data Management in the Testing Process

Managers invest in projects to achieve sustainable improvements. Sustainable improvements for test data management means, first, to integrate the test data management tasks into the test process and, second, to clarify roles and responsibilities. This section explains the best practices following a high level test process (Figure 9).

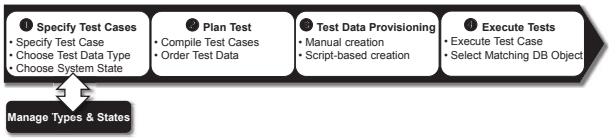

Figure 9: Test Data Management and the Testing Process

Step *Specify Test Cases*

The first step in the high level test process is the specify test case step (Figure 9, ❶). The tester takes the software specification, analyses it, and writes the test cases. No single line of code might exist at this point in time. The tester has to fill out two additional attributes per test case. The attributes are the database test object type and the database test system state. Figure 10 contains a screenshot from HP Quality Center with the two additional attributes. They are implemented in the TDO-CUS module.

Step *Plan Test*

When all test cases are defined, the test manager can start planning (Figure 9, ❷). She compiles the test case set to be executed. It contains newly defined ones plus older regression test cases. This is done in Quality Center. The test manager plans the staffing and defines milestones. She orders the needed infrastructure: servers, databases, or mobile devices. Her new test data management task is to plan the test data activities, which can be done by her testing team. If a centralized test data team exists, the test manager can order her test data with them. This can require some previous coordination. The test data ordering itself is done with the test data order form (Figure 11). It runs outside Quality Center in the TDO-EXT module. TDO-EXT retrieves the list of database test object types. This list is based on the test case set compiled in Quality Center. Next, TDO-EXT queries the system-under-test to get the number of existing objects per test object type. Then the test manager can decide how many test data objects of which type she wants to order. Her test order triggers the actual test data provisioning.

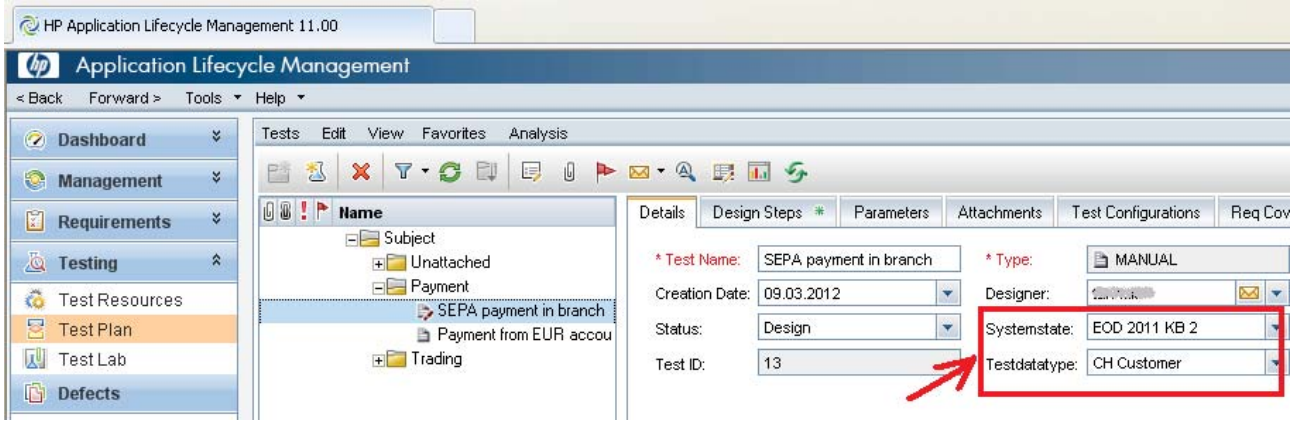

Figure 10: Test Case in HP Quality Center with extensions "Database Test System State" and "Database Test Object Type"

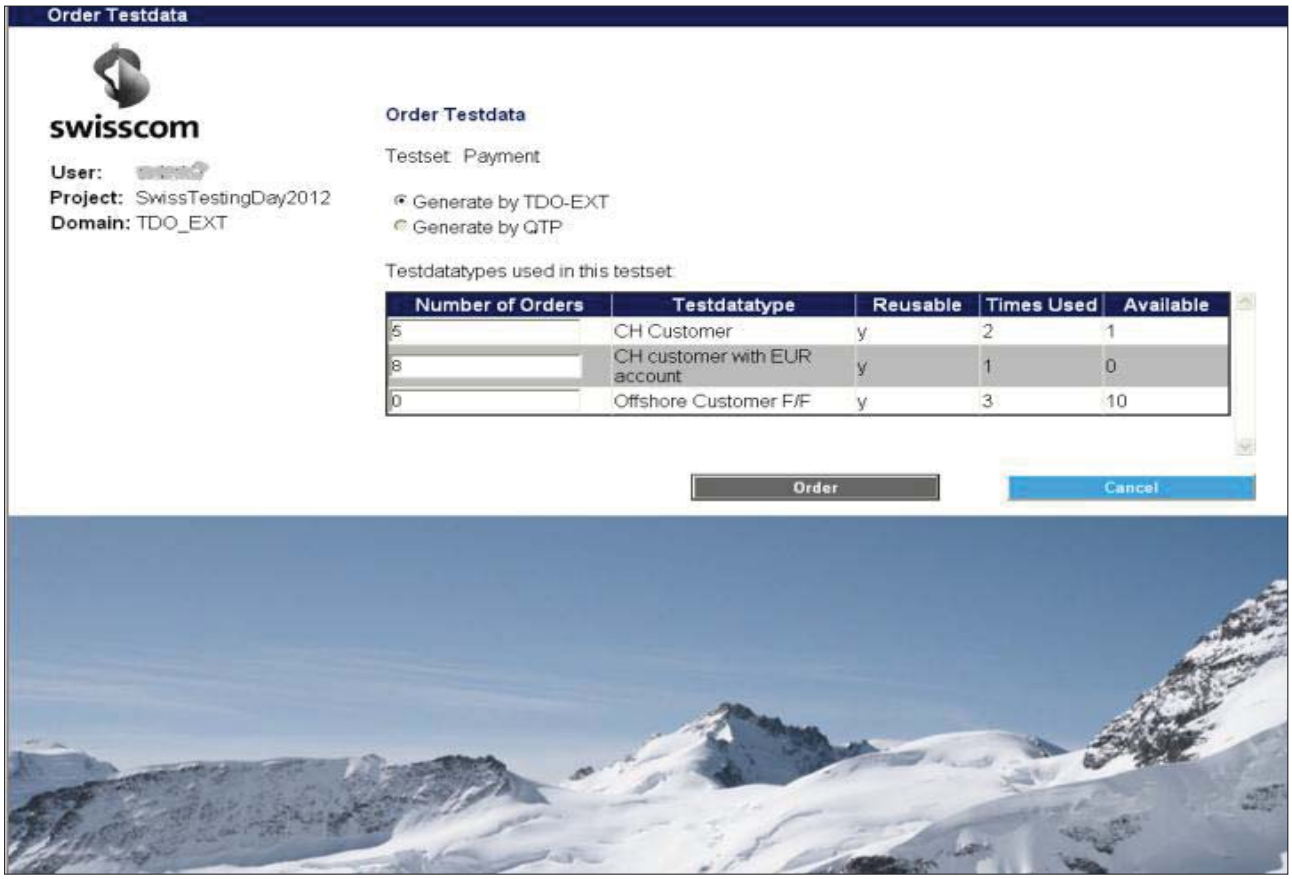

Figure 11: Test Data Order Form

Step *Test Data Provisioning*

"Test data provisioning" is the umbrella term for creating or identifying test data (Figure 9, ❸). Identifying test data can be done by querying the database. This might need more time in a complex IT landscape without federated database schema, but it is often a viable option.

The second option is to create test data. This can be a manual data input via the GUI or by some GUI-level test automation in QTP or Selenium. It can be an input file for batch jobs, or data might be loaded on the database level using SQL, too. The various options are discussed in more detail in [3].

The management has to make two organizational decisions. First, responsibilities have to be defined. Three options exist. Each tester takes care of his data. One team member takes care of all data of the team. Finally, a centralized test data management team can provide the data. The choice depends on the complexity of the environment or needed tool know-how and governance decisions in case of test data anonymization.

The second management decision is the process integration in case of dedicated persons or teams providing test data. The orders have to be tracked and assigned to test data provisioning engineers. This can be done by email, but a ticketing system such as Jira is preferable.

Step *Test Execution*

When the application is ready for testing, test engineers test manually or start the test scripts (Figure 9, ❹). In any case, the test case defines the test object type. It does not point to a concrete test data object. The test data catalogue helps by listing all database test data objects of the needed type. The Swisscom Test Data Organizer implements this functionality in the TDO-EXT module. Quality Center calls it at the start of the test case execution. TDO-EXT queries the TDO-SCHEMA for fitting objects for the type specified in the test case. In the example in Figure 12, the test case is "CH Customer." TDO-EXT identifies "10. Schmid, Florina" as the only suitable object. When pushing the "OK" button, the object is pushed back into Quality Center. Thus, the test report contains the full information which data were used for the test execution. This eases analyzing bugs and retesting them.

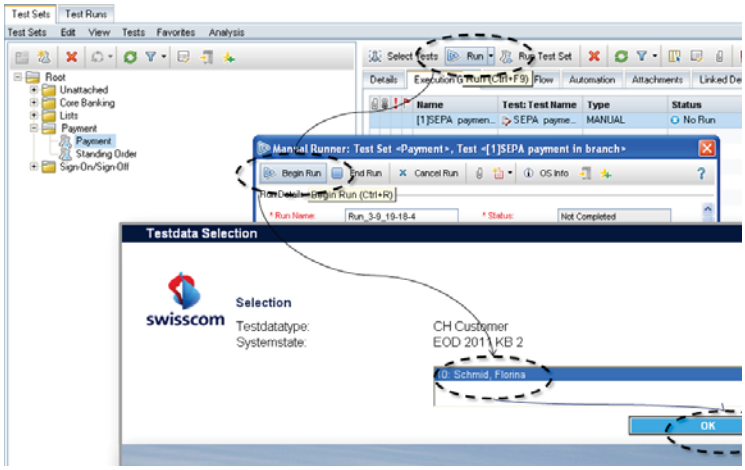

Figure 12: DB Test Object Selection GUI

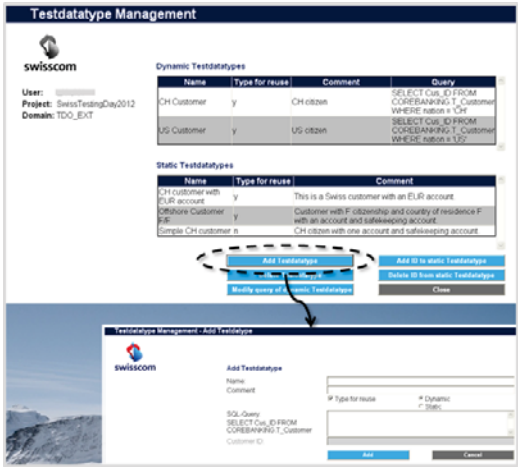

Figure 13: Managing and Adding Test Object Types

Task *Manage Types & States*

Managing database test object types and database system states impacts all test cases and all projects in a particular area. Properly done, there are synergies between projects and testers *if* types and states are coordinated. Then various projects use the same scripts for creating objects of a certain type. This reduces analysis, automation, and maintenance costs. Figure 13 shows the GUI implemented in TDO-EXT for the type management. It allows adding new test object types, static and dynamic, reusable or non-reusable, as described earlier.

Roles and Responsibilities

We conclude this section with a table listing all roles – test engineers, test managers, test analysts, and test data providers – and their tasks (Table 1). This provides a good overview of what test data management means in practice.

Role	TDM tasks
Test Engineer	• Define the database test object type for each test case • Define the database test system state for each test case • Choose and document chosen test objects for manual test execution
Test Analyst	• Manage test object type definitions • Manage system state definitions
Test Manager	• Identify test data needs/order test data
Test Data Provisioning Engineer *)	• Create and manage database test objects • Create and manage database test states

Table 1. Roles and test data management related-tasks – *) might be taken over by test engineers themselves.

Summary

Test centers that deal with many business and database applications invest in test data management for two reasons: improving data privacy, i.e. having less or no production data in test environments, and/or making testing more efficient. This paper describes four concepts to reach the aims:

1. **Database test object types.** They are an easy-to-maintain way to specify with which kind of object a test case is executed. The key point is that the test object type is stable over years, whereas objects in the database change frequently.

2. **Database-application-aware test cases.** They require the specification of test object types and database system states in test cases. This ensures repeatable tests.

3. **Test data catalogues.** They point to concrete objects that "implement" a test object type.

4. **Privacy-aware test environments.** They reduce or remove exposure of production data to testers and developers.

The concepts improve the various steps of a test process, from test case definition and planning to the reliability of the test results. Tool support speeds up the adaption within a test center. The best way is to integrate test data management into an existing tool chain. The idea is to add the missing features to one (or more) of the tools already in use. The Swisscom Test Data Organizer follows this path. It enhances HP's Quality Center with test data management functionality.

To conclude, testers, test managers, and line managers get new ideas from this paper how to improve their test center and their daily work. However, a paper cannot address one important factor. It is the human factor. Improving test data management means improving your organization. This demands guiding your employees through the transition process.

The author thanks Dr. Hans-Joachim Lotzer, Daniel Gehr, and Michael Meister for the valuable discussions, for getting the ideas to work in and with QC, and for the design and implementation of the GUIs.

References

1. Matthes, F.; Schulz, C.; Haller, K.: Testing & quality assurance in data migration projects, IEEE International Conference on Software Maintenance (ICSM), 25-30 Sept. 2011, Williamsburg, VI
2. Haller, K.: White-Box Testing for Database-Driven Applications – A Requirements Analysis, DBTest'09, June 29, 2009, Providence, RI
3. Haller, K.: The Test Data Challenge for Database-Driven Applications, DBTest'10, June 7, 2010, Indianapolis, IN
4. Haller, K.: On the Implementation and Correctness of Information System Upgrades, IEEE International Conference on Software Maintenance (ICSM), 12-18 Sept. 2010, Timisoara, Romania
5. Haller, K.: Web Services from a Service Provider Perspective: Tenant Management Services for Multitenant Information Systems, ACM SIGSOFT Software Engineering Notes, Vol. 36 (1), 2011
6. Haller, K.: Test Data Management–Addressing Data Sensitiveness and Compliance without Sacrificing Productivity, presentation at the Testing & Finance Europe 2012 Conference, May 16th/17th 2012, London, UK
7. Perry, W. E.: Effective Methods for Software Testing, 3rd edition, Wiley Publishing, Indianapolis, IN, 2006
8. Graham, D., et al.: Foundations of Software Testing: ISTQB Certification, rev. ed., Thomson Learning, London, UK, 2008
9. Deng, Y., Frankl, P. G., Chays, D.: Testing database transactions with AGENDA, ICSE'05: 15-21 May, 2005, St. Louis, MO
10. Dai, Zh., Chen, M.-H.: Automatic Test Generation for Database-Driven Applications, SEKE'07, July 9-11, 2007, Boston, MA
11. Willmor, D. and Embury, S.: An Intensional Approach to the Specification of Test Cases for Database Systems, ICSE'06, Shanghai, China, May 20-28, 2006
12. http://www.quest.com/code-tester-for-oracle, last accessed on March 5th, 2012
13. Houkjær, K., Torp, K., Wind, R.: Simple and Realistic Data Generation, VLDB'06, September 12-15, 2006, Seoul, Korea
14. Binning, C., et al.: MultiRQP – Generating Test Databases for the Functional Testing of OLTP Applications, DBTest'08, Vancouver, Canada, June 13, 2008
15. Suárez-Cabal, M., Tuya, J.: Using an SQL Coverage Measurement for Testing Database Applications, SIGSOFT'04/FSE-12, Oct. 31–Nov. 6, 2004, Newport Beach, CA
16. Terrovitis, M., Mamoulis, N., Kalnis, P.: Privacy-Preserving Anonymization of Set-Valued Data, VLDB ,08, August 23-28, 2008, Auckland, New Zealand
17. Zhong, Sh., Yang, Zh., Wright, R.: Privacy-Enhancing k-Anonymization of Customer Data, PODS 2005, June 13-15, Baltimore, MD
18. Up and Away: A Dogfight Over Frequent-Flyer Miles is Distracting Germany's Politicians, The Economist, August 8th, 2002

The Author

Klaus Haller

Klaus Haller works for the testing consulting team of Swisscom IT Services in Zurich. His focus is testing information systems landscapes, including methodology, processes, and organization, especially in the areas of test data management, compliance testing, and IT risk.

↗ klaus.haller@swisscom.com