Challenges for ERP Test Data Generation

Sebastian Wieczorek and Alin Stefanescu
SAP Research
Bleichstraße 8, 64283 Darmstadt
{Sebastian.Wieczorek, Alin.Stefanescu}@sap.com

Abstract

Software development and testing of Enterprise Resource Planning (ERP) systems demands dedicated methods to tackle its special features. As manual testing is not able to systematically test ERP systems due to the involved complexity, an effective testing approach should be automatic. Model-based testing provides by excellence a thorough automation methodology, generating test cases from system specifications. In order to execute these test cases appropriate test data has to be provided. In this paper we discuss modeling, generation and provision of test data for model-based testing of ERP software and identify several challenges in this area.

1 Introduction

In the eve of industrialization of software development, automation is playing a major rôle. As a way to validate the software quality, software testing puts a lot of effort on automation as well. Over the time, we have seen manual testing enhanced by capture and replay capabilities, then test scripting languages were employed for automation, especially for the regression testing. As software systems were becoming more and more complex, new levels of abstractions were introduced both in software development and testing. In keyword driven testing [9,6], the test cases are described in an abstract way, each test step consisting of an action keyword together with a test data reference. Automatic execution of the abstract test cases is achieved once the keywords are mapped to fragments of test scripts. One of the main advantages of keyword-driven testing is an improved test maintainability and automation. A formal, expressive and abstract test description language allows the test designers to concentrate on the test case itself rather than its implementation. However, in order to become executable, consistent and viable test data must be attached to the abstract test steps.

Model-based testing (MBT) [20] is a form of black-box testing [3] that uses structural and behavioral models, described for instance by UML diagrams, to automatically generate abstract test cases, thus automating the test case design process. The

framework making the test cases executable is similar to the one of the keyword-driven approach. MBT is gaining its momentum together with the model driven software development (MDSD) methodology. MBT test case generators aim to cover by test cases model features, e.g. all states or all transitions in a UML state machine or data features, e.g. all boundary values. As most system models are not complete in terms of data modeling, common generators produce abstract test cases which have to be further specified by adding concrete data. Therefore, it is important to have a good modeling framework for test data, such that the appropriate test data are eventually generated along with the generated test cases.

Enterprise Resource Planning (ERP) software [15] is built to support business processes for whole companies. Such software systems are very large: e.g. SAP R/3 consists of over 250 million lines of code [16] and are considered as today's largest software systems. Not only its huge size but also the data volume they are managing makes ERP systems very complex. ERP software integrates many organizational parts and functions into one logical software system, posing its own challenges to testing [10]. When it comes to the data used during testing, questions as below need to be properly addressed: "Where will the test data come from, especially the initial test data pool? How will one produce new data for transactions that require unique data? How one ensures the consistency of master, transactional, respectively test data?"

As we have seen above, test data is very important if we want to have a seamless test automation process. In this paper we will discuss several problems regarding ERP test data that can be identified in the area of functional testing of ERP systems.

The paper is structured as follows. First, we present some preliminaries notions and related work in Section 2. We continue in Section 3 describing the different ERP test data and the relations between them. Section 4 consists of a collection of identified challenges and Section 5 present our future research plans.

2 Preliminaries and related work

In this section we present concepts and methodologies from the literature to which we will refer in the next sections.

A *test case* describes the operational steps through which a certain functionality of property of the System Under Test (SUT) is validated. Following [11], a test case is usually a concatenation of four procedures:

1. *Preamble* (or setup): Sets the test up, getting the SUT into the correct state to run the test.

2. *Body*: Executes a certain scenario or sequence of steps described by the test case in the SUT.
3. *Observation* (or verification*)*: Checks and evaluates the test results.
4. *Postamble* (or teardown): Takes the SUT back in some standard state, where the next test can run.

The execution of the above steps in the SUT is only possible with the appropriate test data.

To ensure the quality of large software systems produced in several development cycles, *regression testing* is used to ensure that additional code and changes to the existing software are not affecting the functionality already implemented and that the requirements are always satisfied. Regression testing consists of a set of test cases that are executed regularly at every stable stage of the development cycle. These test suites should be reproducible and this requires a constant initial system state at the beginning of a test run together with appropriate test data. Such requirement can be easily satisfied by stateless systems or systems with a constant state at the end of each session, e.g. protocol machines. As a positive consequence, test cases can be annotated with static and concrete test data. Other systems can be forcibly brought into a defined initial test state using a preamble procedure. For some systems however the requirement of starting a test in a constant initial state cannot be met. For example it is practically impossible to reset ERP systems because the effort is simply too high even in a developing stage and for a running ERP system this might even not be allowed, e.g. due to legal obligations.

In a software system, state changes do not usually depend only on the performed actions, but also on the input data and the actual internal state, including master data and state variables. As a result various data constraints have to be considered during test design when annotating test cases with data. The current test data constraints focus on transactional data relations during the run of one test case only. Because it was relatively easy to annotate test cases with concrete data using the mentioned restriction, test data modeling was not highly prioritized and so a consolidated approach to link data models and behavior models is still missing [18]. To the best of our knowledge, a classification of different test data constraints especially for ERP systems is missing.

2.1 Related work

Data modeling is usually applied in the field of database layout planning and management. Relational modeling, e.g. using the Entity-Relationship Model, is the most common way to describe database schemas and languages like SQL have been invented to retrieve the data from such relational databases. The object-relational database model recently came to prominence, reflecting the paradigm shift in programming and

providing an object oriented data structure in databases. Related modeling techniques use object-oriented models, enhanced by constraints. The most popular approach utilizes the OMG's UML standard and OCL constraints. A discussion on the adoption of the UML/OCL approach and its possible challenges can be found in [1]. For ERP system tests, data modeling has to be adapted as the data layout information inside the system is distributed and as a whole also might be too complex.

Ontologies in computer science represent a set of concepts within a domain and the relationships between those concepts. Instances of the concepts as well as domain rules are also part of an ontology. Unlike data models, ontologies are supposed to be independent of a particular application of the described domain [19]. Given their genericity, ontologies can be applied to a limited degree in the highly customized ERP systems.

Test data generation has been discussed in the context of automated testing from the very beginning. For white-box testing, approaches like symbolic execution, actual execution, and random testing [8] are used. They are based on code analysis or code execution tracing to find value sets needed to execute predefined paths through a system under test. For black-box testing and in particular model-based testing, test data can be generated from several types of specifications like finite state machines, pre/post models, or UML transition-based models [20,14,5]. Most of the research addresses more the test data generation for each test case independently and less the global consistency of different test data. The problem of test data modeling has been tackled in different ways, by approaches like boundary analysis [12], domain analysis [3] or the classification tree method [7], but these are not very well integrated with the behavior models [18]. Test data modeling using the UML/OCL data modeling approach is a promising alternative [1,4,2]. We believe however that the problems of incomplete data constraint definitions and constraint solving of very complex systems such as ERP systems need further investigations.

3 An introduction to ERP data

In the ERP world, the data can be interpreted from two perspectives: from a business, respectively from a technical perspective.

From a business point of view, data is divided into master data and transactional data:

- *Master data* represents relative static data that remains valid over a period of time and is used in several use scenarios. For example supplier information (such as name and address) or product information (such as size and description) is stored once, seldom changed and can be inserted automatically during several transactions.

- *Transactional data* in contrast is short living, used only for a specific transaction and can always be related to master data. For example ordering of a product will be processed in a sales order transaction. Information like the quantity of products or the delivery deadline is individual for the order and hence transactional data. General product data is in opposition (and as explained above) master data.

From a technical perspective the distinction between master data and transactional data is less relevant. All transactions in a system have to be stored in databases and therefore master data tables only have the purpose of eliminating redundancies. In the above example for a sales order only the transactional data is saved explicitly while master data is referenced. More important from a technical perspective is the distinction between user generated and automatically derived data for a transaction. On first sight this seems to be related to the business data classification and master data without any doubt perfectly fits into the automatically derived data class. On the other hand the choice of which set of master data to use lies in the hand of the user. Also transactional data is not only generated by user input but also might be derived automatically (e.g. the current date) or from a prior transaction. For example the quantity of a product in a sales order might be determined by the current need for production or a request from a customer. Therefore, a technical distinction between system data and input data will be used:

- *System data* serves as the applications internal data stack. It is stored in a database that is directly accessible from the application. Access from the outside is usually very limited.
- *Input data* is all information that has to be provided by users during execution and cannot be derived automatically.

The motivation for such a classification is that data which is stored in the system can be considered as consistent and conform to predefined technical and business constraints (as explained above this also includes former transactional data). The assumption only holds if the system correctly implements all consistency checks and constraint determinations. Consequently, the runtime system only has to check those consistency rules and constraints during a transaction, which are connected to the user generated data input. This is important because the number of transaction critical constraints that have to be checked before saving any data must be reduced to a manageable subset in order to be enforceable in a reasonable, i.e. user convenient, time frame.

Data consistency (e.g. concerning technical, business, and standardization constraints) can be carried at different levels in the system. Common database systems provide means to define simple to very complex and even dynamic constraints on data. The constraints are defined using field properties or guards which are checked every time the associated data is changed. Conformance to the ACID concept (Atomic-

ity, Consistency, Isolation, Durability) further allows rollback of whole transactions, even if constraints are violated at the last processing step. However constraint validation on data in ERP systems is usually implemented outside the database system as early in the data processing (and hence as close to the UI layer) as possible. The main objective for it is to resolve constraints (or roll back transactions) quickly and without blocking additional resources. While consistency checks on the application level is scalable and fast, a rollback inside the database is usually much slower and hence can be seen as the performance bottleneck for ERP systems. Shifting constraint handling to the application also enables ERP system designs which are independent of the database implementation. This is due to the fact that database vendors each have their individual language to define data constraints. Correct implementation of data constraint handling and checking therefore is one of the major goals of ERP application testing.

4 Challenges posed by ERP test data

This section gives an overview of test data characteristics for ERP regression testing and their implications. The discussion starts with system data related issues and concludes with issues regarding input data.

4.1 System data

System data is a necessary ingredient for testing as internal data will also be processed during productive execution. Two special subjects regarding system data for regression testing have to be considered:

- *Test data provision*
- *Test data stability*

The *system test data provision* is strongly connected to the preliminary of regression testing: 'to execute a test run on the same system state' is a very expensive requirement for regression testing in ERP systems. As mentioned before, data constraints are handled outside of the database layer and so even though deleting all data on the databases is possible, loading data directly to the database is difficult. It would basically mean to either manually or automatically insert consistent test data to the system. Manually this is impossible because the complexity of data relations is too high. Automation of the data insertion would mean to implement the expensive systems constraint checking and solving mechanisms, and therefore is also not efficient.

One answer to this issue would be to start the test cases on an empty database. If tests need data inside the system, the obvious solution is to make data insertion

through the application mechanisms part of the test. Theoretically this seems to be working but considering even a simple employment process in an HR module shows the practical weakness. To hire someone, an open position has to be entered to the system. The position demands other information like the unit to which it is assigned. The unit needs information about its manager and the company it belongs to, among others. Further iterations lead to the creation of a vast amount of master data even for a simple test run. Other processes (e.g. a report over all employment processes) additionally need saved transactional data inside the system. There are usually so many internal data dependencies that most test cases would be forced to set up an unmanageable amount of master data in the preamble phase in order to be executable. Moreover, after each test case the database state would have to be reset. Such practice will not scale for ERP systems.

A more realistic solution is to use the empty system and to fill it with common test data by using the application itself only once, before the testing phase. The filling itself can be seen as a set of fundamental test cases. Using this method, mainly the master data stack for test cases is generated, which can be applied for testing all processes that do not rely on saved transactional data. Even though this approach seems to be much more feasible, problems still arise. First, using an untested system to generate a master data stack may be error prone and hence the quality of master data will be poor. Furthermore in early production stages it might even be unfeasible because the mechanisms needed for data insertion have not been fully implemented. Despite the mentioned problems, providing common test data by using already implemented application services and transactions is the only practicable solution so far. Consequently, errors caused by poor test data quality should be part of any ERP testing strategy.

Stability of common test data is the second issue of ERP regression testing. System data will be consumed by some tests, e.g. when a process to dismiss an employee is tested: each time an employee is dismissed the system data will be changed and hence the entity cannot be used for employee related transactions like promotion any more. Other tests might alter common master data unintentionally due to implementation faults in either the system or the test itself. This obviously proves to be problematic for other test cases depending on the same set of data. Finding out whether a failed test was caused by incomplete or changed master data is hard to observe (a failed test even might occur randomly e.g. depending on the execution order of test case steps) and even harder to avoid.

Strategies of cloning master data tables in an initial system state to fasten a later data reset are applied for stability. Using them allows system reset in smaller frequencies (e.g. once a month). However directly coping master data will always result in the loss of stored transactional data, as former references and relations will be destroyed. Also structural changes of the master data (e.g. adding a field for the gender at the personal

data) which are carried out frequently during development phases can result in the invalidation of the master data clone. Therefore cloning master data tables are limited to test data refresh in between development waves.

4.2 Input data

Until now only the test data inside the system has been discussed but in order to test ERP applications also data from the outside has to be provided for the test runs. Mostly this input data will be transactional but in order to test master data modification both kinds of data might have to be added to test cases. Similar to system data, the general feature that makes it hard to deal with input data is the complexity of constraints attached to it. Mainly two features have to be addressed in order to provide valid input data for ERP system tests:

- *Uniqueness of data*
- *Relations to system data*

Statically applying *unique data* to test cases is challenging for regression testing. This is necessary if for example it is prohibited to enter a product with exactly the same features twice into the system. A common solution is to add variability to unimportant features from a technical perspective (e.g. times stamps attached to the identifier). To completely solve the problem, it could also be checked whether certain data is entered into the system and to adapt the input data if necessary. Due to performance concerns and the low risk of duplicated time stamps, this solution is less common.

A more complex problem is caused by test cases *relating input data to special system data* inside the system. Test preambles (cf. Section 2) are used to getting the system under test into a defined state which allows the test execution. As argued above, for most of the test cases it is impossible to add all master data because the complexity is too high. On the other hand, if the deletion of an entry in the master data is subject to a test, it is better to create one additional entry in the test preamble which can then be used. Otherwise this test might interfere with other test cases using the data which is missing after successful test execution and further the test case itself could fail the next time missing the deleted entry, too.

Another way to tackle the problem of required system data is to relatively define the data inside the test cases and compute the concrete values during test execution. In the deletion example only the essential features of the entry could be defined in contrary to a concrete identifier. The test system could then analyze the applications database and exchange the relative with a concrete identifier at runtime. Apart from all the technical problems arising with this approach (e.g. is there a mechanism to retrieve

identifiers implemented and how powerful is it) already in this small example another test related issue becomes apparent. A test case deleting system data could eventually consume data relevant for other tests – even though in some cases postambles could be used to reload the deleted data.

5 Conclusions and future research

Many special characteristics of test data in ERP systems and their influence on black-box testing have been described. Apart from the problems concerning complexity of test data generation for very large system, e.g. constraint solving, also test data provision and consumption have been identified as major issues. We will continue to investigate methods responding to data consumption and strategies of combining those methods in order to find an optimal solution. Most likely categorization of test cases (e.g. system data consumer / system data dependent / system data independent) and then an orchestration of individual strategies will be part of such a solution.

Our research plans will address the modeling of test data for large ERP systems in the context of MBT and its annotation with constraints, most probably in a UML/OCL context. One emphasized difficult aspect is how to provide data for regression testing when the system under test cannot be easily reset. Moreover, model-based testing may also come in two different flavors: online and offline testing [20]. For online MBT, the generation of test cases is dynamic, i.e. the set of test cases already generated and their result on the SUT impacts on the choice of the next generated test case. In this area we see also a challenge to perfect a dynamic test data generation approach able to cope with unpredictable states and test data sets.

The next-generation product of SAP for mid-market, currently code-named "A1S"[1], will employ data types on a generic level (core components) and data types for specific vertical industry [17]. The "A1S" business objects, defined in the Enterprise Service Repository (ESR), are trees of business object nodes. A business object node is structurally defined by a Global Data Type (GDT). In other words, a business object is a structured set of GDTs. As a result the data structures and layouts used inside "A1S" are very diverse and comprise complex associations and interdependencies. We plan to validate our future research solutions addressing test data constraints internally on the new "A1S" system.

Acknowledgements: This work was partially supported by the EU-funded project Modelplex [13].

[1] http://www.sap.com/solutions/A1S

6 References

1. T. Baar. Experiences with the UML/OCL-approach in practice and strategies to overcome deficiencies, In *Proc. of Net.ObjectDays2000*, pp. 192-201, Net.ObjectDays-Forum, 2000.
2. H. Balsters. Modelling database views with derived classes in the UML/OCL-framework. In *Proc. of UML'03*, LNCS 2863, pp. 295-309. Springer, 2003.
3. B. Beizer. *Black-Box Testing: Techniques for functional testing of software and systems.* John Wiley & Sons, Ltd., 1995.
4. M. Benattou, J.-M. Bruel, and N. Hameurlain. Generating test data from OCL specifications. In *Proc. of the ECOOP'2002 Workshop on Integration and Transformation of UML models (WITUML'2002), 2002.*
5. L.C. Briand, J. Cui, and Y. Labiche. Towards automated support for deriving test data from UML statecharts. In *Proc. of UML'03,* LNCS 2863, pp. 249-264. Springer, 2003.
6. H. Buwalda, D. Janssen, and I. Pinkster. *Integrated test design and automation: Using the Testframe method.* Addison-Wesley, 2001.
7. Z. R. Dai, P. H. Deussen, M. Busch, L. P. Lacmene, and T. Ngwangwen. Automatic test data generation for TTCN-3 using CTE. In *Proc. of ICSSEA'05.* CMSL, 2005.
8. J. Edvardsson. A survey on automatic test data generation. In *Proc. of 2nd Conf. on Computer Science and Engineering*, pp. 21–28. ECSEL, October 1999.
9. M. Fewster and D. Graham. *Software test automation*. ACM Press, 1999.
10. M. Helfen, M. Lauer, and H. M. Trautwein. *Testing SAP solutions.* SAP Press, 2007.
11. ISO/IEC 9646-1. *Information technology – open systems interconnection – conformance testing methodology and framework, Part 1: General concepts.* International Standards Organization, 1994.
12. N. Kosmatov, B. Legeard, F. Peureux, and M. Utting. Boundary coverage criteria for test generation from formal models. In *Proc. of ISSRE'04*, pp. 139-150. IEEE Computer Society, 2004.
13. Modelplex: MODELling solution for comPLEX software systems. 6th Framework Programme EU Project. Reference: IST 34081 (IP). Webpage: *http://www.modelplex.org*.
14. J. Offutt, S. Liu, A. Abdurazik, and P. Ammann. Generating test data from state-based specifications. In *Software Testing, Verification and Reliability* 12(1), pp. 25-53. John Wiley & Sons, Ltd., 2003.
15. D. E. O'Leary. *Enterprise Resource Planning systems. Systems, life cycle, electronic commerce and risk.* Cambridge University Press, 2000.
16. G. Pike. Supporting business innovation while reducing technology risk. SAP Insight, June 2006. Online at: *http://www.sap.com/services/programs/pdf/BWP_Supporting_Business_Innovation.pdf*
17. H. Plattner. Trends and Concepts in the Software Industry (part I). Lecture notes from Hasso Plattner Institute. Summer Term 2007. Online at: *http://epic.hpi.uni-potsdam.de/Home/TrendsAndConcepts_I_2007*
18. I. Schieferdecker. Modellbasiertes testen. In *OBJEKTspektrum*, vol. 3, 2007.
19. P. Spyns, R. Meersman, and M. Jarrar: *Data Modelling versus Ontology Engineering.* SIGMOD Record 31(4), 2002, pp. 12-17.
20. M. Utting and B. Legeard. *Practical model-based testing, a tools approach*. Morgan Kaufmann Publishers, 2007.