Status Quo Review Of Existing Testing Practices

Existing Methodology

When reviewing the status quo, companies implementing SAP need to assess what software methodology or approach guides the work products and deliverables of the SAP resources, including the SAP testing team.

Large SAP system integrators such as Deloitte Consulting and IBM offer methodologies and implementation guides such as Thread Manager and Ascendant™ for either upgrading or initial installations of SAP. SAP itself offers the SAP Roadmap methodology embedded within the Solution Manager platform. Recognized bodies such as IEEE, SEI, and the U.S. Department of Defense (DoD) 5000 series directives for life-cycle management and acquisition, to name a few, also provide software methodologies for implementing an ERP/Customer Relationship Management (CRM) solution such as SAP R/3.

Corporations that are missing a recognized methodology for implementing SAP can rely on software approaches that conform to the waterfall, spiral, and evolutionary models. These models offer different approaches for implementing software that include prototyping, dealing with programs that have a large scope, or unstable requirements. Depending on the size of the corporation implementing SAP, it is possible that the corporation already has other large software initiatives and a successful life cycle for doing so that can be leveraged for implementing SAP.

A successful software methodology, whether created in-house or adopted from another body, needs to have templates, accelerators, and white papers for testing ERP applications. Methodologies specifically designed for building software from scratch or from the ground up may not be suitable for implementing an out-of-the-box solution such as SAP and thus not offer any relevant guidance for testing SAP.

The project and test manager must provide special attention to the project's methodologies and how existing testing activities and tasks conform and align to the project's methodologies. If no formal methodology exists within the project, then efforts must be taken to ensure that the testing approach and test plans are adequate for the project to help fulfill testing criteria.

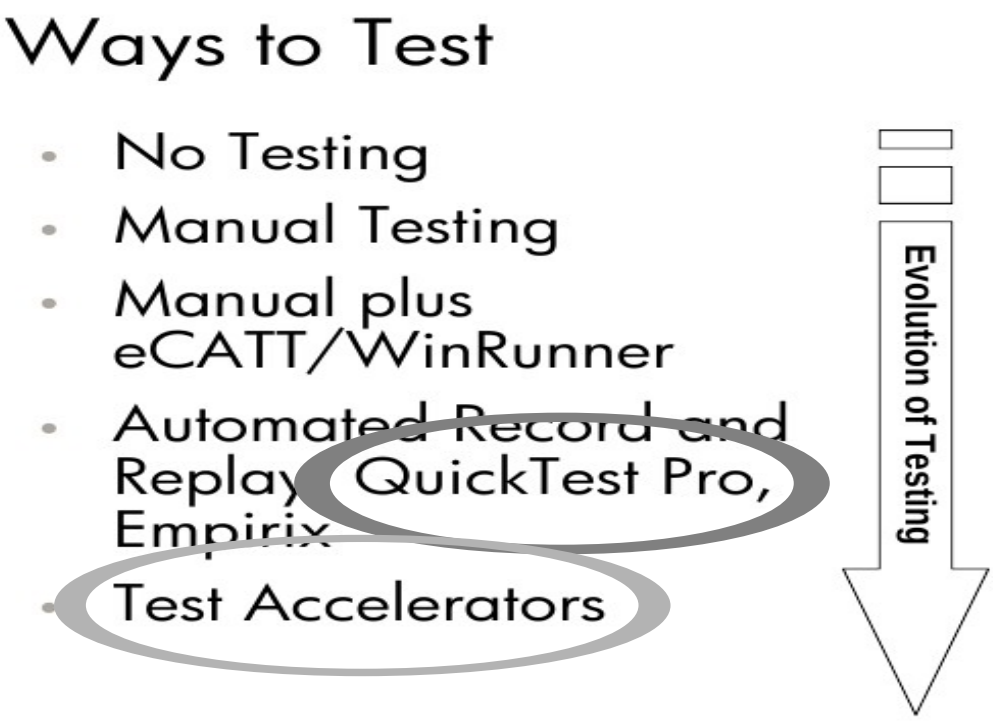

Fig. Various ways to test a software

Drawbacks to manual testing

While manual testing may be the best option for a high percentage of projects, it is not without its shortcomings.

For example:

•Manual tests can simply take too long—testers must tediously document each step of a test case and manually execute each test, reproduce defects, and so on.

• The dramatic increase in complexity of today's computing environments is amplifying test coverage requirements, creating more pressure to move to automated testing.

•Corporate globalization and geographically dispersed teams create a need for standardized testing processes, which manual testing does not readily facilitate.

•When there is no automated process for testing, there is typically no automated way to keep documentation synchronized with the testing process; each element of the test plan is a separate entity and every change must be managed and maintained individually.

•Manual tests are subject to higher risk of mistakes and oversights than automated tests.

The disadvantages of record and playback only become apparent as you begin to use the tool over time. Capture replay always looks very impressive when first demonstrated, but is not a good basis

for a productive long-term test automation regime. The script, as recorded, may not be very readable to someone picking it up afterwards. The only value of an automated test is in its reuse. A raw recorded script explains nothing about what is being tested or what the purpose of the test is. Such commentary has to be inserted by the tester, either as the recording is made (not all tools allow this) or by editing the script after recording has finished. Without this information any maintenance task is likely to be difficult at best.

A raw recorded script is also tied very tightly to the specifics of what was recorded. Depending on the tool, it may be bound to objects on the screen, specific character strings, or even screen bitmap positions. If the software changes - correction: when the software changes - the original script will no longer work correctly if anything to which it is tightly bound has changed. Often the effort involved in updating the script itself is much greater than re-recording the script while running the test manually again. This usually does not give any test automation benefit. For example, the values of the inputs recorded are the exact and detailed values as entered, but these are now 'hard-coded' into the script. The recording is simply of actions and test inputs. But usually the reason for running a test is to look at the test outcome to see whether the application does the right thing.

Simply recording test inputs does not include any verification of the results. Verification must be added in to any recorded script in order to make it a test. Depending on the tool, it may be possible to do this additional work during recording (but it takes additional time and effort) or during the first replay, otherwise it will be necessary to edit the recorded script.

Manual tests are often labor intensive, time consuming, inconsistent, boring, and lengthy, and comparison of test results is tedious and error prone. At first glance, these problems look ideal for test automation, and indeed that may be true. However, it is not necessarily the only solution to these problems.

The first question to ask is whether these manual tests actually give value for money. If they are too lengthy, weeding out ineffective or redundant tests could shorten them. This may enable them to be run manually within a shorter time frame. If the tests take too much elapsed time to run manually, perhaps recruiting more testers would help. If the tests are very labor intensive, perhaps they could be redesigned to require less effort per test, so the manual testing would be more productive. For example, a test may require testers to sit at different machines in different rooms. If all of the test machines were moved into one room, one tester may be able to oversee two or more machines at

the same time.

If the test input or comparison of results is error prone, perhaps the test procedures are unclear. Have the testers been trained in how to input, execute, and analyze the tests correctly? Are they aware of the importance of the correctness of test results?

Comparison of test results is probably one of the best uses of a computer. Most test execution tools include some comparison facilities. However, most operating systems also have comparison utilities that can be used to good effect, whether or not you have a comparison tool.

If executing the current tests is boring, this probably does indicate a need for tool support of some kind. Things which people find boring are often done better by a computer.

Setting up test data or test cases is repetitive and 'mechanical'; the testers find it boring and make too many 'simple' errors. This problem is a good candidate for a test execution tool. On the other hand, why are test cases and test data being set up in this way? It may be better to organize the test data into pre-packaged sets which could be called upon when needed, rather than setting them up every time, particularly if this is an error-prone process.

Test documentation serves different purposes. Test plans contain management information about the testing process as it should be carried out. Test scripts contain information about the detail of tests to be run, such as what the inputs and test data arc. Test reports contain information about the progress of tests that have been run. An area where test execution tools can help overcome documentation problems is where inadequate records are kept of what tests have been executed. If careful records are not kept, tests may be repeated or omitted, or you will not know whether or not tests have been run. The test log does provide an audit trail (although it may not necessarily be easy to find the information you want from the raw logs produced by the tool).

An automated solution often 'looks better' and may be easier to authorize expenditure for than addressing the more fundamental problems of the testing process itself. It is important to realize that the tool will not correct a poor process without additional attention being paid to it. It is possible to improve testing practices alongside implementing the tool, but it does require conscious effort.

The right time is:
when there are no major organizational upheavals or panics in progress;

when one person has responsibility for choosing and implementing the tool(s);
when people are dissatisfied with the current state of testing practice;
when there is commitment from top management to authorize and support the tooling-up effort.

If one or more of these conditions do not apply to your organization it does not mean that you should not attempt to introduce test automation. It merely implies that doing so may be somewhat more difficult.