WHITE PAPER | APRIL 2016

Test Data Management

How it may be the only way to drive Continuous Delivery

Huw Price
CA Technologies

Table of Contents

Section 1

Introduction to Continuous Delivery

Continuous Delivery has become somewhat of a buzzword in the software development world. As such, numerous vendors promise that they can make it a reality, offering their tools as a remedy to the traditional causes of project delays and failure. They suggest that by adopting them, organizations can continually innovate and deliver quality software on time, and within budget.

The appeal of Continuous Delivery is understandable. In this modern application economy, organizations rely on software to deliver value to their customers. Business and IT needs are therefore more closely aligned, and a company's position in a market depends on their ability to deliver value to the consumer on a day-to-day basis. IT teams need to be able to respond quickly to changing market and consumer expectations, developing software that delivers on changing business critical needs, while reducing testing costs and time.

Though many tools claim to drive Continuous Delivery, they are primarily logistical, and can only be deployed late in the development lifecycle. DevOps is seen only in terms of Operations, with tools taking software that's already been designed, then supporting development and regression testing. The systemic issues that arise earlier on, from the requirements gathering stage onwards, remain and cause costly bottlenecks and project delays that make the successful implementation of Continuous Delivery impossible.

In particular, the importance of Test Data Management (TDM), or having the right data delivered to the right place, at the right time, for testing purposes is overlooked. Poor test case design and the inefficient provisioning of poor quality data means that test teams find themselves without the data required to fully test a system. Quality is compromised in favor of delivering software on time and within budget.

Section 2

Poor TDM Prevents Continuous Delivery

For most organizations and vendors, TDM starts and ends with copying, masking, and possibly subsetting production data. These TDM strategies are purely logistical, focusing only on moving data, while data is overlooked when designing project requirements. Once production data has been copied, and migrated to a development, testing or QA environment, it is often called a "gold copy" – i.e., a perfect set of test data, with which testers can execute any tests needed to fully test a system.

Masking and subsetting are a good place to start and go part of the way to resolving the traditional pain points of infrastructure costs and compliance. But, on their own, subsetting and masking a production database carries many of the issues inherent to using production data itself. These issues make the successful implementation of Continuous Delivery impossible.

Wasted Time

Most organizations lack a central TDM team or provisioning service, so that data has to be found or created by local teams. Testers can spend up to 50% of their time looking for data, and as much as 20% of the total SDLC is spent waiting for it. This leads to testing bottlenecks which make true agility and Continuous Delivery impossible. In one instance, Grid-Tools (now part of CA Technologies) came across a team who were supposed to engage in three week "sprints", but had to spend four weeks preparing the data for the sprint.

Masking or subsetting production data entails having testing teams to actually go and find the data to use. This is a time consuming, arduous task, exacerbated by the inconsistent storage of data in uncontrolled spreadsheets. For example, Personally Identifiable Information (PII) might be added in a "notes" column, for instance if there are three columns for credit cards, but one customer happens to have four. What's more is that the data must be referentially intact. The more complex the data is, the harder this is to achieve, and the time and effort spent masking data often outweighs the time testers might have spent making it from scratch.

Manually creating data from scratch can be equally as time consuming. Further, as the data is usually created with specific test cases in mind it quickly becomes outdated and irrelevant, as the real world constantly changes. A commonly given example is trading patterns and currency exchange rates, where data becomes outdated on a daily basis. When this happens, data has to either be updated - another time consuming delay – or, more often, "burned". This is likewise the case when a version changes, or if someone requests a data refresh, or when having to create data every time a virtual environment changes.

It is rarely possible to share and re-use manually created data between teams. This inability to leverage earlier effort spent producing data creates extra work, and forces organizations to make a straight and undesirable choice between reducing cost and time-to-market versus delivering valuable software applications with all of the required functionality.

Dependency Constraints

Much of a tester's time is wasted waiting for data to be provisioned, or to become available. Organizations and vendors often view the SDLC as a series of linear stages, where one team finishes something, passing it on to the next. Teams therefore have to wait for "fit for purpose" data to become available from upstream teams, and might find themselves unable to use required data as another team is working with it.

This is in contrast to agile parallel development principles at the core of Continuous Delivery, where every moment of an individual's time and ability to innovate is used for its maximum value. What's more, a change made by one team might affect data in such a way that when another team uses it, their application tests will fail for apparently no reason.

If teams are having to manually create data, or wait days or weeks for it to become available, they cannot be expected to quickly respond to changing requirements, and deliver fully tested software.

Poor Quality

The real issue with using production data in non-production environments is quality. Production data typically only provides 10-20% functional coverage, and so any sampling methods are unlikely to provide data that meets all the test cases required to build a new subsystem. A lot of production data is very similar, being drawn from common or "business as usual" transactions, and is sanitized by its very nature to exclude the bad data that will break systems. Testing therefore tends to place a higher focus on "happy path" and less on non-functional and negative testing.

However, negative testing should constitute around 80% of testing, as it is these outliers and boundary scenarios that cause systems to collapse. As long as testing focuses on happy paths, defects will invariably make it into production, leading to rework, critical delays, spiraling costs, and potential project failure. Industry research shows that it takes 50 times longer to fix a bug during testing, than finding it in requirements stage[1], with such delays and inability to respond to change making continuous delivery impossible.

The functional coverage of production data can be supplemented by manual creation, but this is an inexact, time-consuming and unscientific method. There is no way of verifying that maximum coverage has been achieved, nor of knowing that the data remains referentially intact.

High Costs

Finally, in addition to the high costs of rework and late delivery, the actual copying of production data is a prohibitively slow and expensive process. Report shows that some organizations find themselves with as many as 20 copies of a single database, incurring high expenditure on hardware, licenses and support costs.

Further, high volume data storage can be very expensive. A related third party data storage research points out that moving data to lower-cost tiers including the cloud temporarily reduces the cost of storage, but it does not solve or address the fundamental problems of data growth. In other words, any TDM policy which ends with migrating, subsetting and masking data cannot address the issue of how to affordably store the data which is processed by ever-more complex, modern, composite applications.

There is also the more costly danger of non-compliance, which is not resolved even when production data is masked. An independent study revealed that the cost of data breach in 2014 increased by 15% in the average cost of a data breach, to $3.5 million per record, while the new European Union (EU) data protection directive, due for enforcement in 2016, will result in the maximum fine levied for a data breach being equivalent to €100 million[2] or 5% of global turnover – whichever is higher.

Masking production data does not guarantee compliance, as the real danger remains human error, whereby over 50%[3] of data breaches can be related to insider behavior. Further, when masking, the referential integrity of data must remain intact, otherwise application testing might break down. But, the more complex the data is, the easier it is to crack, as more pieces of information can be correlated.

The only real way to avoid such high infrastructure costs, and to ensure that sensitive data is not leaked, is to not use production data. Although such expenses are not peculiar to organizations wishing to implement a continuous delivery framework, they certainly stand in contrast to its principles. They prevent organizations from being able to deliver quality software that delivers on changing business demands quickly, and within budget.

Section 3

Requirements Based Approach to TDM

Organizations wishing to implement Continuous Delivery must rethink their testing and development processes rather than simply reshuffle them. They need to reconsider their approach to TDM. A complete, end-to-end approach to Test Data Management, driven by requirements, allows organizations to do shift left testing, mitigate risk, and minimize defect creation, thereby delivering quality software faster, and for less.

Rather than thinking of data on a test case by test case basis, organizations should think of data in terms of design decisions – in terms of the requirements themselves, designing test cases with data linked directly to them. Tailoring test data to requirements ensures that it is 'fit for purpose', while the ability to provision it quickly to test teams means that they can quickly respond to the changing demands of the business.

In this respect, Continuous Delivery cannot start late in the SDLC: it must begin with an idea – from the requirements themselves – so that test and development teams can quickly respond to changing business needs, working from this changing idea throughout. Teams need to be able to easily validate and verify the requirements, deriving test cases, linked to expected results and virtual data, directly from them. Such model based testing is what it properly means to "shift left" and continually deliver software – to condense all the work of the development lifecycle into the requirements gathering stage, with all subsequent work flowing easily from it, even if the requirements change.

Building Better Requirements

The requirements themselves need to contain all the qualitative information about a system required for testing, so that testers can derive use cases and test cases directly from the initial "idea", with traceability introduced between them. Such traceability is necessary if testers are to quickly update their tests when requirements change.

With proper tooling, flowchart modeling can produce a flowchart which contains all the qualitative information about a system needed for testing, in spite of the simplicity of designing the flow itself. CA Agile Requirements Designer (formerly Grid Tools Agile Designer) underpins a flowchart with all the functional logic needed to automatically derive the smallest number of test cases with maximum coverage, without leaving the business "gawping" in the way that cause and effect modeling or pairwise approaches might[2].

Further, a flowchart increases the likelihood that requirements will be unambiguous and complete. They break the disparate 'wall of words' and cumbersome diagrams which usually constitute requirements down into small, digestible chunks. These processes reflect the cause and effect logic of a system, in effect making up a series of 'what if, then this' statements.

Not only does this help reduce the 56%[4] of defects that emerge from ambiguities in requirements, but it forces the requirements team to think in terms of modeling a system – in terms of constraints, restraints and boundary conditions. From such requirements, modeled as a flowchart, then, every possible path through a system can be identified. Test cases which offer 100% functional coverage can then be derived, so that testing covers every possible path through a system, including negative paths and unexpected results[3].

With an active flowchart, these test cases can further be linked to complexity metrics, virtual data, test data, automation scripts, expected results and backlogs, concentrating the effort of the SDLC into the requirements gathering stage.

Provisioning 'Fit for Purpose' Data

Automating test case design processes with tools like CA Agile Requirements Designer allows testers to generate the right test cases needed to test requirements optimally and to their satisfaction. This, then, brings us to the central claim of this paper: that better TDM is required for continuous delivery. Testers need access to 'fit for purpose' data that can help them achieve 100% coverage of test cases, delivered to the right place, at the right time.

Quality

As discussed, production data cannot provide the coverage needed to fully test a system. Synthetically generating test data, by contrast, can produce small, rich sets of data that cover all possible scenarios, even when an event has not occurred before. Each real-world scenario can be thought of as another data point, and so synthetic data can be created even for new and upcoming scenarios. CA Test Data Manager (formerly CA Data Finder or Grid-Tools' Data Maker) uses intelligent data profiling techniques to take an accurate picture of a data model, generating rich, sophisticated data that provides 100% of functional variations based upon it.

With a requirements based approach, the generated data is matched to a test case, ensuring that it is fit to serve each individual tester's needs. Such "test case data generation" makes it more likely that test teams will find defects first time round, avoiding the time-consuming rework that makes continuous delivery impossible.

Time

The time spent manually creating or manipulating data, or waiting for it to become available, stands in contrast to the principles of continuous development, as described.

By contrast, automated tools for data creation, such as CA Test Data Manager, can work directly with RDBMs or ERP API layers, allowing users to generate data as quickly as their processing power will allow. Bulking scripts can double the amount of data an organization has, as fast as the database infrastructure can handle it. This ensures that 'fit for purpose data' exists in a matter of hours, not weeks, so that testers have the data they need, when they need it within a sprint.

Powerful test matching functionality also means that if data exists, it can be identified and mined from multiple, disparate sources, before being matched and allocated to the appropriate test cases. Such test matching has proven to reduce the time taken to find and provision data by 95% when compared to manual processes, and also allows teams to know which tests will fail due to data issues before they run.

Storing test data in a centralized data warehouse can further avoid the bottlenecks caused when teams are waiting for data to become available downstream. The CA Test Data Manager On Demand Test Data portal offers dynamic form building, which allows users to select what sort of data they want, based on specific criteria, such as a type of credit card or geographical location. This form will either allocate the data needed for the test case, or will create new data if none exists. This means that engineers have more time to fix the defects exposed by testing, instead of spending 50% of their time waiting for data to be provisioned. It also means that organizations can centralize data ownership under the IT security team, only provisioning sensitive data to the authorized staff who request it.

Cost

Finally, a better TDM policy reduces the risk of cost over-runs. Detecting and resolving defects earlier can reduce the creation of defects by up to 95%[5] and realize savings of over $50k[5] above per defect. What's more, using smaller, richer subsets of data can reduce infrastructure costs by up to $50k[5] per database, while running fewer, better quality tests also drives down testing time and cost significantly. The danger of costly non-compliance is also removed when using synthetic data, as sensitive information does not need to leave production environments.

Section 4

Summary

Copying production data into non-production environments as the source of "truth" prevents the successful implementation of continuous delivery. Test teams are likely to find themselves without fit for purpose data when they need it to fully test software. They are therefore unable to quickly respond to changing business requirements. While subsetting and masking tools resolve certain pain points, production data is not of sufficient quality to uncover and resolve defects.

Adopting an end-to-end requirements-based approach to software development and treating test data as a valuable reusable asset is paramount to the successful implementation of continuous delivery. Working from an "idea" throughout, teams can quickly respond to changing business needs, using 'fit for purpose' data, delivered to the right place, at the right time, to fully test software in the shortest number of test runs required.

This approach allows organizations to create efficiencies in their test data provisioning, whilst also mitigating the risk of delays, rework and spiralling costs that make Continuous Delivery impossible. It leads to the continual delivery of valuable software that delivers on critical business needs, on time, and for less.

Section 5

References

1 http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/

2 http://www.agile-designer.com/resources/test-case-generation-bloor-market-report/

3 See Llyr's Wyn Jones' Primer of A Critique of Testing for a conceptual, mathematical treatment of the advantages of flowchart modelling, available at http://www.agile-designer.com/resources/critique-testing-primer/

Section 6

About the Author

Co-founder of Grid-Tools & VP at CA Technologies

Voted "IT Director of the Year 2010" by QA Guild, Huw Price has been the lead technical architect for several US and European software companies. Specialising in test automation tools he has launched numerous innovative products, which have re-cast the testing model used in the software industry.

1 Ponemon Institute: 2014 Cost of Data Breach

2 EU Proposed Data Protection Regulation

3 2013 Data Breach Investigations Report

4 Bender Report: Requirements Based Testing

5 Metrics collected from Grid-Tools' implementation experience

CS200-196723_0416