ACHIEVING TEST AUTOMATION SYNERGY

CI/CD PIPELINES + TEST AUTOMATION + TEST DATA GENERATION

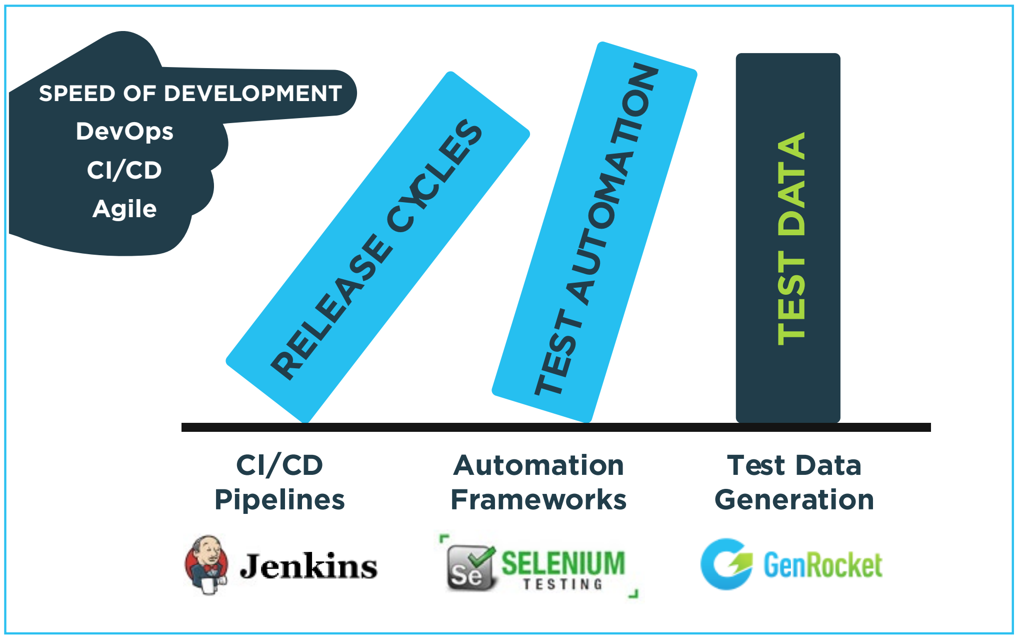
How **Test Data Generation** integrates with CI/CD pipelines and test automation tools to unlock the full potential of accelerated software development.

TABLE OF CONTENTS

ACHIEVING TEST AUTOMATION SYNERGY

CI/CD PIPELINES + TEST AUTOMATION + TEST DATA GENERATION

1 EMBRACING QUALITY AT SPEED



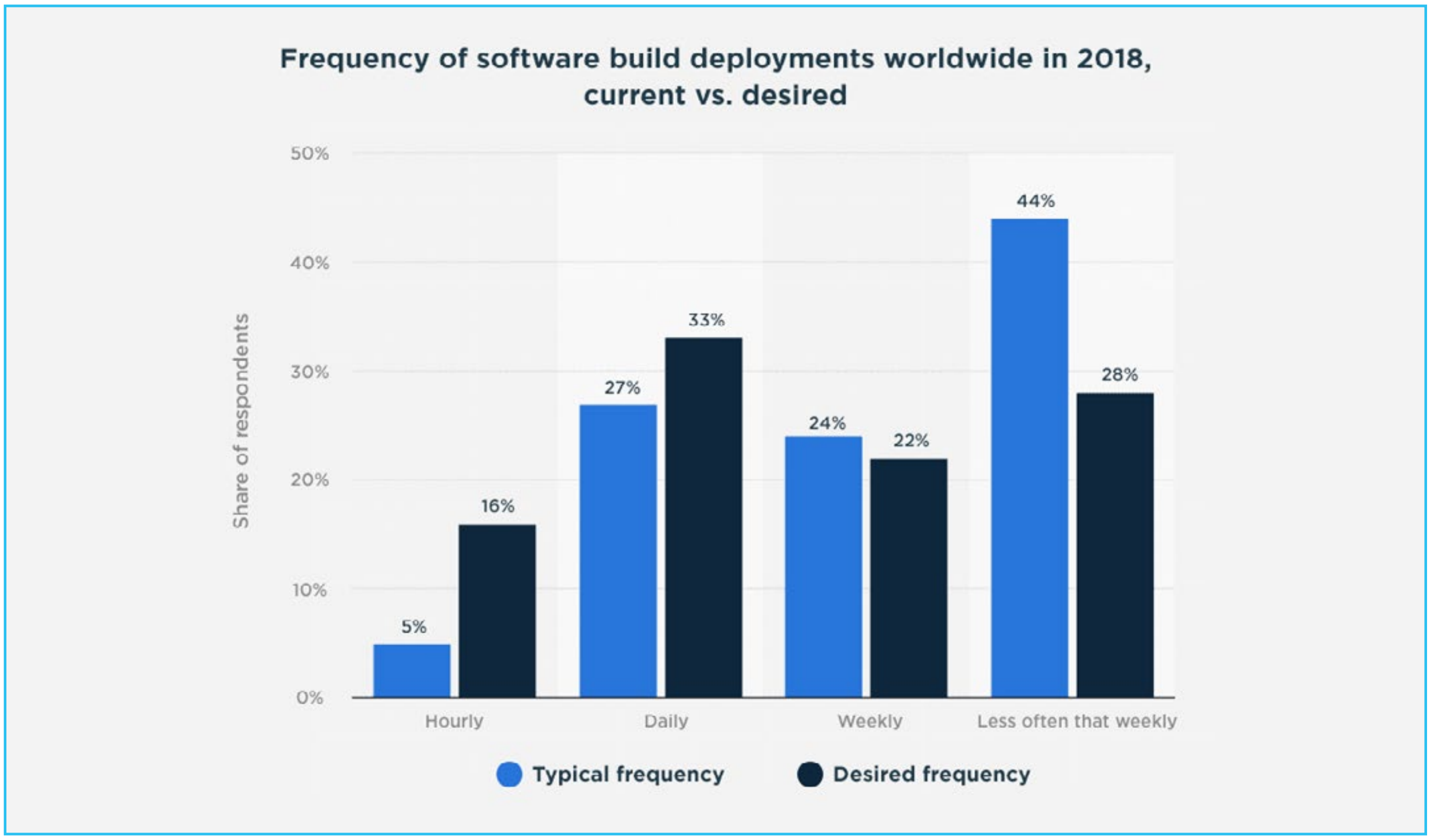
Software engineering teams are accelerating the speed of development through the adoption of Agile and DevOps methodologies combined with the deployment of CI/CD pipelines. This trend has made test automation essential for QA organizations to keep pace with development.

Faster release cycles are driving automation in every testing category. Test automation has become the key to ensuring quality at the speed of continuous delivery.

This eBook will introduce *Test Data Generation* (TDG) as the next generation of Test Data Management (TDM) and will illustrate how this new and innovative technology can help your organization achieve the full synergy of test automation in a continuous delivery environment.

Release Cycles Keep Getting Faster

For most development environments, DevOps has become a standard approach for compressing release cycles from months and weeks to just days and hours. Market data now characterizes DevOps as widely adopted, with over 50% of organizations having implemented it and over half of the remaining organizations planning to implement the approach within 12 months (Forrester).



In similar fashion, CI/CD pipeline tools have become the go-to technology for managing faster release frequencies and Jenkins leads the way with 70% market share of all CI/CD server installations (Datanyze).

Almost half of all development organizations are now aspiring to achieve daily or hourly release frequencies (49% according to Statista), while the number who are still planning to release less often than once per week has dropped from 44% to just 28%. The use of CI/CD pipelines with test automation is essential for meeting these accelerated release frequency goals.

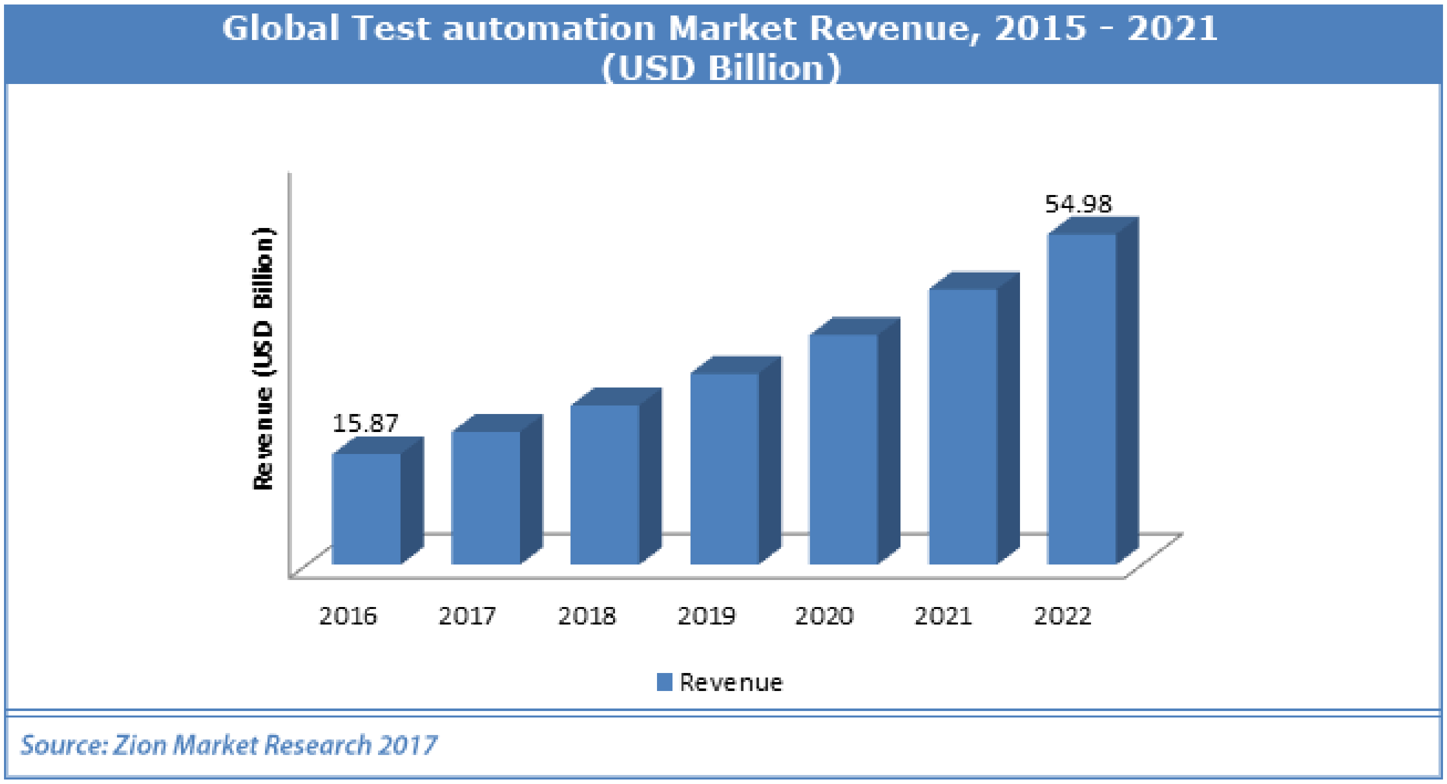
Test Automation to the Rescue

Test automation enables exponentially faster testing than manual methods at a lower cost. It enhances the ability to perform unit, functional, regression, integration and performance testing at higher velocity and frequency while offering greater control over the consistency of testing, the coverage of code and the quality of software released to production.

Test automation tools are essential enabling technologies for implementing continuous integration and testing for successful CI/CD pipelines.

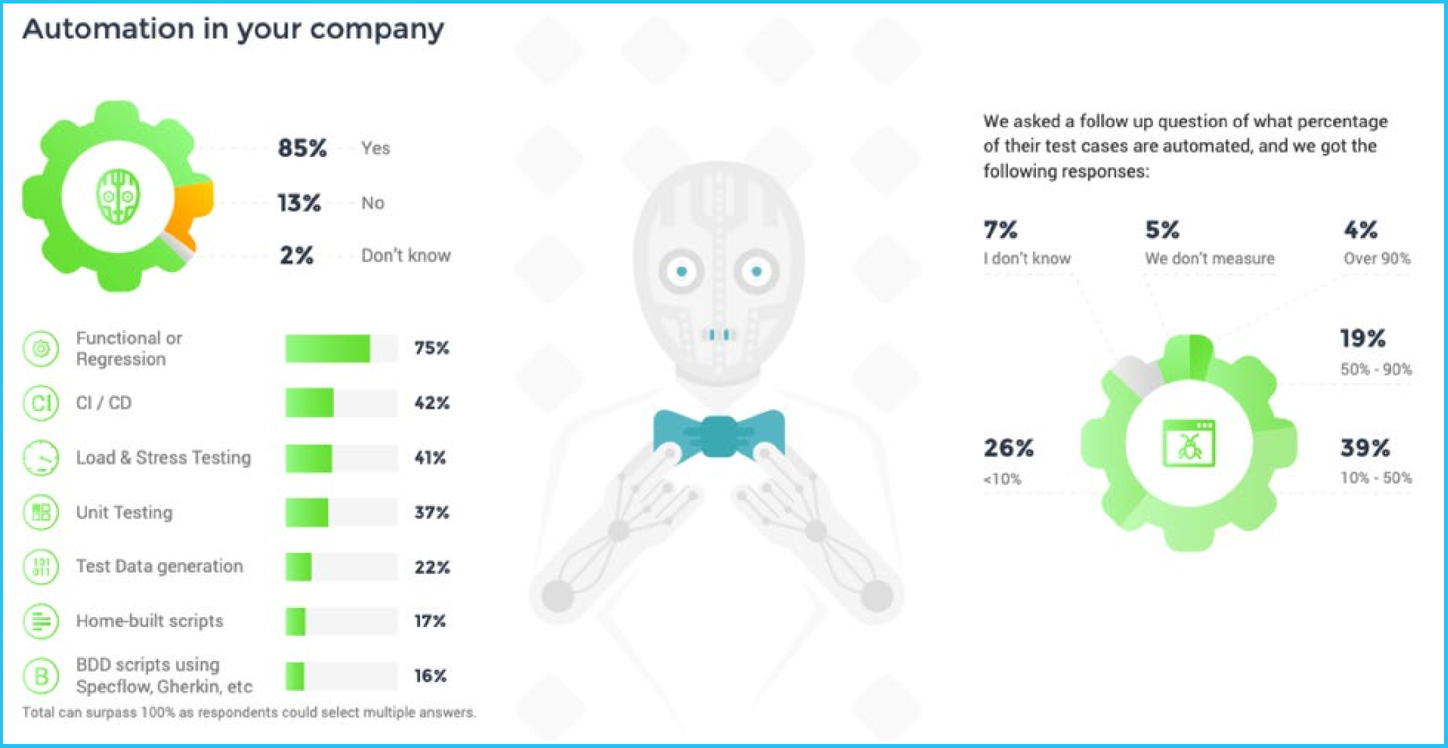
The global market for test automation is expected to grow at a rapid pace, more than tripling in size between 2015 and 2022 as it grows annually at 23% to reach almost $55 billion during the forecast period (Zion Market Research).

This heightened demand for test automation solutions has led to a wide assortment of test tools and test automation frameworks to orchestrate them. Chief among these testing frameworks is Selenium, a suite of open source software testing tools used by leading software-driven companies like Netflix, Google, HubSpot, and Fitbit. According to iDatalabs, Selenium is in production at more than 29 thousand companies, holds a 27% market share, and maintains a commanding lead over the rest of the pack in the test tools arena.



Test Automation Brings New Challenges

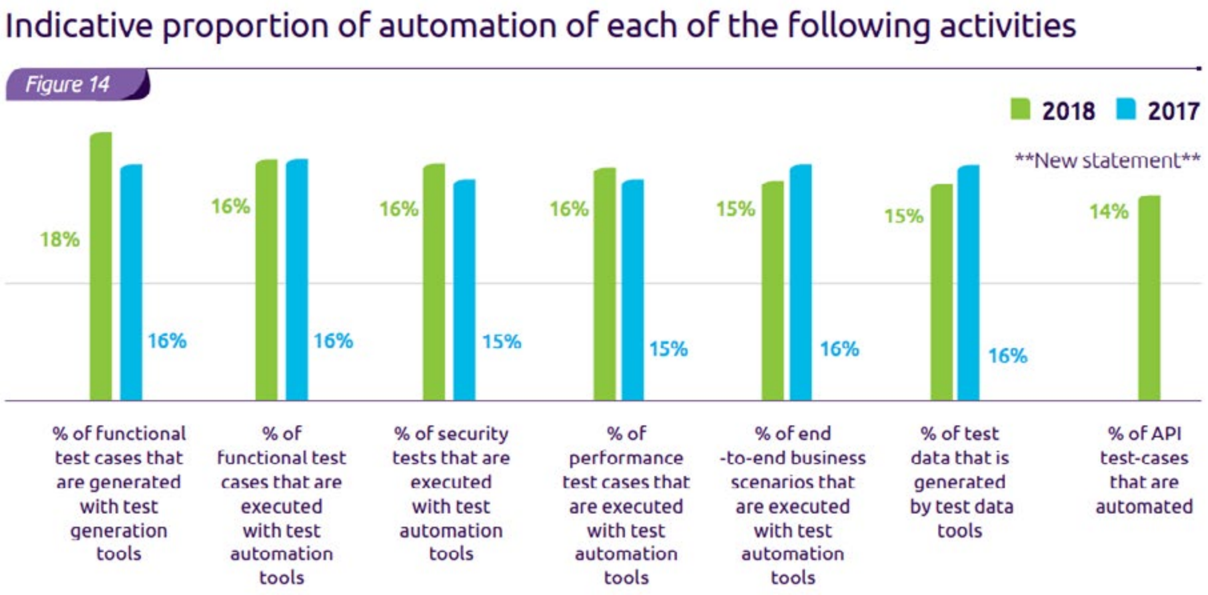
Intense market interest and aggressive growth projections would seem to imply that test automation is something of a miracle cure for meeting the demand for continuous testing and integration solutions. A survey of more than 1600 QA professionals in 60 countries conducted by QA Intelligence found that **85 percent of organizations have already implemented some form of test automation.**



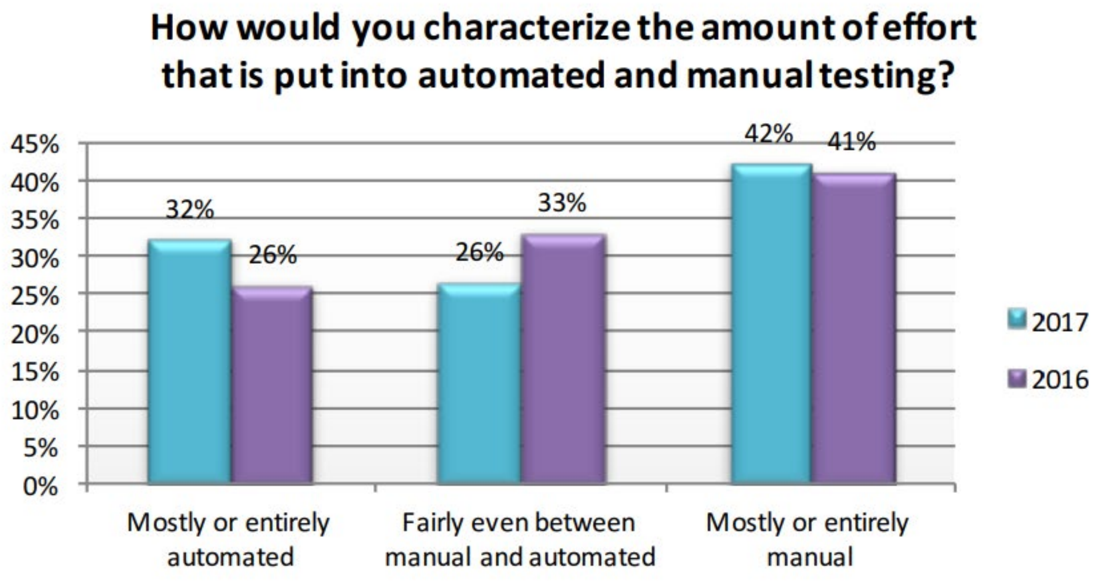
At first glance, it would appear test automation has already reached wide-scale adoption. However, a second and more revealing question posed to the same group of respondents found that **only 23% of organizations had automated more than 50% of their test cases and only 4% had automated over 90% of their test cases.** Apparently, adoption of test automation is one thing, while full-scale deployment is quite another matter.

The most recent edition of the World Quality Report breaks down the penetration of test automation into various test categories. In a comparison of automation levels between 2018 and 2017, the report shows that no testing category had a proportion of their test cases automated to a level greater than 18% with very little advancement from 2017 levels to 2018 levels.

Cleary, organizations are facing significant challenges with full deployment of their test automation tools.



Another survey of 732 software professionals conducted by Saucelabs in 2017 found the majority of their efforts were still being applied to manual testing rather than automated testing.



Why is automation failing to dominate the day-to-day operational world of software testing?

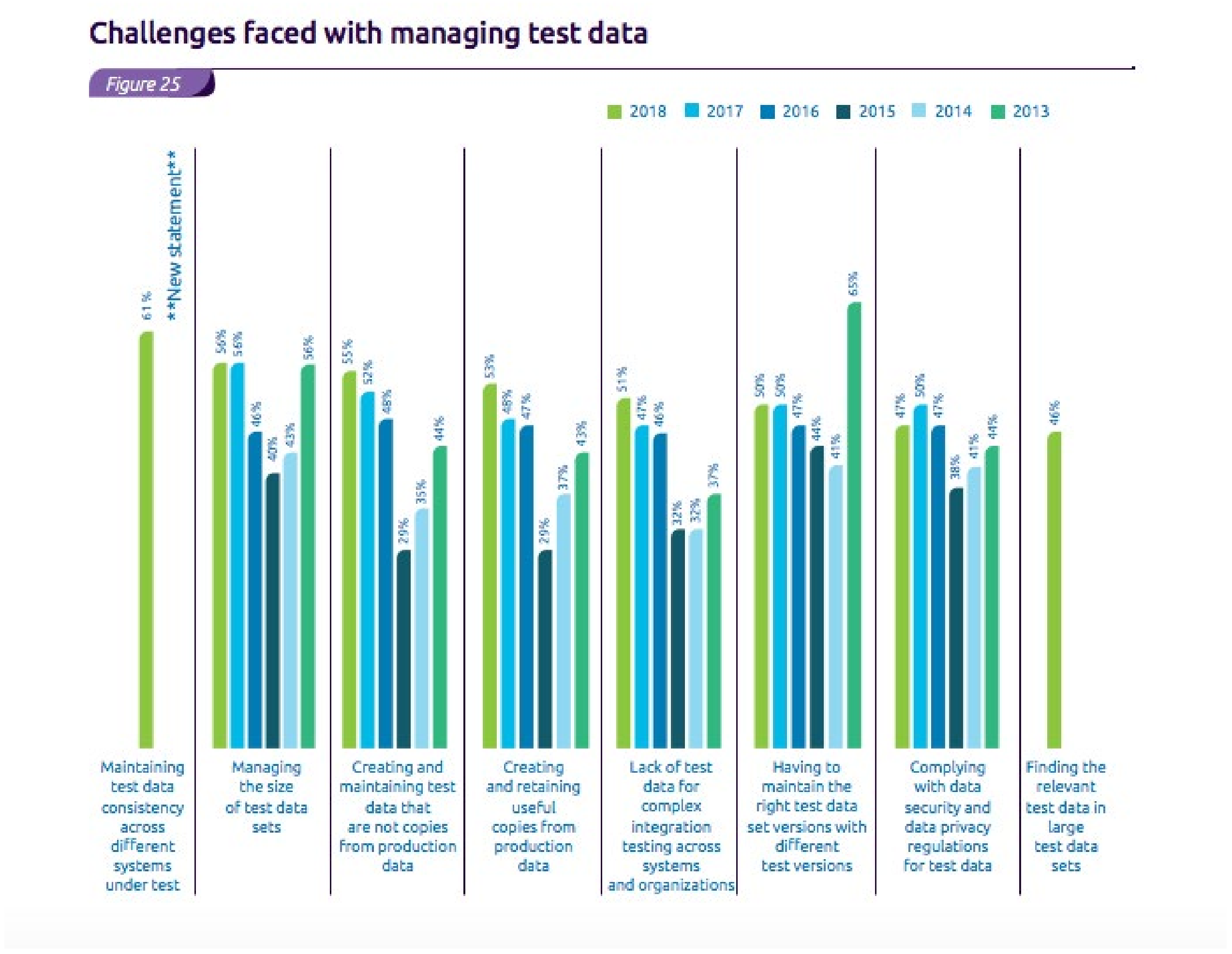
What challenges are preventing its full deployment and associated benefits in cost reduction, higher efficiency, increased coverage and improved quality?

The 2018 World Quality Report, examined this area as well. Their study found 61% of respondents have difficulty automating their QA and testing processes because applications change too much with each release, indicating they have trouble building an adaptable test automation solution. This can be viewed as a temporary problem as QA teams learn how to use and adapt their tools more effectively and efficiently over time.

However, a second and more systemic problem was identified by the research. Almost half of the survey respondents (48%) reported challenges with predictable and reusable test data for their test environment, making it especially challenging to automate testing.

To achieve continuous testing and integration, one needs access to continuous test data as well.

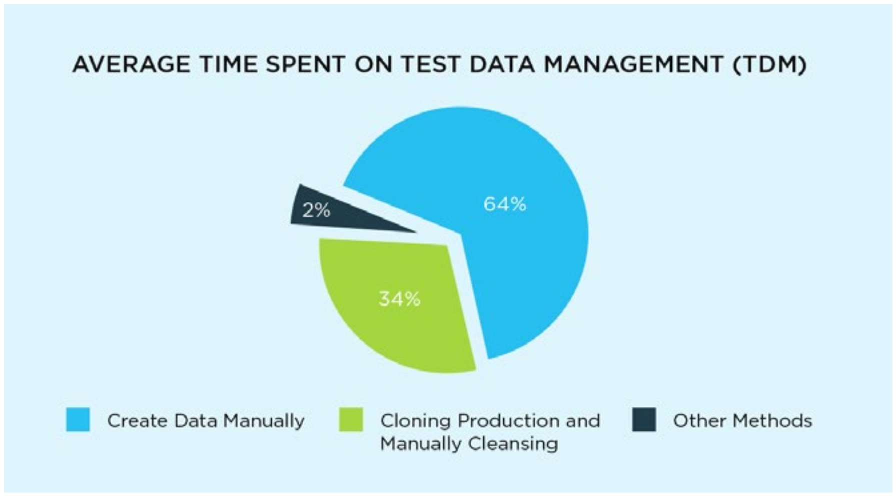
As stated in the World Quality Report, *"66% of our respondents said that they used spread sheets to manually generate new test data for multiple iterations of testing... another 62% of respondents said that they copied production data which they anonymized before testing... Clearly, the maturity of test data provisioning is not changing in enterprises."*



The lesson to be learned is obvious: By itself, test automation is only half of the solution needed by testers to fully and successfully automate a majority of their test operations. The other half of the solution is an automated approach for provisioning quality test data at the scale and pace required for continuous testing.

Test Data Generation can provide an automated approach that unlocks the full potential of test automation.

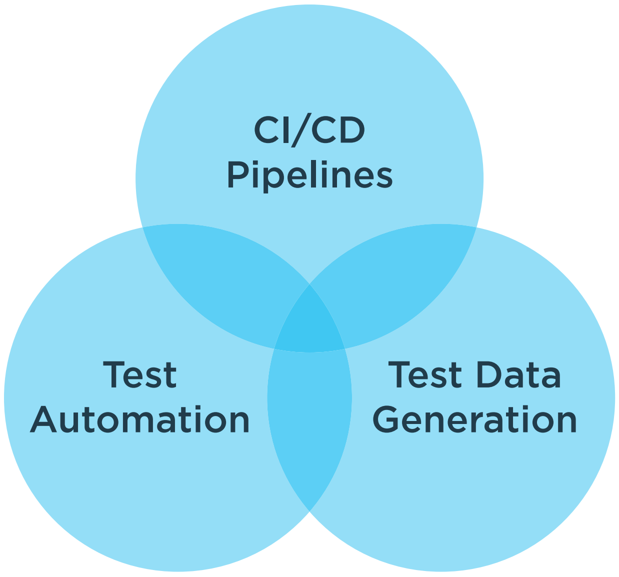
2 THINKING DIFFERENTLY ABOUT TEST DATA



There is increasing demand for high quality test data to be provisioned instantly and on-demand to enable continuous testing and integration. This puts tremendous pressure on the traditional TDM model of copying, subsetting and masking production data. It takes too long and requires too much effort.

Until now, the only TDM alternative has been to manually create rows of test data in Excel, limiting the volume and variety of data for use by test automation tools.

TDG removes the time delay associated with the use of production data and the limitations on volume and variety associated with manually created data.



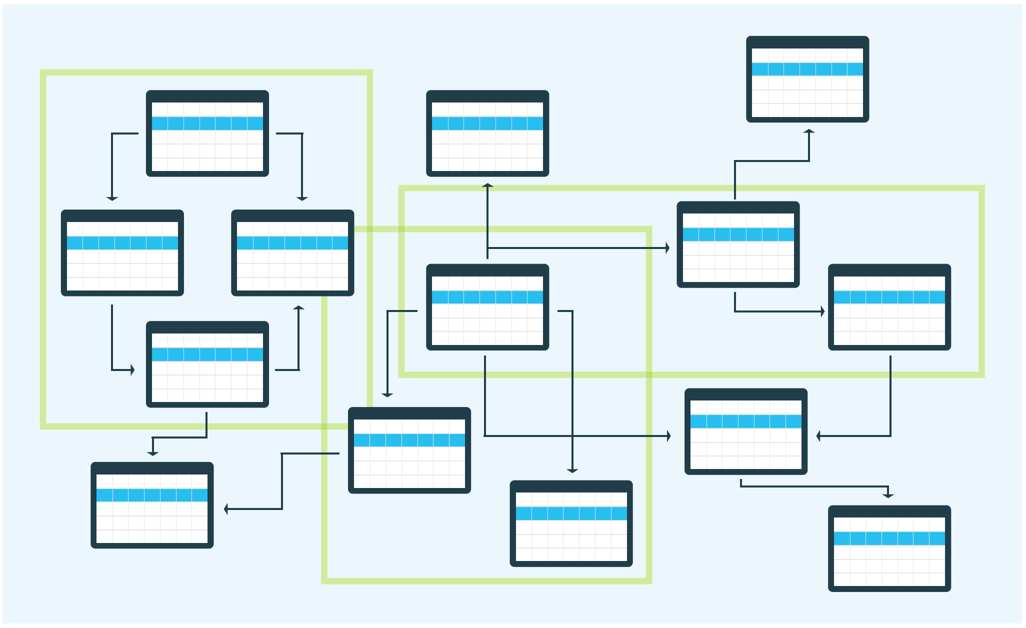
TDG technology allows test data to be generated instantly, with predefined attributes and based on the organization’s data model. With TDG, test data can be controlled, conditioned and generated in a consistent manner whenever it’s needed. And unlike production data, TDG produces synthetic data that is completely secure, removing the necessity for data masking. Test data provisioning is becoming easier and more automated as the traditional TDM model evolves into a TDG model.

When CI/CD pipelines are combined with test automation tools and TDG technology, the full synergy of test automation is achieved.

GenRocket is a technology leader and thought-leader in this space. **The GenRocket TDG™ platform is seamlessly integrated with Jenkins CI/CD pipelines and Selenium test automation tools enabling the combined automation of software testing and test data generation.**

Together, these complementary technologies help organizations realize the full potential of automation with fully integrated platforms that accelerate software testing to keep pace with the shorter release cycles of software development.

Production Data Subsetting – No Longer Required



By its nature, a data subset is a sample of application inputs taken from a production database. There is little control over the nature of the data and testers may have to manually review the data to assess its value to the test case. While it provides a realistic representation of application data, it does not represent all of the potential edge cases or provide a vehicle for negative testing. It is what it is - realistic, but uncontrolled test data that must be masked to remove sensitive information prior to its use for testing code.

Why not generate *real-time synthetic test data* in any pattern imaginable to provide greater coverage of all data permutations and fully exercise your code under all possible scenarios. It's faster, easier, and more secure than subsetting production data and eliminates the requirement for data masking.

Test Data Refresh – No Longer Required

The principle of test isolation refers to the independence of each test from every other test. However, what often takes place particularly in the context of testing web-based transactions, is the **test data is changed in some way by the application during the testing of a stateful workflow**. This invalidates the use of this data for regression testing by other related test cases. The test data can no longer be used consistently across independently run tests.

Imagine a functional test for credit card transactions that calculates reward points based on the account balance of the card holder. The test case must simulate credit transactions, change account balances and compute reward points dynamically. During test execution, test data is altered by the calculations being tested and must be refreshed for subsequent testing.

View a video showing a TDG solution for dynamic test data.

Data refresh is not a concern when using GenRocket's TDG platform. For each test case that is run, a fresh copy of real-time synthetic test data is generated on-demand for that test. Test data sourced via GenRocket TDG appears dynamically when it's needed by the test case and disappears when the test operation is complete. The next time the test is run, the data reappears in the exact state required.

Appears

Ms. Tereasa F. Saldana	001-01-0001
Mr. Everette Q. Groom II	001-01-0002
Mr. Jules U. Hackney Jr.	001-01-0003
Mrs. Kristina J. Brick	001-01-0004
Mr. Francisco M. Grimes II	001-01-0005
Dr. Iona D. Starrett	001-01-0006
Ms. Patricia O. Ingraham III	001-01-0007
Ms. Tracee M. Farah	001-01-0008
Mr. Alva I. Ziegler Jr.	001-01-0009
Dr. Mike T. Youngblood II	001-01-0010

Disappears

Reappears

Ms. Tereasa F. Saldana	001-01-0001
Mr. Everette Q. Groom II	001-01-0002
Mr. Jules U. Hackney Jr.	001-01-0003
Mrs. Kristina J. Brick	001-01-0004
Mr. Francisco M. Grimes II	001-01-0005
Dr. Iona D. Starrett	001-01-0006
Ms. Patricia O. Ingraham III	001-01-0007
Ms. Tracee M. Farah	001-01-0008
Mr. Alva I. Ziegler Jr.	001-01-0009
Dr. Mike T. Youngblood II	001-01-0010

Additionally, test data can become "stale" over time. Data structures can change for user accounts, business transactions, product inventory, or other data elements. Test data must be refreshed to reflect these changes.

Whenever a GenRocket test data scenario is invoked by an automated test script, a fresh copy of test data, in its original state, is generated for that test. Any changes to the data model will be automatically incorporated and the TDG engine will generate new test data exactly matching the revised model.

Secure Test Data - No more PII

Many organizations are mandating the elimination of *Personally Identifiable Information* (PII) from testing to avoid the risk of a data breach and a potentially enormous cost in terms of liability, lost revenue and the penalties for non-compliance with privacy regulations. Organizations in retail, healthcare, banking and insurance are especially vulnerable to these data privacy risk factors.

By its nature, synthetic test data does not contain PII. All user account data, credit information, transaction values, bank balances, healthcare records or other forms of sensitive information are artificially generated by the TDG engine. From a data security perspective, testing with data generated by GenRocket TDG is risk-free.

Centralized Provisioning – No Longer Required

GenRocket TDG changes the way test data is provisioned from a centralized service model to a distributed self-service model. With traditional TDM, once test data requirements have been defined, a production data subset is requested. Once obtained, sensitive data must be masked and additional data may be added manually to increase the volume and variety of data patterns. Production test data must be refreshed on a regular basis to ensure consistency and accuracy.

With GenRocket TDG, the process is simplified. Test data scenarios are written to specify the volume and variety of data needed to maximize coverage. As tests execute, synthetic test data is generated in real-time by test scripts that call for test data using APIs. Any tester can use or modify an existing test data scenario to jumpstart the provisioning process.

Traditional TDM	➡ GenRocket TDG
Centralized Provisioning	Self-Service Provisioning
Test data requirements defined	Test data scenario created
Data requested from production	Tester runs test
Production copies data subset	Test data is generated
TDM operator masks data	
Data is added to increase coverage	
Tester runs test	
TDM operator refreshes data	

Quality at the Speed of Development

In real-world software testing environments, GenRocket TDG has been shown to accelerate the test data provisioning process by more than 1000%. The time to source the data is greatly reduced and the time to generate quality and controlled test data at scale occurs on-demand and in real-time at over 10,000 rows per second. Here are some benchmarking examples of test data acceleration taken from actual QA departments at GenRocket customers in various industries.

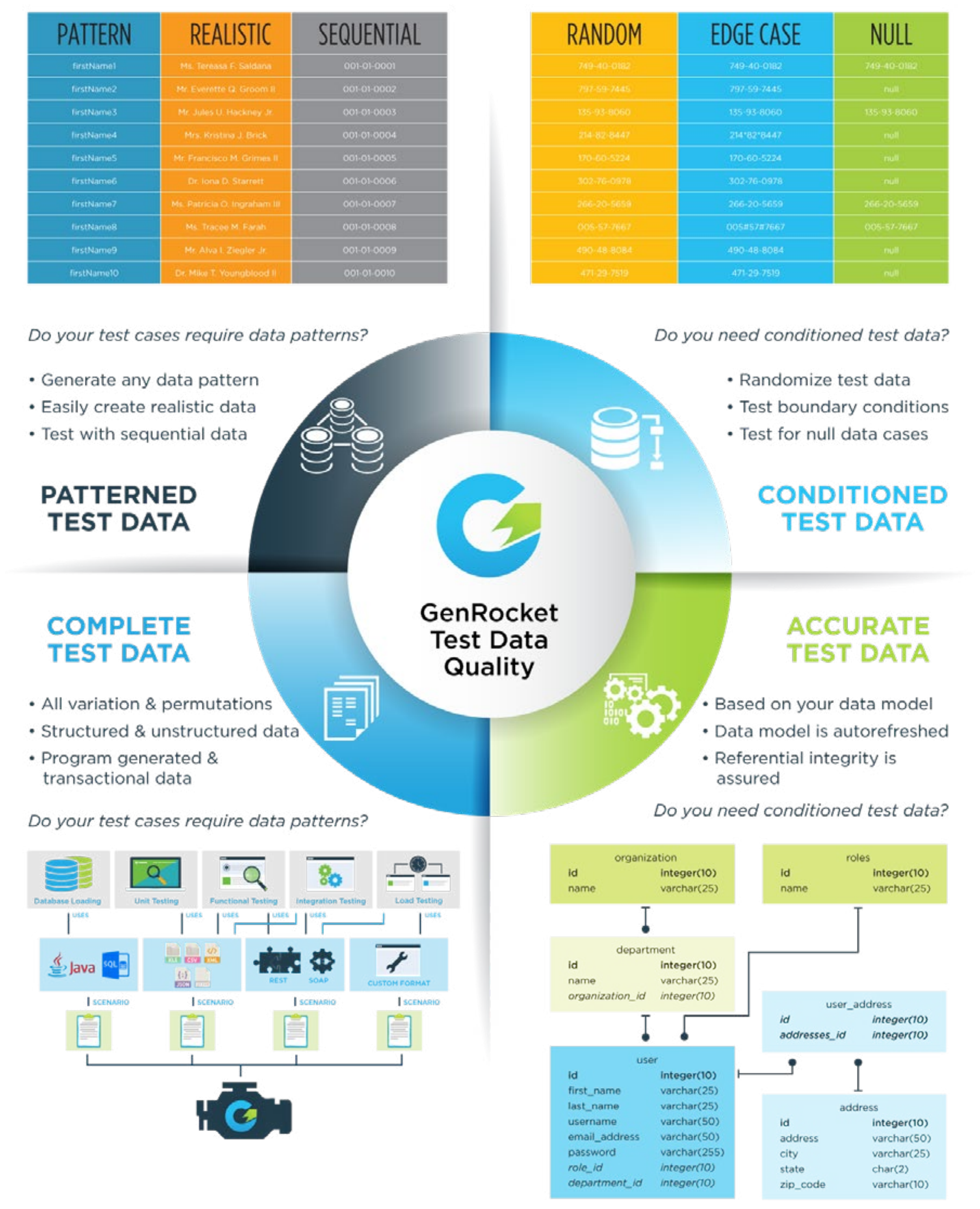
DOMAIN	SCENARIO	VOLUME OF DATA	# OF ELEMENTS / COLUMNS	TIME TAKEN WITH TRADITIONAL APPROACH	TIME TAKEN WITH GENROCKET
Finance	Real-time synthetic data replacement	50,000	2 databases 15 tables 110 columns	~ 16 hours	~ 20 minutes
Retail	Data generation for functional and performance testing of e-commerce	1,000,000	2 databases 30 tables 200 columns	~ 30 hours	~ 15 minutes
Health Care	Data generation for integration testing	500,000	2 databases 28 tables 105 columns	~ 24 hours	~ 10 minutes

Test Data Generation is quickly becoming an accepted approach for keeping pace with the speed of software development in the age of DevOps and continuous delivery.

TDM is evolving to TDG to simplify and accelerate the process while manual test data creation is giving way to a greater use of automation.

3 THE IMPORTANCE OF TEST DATA QUALITY

At GenRocket, we believe the quality of the testing process is only as good as the quality of the test data being used for those tests. When generating test data for your test automation framework, it's just as important to generate data with a very high standard of quality as it is to provision it rapidly. The diagram below illustrates the four dimensions of test data quality delivered by GenRocket's Test Data Generation platform:



PATTERN	REALISTIC	SEQUENTIAL
firstName1	Ms. Tereasa F. Saldana	001-01-0001
firstName2	Mr. Everette Q. Groom II	001-01-0002
firstName3	Mr. Jules U. Hackney Jr.	001-01-0003
firstName4	Mrs. Kristina J. Brick	001-01-0004
firstName5	Mr. Francisco M. Grimes II	001-01-0005
firstName6	Dr. Iona D. Starrett	001-01-0006
firstName7	Ms. Patricia O. Ingraham III	001-01-0007
firstName8	Ms. Tracee M. Farah	001-01-0008
firstName9	Mr. Alva I. Ziegler Jr.	001-01-0009
firstName10	Dr. Mike T. Youngblood II	001-01-0010

RANDOM	EDGE CASE	NULL
749-40-0182	749-40-0182	749-40-0182
797-59-7445	797-59-7445	null
135-93-8060	135-93-8060	135-93-8060
214-82-8447	214*82*8447	null
170-60-5224	170-60-5224	null
302-76-0978	302-76-0978	null
266-20-5659	266-20-5659	266-20-5659
005-57-7667	005#57#7667	005-57-7667
490-48-8084	490-48-8084	null
471-29-7519	471-29-7519	null

Do your test cases require data patterns?

- Generate any data pattern
- Easily create realistic data
- Test with sequential data

PATTERNED TEST DATA

Do you need conditioned test data?

- Randomize test data
- Test boundary conditions
- Test for null data cases

CONDITIONED TEST DATA

GenRocket Test Data Quality

COMPLETE TEST DATA

- All variation & permutations
- Structured & unstructured data
- Program generated & transactional data

ACCURATE TEST DATA

- Based on your data model
- Data model is autorefreshed
- Referential integrity is assured

Do your test cases require data patterns?

Do you need conditioned test data?

organization	
id	integer(10)
name	varchar(25)

roles	
id	integer(10)
name	varchar(25)

department	
id	integer(10)
name	varchar(25)
organization_id	*integer(10)*

user_address	
id	*integer(10)*
addresses_id	*integer(10)*

user	
id	integer(10)
first_name	varchar(25)
last_name	varchar(25)
username	varchar(50)
email_address	varchar(50)
password	varchar(255)
role_id	*integer(10)*
department_id	*integer(10)*

address	
id	integer(10)
address	varchar(50)
city	varchar(25)
state	char(2)
zip_code	varchar(10)

PATTERNED TEST DATA

The GenRocket Test Data Generation platform provides hundreds of test data generators that can be combined to create test datasets with any pattern of data you can imagine. Patterned test data can methodically reproduce a sequence of data variations for a given input field or set of fields making it very useful for integration testing and load testing.

Here are some typical examples:

- An ordered list of user account names with a mix of alpha-numeric characters
- A realistic list of names and addresses for multiple countries and languages
- A sequential list of formatted social security numbers for any range of values

Simply create and run a *Test Data Scenario*, which is a small instruction set used by the GenRocket engine to generate the desired test data in real-time.

CONDITIONED TEST DATA

Perhaps your test data needs to represent a specific set of conditions to find anomalies associated with boundary conditions. Or your test case may be designed for negative testing to detect errors associated with invalid or missing data. TDG allows comprehensive testing by providing conditioned test data to cover a wide variety of use cases.

Here are some typical examples:

- Randomized test data to detect unexpected outcomes from the code
- Edge case data to fully exercise the code for all boundary conditions
- Null test data to test an application's response to missing data inputs

GenRocket's TDG platform allows testers to define their own test data requirements and generate conditioned test data in real-time and on-demand.

COMPLETE TEST DATA

The best way to maximize code coverage is to maximize the completeness of the test data. With GenRocket TDG, testers can generate many more variations of test data than can possibly be found in a production data subset. Permutation testing, for example, is useful for finding false positives in applications with an extensive use of calculations that produce statistical results.

Financial service applications that handle a high-volume of transactions must be rigorously tested with a large-scale transaction history containing past or future dates. They also require the ability to blend synthetic data with program data to accurately test transaction outcomes.

Another example is a machine learning algorithm that must be trained and validated using a large test dataset that represents real-world information with a variety of values in structured or unstructured formats.

GenRocket TDG allows full control over generating:

- All variations and permutations of data
- Conditions, percentages, edge cases and null data
- All variations of historical and future data
- Multiple formats for data interface testing
- High volume data for load and performance testing

ACCURATE TEST DATA

Accuracy is a measure of the precision of the data, the validity for how faithfully it represents the database and the timeliness for how often the data is refreshed and updated. GenRocket TDG generates test data based on the data model for the target application database. When the data model changes, GenRocket TDG automatically updates its test data scenarios to represent the most current version of the data model.

GenRocket TDG also maintains *referential integrity* for all parent-child-sibling relationships among data tables throughout the TDG process. The more complex the application and its database structure, the more value QA professionals will realize from GenRocket's ability to ensure referential integrity.

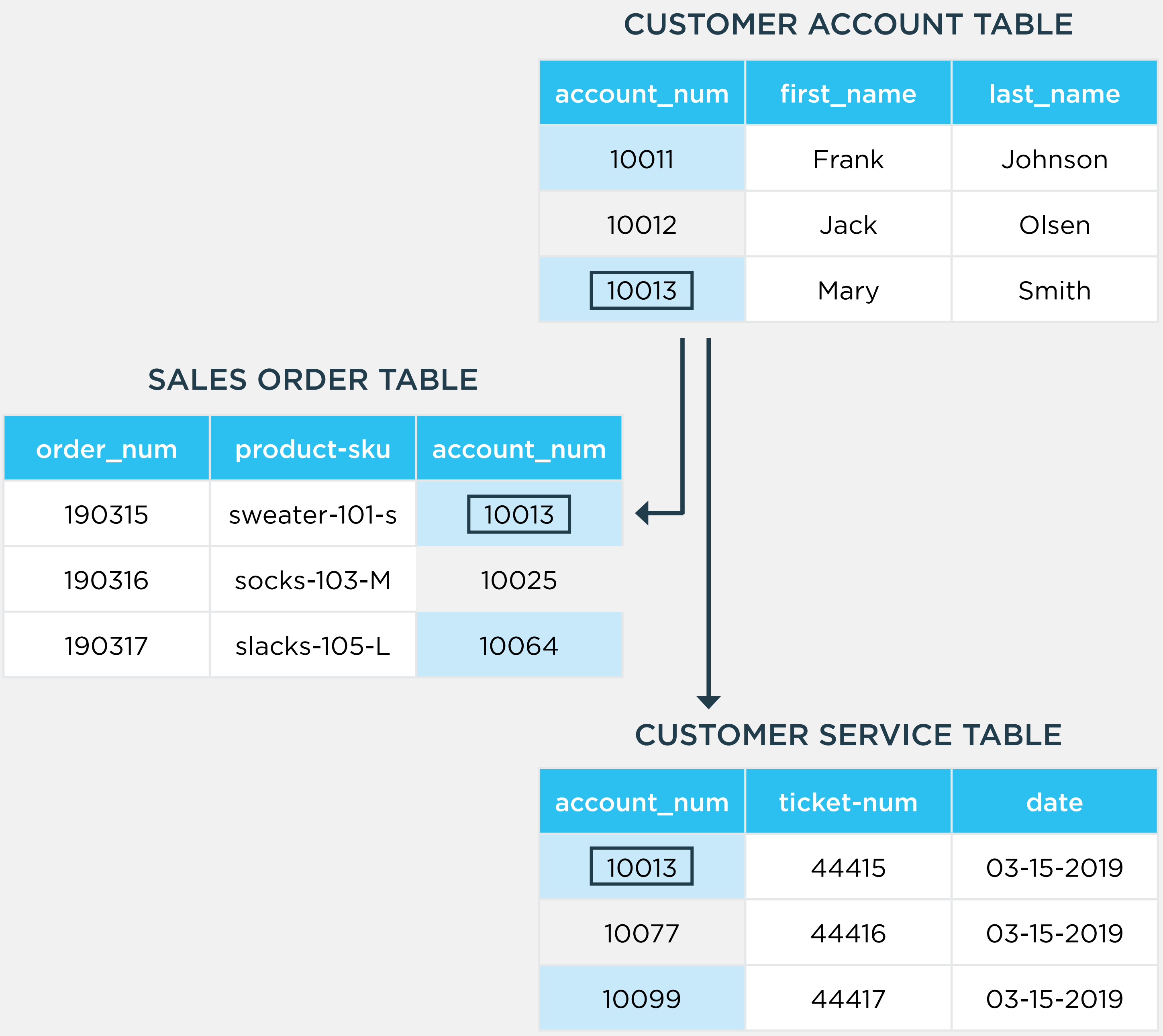
- GenRocket's TDG engine generates test data based on your data model
- All test data is refreshed based on the most current version of the data model
- Parent-child-sibling relationships are always maintained

The Importance of Referential Integrity

Referential Integrity is a term used to describe the relationship between tables in a database and the importance of preserving the links between primary and foreign key fields that connect them together in a logical arrangement. A primary key is a field in a parent data table that links to a foreign key in a child or sibling data table.

For example, if the relationship between a customer account entry (parent) and related entries in a sales order table (child) and a customer service table (sibling) are not maintained, then changing or deleting one row can result in orphaned rows of data in other tables.

In a production environment, loss of referential integrity between data tables can result in orders not being fulfilled, erroneous charges on customer billing statements or sales returns not being processed.



CUSTOMER ACCOUNT TABLE

account_num	first_name	last_name
10011	Frank	Johnson
10012	Jack	Olsen
10013	Mary	Smith

SALES ORDER TABLE

order_num	product-sku	account_num
190315	sweater-101-s	10013
190316	socks-103-M	10025
190317	slacks-105-L	10064

CUSTOMER SERVICE TABLE

account_num	ticket-num	date
10013	44415	03-15-2019
10077	44416	03-15-2019
10099	44417	03-15-2019

To be effective, software testing must use test data that preserves referential integrity between parent-child-sibling data table relationships to validate code that processes database transactions. Otherwise, defects will be missed, or false positives will be flagged during the testing process.

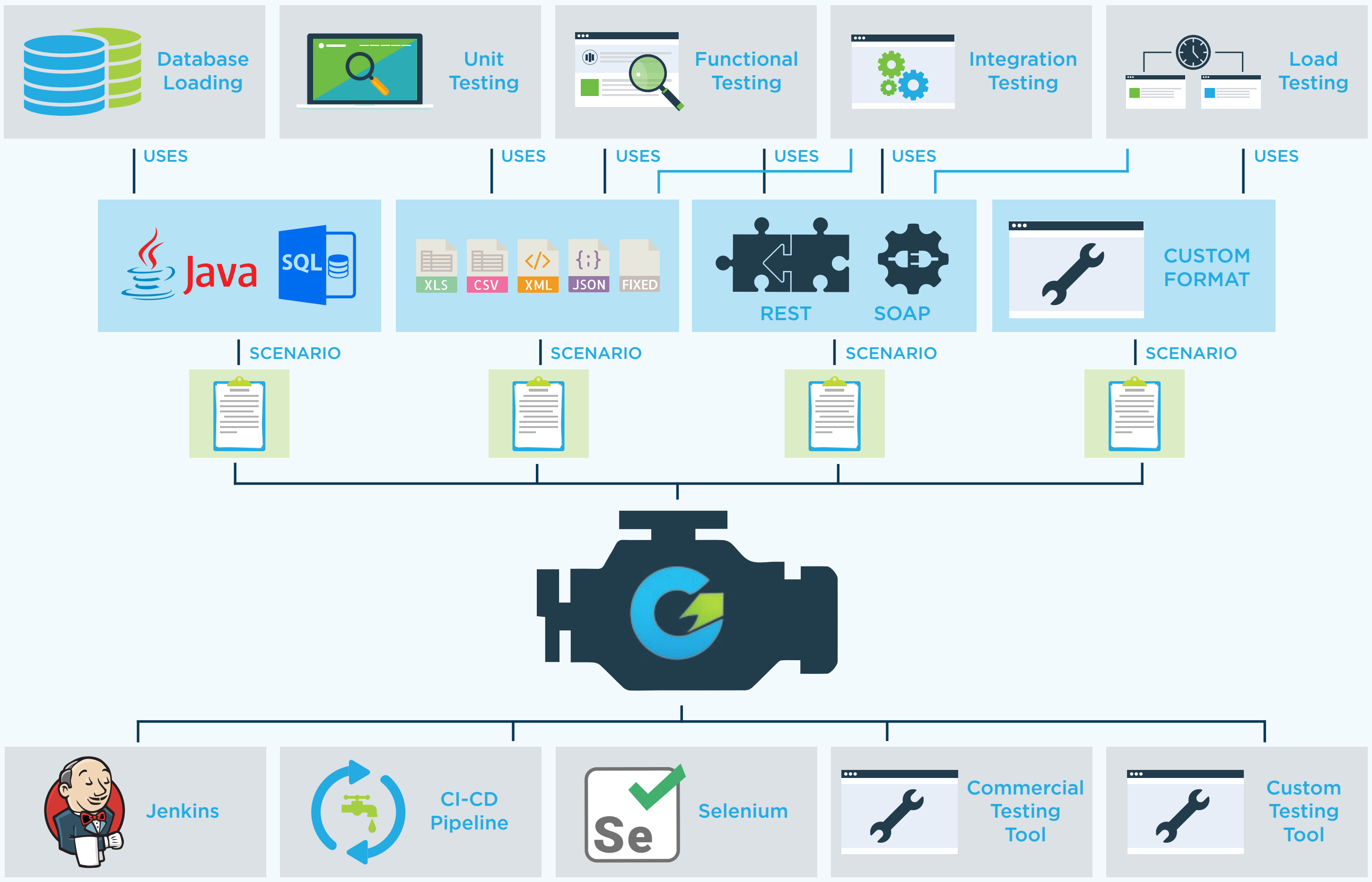
GenRocket TDG holds the only patent for maintaining referential integrity while generating real-time synthetic test data. It is a critically important element of data quality that enables QA teams to perform rigorous testing to maximize the efficiency of defect identification and as a result, the quality of software released to production. In the next chapter, we discuss ways you can make use of this new level of test data quality by automating the generation of test data within your CI/CD pipeline server and test automation framework.

4 INTEGRATING TDG WITH CI/CD

As previously stated, when CI/CD pipelines are combined with test automation tools and TDG technology, the full value of test automation is achieved. Technology integration is the key to realizing “Test Automation Synergy”. In this chapter, we describe how GenRocket TDG can be easily integrated with CI/CD pipeline servers like Jenkins and test automation frameworks like Selenium.

Let’s start with a high-level overview of the GenRocket platform to understand its modular architecture, its flexible capabilities and how it interfaces with automation tools and technologies like Jenkins and Selenium. As illustrated below, the GenRocket TDG engine supports virtually any testing requirement where quality test data is needed.

The top layer in the diagram represents use cases for various categories of testing. Test cases for performing these operations can request test data from the GenRocket TDG engine in a number of ways including database queries using Java database connectivity, test scripts written for XLS, CSV, XML or JSON files, API calls for REST or SOAP services, or a custom format for an industry-specific application.



For each test case, a *GenRocket Test Data Scenario* is used to control the operation of the TDG engine.

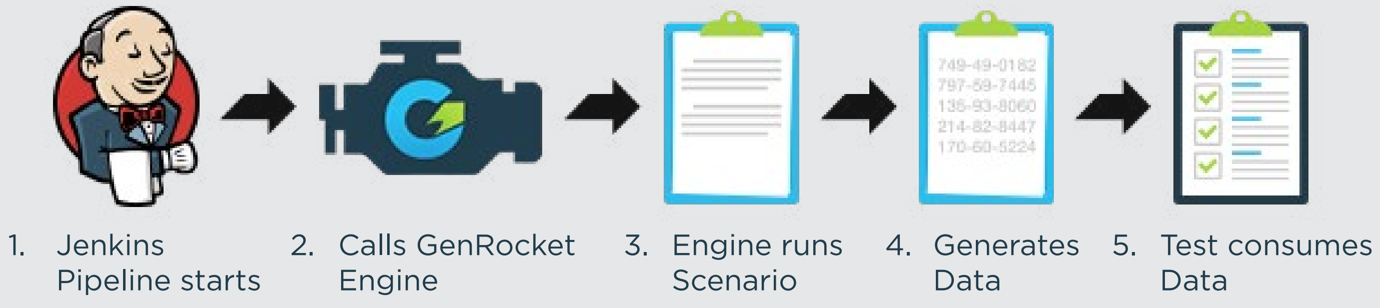
GenRocket Scenarios contain the test data specifications required for a given test case. They define the data structure (e.g., data table fields and their relationships), the attributes of the data (e.g., first name, last name, social security number, etc.), instructions for controlling the volume and variety of data (e.g., a pattern of 1,000 records in sequential order by user ID), and the output data format required (e.g., XML, JSON, CSV, SQL, JDBC, REST, Webservice, etc.).

GenRocket TDG can interface to a wide variety of testing tools and frameworks to fully automate the process of generating test data in real-time and on-demand for virtually any testing operation or application environment.

A Fully Integrated Test Automation Environment

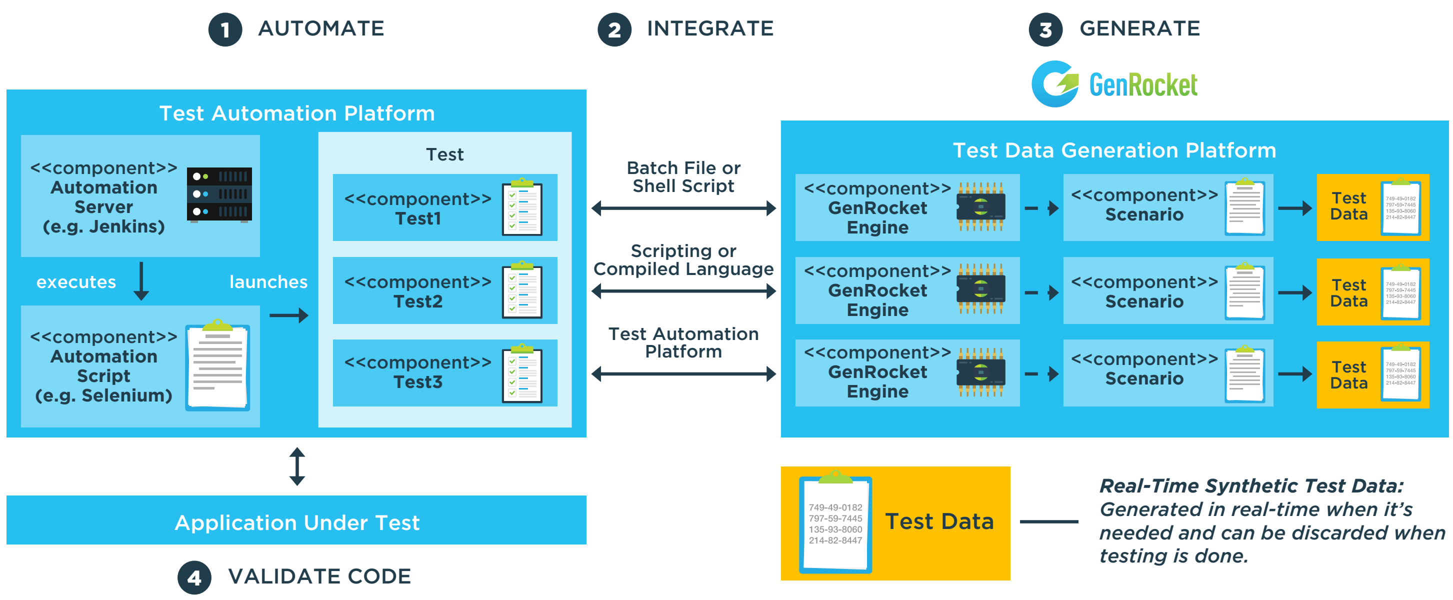
If you are already using Jenkins and Selenium, or a similar test automation framework, building a fully automated test automation environment that includes TDG is easy and straightforward. A simple process flow for using GenRocket TDG in conjunction with the Jenkins CI/CD pipeline is illustrated by the diagram below.

Here's how it works: (1) The Jenkins pipeline starts, (2) and calls the GenRocket engine, (3) which in turn executes a pre-defined test data scenario that (4) generates controlled test data according to an exact specification in real-time so that (5) the test case can immediately consume and process the data at runtime.



An architectural model for this process flow illustrates how each component of the test automation framework can be seamlessly connected to provide a "zero touch" model for continuous testing in a DevOps environment. In the block diagram below, the CI/CD pipeline server with its test automation framework are both represented on the left and the GenRocket TDG platform for on-demand test data provisioning is on the right.

FULLY INTEGRATED TEST AUTOMATION ENVIRONMENT


1 AUTOMATE
2 INTEGRATE
3 GENERATE
GenRocket
Test Automation Platform
<<component>> Automation Server (e.g. Jenkins)
executes
launches
<<component>> Automation Script (e.g. Selenium)
Test
<<component>> Test1
<<component>> Test2
<<component>> Test3
Batch File or Shell Script
Scripting or Compiled Language
Test Automation Platform
Test Data Generation Platform
<<component>> GenRocket Engine
<<component>> Scenario
Test Data
Application Under Test
4 VALIDATE CODE
Test Data
Real-Time Synthetic Test Data: Generated in real-time when it's needed and can be discarded when testing is done.

The methodology for using these platforms as an integrated environment for continuous testing can be summarized by these 4 simple stages:

1. **Automate Testing:**
   Define the test cases and test data requirements for the application in order to perform functional testing, regression testing, performance testing, etc.

2 **Integrate Platforms:**
   A batch file or shell script can invoke a test data scenario in the GenRocket TDG engine or an API can call for test data via scripting or compiled language.

3. **Generate Test Data:**
   GenRocket TDG will generate test data as the test executes using pre-determined test data specifications and in the output format required.

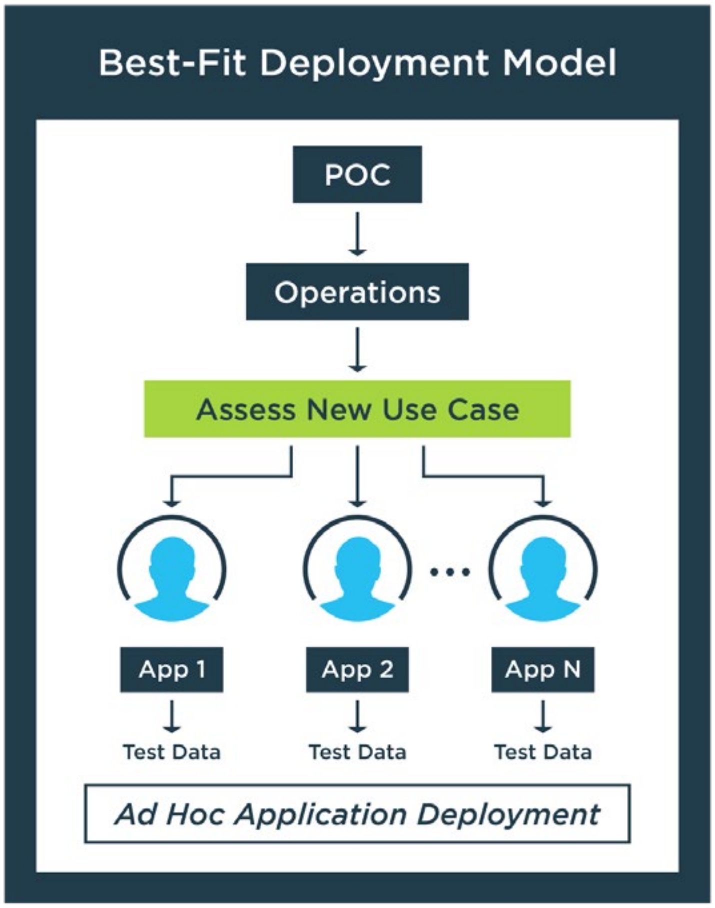
4. **Validate Code:**
   Your application can now be continuously tested with simple functional testing or comprehensive integration testing to identify and remedy defects before they reach production.

Together, these tools and technologies comprise a fully integrated environment that enables full scale deployment of test automation across the QA organization, throughout the continuous delivery development cycle.

GenRocket TDG Deployment Models

The adoption and deployment of any new technology can take time and no single approach will work for every organization. However, at GenRocket we have seen two models for deployment used by our customers. Both have proven to be effective for accelerating their testing processes while increasing the quality of test operations.

The Best Fit Deployment Model



"Best-Fit" deployment introduces TDG one test case at a time to realize the greatest impact on operational efficiency. Applications are assessed to determine their test data requirements and the need for continuous testing to keep pace with development. After each test case is created and the test data specified, testing is automated using the integration framework described above.

Deployment starts with a *Proof of Concept* (POC) to introduce GenRocket technology into the organization and prove-in the use of GenRocket TDG for meeting test data requirements. A POC typically involves 1 to 3 use cases with appropriate test data scenarios to get experience with the technology.

After the first test case is automated for testing with TDG, additional test cases are assessed one-by-one using GenRocket's guidelines for TDG deployment.

Guidelines for assessing best-fit use cases are documented in *Test Data Generation Use Cases*, a GenRocket publication that shares deployment experiences from our most effective TDG customer engagements.

The document provides a useful checklist for identifying the applications and test cases that will derive immediate benefit from TDG.

This approach provides your QA team with "quick wins" to demonstrate the value of GenRocket TDG and justify its continued deployment across the organization.

Best-Fit TDG Deployment

Test Data Generation Deployment Checklist			
TDG Guidelines	**Applications**	**Test Operations**	**Data Source**
1. Fast Delivery	*Application Name*	*Test Cases*	☐ TDG
2. Data Privacy	*Application Name*	*Test Cases*	☐ TDG
3. Boundary Conditions	*Application Name*	*Test Cases*	☐ TDG
4. Blended Test Data	*Application Name*	*Test Cases*	☐ TDG
5. Volume Data	*Application Name*	*Test Cases*	☐ TDG
6. Data Refresh	*Application Name*	*Test Cases*	☐ TDG
7. Special Data Formats	*Application Name*	*Test Cases*	☐ TDG
8. New Applications	*Application Name*	*Test Cases*	☐ TDG
9. Dynamic Test Data	*Application Name*	*Test Cases*	☐ TDG
10. CI/CD Pipelines	*Application Name*	*Test Cases*	☐ TDG

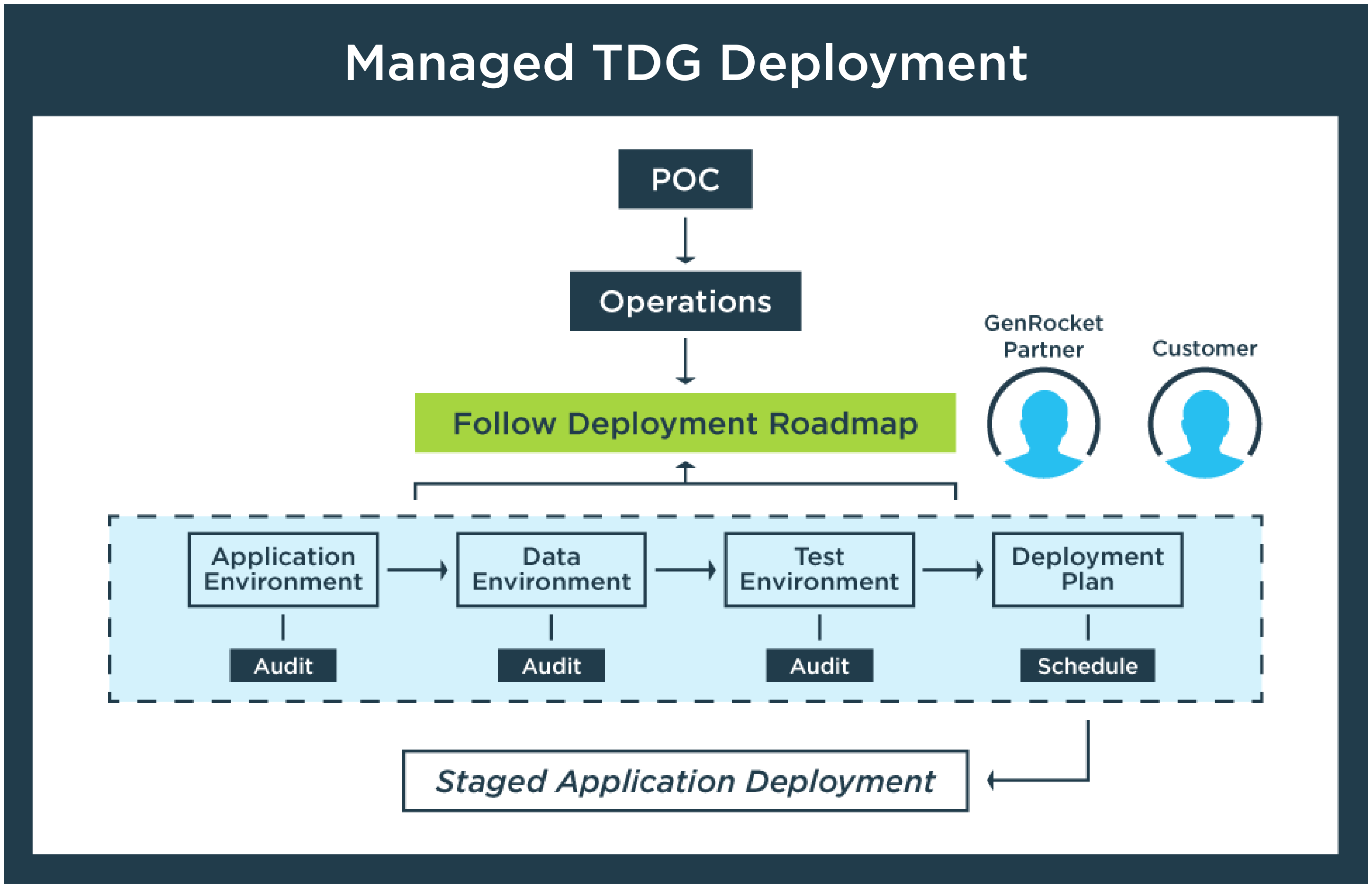
The Managed Deployment Model

Managed Deployment, a more structured approach for GenRocket TDG deployment, is based on proven best practices and assisted by a *GenRocket Certified Partner.* Like the Best Fit Deployment model, this model begins with a POC to prove-in TDG technology with 1 to 3 use cases that represent the most pressing QA challenges.

After successfully completing a POC, the GenRocket Partner will facilitate the TDG deployment process by following a Deployment Roadmap.

A deployment team will be established to audit the *Application Environment,* the *Data Environment*, and the *Test Environment* before building out the *Deployment Plan.*

Based on that plan, the deployment team will methodically roll-out the use of TDG technology for application testing in a staged approach. This approach minimizes disruption as the QA team gains proficiency with GenRocket TDG and ensures a successful roll-out.



A Managed TDG Deployment Roadmap includes analysis of the following:

- **APPLICATION ENVIRONMENT**
  A written audit of the target applications and infrastructure
- **DATA ENVIRONMENT**
  The information flow and data dependencies between systems
- **TESTING ENVIRONMENT**
  The test data tools that are in use and their integration requirements
- **TDG DEPLOYMENT**
  Staged and prioritized deployment of the TDG provisioning processes

TDG Deployment Roadmap

APPLICATION ENVIRONMENT

Understand the applications and infrastructure that supports them and underlying databases
Deliverable > ***Applications Audit:*** Identifies applications and infrastructure.

DATA ENVIRONMENT

Understand the information flow through the application environment and the data.
Deliverable > ***Data Audit:*** Identifies data flow and data dependencies.

TESTING ENVIRONMENT

The current testing resources, test processes, testing tools and data provisioning processes.
Deliverable > ***Test Audit:*** Test data tool and framework integrations.

TDG DEPLOYMENT

The TDG deployment plan for the organization.
Deliverable > ***TDG Deployment Plan:*** Prioritized plan for application testing, revised processes for test data provisioning, data refresh and test data integration with tools and frameworks.

The deployment team will be composed of the following roles and responsibilities:

Project Manager – Oversees and manages the TDG implementation project.

Data Architect – Understands the data model and the relationships between the applications and databases.

Solution Architect – Understands the test data requirements for testing applications and sets up a project in GenRocket's TDG platform to meet those needs.

Data Specialist / Testers – Uses prebuilt test data generation scenarios, or builds and modifies new scenarios, to generate test data for a variety of test cases.

The *Managed Deployment* process can be implemented across very large and sophisticated testing organizations in a matter of weeks. Operational efficiencies will be realized almost immediately as provisioning cycles are drastically reduced and overall testing velocity accelerates from use of automation.

In the next chapter, we will dig deeper into the many use cases for GenRocket TDG for all areas of software testing as we explore strategies that will maximize the ROI on your investment in test automation and TDG technologies.

5 MAXIMIZING TEST AUTOMATION ROI

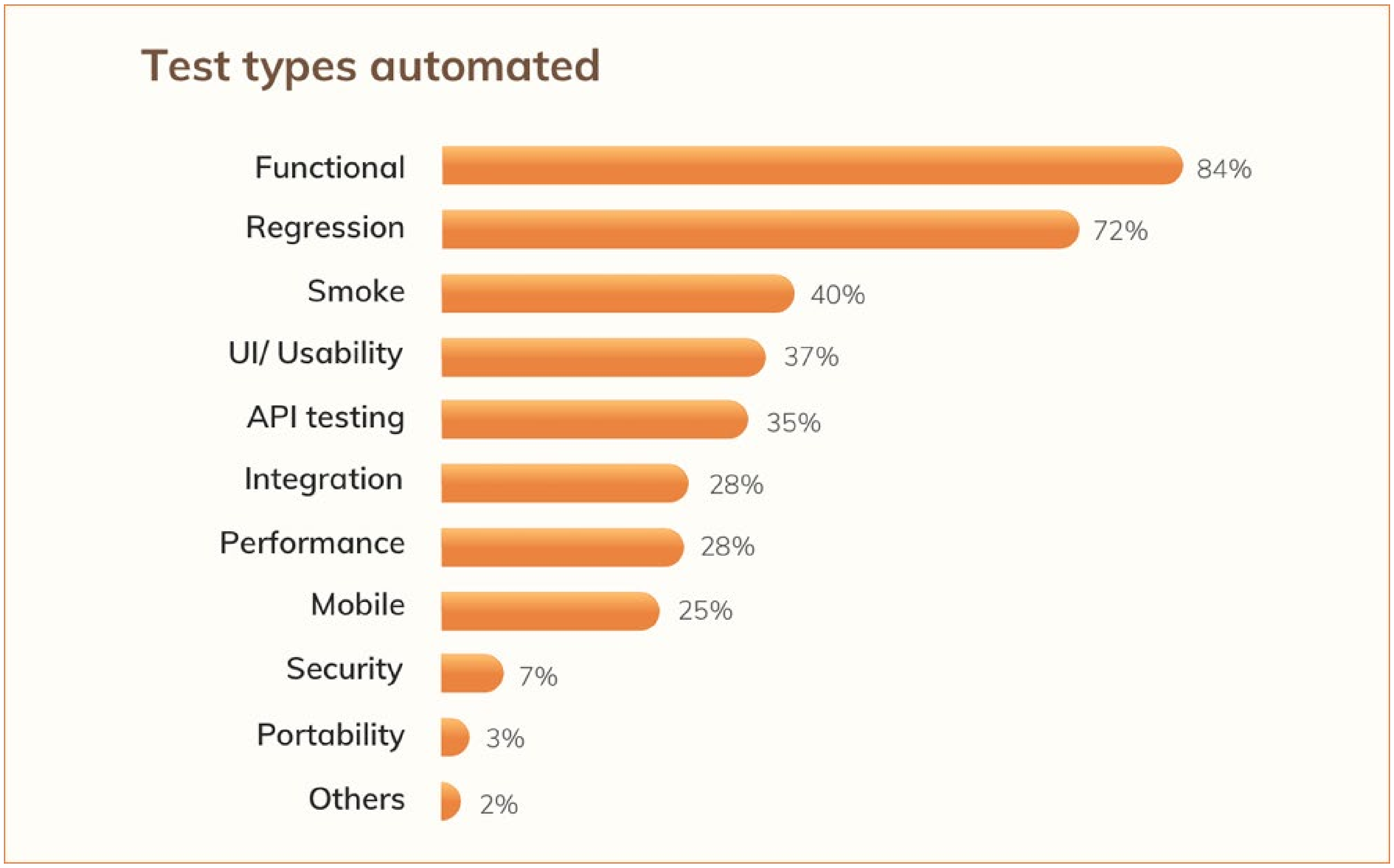
Test Data Generation (TDG) is the perfect companion technology for test automation. By their nature, test automation tools require that test data be provisioned at high velocity. With GenRocket TDG, provisioning speed is not a problem because test data can be generated instantly when the automated tests run.

With the help of TDG, realizing the full benefit of continuous testing can be a very achievable goal. However, transitioning from traditional test data provisioning to TDG can be perplexing. For many QA professionals, it's something of a "where do I begin" dilemma. It's a classic change management challenge to overcome.

Change is always difficult, and the inertia of current practices can be hard to break. **One of the principles of successful change management is to identify *quick wins* that establish early success and reinforce the team's commitment to a new direction.** Quick wins combine achievable outcomes and measurable benefits with the use of best practices that can be used and expanded over time.

Selecting quick wins for GenRocket TDG deployment should be guided by your test automation deployment. Which tests you choose to automate first will have a profound effect on productivity gains. Because some categories of testing provide greater ROI than others in terms of time and cost savings, it's best to identify areas of testing where automation and test data generation will have the greatest impact.

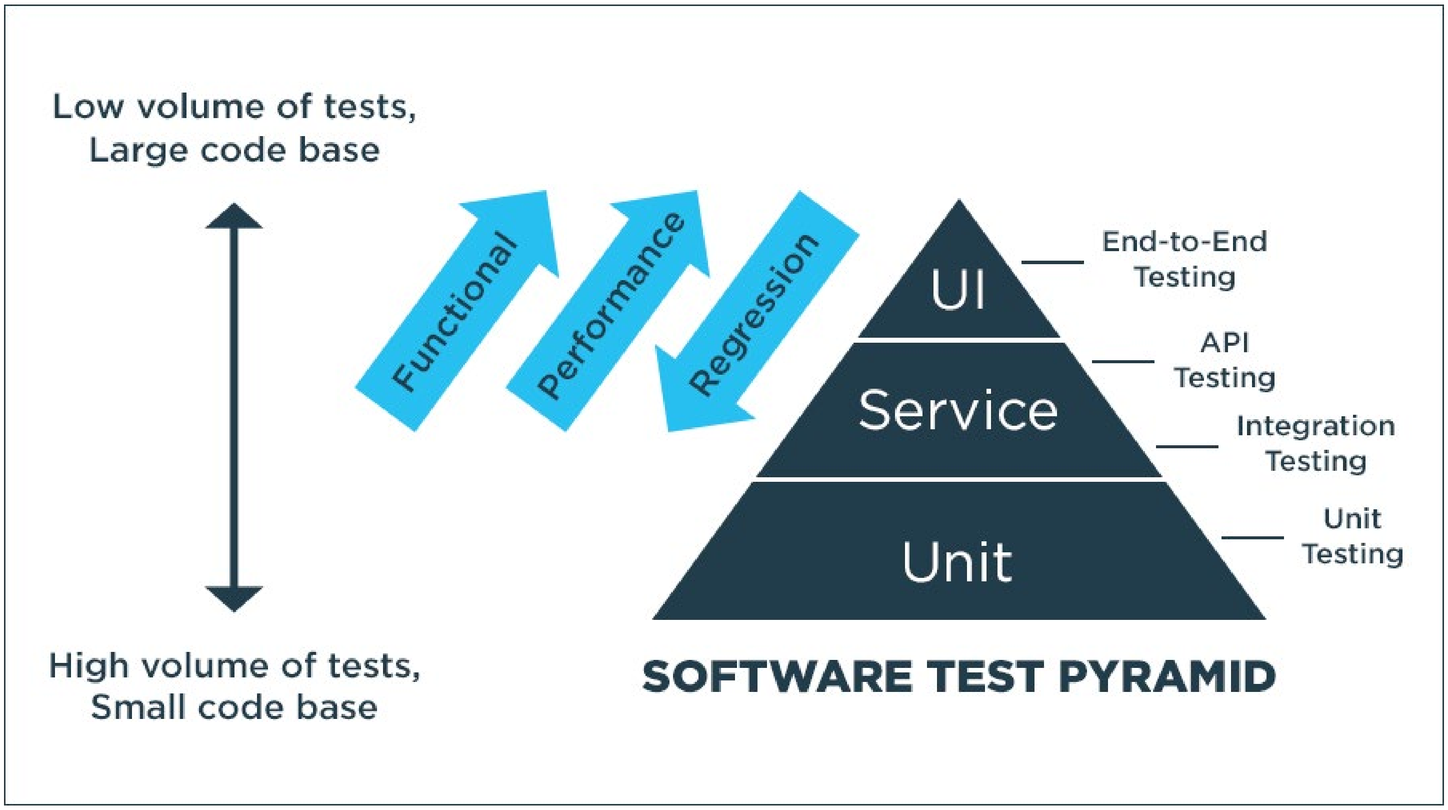
According to a survey of 2,291 QA professionals conducted by ToolsQA, **functional and regression testing are the most common areas for the application of test automation.**



Functional testing is an essential role performed by QA to verify the system or application fully meets the requirements of its functional specification.

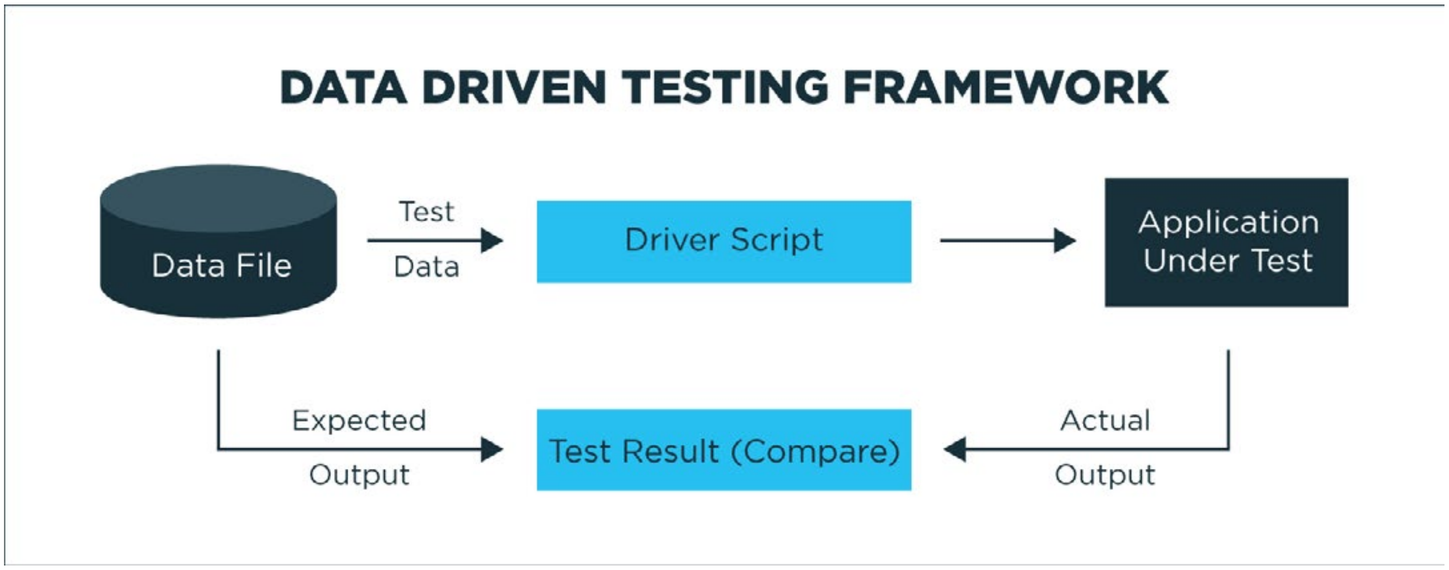
Non-functional testing exercises the performance, reliability, scalability, compatibility, security, and other non-functional aspects of the software.

The *Software Test Pyramid* provides a useful way to look at the operational nature of testing. Lower on the pyramid are tests performed at high volume on a smaller code base. Tests that are higher on the pyramid are performed at lower volume on a larger code base.



Functional testing can be performed at all layers of the software test pyramid - *Unit Testing, Service Testing and User Interface Testing*. Functional tests performed at the unit testing layer, often referred to as *white box testing*, are often performed by developers who examine source code internals to ensure they conform to the business logic the software is meant to execute.

Functional testing performed at the *Service and UI layers* is often referred to as *black box testing* and requires no knowledge of internal code structure or logic. The concept is to provide inputs to a "black box" and compare actual results with expected results. This kind of functional testing lends itself to the practice of *Data Driven Testing,* a testing approach where an external data source is used to feed the test case with a large number of data variations in a controlled fashion to fully exercise the code. This technique enables positive testing (based on likely, valid and complete input data) and negative testing (based on unlikely, invalid and missing input data).



The middle of the software test pyramid represents a sweet spot for test automation tools integrated with test data generation. Service layer tests for functional, performance and regression testing all share four important attributes that maximize ROI. That's because service layer tests represent tests that:

- Require a high-volume of test cases
- Involve tests that are highly repetitive
- Require no knowledge of source code internals
- Are data-intensive and require controlled test data inputs

Software tests that are performed manually, and on a repeated basis, should be automated first to save time, reduce project cost and apply *Data Driven Testing* principles to maximize coverage and quality.

Therefore, the best types of tests to start deploying GenRocket TDG with automated testing are:

- Data-driven functional testing
- Functional tests repurposed for performance testing
- Functional and performance tests that rerun for regression testing

This approach provides the best path to quick wins because each functional test can be repurposed for other test categories with test data that is based on reusable test data scenarios.

An Example of Test Automation, TDG and Data-Driven Testing

Here is a simple example that illustrates the power of test automation, TDG and data driven testing. Let's assume that we want to test a login system having multiple input fields with 1,000 different data sets.

To test this, you can take the following different approaches:

Approach 1) Create 1,000 scripts one for each dataset and run each test separately one by one.

Approach 2) Manually change the value in the test script and run it multiple times.

Approach 3) Import test data from an excel sheet, fetch data from excel rows one by one and execute the script.

Approach 4) Create a Test Data Scenario using GenRocket's TDG platform to replace multiple test scripts and generate 1,000 iterations of patterned, conditioned, controlled test data (or any volume of data needed) at the rate of 10,000 rows per second.

Clearly Approach 4 is the preferred method for test automation. An investment of 10 minutes to create a Test Data Scenario for the TDG platform translates to test execution time of just a few seconds with a test case that can be reused for performance testing and regression testing. Below is an example of the kind of test data that can be defined and generated by the GenRocket TDG. Notice the many variations that are possible and the controlled manner in which the test data can be used by test cases.

Pattern	Realistic	Sequential	Random	Edge Case	Null
firstName1	Ms. Tereasa F. Saldana	001-01-0001	749-40-0182	749-40-0182	749-40-0182
firstName2	Mr. Everette Q. Groom II	001-01-0002	797-59-7445	797-59-7445	null
firstName3	Mr. Jules U. Hackney Jr.	001-01-0003	135-93-8060	135-93-8060	135-93-8060
firstName4	Mrs. Kristina J. Brick	001-01-0004	214-82-8447	214*82*8447	null
firstName5	Mr. Francisco M. Grimes II	001-01-0005	170-60-5224	170-60-5224	null
firstName6	Dr. Iona D. Starrett	001-01-0006	302-76-0978	302-76-0978	null
firstName7	Ms. Patricia O. Ingraham III	001-01-0007	266-20-5659	266-20-5659	266-20-5659
firstName8	Ms. Tracee M. Farah	001-01-0008	005-57-7667	005#57#7667	005-57-7667
firstName9	Mr. Alva I. Ziegler Jr.	001-01-0009	490-48-8084	490-48-8084	null
firstName10	Dr. Mike T. Youngblood II	001-01-0010	471-29-7519	471-29-7519	null

The sample test data in the table above represents just 10 rows of data. Think of the time savings and coverage improvement when 1,000 rows (or more) are generated in a matter of seconds. Now consider the ability to run load, stress, or performance tests with a million rows of data generated by a TDG platform in a matter of minutes.

Combined Functional, Performance and Regression Testing

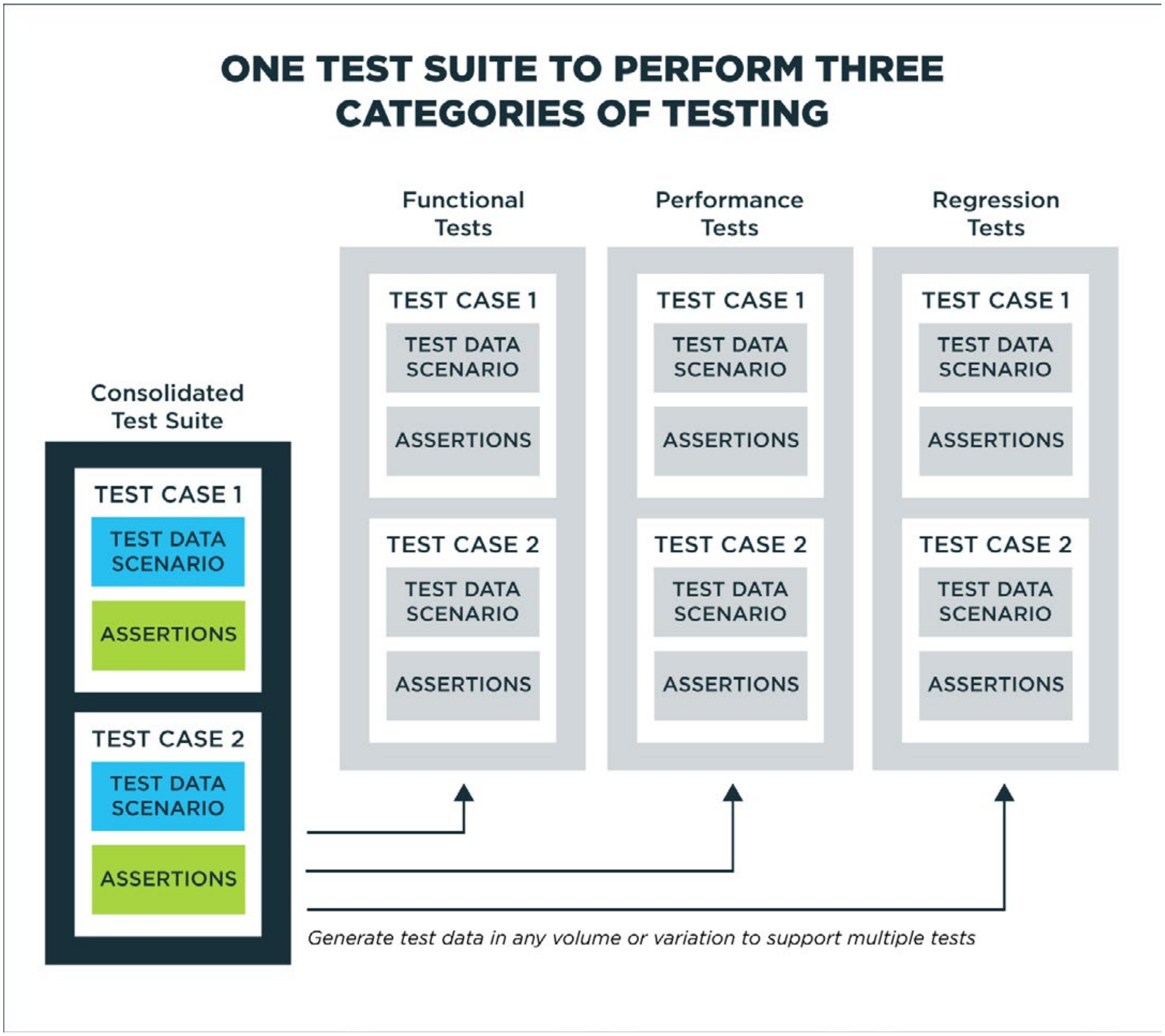
One way to maximize the ROI of your test automation investment and get some quick wins during your transition to TDG is to combine multiple test operations into a single testing process. Start by automating your functional test cases and configure your test data scenarios to maximize coverage. Specify test data for both positive and negative testing purposes. Now include the assertions that validate code functionality and identify defects. Highly controlled test data helps to minimize false negatives (when good code fails an assertion test) and false positives (when bad code passes an assertion test). Both conditions contribute to a higher defect rate and can erode confidence in the QA process.

TDG ushers in a new approach to testing where the design of test data is integral to the design of the test case. **Controlling the nature of the test data used by a given test case is critical for identifying defects, while maximizing test data volume and variation is critical for maximizing coverage.**

As a separate test case, change your assertions to test the performance of the application by increasing its load conditions. With GenRocket TDG, the volume of your test data can be scaled up or down simply by changing the loop count for a given test data scenario. You can vary the load in a structured, methodical way to systematically assess the impact data volume on performance.

Additionally, **you can perform automated regression testing simply by rerunning the functional and performance tests for each new release**. Regression testing must always be performed with consistent test data. With GenRocket TDG, test data is always fresh because it is newly generated for every test. And the identical data (even if it is random data) is generated every time.

The diagram below illustrates the concept of a consolidated test suite that performs functional testing and also performs performance testing by varying the application load with increasing data volume. These tests can be rerun for regression testing.



If your performance testing calls for a variety of test cases, you can still leverage your functional tests by repurposing them, and their test data scenarios, for various types of performance testing such as:

- Capacity Testing: Increasing the number of application users
- Load Testing: Increasing the number of simultaneous transactions
- Volume Testing: Increasing the volume of data handled by the system
- Stress Testing: Testing application behavior at loads beyond capacity
- Soak Testing: Refers to examining the impact of load over time

Regardless of the type of test, the important concept to keep in mind is this:

With TDG, the speed and effectiveness of automated testing is no longer dependent on the availability of test data.

Any volume or variety of data can be configured by a test data scenario and generated in real-time by the TDG engine. Carefully selecting which test procedures to automate first and using the power and flexibility of GenRocket TDG to enable multiple categories of testing is the best way to maximize your test automation ROI.

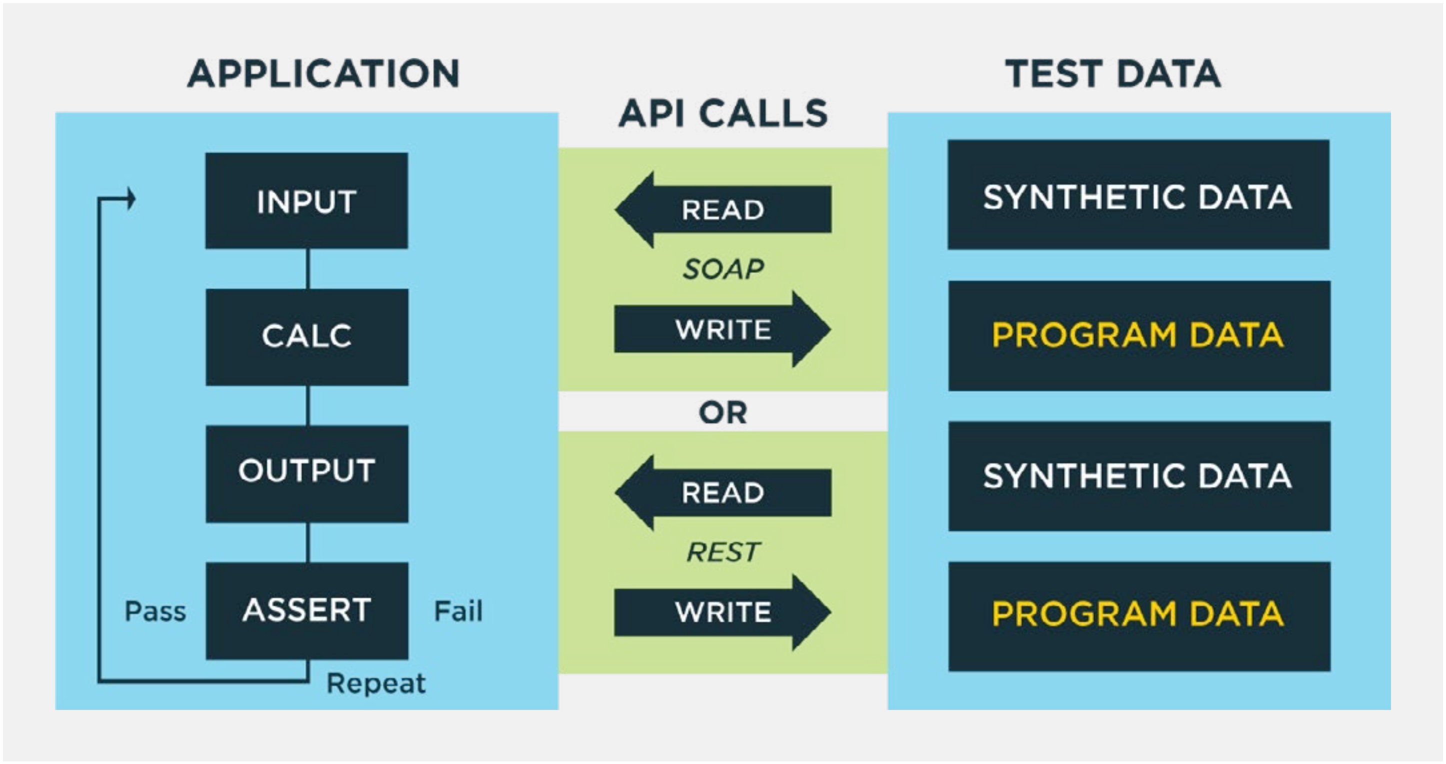
6 INNOVATING TEST DATA SOLUTIONS

In the previous chapter, we described the use of Test Data Generation to support **automated functional testing**, repurposing test cases and test data scenarios for **automated performance testing** and rerunning tests for **automated regression testing**. This is an effective way to create immediate synergy between your test automation tools and your TDG platform.

In this chapter, we explore some additional ways to utilize GenRocket TDG across all of your applications to drive additional operational efficiencies, benefits and value. **The GenRocket platform is adaptable to almost any testing requirement and ways to leverage the power of TDG are limited only by the tester's imagination**. As you read the following test case descriptions, think about your own application requirements and how you might adapt these ideas to innovate new test data solutions for your testing environment.

Testing Workflows with Program-Driven Test Data

In transactional environments, like an airline reservation system or a credit card transaction processing system, the data used by the application during testing can be fluid and subject to multiple program-driven changes. With GenRocket TDG, testers are able to supply test data for any workflow condition by integrating the GenRocket API within the test case and dynamically loading *test data scenarios* to generate the required test data on-demand and in real-time



The GenRocket API allows a test case to dynamically modify the data conditions within the test data scenarios which enables them to dynamically generate synthetic test data that simulates real world production data in real-time.

The API allows testers to have complete control over the entire testing workflow and have the software itself make branching decisions to determine what scenarios to load and dynamically modify how the test data will be generated as the application continues to run.

Here is the basic program workflow using the GenRocket API:

1. Load the first GenRocket *test data scenario*
2. Modify test data scenario conditions
3. Run the test data scenario
4. Test data is generated, then a SOAP or REST request is made, then the response is saved
5. The test script parses the response and makes a branching decision for loading the next scenario
6. Another (or the same) GenRocket test data scenario is selected and loaded
7. The system branches back to Step 2

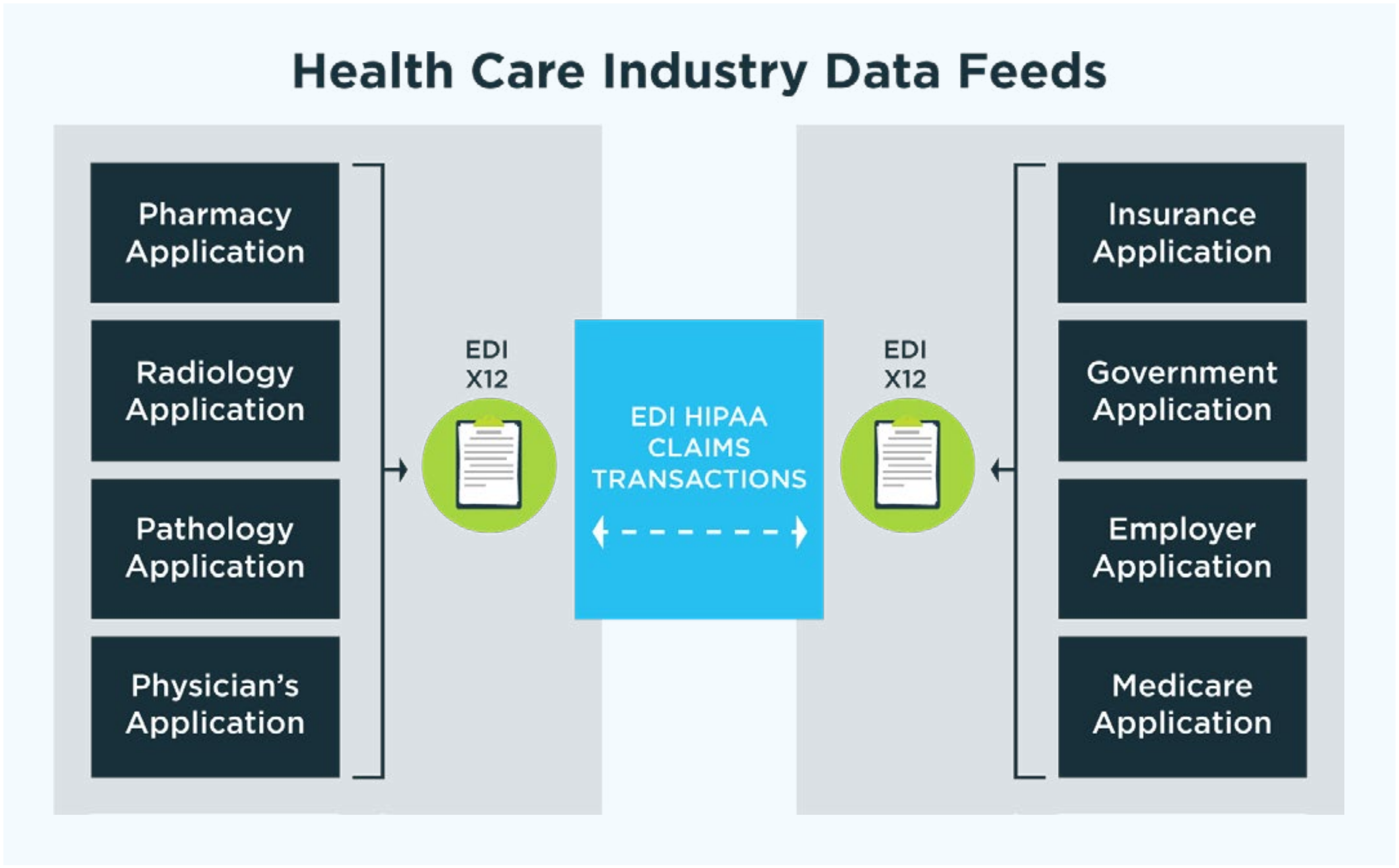
Learn more about testing workflows with program-driven test data in this airlines reservations system use case.

Through the use of GenRocket's API to control program-driven test data, testers can create and automate highly adaptable, intelligent and dynamic test cases for any workflow. The simulation of any real-world scenario with dynamically generated test data allows developers to continuously improve algorithms and code performance while testers catch application defects before code is released to production.

Testing Data Interfaces and Real-Time Data Feeds

Depending on the industry, many QA organizations must test a variety of data interfaces and simulate real-time data feeds used by their applications. Some data feeds are tightly specified by data interchange standards, others are de facto standards or can be proprietary in nature.

For example, the EDI X12 (Electronic Data Interchange) standard widely used in healthcare and other industries, is governed by the Accredited Standard Committee (ASC). It defines the format and data structures for commonly used transactions that are exchanged between business and trading partners. In healthcare, EDI X12 allows online claims processing between insurance companies, healthcare providers, pharmaceutical firms and other medical suppliers or service organizations who must share sensitive patient care information.



In banking and financial services, some data interface standards are governed by trade associations such as NACHA, which defines messaging standards for the movement of electronic bill payments and direct deposits over the ACH network. Other standards have become de facto standards, like the FIX protocol used for the real-time electronic exchange of securities transactions.

Over the years, data interchange standards have evolved into a blend of proprietary formats adopted by industry groups and others that are formalized and controlled by international standards bodies.

This complicates the testing process for applications that rely on data feeds. **Testing data interfaces requires test data that accurately simulates real-world transactions and structures the data in the precise data format used by that application** (e.g., insurance claims, bank deposits, stock trades). The use of production data for this purpose is problematic for the following reasons:

- The use of sensitive customer or patient data puts data privacy and regulatory compliance at risk
- Boundary condition and negative testing is not well-supported by production data subsets
- Manually creating synthetic data for simulating test data feeds is both complicated and tedious

Testing data feeds is a perfect use case for Test Data Generation technology. If the TDG platform is pre-programmed to format data according to a particular data interface specification, then generating controlled, patterned and conditioned test data is simply a matter of defining the attributes of the data fields required by the test case and automating a simulated data feed.

This is precisely how GenRocket's TDG platform operates. There is a wide variety of data interface formats already supported by the GenRocket platform. Many more are on our product roadmap and virtually any data feed, whether it's an industry standard or a proprietary format, can be created as a custom test data receiver in the GenRocket platform.

Here is list of data interface formats that are available now or in development for future release.

- EDI X12
- HL7
- NACHA
- BA12
- FIX
- SWIFT
- Complex XML (XSD-driven)

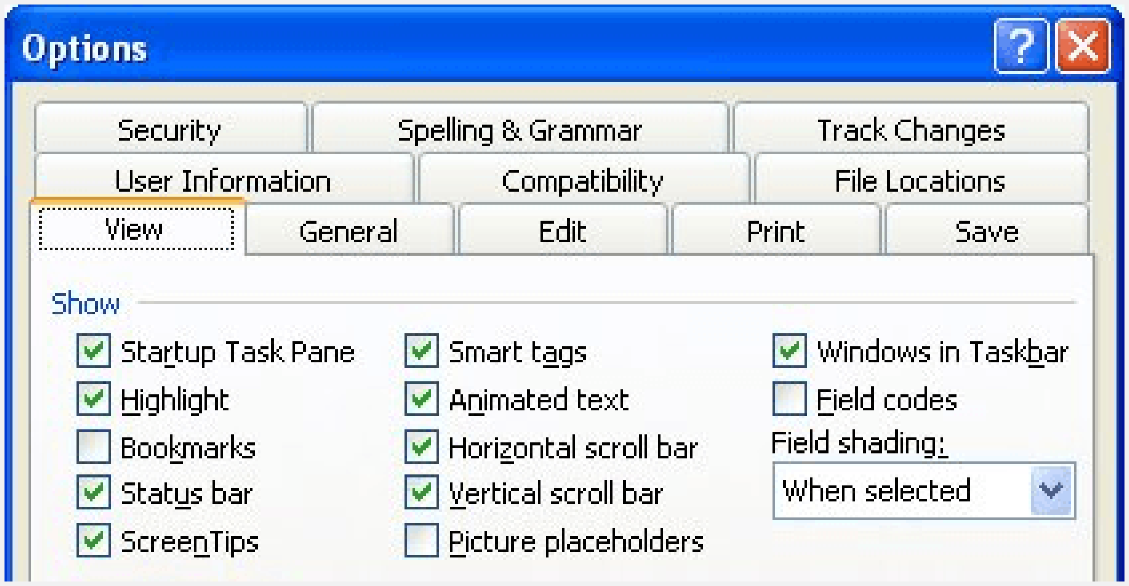
If you would like to learn more about data feed testing using GenRocket's TDG platform, read our blog post, Solving the Test Automation Challenge for EDI Data Feeds and download our case study, Test Data Generation for Healthcare Markets.

Streamlining Combinatorial Testing with Pairwise

Combinatorial testing is required whenever an application must process a large number of input variables (e.g., web forms for shopping carts, advanced site search filters, software configuration settings) or must execute across a wide variety of operating environments (e.g., different hardware platforms, operating systems, and browsers across multiple release levels).

Before you know it, you are in "combinatorial explosion" of testing possibilities. For example, **10 input fields, each with 10 possible input variables, represents 10 billion testing combinations**. For many applications, testing all possible combinations of input variables is next to impossible.

A Real-World Example



Show

- [✓] Startup Task Pane
- [✓] Highlight
- [ ] Bookmarks
- [✓] Status bar
- [✓] ScreenTips
- [✓] Smart tags
- [✓] Animated text
- [✓] Horizontal scroll bar
- [✓] Vertical scroll bar
- [ ] Picture placeholders
- [✓] Windows in Taskbar
- [ ] Field codes

Field shading: When selected

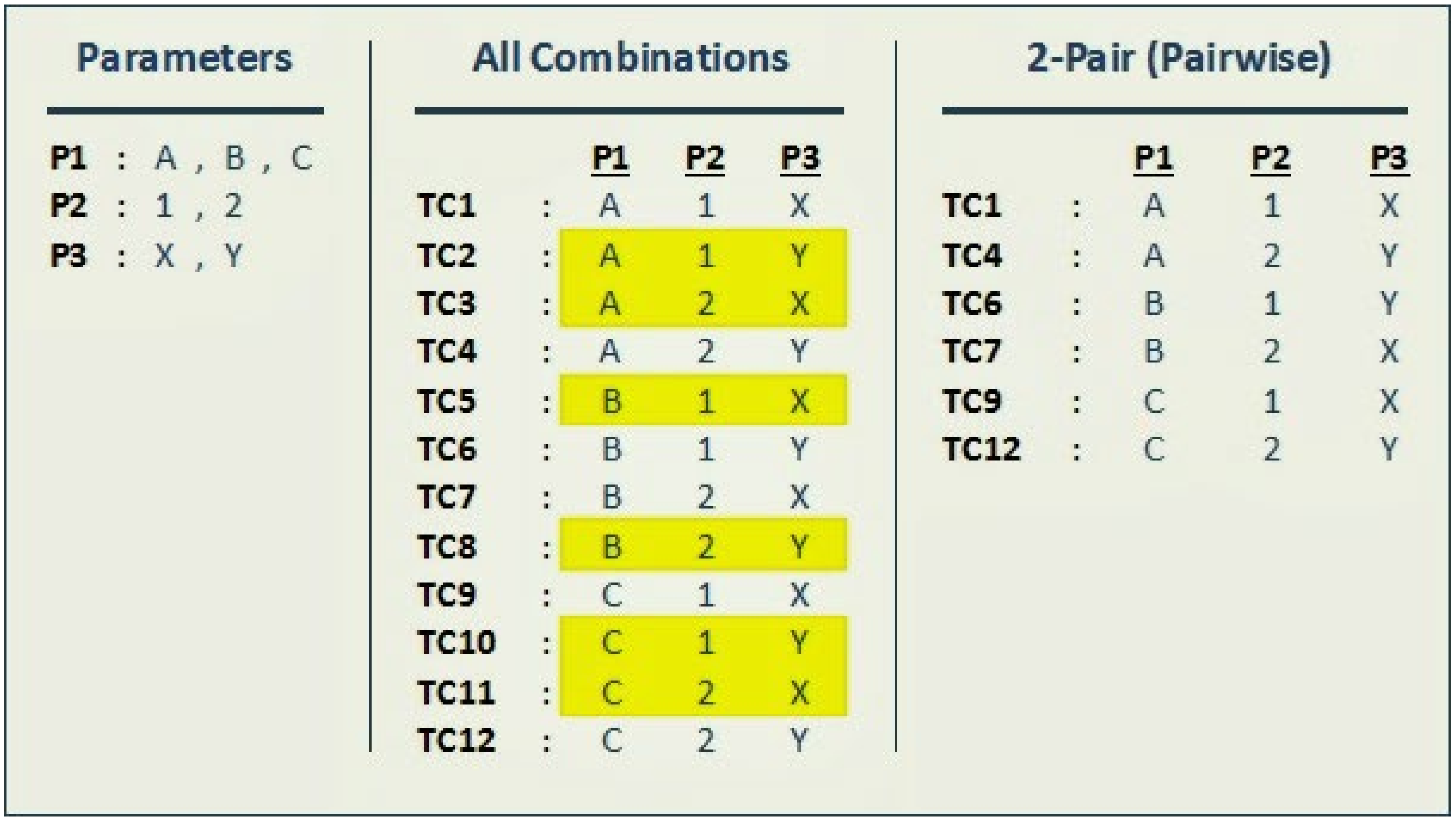
Portion of Option Dialog in Microsoft Word

Consider this real-world example for software that many of us use every day. The dialog box in the illustration from Microsoft Word represents 12,288 possible testing combinations. Here is the math behind that number: 2^{12} to represent 12 checkboxes each having 2 input possibilities that are multiplied by 3 for the "Field sharing" menu which contains 3 items.

Pairwise testing (or "all pairs testing") is a testing methodology that drastically reduces the number of test combinations required while maximizing test coverage and defect identification.

NIST research showed that most software bugs are caused by one or two parameters, with progressively fewer defects caused by the interaction of three or more parameters. This finding is referred to as the interaction rule and has dramatic implications for testing software with a high number of input variable combinations.

The International Software Testing Qualification Board, or ISTQB **defines Pairwise Testing as a black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters.** Here's how it works. Imagine a software application module that has 3 input parameters with each one having three possible responses as illustrated in the chart below.



Parameters

P1	:	A , B , C
P2	:	1 , 2
P3	:	X , Y

All Combinations

		P1	**P2**	**P3**
TC1	:	A	1	X
TC2	:	A	1	Y
TC3	:	A	2	X
TC4	:	A	2	Y
TC5	:	B	1	X
TC6	:	B	1	Y
TC7	:	B	2	X
TC8	:	B	2	Y
TC9	:	C	1	X
TC10	:	C	1	Y
TC11	:	C	2	X
TC12	:	C	2	Y

2-Pair (Pairwise)

		P1	**P2**	**P3**
TC1	:	A	1	X
TC4	:	A	2	Y
TC6	:	B	1	Y
TC7	:	B	2	X
TC9	:	C	1	X
TC12	:	C	2	Y

In this simple example, 12 test cases would be required to test all combinations of variables. However, by testing only the 2-pair combinations, the number of test cases can be reduced to 6 test cases – a reduction of 50% of test cases required.

As the number of variables increases, the number of testing combinations grows exponentially and the use of pairwise with test case reduction becomes increasingly powerful. Pairwise allows testers to greatly improve test coverage without the need to test every conceivable combination.

There are methods to further reduce the number of test cases using pairwise. One common approach is to create *equivalence partitions* and identify *boundary values*. With equivalence partitioning, test cases are divided into classes of input data, like state values for text fields (e.g., valid integer, invalid integer, alpha/special character) or a range of numbers for numerical values. Then boundary values can be established to constrain the size of a numerical range or test the edge cases to identify the more likely errors that often occur at the boundaries.

GenRocket's TDG platform can be used to conduct pairwise testing by generating test data that represents realistic input variable pairs controlled by *constraints* to limit the number of test cases required.

Constraints also ensure that variable combinations that are not possible in real life are not included in the test. For example, for a given alarm system under test, if the state of its *Switch* variable is *Off*, then testing the state of its variables, *Security Level* and *Action*, would be impossible.

Following is GenRocket's approach for maximizing test coverage with Pairwise:

1. **Reduce the Number of Tests:**
   With GenRocket Pairwise Combinatorics
2. **Generate the Right Test Data:**
   With GenRocket's Test Data Scenarios using Constraints
3. **Automate Pairwise Test Execution:**
   By integrating GenRocket with test automation tools

GenRocket's pairwise solution enables an efficient testing process that significantly reduces the number of tests, maximizes code and data coverage, minimizes false positives and improves overall testing accuracy. The more complex the combinatorial testing challenge, the greater the value of GenRocket's TDG platform.

Using TDG for AI and ML Applications

Artificial Intelligence (AI) and Machine Learning (ML) are two of the hottest buzzwords in the field of Information Technology. AI is behind the growing popularity of the Virtual Digital Assistant (VDA) as popularized by Google Home, Siri, Cortana and Alexa and used by consumers to answer questions and automate everyday tasks. Businesses are increasingly using VDAs for sales, marketing and customer service applications as well.

ML is a subset of artificial intelligence and is the enabling technology behind the rapidly growing field of predictive analytics. Machine learning uses sophisticated algorithms that allow computers to recognize patterns from current and historical data, learn from those patterns and then make predictions about future outcomes.

Machine Learning is used in a wide variety of business applications including:

- Recommendations Engine
- Fraud Detection
- Personalized Marketing
- Operational Efficiency
- Dynamic Pricing
- Risk Reduction
- Health Care Applications
- Insurance Applications
- Predictive Maintenance

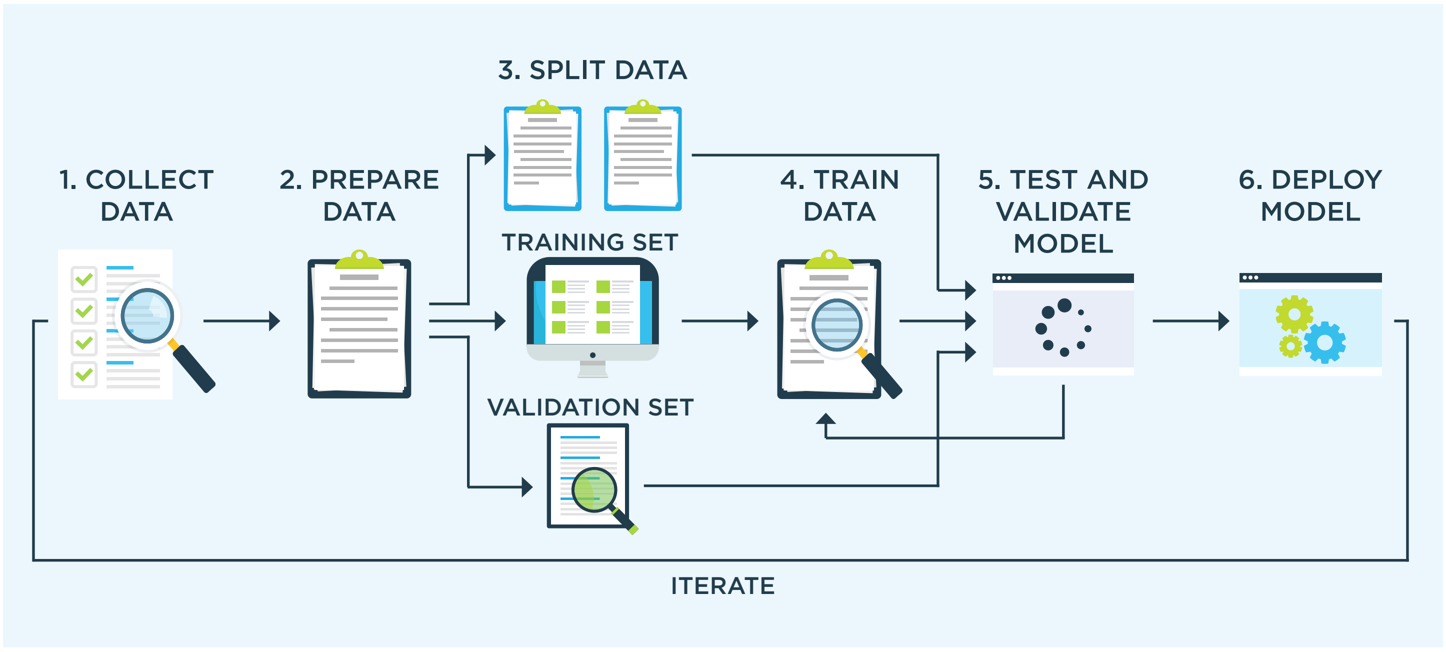
When developers and data science practitioners think about new applications for AI, ML and predictive analytics, they often think the bulk of the work will be in the development of the algorithms and how to code them. **One of the biggest challenges in developing applications for AI and ML is provisioning the data used to train, validate and test their algorithms for accuracy and robustness.**

When perfecting a new algorithm for AI and ML applications, it is important to remember this:

High Quality Training Data & Test Data at Scale = Accuracy of AI & ML Algorithms

The greater the volume and variety of training data used, the more accurate and robust the model for predicting future outcomes will be. This creates a major challenge for testers: How to provision a high volume of high-quality training data without spending an enormous amount of time collecting, labeling, classifying, cleaning, pruning, normalizing, and formatting the data with the help of domain experts who understand the data requirements.

Also, important to remember is the need for three different kinds of data during the development process: One dataset to train the model, one dataset to validate the model and one dataset to test the model.



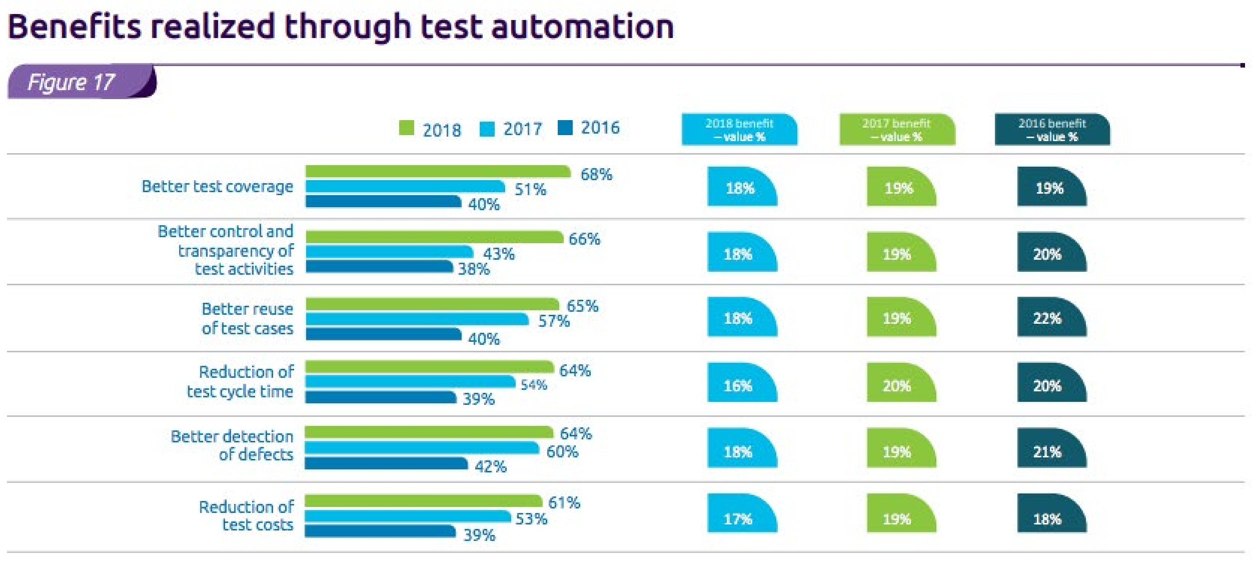
These three datasets must be different to ensure the integrity of AI or ML algorithms and how they will perform in real-world scenarios. That's where GenRocket's ability to generate high-volumes of data based on a predefined data model, data attributes and patterns of data variation is a perfect match for AI and ML application development.

Once the domain expert specifies the data requirements, GenRocket's TDG engine generates controlled and conditioned data at the rate of 10,000 rows per second. By partitioning the engine into multiple instances, performance can be increased exponentially to make same-day provisioning for datasets having millions or even billions of rows a possibility.

If you would like to learn more about GenRocket's TDG solution for training and testing AI and ML applications, read the case study Test Data Solutions for Artificial Intelligence and Machine Learning Applications.

7 REALIZING OPERATIONAL BENEFITS

The 2018-19 edition of the World Quality Report, a comprehensive research report covering the key trends shaping quality assurance and testing, examined the benefits realized through test automation. Based on a survey of 1700 executives across 10 different industry sectors and 32 countries, the report identified a broad range of benefits that describe the different ways test automation brings value to their organizations.



While the results show a mostly even distribution of responses, "better test coverage" was identified as the leading benefit realized by QA executives. Although the number of respondents who identified this benefit is higher than recorded in previous years, it's hard to say exactly how much their test coverage actually improved as a result of automation. Did coverage only improve by 10% resulting in a marginal benefit, or was it dramatically higher leading to a game-changing level of improvement?

Test automation is capable of accelerating the testing process exponentially, and with the addition of TDG, it allows QA organizations to run more tests with greater test data variety and volume to maximize coverage.

From the findings above, it’s hard to say the full potential of test automation was realized by the organizations surveyed. One can only assume that given the challenges cited for test data provisioning, combined with the learning curve for adopting new technology, most organizations likely have a long road ahead before they reach full coverage and realize the full benefits of *quality at speed.*

When asked about the principle benefits they ***failed to realize***, 36% of respondents said, “better detection of defects”. This missed expectation supports the notion that testers are still in search of the right variety and volume of test data needed to fully test their code and to avoid the problems of false positives and false negatives.

The message from the market research is clear, QA leaders are fully committed to test automation as the best long-term strategy for improving test coverage and maximizing the quality of code released to production. However, they struggle with transitioning from early adoption of the technology to its full deployment.

QA managers expect several benefits from test automation:

- Better test coverage
- Better control and transparency of test operations
- Better reuse of test cases
- Reduction of test cycle time
- Better detection of defects
- Reduction of test costs

And according to the 2018-19 World Quality Report, over 60% of QA executives reported seeing some level of improvement in each of these areas. As they overcome the challenges of test data provisioning and improve their proficiency in the application of test automation technology, those benefits will become more fully realized.

Achieving Test Automation Synergy

Test automation synergy is all about maximizing the benefits of combining CI/CD pipelines with test automation and test data generation. The answer to achieving *quality at speed* lies at the intersection of these three technologies. If one is omitted, or their best practices not followed, the level of quality and operational efficiency realized will fall short of expectations.

The following six steps summarize the keys for creating synergy between CI/CD pipelines, test automation and test data generation:

1. Integrate TDG with CI/CD and Test Automation

Establish a fully integrated test automation environment that includes your CI/CD server (e.g., Jenkins), your testing framework and tool set (e.g., Selenium), and your test data generation platform (e.g., GenRocket). Start with a *Proof of Concept*, select appropriate use cases that will create *quick wins,* and map out your test automation deployment strategy. GenRocket has a structured POC program that will assist you in getting started and ensuring a successful experience with this new and powerful technology combination.

2. Enable Self-Service Provisioning

Identify some functional, performance and regression test combinations and provide testers with access to the integrated test automation environment. Ensure they are trained in the use of productivity features such as *GenRocket Presets* (Test Data Scenario templates) and *Test Data Suites* (a scriptless wizard for Test Data Scenario creation). The nature of the test data that can be generated is limited only by the tester's imagination. GenRocket TDG enables a self-service alternative to the centralized model associated with traditional Test Data Management systems.

3. Facilitate Collaborative Test Design

Encourage testers to share their knowledge of test case and test data designs. Take advantage of shared repositories for reusable test cases and test data scenarios. This provides testers with ready access to pre-built tests that generate test data on-demand. Where possible, encourage testers to reuse and repurpose their test case/test data designs for multiple test operations. To create a secure environment GenRocket provides the ability to control access to specific TDG resources using the *GenRocket Teams* feature.

4. Phase Out Production Data

Transition the use of production data to *Real-Time Synthetic Test Data* generated by the TDG platform. This approach allows you to phase out the use of sensitive customer data, improves test data quality and accelerates the test data provisioning process. It also reduces the bottleneck associated with centralized provisioning and allows the QA team to directly control the nature of the test data used for testing. Gradually increase the use of real-time synthetic test data to 90% of the data required for testing.

5. Eliminate Data Masking

Prioritize your transition from production data to real-time synthetic test data by isolating *Personally Identifiable Information* (PII) and thereby eliminating the requirement for data masking. This saves time and eliminates the cost and complexity of traditional Test Data Management systems. At the same time, this step removes the risk of a data privacy breach while ensuring compliance with all data privacy laws.

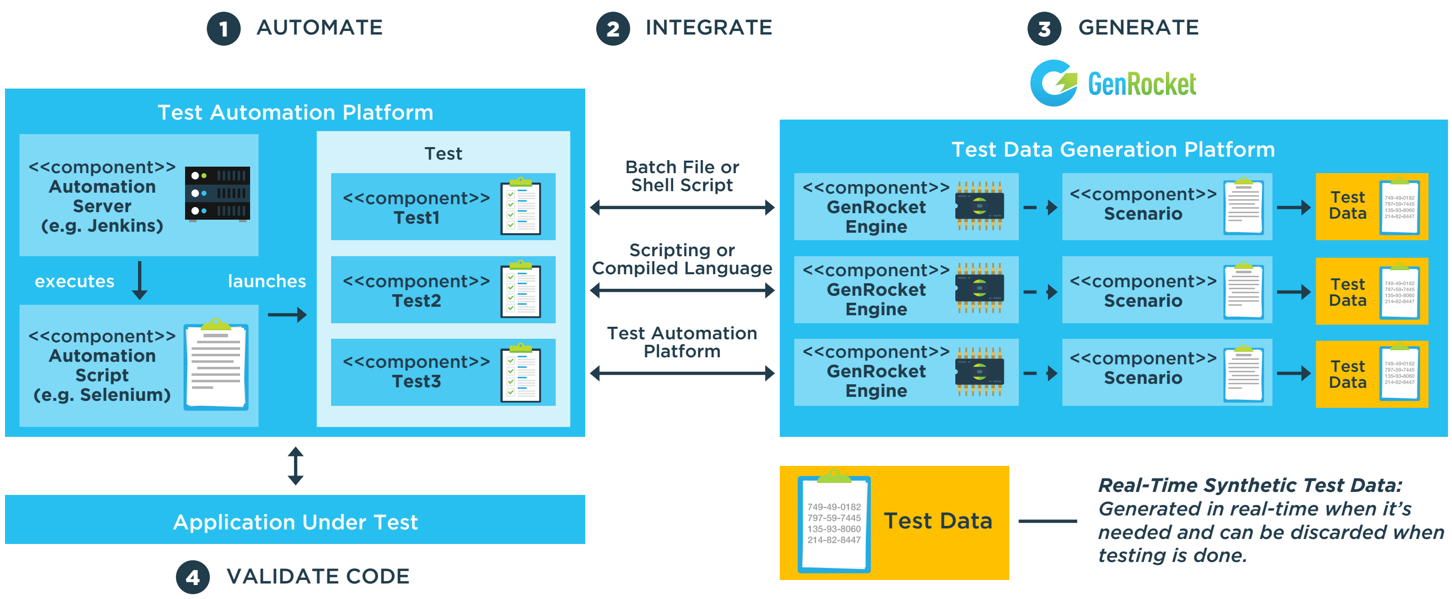
6. Implement TDG Best Practices

Stay current on the best practices for test case and test data design methods. Utilize *GenRocket University*, an online educational resource, to advance the skill level of your testing staff. Consult with *GenRocket Certified Partners* for the most effective and efficient way to deploy and operate the TDG platform. This will streamline the process of scaling up your test automation environment as you maximize the benefits of test automation as well as your return on investment.

Take the First Step Toward Test Automation Synergy

Are you ready to begin the journey toward achieving test automation synergy? Getting started is easy and GenRocket is ready to provide all the assistance you will need to successfully implement a *Fully Integrated Test Automation Environment as illustrated by the diagram below.*

FULLY INTEGRATED TEST AUTOMATION ENVIRONMENT



1 AUTOMATE
2 INTEGRATE
3 GENERATE
GenRocket
Test Automation Platform
<<component>> Automation Server (e.g. Jenkins)
executes
launches
<<component>> Automation Script (e.g. Selenium)
Test
<<component>> Test1
<<component>> Test2
<<component>> Test3
Batch File or Shell Script
Scripting or Compiled Language
Test Automation Platform
Test Data Generation Platform
<<component>> GenRocket Engine
<<component>> Scenario
Test Data
Application Under Test
4 VALIDATE CODE
Test Data
Real-Time Synthetic Test Data: Generated in real-time when it's needed and can be discarded when testing is done.

Start with a live demonstration of GenRocket TDG integrated with CI/CD and test automation environment. Ask questions of TDG experts and learn how your QA organization can benefit from the power of TDG technology.

Then work with a GenRocket Certified Partner to conduct a Proof of Concept based on 1, 2 or 3 representative test cases that will prove-in the technology and allow your team to experience the power and impact of GenRocket TDG first hand. Just click the link below to get started.

REQUEST A LIVE DEMONSTRATION

GenRocket

Think Differently About Test Data

Contact Us

GenRocket, Inc.
2930 East Ojai Ave.
Ojai, CA 93023, USA

1-805-836-2879

info@genrocket.com

Visit us at **www.genrocket.com**