Test Data Management

Patterns and Anti-Patterns for a Data Factory

NIALL CRAWFORD
CHIEF INFORMATION OFFICER, ENOV8

CONTENTS

INTRODUCTION

When it comes to managing your software development lifecycle (SDLC), one of the most challenging aspects is effectively managing and deploying data.

This involves building "data-ready" test environments that will effectively support your engineering practices across the development and test lifecycle — a discipline better known as **test data management (TDM)**.

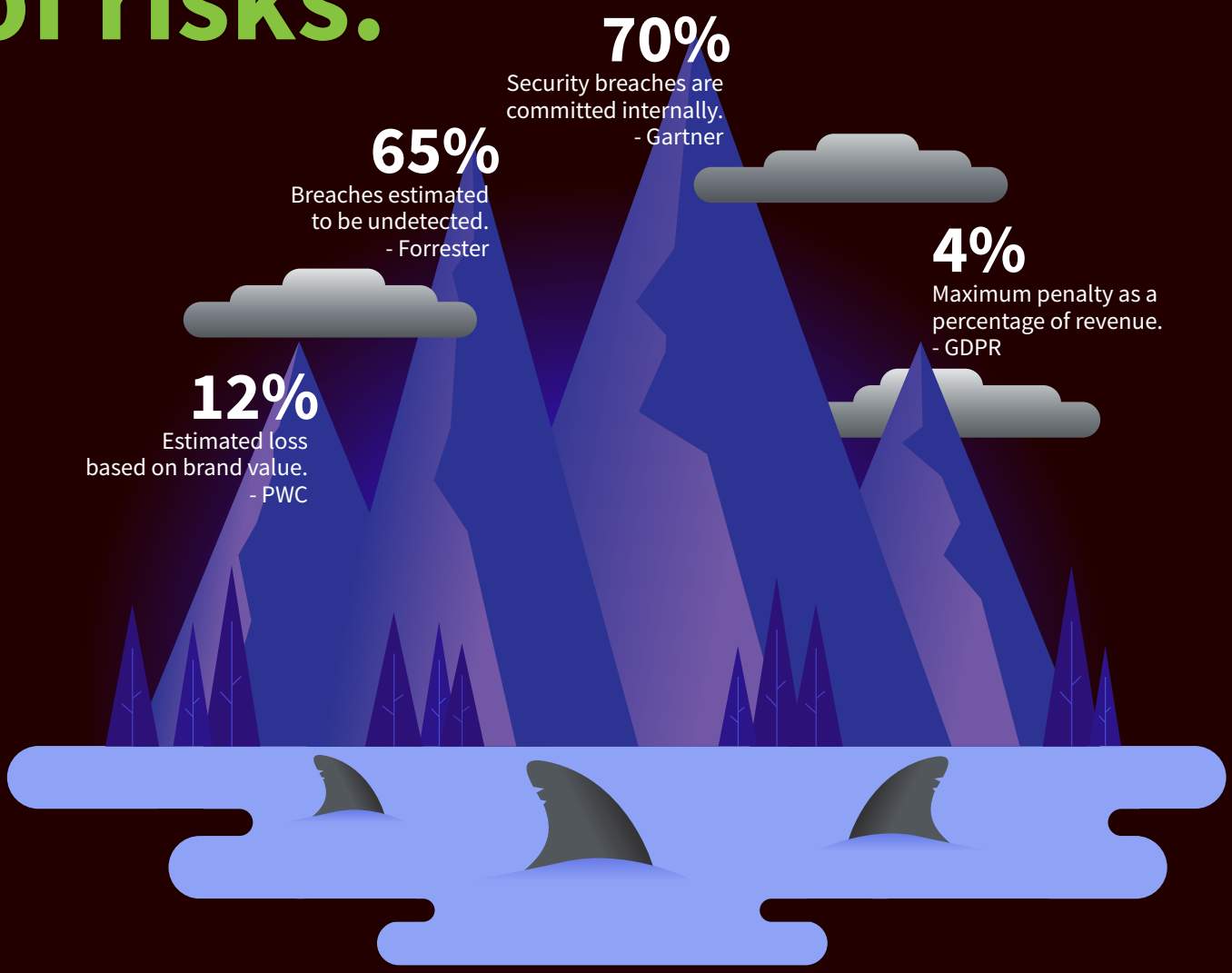
NATURAL COMPLEXITY OF DATA

When talking about test data management, it is important to talk about the natural complexity of data, which can often make TDM more difficult to manage.

Complexities that include:

- **Heterogenous data architectures**
  - It is not uncommon for organizations to have a mix, including relational databases, hierarchical databases, NoSQL databases, structured flat files, unstructured files, binaries, big data solutions, and in-memory.
- **Reliance on multiple database vendors**
  - Each database vendor promotes their own proprietary "query" languages and methods, for example, Oracle, Microsoft, IBM, SAP, Sybase, MongoDB, Teradata, and open source like Apache.
- **Different hosting platforms**
  - Different hosting platforms consist of distributed (Linux, Windows, UNIX) and legacy (Z/OS, AS400) systems, each requiring their own subject matter experts.
- **Database design**
  - Ultimately, database design is dependent on the application designer and function, ranging diversely from a few tables (perhaps leveraging a two-table object-oriented design) through to a complex spaghetti (perhaps thousands of tables and 100K of columns) evolved over time.
- **Relationships between data points**
  - Data points consist of relationships and constraints that can be managed within the database itself and can also be managed outside the database in the application logic, or even externally via upstream-downstream microservices.

Sea of data and a mountain of risks.



DATA COMPLIANCE SUITE (DCS) BENEFITS

For the technical & non-technical alike, an "automated intelligence" solution that identifies sensitive data, remediates the risk and validates your compliance success.

SIMPLICITY

- Easy to Learn
- Painless to Onboard
- Straightforward Workflow
- Informative Dashboards

COMPLIANCE

- Understand Data & Risk
- Protect Customer Identify
- Avoid a Security Breach
- Align to Regulations

AGILITY

- AI based Threat Analytics
- Rapid Risk Remediation
- Automated Validation
- Self Service Operations

DEVSECOPS

- DevOps Library
- Data Fabricator
- Data View
- Easy CICD Integration

CONTACT US TODAY
www.enov8.com

enov8

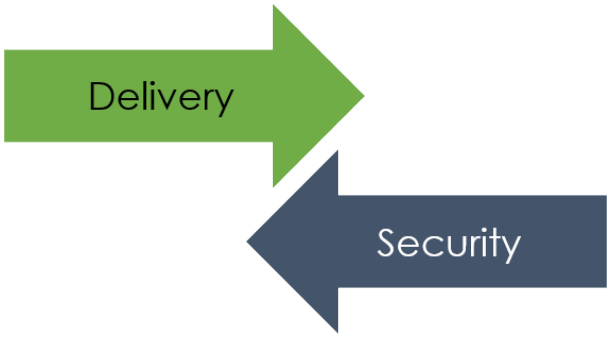
THE BROAD CHOICE OF DATA TOOLING

Another key component to TDM is the vast choice of tools that aim to support your data management requirements and operations, ranging from tools that are more DevOps oriented to those more DB specific.

This may include tools to support your database design; editing or coding; configuration management of data assets like stored procs and schemas; tools to create new data content; tools to extract or clone old data; various data transformation solutions, for example, aging; methods for testing or analyzing data quality; database monitoring solutions; tools for provisioning/loading; and vendor-specific database management and tuning platforms.

DELIVERY PRIORITIES

There are often different priorities across your release team and enterprise, which can be loosely described as a focus on "delivery" (go fast) versus "security" (be careful).



A set of differing objectives leads us to realize "one size does not fit all," and we may inevitably gravitate to more than one delivery mode, or at the very least, one "data bimodal."

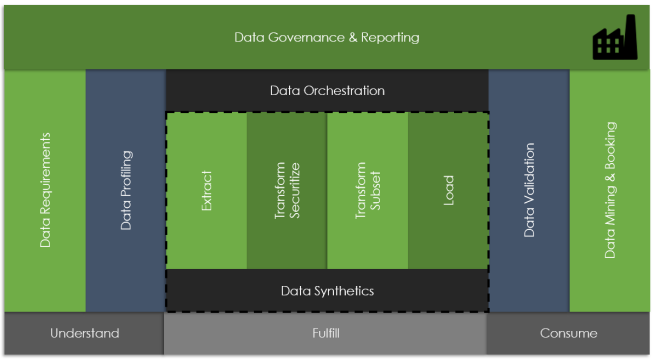
With that in mind, let us review the multi-faceted disciplines of TDM, with a focus on what TDM looks like more broadly and each of its subcomponents and demands.

HOLISTIC TEST DATA FACTORY

So what might a robust test data management framework look like? In the example below, we highlight the primary methods, or facets, you may need to consider when implementing a TDM data factory.

Author's Disclaimer:

The following "data factory" diagram was originally designed to help larger client enterprises contextualize the broader aspects of test data management and data securitization. Smaller organizations, or independent teams, may have some flexibility in choosing which of these methods adds value and which might be displaced as unnecessary.



Some TDM methods for consideration in your data factory include:

- Data requirements management
- Data profiling
- Data synthetics
- Data sub-setting
- Data virtualization
- Data masking (or encryption)
- Data validation
- Data orchestration
- Data viewing
- Data exemptions

TEST DATA MANAGEMENT METHODS

With the above in mind, let us now explore the key facets of test data management, exploring the primary objectives and goals of TDM, best practices (referred to as "patterns") supporting TDM, and behavior that tends to be counter-productive ("anti-patterns").

DATA REQUIREMENTS

GOAL	The data provisioned should align with our test needs
PATTERN	Ensure data requirements are not forgotten
ANTI-PATTERN	Capturing test data requirements too late

PATTERN

Ensure data requirements are not forgotten (or identified too late). For each user story, ensure that your data needs (tasks) are captured and articulated effectively.

> ***Story-001 Data-Task:***
> *I need Loan Data,*
> *Where Customer is Female &*
> *Loan Created between:*
> *1/JAN/2021 to 31/JULY/2021*

Benefits

- Re-work avoidance
- Timely preparation of environments
- Testability

ANTI-PATTERN

Capturing test data requirements too late. Identifying data requirements after the environments are provisioned will invariably result in long delays and inhibit broader Dev, Test, and QA endeavors.

Example data profiling report

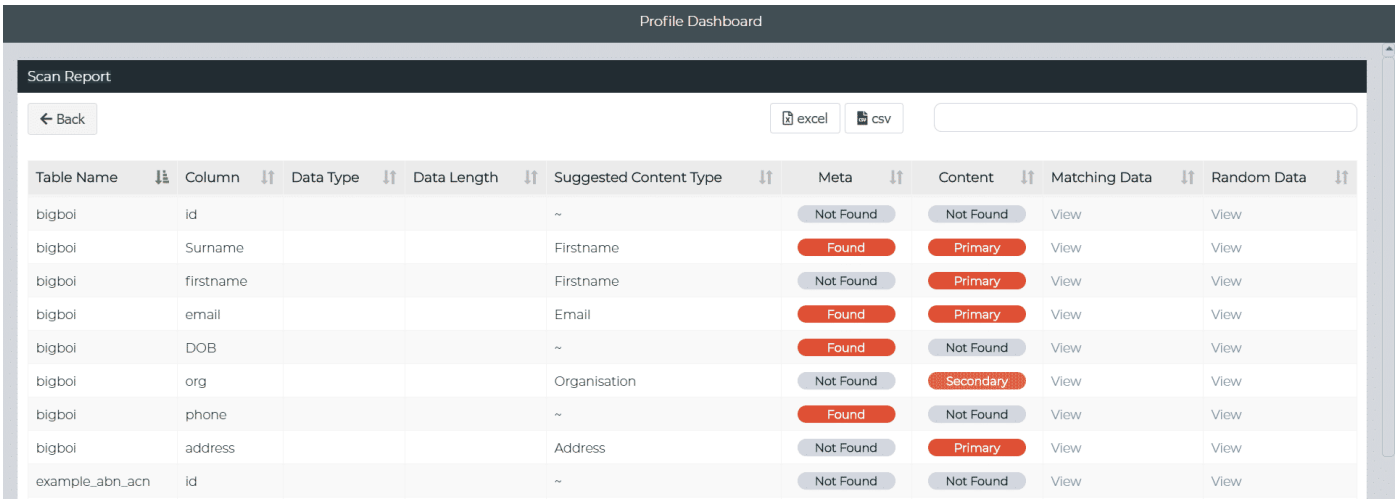


Table Name	Column	Data Type	Data Length	Suggested Content Type	Meta	Content	Matching Data	Random Data
bigboi	id			~	Not Found	Not Found	View	View
bigboi	Surname			Firstname	Found	Primary	View	View
bigboi	firstname			Firstname	Not Found	Primary	View	View
bigboi	email			Email	Found	Primary	View	View
bigboi	DOB			~	Found	Not Found	View	View
bigboi	org			Organisation	Not Found	Secondary	View	View
bigboi	phone			~	Found	Not Found	View	View
bigboi	address			Address	Not Found	Primary	View	View
example_abn_acn	id			~	Not Found	Not Found	View	View

AUTOMATED DATA PROFILING

GOAL	Better understand our platform data
PATTERN	Implement "automated" profiling methods to promote data literacy
ANTI-PATTERN	Profiling data manually

PATTERN

Implement "automated" profiling methods to promote data literacy, i.e., ensure an understanding of your data structure and content, including the discovery of schemas, tables, columns, relationships, constraints, and sensitive risks like personally identifiable information (PII).

Examples of PII Risks

- *Full name*
- *Address*
- *Email*
- *Phone number*

Benefits

Data literacy will contribute to SDLC delivery activities such as development, testing, and support tasks like data synthetics.

"Data risk" literacy will benefit those involved in compliance and security endeavors and support tasks like data masking.

ANTI-PATTERN

Profiling data manually. It is not uncommon for the data sources to have thousands of columns and billions of data points. Manual profiling methods, like team workshops, will probably fail for anything but the simplest of data structures.

DATA SYNTHETICS

GOAL	Require "fake data" for developer and test activities
PATTERN	Build synthetic test data for the purpose of development and testing
ANTI-PATTERNS	• Profiling data manually • Assuming synthetic data is suitable for all test phases

PATTERN

Build synthetic (i.e., fabricated, fake, or dummy) test data for the purpose of development and testing.

Note: There are two main reasons why synthetic test data is generated: (1) to avoid using "production" data that may contain sensitive information and (2) to "build" data that may not exist in production, for example, data related to a new product or story. Fake data is particularly useful in the lower test environment or early phases of the test lifecycle, e.g., unit and system.

Benefits

- Easy to share and reuse

- Rapid provisioning
- Improves security as it avoids the risk of "production data" being used in earlier phases of the lifecycle

ANTI-PATTERNS

Using fake data in higher environments. Data, and its underlying structure, is rarely simple, and at times, it can be incredibly complex. This includes relationships between the tables and columns, which can be managed outside of the database itself. It is not difficult to create data that is not "fit for purpose" and does not match the platform business rules.

Don't assume that "synthetic data" is universally suitable for all test phases. This includes higher environments like SIT, UAT, and performance. The value of "fake data" will usually diminish later in the lifecycle as acceptance and cross-platform integration coverage increases.

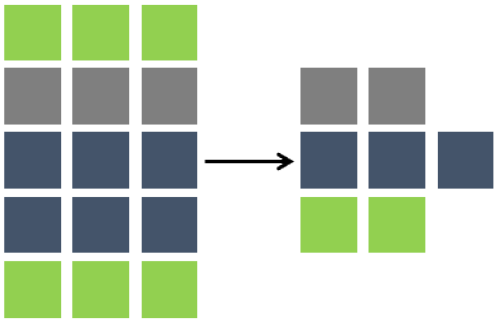
DATA SUB-SETTING

GOAL	Keep data small and easy to provision
PATTERN	Employ small data footprints
ANTI-PATTERN	Loose or weak subset design

PATTERN

Data sub-setting is the act of reducing your data footprint by selecting data based on a "repeatable" criterion. By employing small data footprints, the DevOps teams will find it easier to provision data, share data, and use standalone methods like containers or sandpits.

> ***Simple Example:***
> *Only use data related to Customers*
> *Where name beginning with C%*



Benefits

- Less storage costs
- Reduced batch/processing time

ANTI-PATTERN

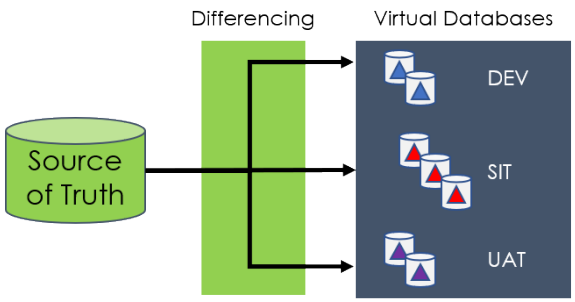
Loose or weak subset design. Sub-setting needs to be "precise;" otherwise, the process will result in "required" data going missing and/or relationships being lost. A loss of data integrity is likely to break business logic and cause the platform to cease functioning.

DATA VIRTUALIZATION

GOAL	Keep data small and easy to provision
PATTERN	Identify data "deltas" or "differencing" between the application "source of truth" and local copy
ANTI-PATTERNS	• Use de-sensitized data in source • Transform the delta source

PATTERN

Data virtualization is a modern, and somewhat safer, alternative to sub-setting that identifies data "deltas" or "differencing" between the application "source of truth" and your local copy.



Tip: Think of the virtual copy as a snapshot of the changes (deltas) made by the development or test team. Unchanged data still resides back in the "source."

Benefits

- Rapidly provision copies of your DB for Dev and Test
- Small storage footprints

ANTI-PATTERNS

Using de-sensitized data in source. Virtualizing the data means that the teams still have access, indirectly, to the original source. If virtualization is used, then ensure the sensitive "source" data has been obfuscated.

Transforming the delta: If the data needs to be transformed, then transform the source. Trying to transform the virtualized DBs can cause the virtual copies to explode in size (due to over delta-ing), losing their original portability value.

DATA MASKING

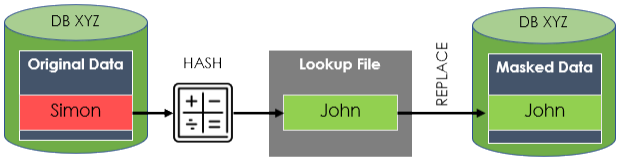
GOAL	Real data sensitivity should be obfuscated and non-reversible
PATTERN	Dataset is copied but the PII and/or sensitive data has been obfuscated
ANTI-PATTERNS	• Securing too much • Securing inconsistently

PATTERN

Data masking is a data security technique in which a dataset is copied but the PII and/or sensitive data has been obfuscated.

Ideally, obfuscated data should be non-reversible to further tighten security, and for the benefit of DevTest, it should still look and feel production-like.

> ***For Example:*** *Replace First name*
> *Simon with John*
> *Mary with Sally*



Benefits

- Aligns to privacy regulations, e.g., GDPR, HIPPA, and PCI
- Helps QA/Test by looking and behaving like production

ANTI-PATTERNS

Securing too much. When you mask, there is the inevitable risk of breakage or loss of integrity. As such, keep your masking scope "tight," i.e., focus on primary PII and don't be tempted to "over secure" low value artifacts. Less is more (and quicker).

Securing inconsistently. If you change "Simon" to "John" here, then ensure you change "Simon" to "John" there. Inconsistent masking within an application or across applications is likely to cause serious data integrity issues.

DATA ENCRYPTION

GOAL	Real data sensitivity should be obfuscated and reversible
PATTERNS	• Obfuscate information so that it can't be read • Decipher/reverse engineer data using a secret key
ANTI-PATTERNS	• Using encryption where encryption is not sensible • Breaking formatting rules

PATTERN

Use data encryption as a method of (1) obfuscating information so that it can't be read and (2) being able to decipher (reverse engineer) that data using a secret key.

> *Note: There are various industrial strength methods, for example: DES, AES, and RSA. There are also weaker/lighter methods like cyphering.*
>
> ***Example Encryption:***
> *Input Data: Smith*
> *Output Data: BPMtbFk5BrZ1OnpwS8vpvw==*
> *Note: AES128 with 16-digit secret.*

A primary differentiator with masking is the option to reverse the data later. This is potentially useful if the data was encrypted to support safe transfer, for example, to a trusted site.

Benefits

- Consistent
- Reversable

ANTI-PATTERNS

Encryption where encryption is not sensible. A primary issue with encryption is that data no longer looks production-like, i.e., the data loses its shape. For example: "Jones" doesn't look like *"XPMtbFk7npwS8vpvw==". Only use encryption if pattern breakdown is acceptable.*

Breaking formatting rules. Another risk of encryption is that data may be transformed into a format that doesn't support the application logic or contradicts table column definitions or constraints — a transformation that would likely result in application breakage and non-useability.

DATA VALIDATION

GOAL	Validate that sensitive data has been removed
PATTERN	Implement a screening process to ensure sensitive data protection
ANTI-PATTERN	Manual inspection

PATTERN

With thousands to tens of thousands of columns, the risk of missing PII is likely. Implement a screening process to ensure the sensitive data has been remediated and has not slipped through the net.

Benefits

- Compliance
- Security

ANTI-PATTERN

Manual Inspection. Manual inspection of point data isn't ideal, nor reliable, especially when you are dealing with millions of data-point changes. Ideally, you need to use automated methods to demonstrate that the broader "production patterns" are now absent.

DATA VIEWS

GOAL	Data should be easy to consume and find — and free of contention
PATTERN	Implement methods that simplify the real-time viewing of your data
ANTI-PATTERN	Use a data list

PATTERN

A lot of time can be wasted in finding the correct test data for your test work. This process can be further "disadvantaged" by others using the same data and then altering it unexpectedly.

Implement methods that simplify the real-time viewing of your data, mining for "fit-for-purpose" pools of test data. Also, it is important to help "reserve" data pools to limit opportunity for overwrite.

> *Example:*
> ***Create a Data View called Data-Pool1***
> *Where Customer Last Name Like C%*
> *Gender is Female AND*
> *Loan Created between:*
> *1/JAN/2021 to 31/JULY/2021*
> ***Assign Data-Pool1 to Tribe1***

Benefits

- Find data instantly
- Avoid data-related defects (overwrite)

ANTI-PATTERN

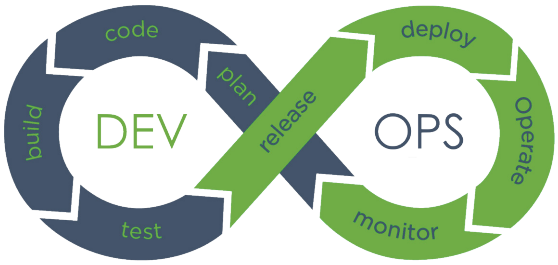
Use a data list. Using a list (for example, a spreadsheet) to maintain a set of data points for testing is a common but somewhat stagnant practice. Lists are slow to build; they usually represent a point in time ("time of extract") that users will gravitate toward the same data points (see top of Refcard). Typically, they get out of date quickly due to the data or testable requirements changing.

DATA ORCHESTRATION

GOAL	Data operations should be streamlined and self-serviceable
PATTERN	Identify key test data operations
ANTI-PATTERNS	• Playing "the hero" • Selective automation

PATTERN

Data operations are often the slowest activities when it comes to preparing your environments. This is partly due to the date complexity, likelihood of breakage, and the data SMEs often being busy single points of failure (SPOF).



It is important to identify the key test data operations with the intent of standardizing and automating them to align with your broader DevOps and CI/CD practices.

Benefits

- Repeatable, i.e., reliable
- Fluidity, i.e., less time waiting
- Self-serviceable

ANTI-PATTERNS

"The Hero": Reliance on a team, or individual, that goes to "extraordinary lengths," usually working lots of hours to help the team both test and deliver within deadlines. However, the heroics — and the reliance on these "heroics" — is continual, and mistakes, due to overwork, are common.

Selective automation. When specific data operations are excluded from automation because the team or individual responsible doesn't see the value of automation and/or fears that automation may make them redundant.

DATA EXEMPTIONS

GOAL	Recognizing scenarios when using production data is impactful

PATTERNS	• Use production data when necessary • Provide a service request method
ANTI-PATTERN	Promoting exemptions as a standard practice

PATTERNS

Although not ideal, sometimes we may need to use production data. This may be required during a serious production event, where use of non-production data may limit test and fix.

Without promoting the concept of "data exemptions" as it is a last resort, provide a service request method where the request can be raised, risk documented, request approved, and length recorded (the window of risk).

Benefits

- Auditability
- Ownership of decision and risk

ANTI-PATTERN

Promoting exemptions as a standard practice: No individual — other than your customer — has the right to use customer information. This breach of trust should be deemed a last resort and a very rare occurrence.

DATA REPORTING

GOAL	Understand data operations to make improvements
PATTERN	Implement automated methods — DataOps and DataSec reporting
ANTI-PATTERNS	• Not reporting • Manual reporting

PATTERN

Implement automated methods so that we can understand our behavior and continually improve. Data reporting can be loosely divided into two camps:

1. **DataOps (Data DevOps) Reporting**
   - Focus on the day-to-day operations.
   - For example, for each DataOps Type*:
     - Ticket requests
     - Lead time (request to completion)
     - Execution time
     - Frequency
     - Success rate

**: A DataOps type might be a provisioning task, or something else, for example, a subset, mask, age, etc.*

2. **DataSec (Compliance) Reporting**
   - Focus on ensuring regulatory needs of security, compliance, and audits.
   - For example:
     - Coverage of platforms risk profiled
     - Coverage of platforms secured
     - Exemption reports

Benefits

- Measure of operational efficiency
- Measure of security coverage
- Baselines to improve from

ANTI-PATTERNS

Not reporting: If you don't measure, you will struggle to improve it.

Manual reporting: Attempting to report on DataOps or DataSec manually is destined to fail. The information is too broad and will be moving too fast. Ultimately, the information will fall into disrepair and be unreliable.

CONCLUSION

There is more to TDM than just ETL! Test data management should be an essential part of an organization's test environment and release management operations.

Consider these aforementioned TDM, or DataOps, methods as part of your data factory and look to embrace DevSecOps practices in promotion of:

- Streamlined delivery
- Operational automation
- Cross-team collaboration
- Repeatable and reliable processes
- Improved and proactive security
- Platform readiness and stability
- Security and data literacy
- End-product quality

How does your data factory measure up?

REFERENCES

Further reading:

- DZone: The Data Bimodal
- DZone: Environment Anti Patterns
- Enov8: What is TDM?
- Enov8: Test Data & Data Compliance in DevOps
- Enov8: Environment Maturity Assessment

WRITTEN BY NIALL CRAWFORD,

CHIEF INFORMATION OFFICER, ENOV8

Niall, a passionate software engineer, has been in the IT industry for over 30 years. From writing games in his teens, through to Software Developer, Architect and eventually into Executive IT Management.

Niall is currently the CIO and co-Founder of Enov8. Enov8 specializes in helping organizations better understand and manage their IT landscape, including IT and test environments, data, and releases.

DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of thousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 | 919.678.0300