Testing your ETL pipelines:

test when writing
test when deploying
test when running

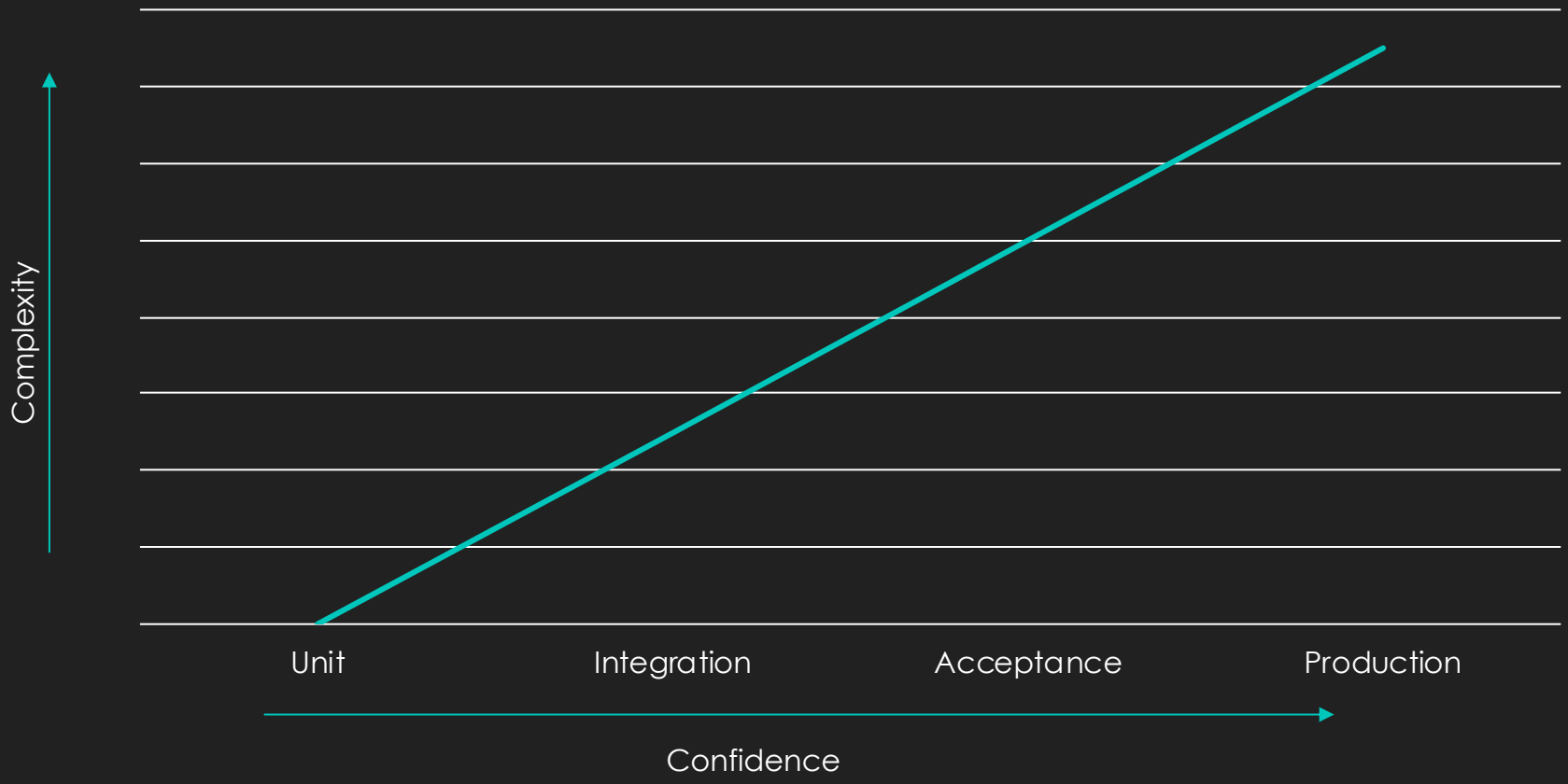
Ed Elliott | https://The.AgileSQL.club | ed@agilesql.co.uk

Slides: https://The.AgileSQL.club/etl-testing

Types of Tests

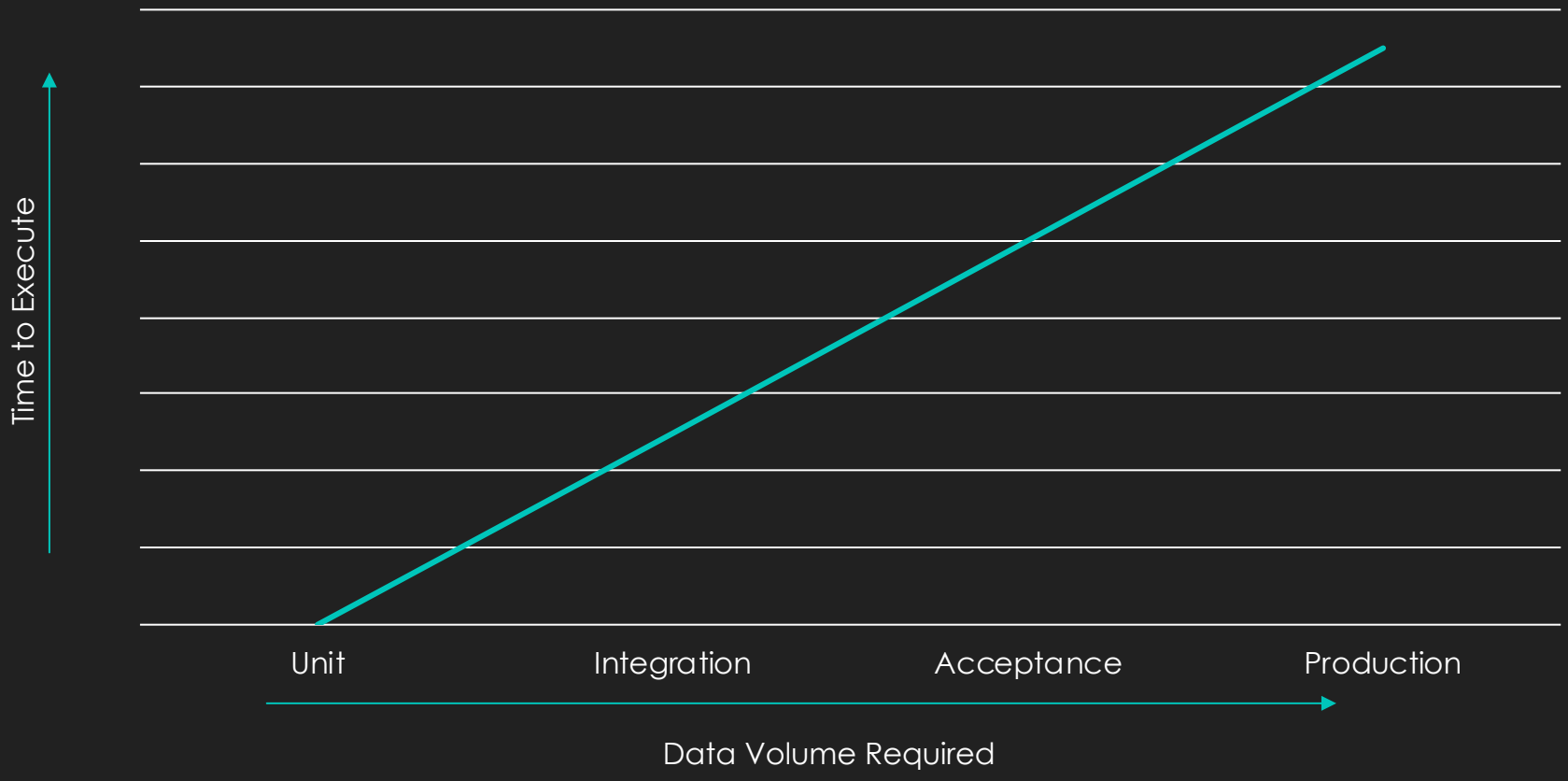
- Unit
- Integration
- Acceptance
- Data Quality
- Production

Complexity vs Confidence



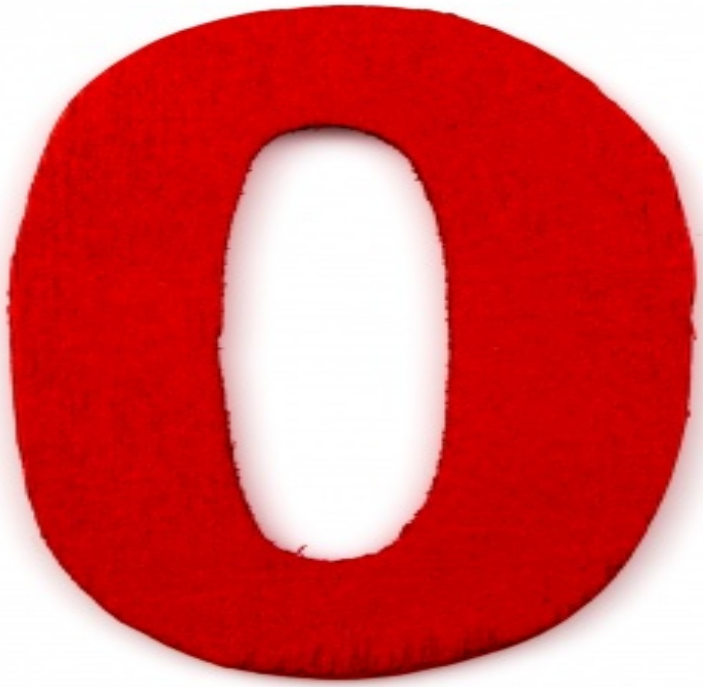
Ed Elliott | https://The.AgileSQL.club | ed@agilesql.co.uk

Time to Execute vs Data Volume Requirements



Ed Elliott | https://The.AgileSQL.club | ed@agilesql.co.uk

Unit Tests



Unit Tests

- Documentation
- Developer understanding
- Code behavior
- Don't need data (better if they don't have any data)

Unit Tests

- Are Specific:
  - Post Code Validator Checks:
    - Length is correct
    - Starts with two characters
    - Either two or three numbers
    - Ends with two characters
    - Poop Emoji is not valid
    - Negative Numbers are not valid

Unit Tests

- For SQL - only real choice is tSQLt – more on this later

Integration Tests

- Great starting point
- Shows that a process completes (probably successfully)
- Write some data / execute / Read some data

SSIS Integration Test - SETUP

1. DELETE data in DESTINSTION
2. DELETE data in SOURCE
3. WRITE data in SOURCE

Setup using SQL

```sql
DELETE FROM Source;
DELETE FROM Destination;

INSERT INTO Source
VALUES('known value 1', 1), ('known value 2', 2), ('known value 1000', 1000)
```

setup.cmd

```
REM Run SETUP scripts probably have %1 or %%1 in somewhere

sqlcmd -E -S server -i ./setup.sql
```

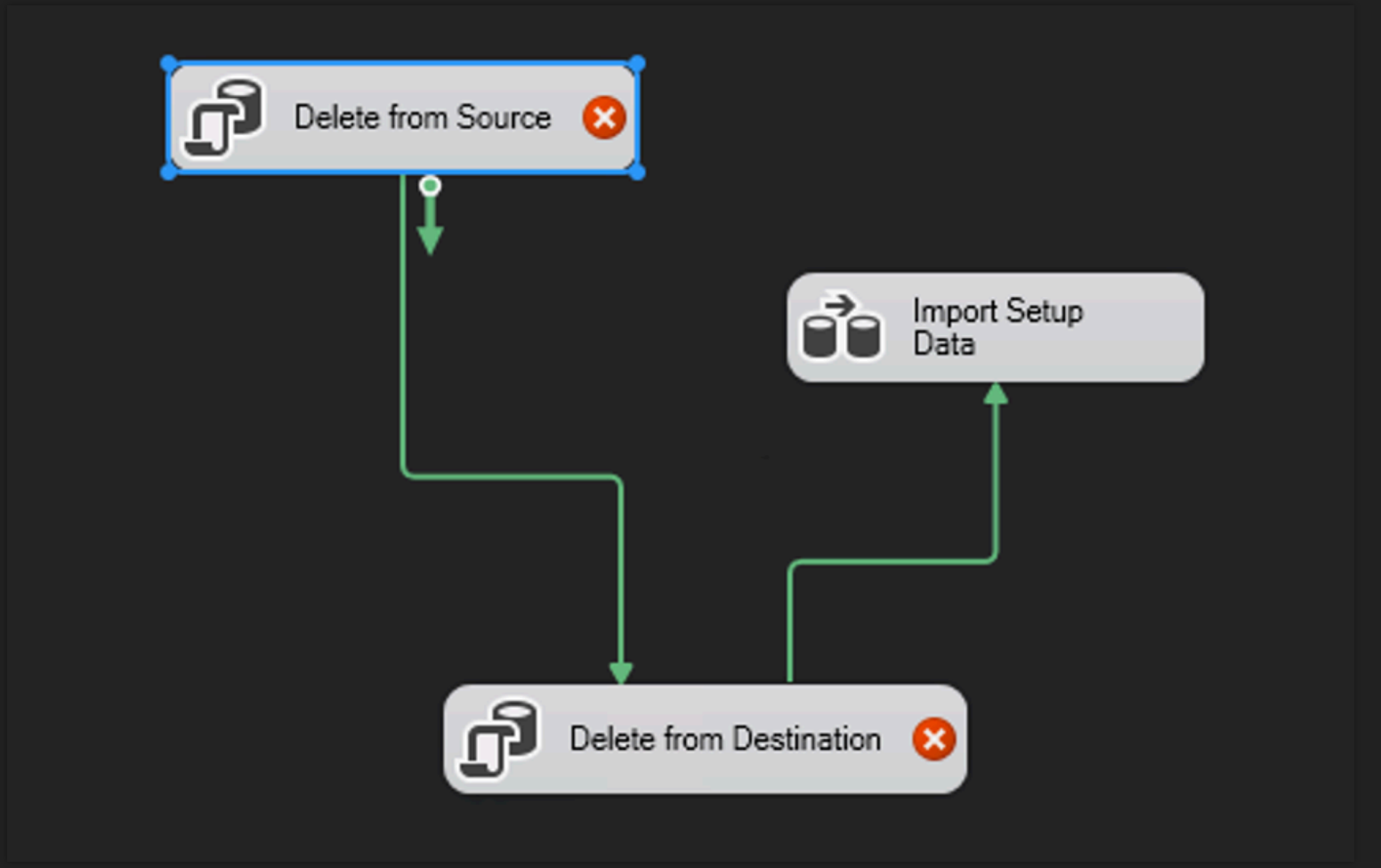
Setup using Powershell

setup.ps1

```powershell
Invoke-SqlCmd -E -S server -i -Q "DELETE FROM Source"
Invoke-SqlCmd -E -S server -i -Q "DELETE FROM Target"

Get-SetupRows | Insert-SetupRows
```

Setup using SSIS



Setup using BAK

```
USE [master]

RESTORE DATABASE [source] FROM  DISK = N'C
RESTORE DATABASE [target] FROM  DISK = N'C

GO
```

Setup

Sometimes life gives you <insert expletive>.

SSIS Integration Test - Execute

1. Run Package
2. Check DTSExec EXIT code
3. NEED SQL Server developer (probably)

SSIS Integration Test - Execute

SUCCESS != NOT “FAIL”

SUCCESS = “SUCCESS”

Execute SSIS using DTExec

```
START /WAIT dtexec /f "c:\pkgOne.dtsx" /l "DTS.LogProviderTextFile;c:\log.txt"
IF NOT ERROR_LEVEL 0 THEN GOTO PANIC_STATIONS
```

Execute SSIS using C#

```
using Microsoft.SqlServer.Management.IntegrationServices;
using System.Data.SqlClient;

namespace run_ssis_package
{
    class Program
    {
        static void Main(string[] args)
        {
            // Variables
            string targetServerName = "localhost";
            string folderName = "Project1Folder";
            string projectName = "Integration Services Project1";
            string packageName = "Package.dtsx";

            // Create a connection to the server
            string sqlConnectionString = "Data Source=" + targetServerName +
                ";Initial Catalog=master;Integrated Security=SSPI;";
            SqlConnection sqlConnection = new SqlConnection(sqlConnectionString);

            // Create the Integration Services object
            IntegrationServices integrationServices = new IntegrationServices(sqlConnection);

            // Get the Integration Services catalog
            Catalog catalog = integrationServices.Catalogs["SSISDB"];

            // Get the folder
            CatalogFolder folder = catalog.Folders[folderName];

            // Get the project
            ProjectInfo project = folder.Projects[projectName];

            // Get the package
            PackageInfo package = project.Packages[packageName];

            // Run the package
            package.Execute(false, null);

        }
    }
}
```

Execute SSIS using C# and DTExec

```
Process p = new Process();

// Redirect the output stream of the child process.

p.StartInfo.UseShellExecute = false;

p.StartInfo.RedirectStandardOutput = true;

p.StartInfo.FileName = @"C:\Program Files\Microsoft SQL
Server\90\DTS\Binn\DTExec.exe";

p.StartInfo.Arguments = "/F badfile";

p.Start();

Debug.WriteLine(p.StandardOutput.ReadToEnd());

p.WaitForExit();
```

SSIS Integration Test - Verify

1. Check LOGGING
2. Check data in DESTINATION

Verify using SQL

```
SELECT COUNT(*) FROM target.schema.table
    WHERE ConditionsMatch

Check some Row Counts

Check some specific values

Check some aggregations

Check something cool
```

Verify using …

ADF Integration Testing

- More likely write file / delete file / delete data
- More likely verify file contents

ADF Execute

```
$run = (Invoke-AzureRmDataFactoryV2Pipeline -ResourceGroupName "ADF" -
DataFactoryName "WikiADF" -PipelineName "DPWikisample")


while(-not(Get-AzureRmDataFactoryV2PipelineRun -ResourceGroupName "ADF" -
DataFactoryName "WikiADF" –PipelineRunId $run).Status –eq “Success”){
    Start-Sleep –Seconds BalanceBetweenAnnoyanceAndWastefulness
}
```

Integration Key Points

- 1 – 1000 Rows
- Command line OR API – ANYTHING IS Testable
- Show that one process works end to end

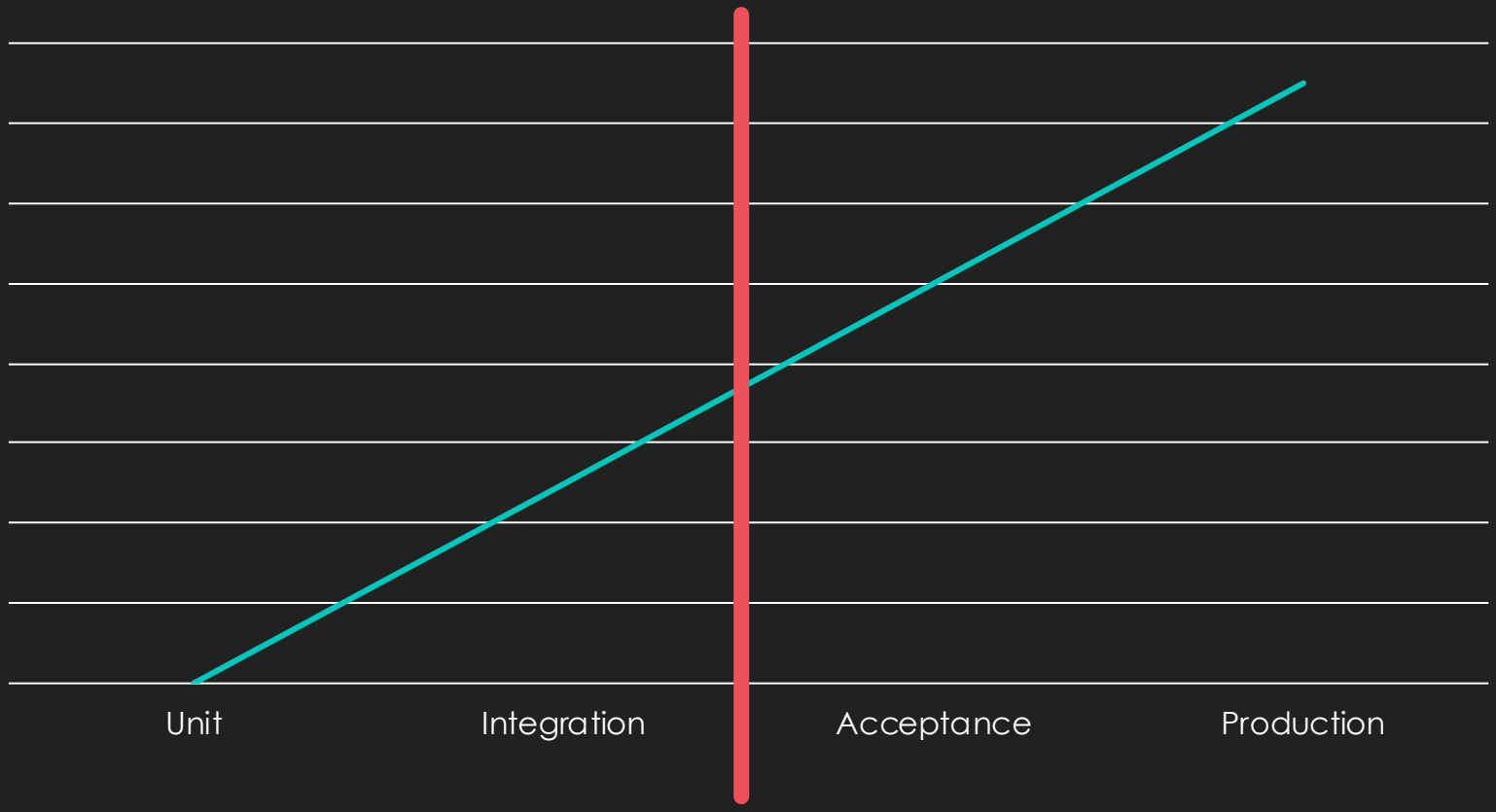
Acceptance Tests

MORE DATA, REALISTIC DATA with something pretty for the business to get excited about

Acceptance Tests

Show that the BUSINESS PROCESS is working as the BUSINESS NEEDS

Developer vs Business Audience



Ed Elliott | https://The.AgileSQL.club | ed@agilesql.co.uk

Types of code

- Business Logic
- Infrastructure

Unit Tests

What type of PIPELINE do YOU have?

Separating Business Logic Helps…

- Write UNIT Tests
- Code Review
- Developer Frustration Levels

Types of code / How to test

- Business Logic
  - Unit Tests
  - Integration Tests
  - Acceptance Tests
- Infrastructure
  - Integration Tests
  - Acceptance Tests

“EVERY ETL run has the potential to be on the receiving end of infinite monkeys trying to type out the works of Shakespeare” – ed elliott, 2020

Data Quality

- CONSTRAINTS
- Delta Lake Expectations (COMING SOON!)
- https://greatexpectations.io/
- Home grown checks

Great Expectations

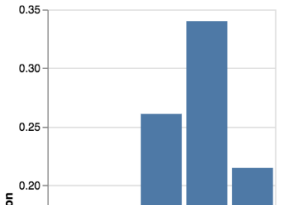
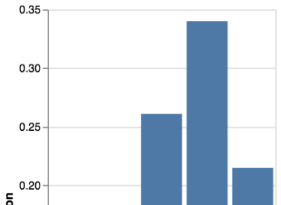
Table-Level Expectations

Status	Expectation	Observed Value
✓	Must have between `990` and `1010` rows.	1000
✓	Must have exactly `5` columns.	5
✓	Must have these columns in this order: `id`, `user_id`, `movie_id`, `rating`, `rated_at`	['id', 'user_id', 'movie_id', 'rating', 'rated_at']

rated_at

Status	Expectation	Observed Value
✓	values must never be null.	100% not null
✓	values must always be between `1996-09-19 20:05:10` and `1999-04-22 13:41:42`.	0% unexpected

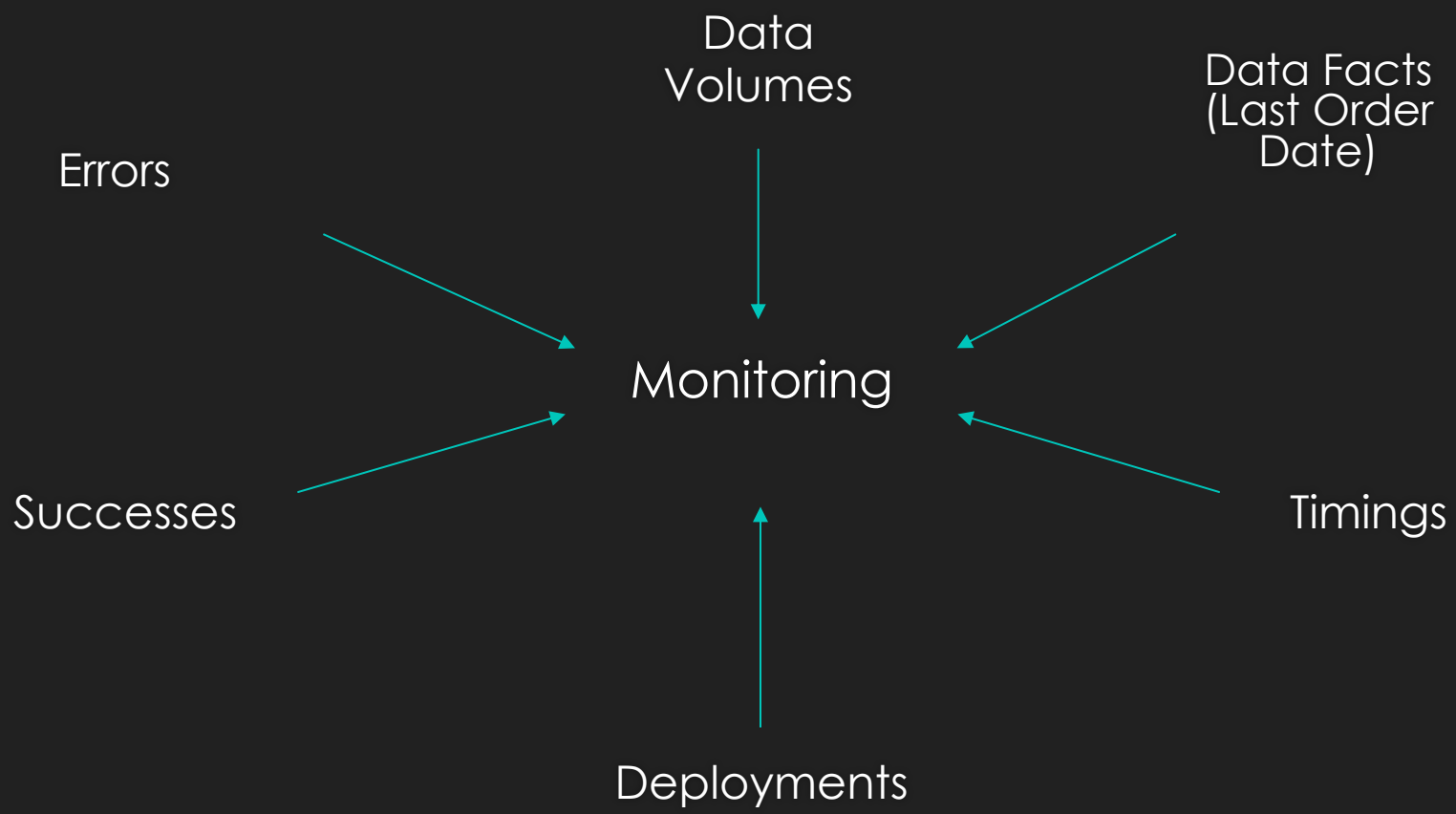
rating

Status	Expectation	Observed Value
✓	values must never be null.	100% not null
✓	distinct values must belong to this set: `1` `2` `3` `4` `5`.	[1, 2, 3, 4, 5]
✓	Kullback-Leibler (KL) divergence with respect to the following distribution must be lower than 0.6. 	KL Divergence: None (-infinity, infinity, or NaN) 

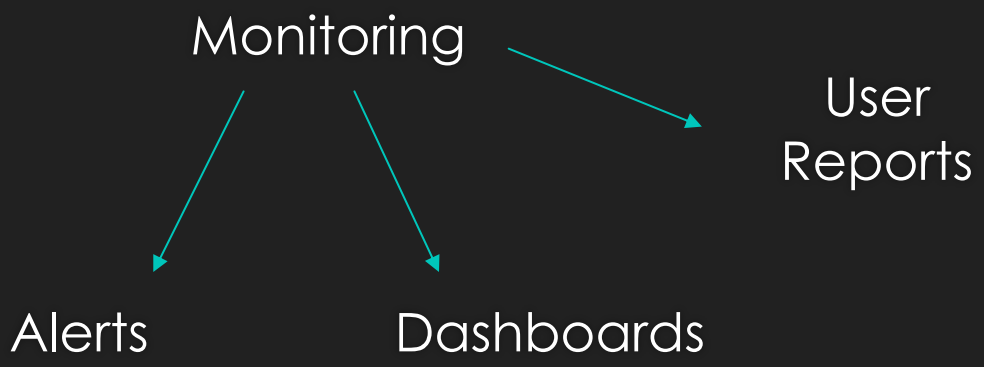
Simpler Checks

- Row Counts
- Averages
- Shape of the data

Monitoring


Data Volumes
Data Facts (Last Order Date)
Errors
Monitoring
Successes
Timings
Deployments

Monitoring



Types of code / How to test

- Business Logic
  - Unit Tests
  - Integration Tests
  - Acceptance Tests
- Infrastructure
  - Integration Tests
  - Acceptance Tests
- Upstream Data Quality
  - Monitoring
- Infrastructure
  - Monitoring

Where to start?

- Write 1 Test
- Write 1 Integration Test
- Maybe never unit tests
- Keep tests passing
- Include in dev pipeline