UDM Framework: Data Quality & Governance

Built-In Quality Assurance and Governance Features

Introduction

Data quality and governance are not afterthoughts in the UDM Framework—they're built into its core architecture. The Fallout system, source system traceability, and audit capabilities provide comprehensive data quality management and governance that meets regulatory and business requirements.

The Fallout System: Heart of Data Quality

Philosophy: Fail Gracefully, Track Everything

Traditional Data Quality Approach:

```
flowchart LR
    BadData[Bad Data] --> Fail[ETL Fails]
    Fail --> Stop[Pipeline Stops]
    Stop --> NoData[No Data Loaded]
    NoData --> Manual[Manual Investigation]
    Manual --> Fix[Fix]
    Fix --> Rerun[Rerun]

    style BadData fill:#ffcdd2
    style Fail fill:#f44336,color:#fff
    style Stop fill:#f44336,color:#fff
    style NoData fill:#ffcdd2
```

Problems:

- All-or-nothing: One bad record blocks thousands of good records
- Limited visibility: Error logs are cryptic and scattered
- Slow resolution: Manual investigation required for each failure
- Business impact: Reports missing data without explanation

UDM Fallout Approach:

```
flowchart LR
    BadData[Bad Data Detected] --> Log[Logged to Fallout]
    Log --> Continue[Pipeline Continues]
    Continue --> GoodData[Good Data Loaded]
    GoodData --> Track[Transparent Tracking]
    Track --> Analyze[Systematic Analysis]
    Analyze --> Resolve[Resolution]
    Resolve --> Reprocess[Auto-Reprocessing]

    style BadData fill:#fff4e1
    style Log fill:#e8f5e9
    style Continue fill:#c8e6c9
    style GoodData fill:#4caf50,color:#fff
    style Track fill:#e1f5ff
    style Analyze fill:#e1f5ff
    style Resolve fill:#c8e6c9
    style Reprocess fill:#4caf50,color:#fff
```

Benefits:

- Graceful degradation: Good data flows through, bad data tracked separately
- Complete visibility: Every issue logged with context
- Faster resolution: Systematic analysis and prioritization
- Business transparency: Stakeholders see what's missing and why

---

Fallout Architecture

Core Tables

Fallout.Fallout: Main tracking table

```
CREATE TABLE [Fallout].[Fallout](
    [FalloutId] [int] IDENTITY(1,1) NOT NULL,
    [ErrorCodeId] [int] NOT NULL,                    -- Type of error
    [SourceSystemId] [bigint] NOT NULL,              -- Which source
    [SourceSystemIdentifier] [nvarchar](100) NOT NULL, -- Which record
    [InsertDate] [datetime] NOT NULL,                -- When detected
    [SolveDate] [datetime] NULL,                     -- When resolved
    CONSTRAINT [PK_Fallout] PRIMARY KEY ([FalloutId])
)
```

Fallout.ErrorCode: Error taxonomy

```
CREATE TABLE [Fallout].[ErrorCode](
    [ErrorCodeId] [int] NOT NULL,
    [ErrorDescription] [nvarchar](200) NULL,
    [ErrorCategory] [nvarchar](50) NULL,
    [ResolutionGuidance] [nvarchar](max) NULL,
    CONSTRAINT [PK_ErrorCode] PRIMARY KEY ([ErrorCodeId])
)
```

Standard Error Codes

Referential Integrity (11-19):

- 11: Missing Foreign Key - Lead/Opportunity
- 12: Missing Foreign Key - Channel/Marketing
- 13: Missing Foreign Key - Status/Lookup
- 14: Missing Foreign Key - Product
- 15: Missing Foreign Key - Customer/Contact
- 16-19: Additional FK errors (domain-specific)

Data Format (20-29):

- 20: Invalid Data Type (type conversion failure)
- 21: Invalid Date Format
- 22: Invalid Numeric Value
- 23: Invalid Email Format
- 24: Invalid Phone Format
- 25-29: Additional format errors

Business Logic (30-39):

- 30: Business Rule Violation - General
- 31: Negative Amount (where positive expected)
- 32: Future Date (where past expected)
- 33: Invalid Status Transition
- 34: Duplicate Transaction
- 35-39: Additional business rule errors

Data Quality (40-49):

- 40: Duplicate Record
- 41: Missing Required Field
- 42: Orphaned Record
- 43: Data Inconsistency
- 44-49: Additional quality issues

Fallout Detection Patterns

Pattern 1: Missing Foreign Key

```
// ADF Dataflow transformation
Records, DimensionTable lookup(
    Records@FKColumn == DimensionTable@IdColumn,
    multiple: false,
    pickup: 'first'
) ~> LookupDimension

// Split: Found vs. Not Found
LookupDimension split(
    isNull(DimensionTable@IdColumn),
    disjoint: false
) ~> SplitOnFK@(FalloutPath, SuccessPath)

// Route to fallout
SplitOnFK@FalloutPath derive(
    ErrorCodeId = 15,  // Missing Customer FK
    InsertDate = currentTimestamp()
) ~> AddFalloutMetadata

// Deduplicate (don't log same error twice)
```

```
AddFalloutMetadata, ExistingFallout exists(
    AddFalloutMetadata@ErrorCodeId == ExistingFallout@ErrorCodeId
    && AddFalloutMetadata@SourceSystemId == ExistingFallout@SourceSystem
    && AddFalloutMetadata@SourceSystemIdentifier == ExistingFallout@Sourc
    && isNull(ExistingFallout@SolveDate),
    negate: true
) ~> NewFalloutOnly

NewFalloutOnly sink(...) ~> FalloutSink

// Continue with valid records
SplitOnFK@SuccessPath ~> NextTransformation
```

Pattern 2: Business Rule Violation

```
-- Stored procedure validation before dataflow
CREATE PROCEDURE [dbo].[ValidateBusinessRules]
    @SourceSystemId BIGINT
AS
BEGIN
    -- Detect negative amounts (where should be positive)
    INSERT INTO Fallout.Fallout (
        ErrorCodeId,
        SourceSystemId,
        SourceSystemIdentifier,
        InsertDate
    )
    SELECT DISTINCT
        31,  -- Negative amount error
        @SourceSystemId,
        f.SourceSystemIdentifier,
        GETDATE()
    FROM Flow.Sales_Order_16 f
    LEFT JOIN Fallout.Fallout existing
        ON existing.SourceSystemId = @SourceSystemId
        AND existing.SourceSystemIdentifier = f.SourceSystemIdentifier
        AND existing.ErrorCodeId = 31
        AND existing.SolveDate IS NULL
    WHERE f.Amount < 0
      AND existing.FalloutId IS NULL  -- Not already in fallout
```

```
    -- Remove invalid records from Flow
    DELETE FROM Flow.Sales_Order_16
    WHERE Amount < 0
END
```

Fallout Monitoring and Reporting

Daily Fallout Dashboard

Key Metrics:

```
-- Total open fallout by source
CREATE VIEW [Fallout].[VW_FalloutBySource] AS
SELECT
    ss.SourceDescription,
    COUNT(*) as OpenFalloutCount,
    MIN(f.InsertDate) as OldestFallout,
    MAX(f.InsertDate) as LatestFallout
FROM Fallout.Fallout f
JOIN Reference.SourceSystem ss ON f.SourceSystemId = ss.SourceSystemId
WHERE f.SolveDate IS NULL
GROUP BY ss.SourceDescription
GO

-- Fallout by error category
CREATE VIEW [Fallout].[VW_FalloutByCategory] AS
SELECT
    ec.ErrorCategory,
    ec.ErrorDescription,
    COUNT(*) as FalloutCount,
    CAST(100.0 * COUNT(*) / SUM(COUNT(*)) OVER () AS DECIMAL(5,2)) as Per
FROM Fallout.Fallout f
JOIN Fallout.ErrorCode ec ON f.ErrorCodeId = ec.ErrorCodeId
WHERE f.SolveDate IS NULL
GROUP BY ec.ErrorCategory, ec.ErrorDescription
GO

-- Fallout aging analysis
CREATE VIEW [Fallout].[VW_FalloutAging] AS
```

```
SELECT
    CASE
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 1 THEN '0-1 days'
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 7 THEN '2-7 days'
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 30 THEN '8-30 days'
        ELSE '30+ days'
    END as AgeRange,
    COUNT(*) as FalloutCount
FROM Fallout.Fallout
WHERE SolveDate IS NULL
GROUP BY
    CASE
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 1 THEN '0-1 days'
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 7 THEN '2-7 days'
        WHEN DATEDIFF(DAY, InsertDate, GETDATE()) <= 30 THEN '8-30 days'
        ELSE '30+ days'
    END
GO
```

Daily Fallout Report (for data stewards):

```
-- Email-ready fallout summary
SELECT
    ss.SourceDescription as [Source System],
    ec.ErrorDescription as [Error Type],
    COUNT(*) as [Record Count],
    MIN(f.InsertDate) as [First Occurrence],
    MAX(f.InsertDate) as [Latest Occurrence],
    DATEDIFF(DAY, MIN(f.InsertDate), GETDATE()) as [Days Open]
FROM Fallout.Fallout f
JOIN Reference.SourceSystem ss ON f.SourceSystemId = ss.SourceSystemId
JOIN Fallout.ErrorCode ec ON f.ErrorCodeId = ec.ErrorCodeId
WHERE f.SolveDate IS NULL
  AND f.InsertDate >= DATEADD(DAY, -30, GETDATE())  -- Last 30 days
GROUP BY ss.SourceDescription, ec.ErrorDescription
HAVING COUNT(*) >= 10  -- Only significant volumes
ORDER BY COUNT(*) DESC
```

Fallout Trends Analysis

Quality Improvement Tracking:

```
-- Monthly fallout rate trend
SELECT
    YEAR(f.InsertDate) as Year,
    MONTH(f.InsertDate) as Month,
    COUNT(*) as FalloutCount,
    (SELECT COUNT(*)
     FROM Sales.Order o
     WHERE YEAR(o.OrderDate) = YEAR(f.InsertDate)
       AND MONTH(o.OrderDate) = MONTH(f.InsertDate)
    ) as TotalRecords,
    CAST(100.0 * COUNT(*) /
        NULLIF((SELECT COUNT(*)
                FROM Sales.Order o
                WHERE YEAR(o.OrderDate) = YEAR(f.InsertDate)
                  AND MONTH(o.OrderDate) = MONTH(f.InsertDate)), 0)
    AS DECIMAL(5,2)) as FalloutRate
FROM Fallout.Fallout f
WHERE f.InsertDate >= DATEADD(MONTH, -12, GETDATE())
GROUP BY YEAR(f.InsertDate), MONTH(f.InsertDate)
ORDER BY Year, Month
```

Fallout Resolution Workflow

```
flowchart TD
    Start[Daily Fallout Report] --> Prioritize[Step 1: Prioritize<br/>By

    Prioritize --> Analyze[Step 2: Root Cause Analysis<br/>Query Fallout

    Analyze --> Decision{Issue Type?}

    Decision -->|Missing FK| FixDim[Fix Dimension Data<br/>Load missing r
    Decision -->|Data Quality| FixSource[Fix at Source<br/>Correct source
    Decision -->|Business Rule| AdjustRule[Adjust Business Rule<br/>Updat

    FixDim --> Reprocess[Step 3: Reprocess<br/>Re-run ETL Pipeline]
    FixSource --> Reprocess
```

```
    AdjustRule --> Reprocess

    Reprocess --> Check{Records<br/>Load Successfully?}

    Check -->|Yes| AutoResolve[Step 4: Auto-Resolution<br/>SolveDate Upda
    Check -->|No| ReAnalyze[Re-analyze<br/>Different root cause]

    ReAnalyze --> Analyze

    AutoResolve --> Monitor[Step 5: Monitor<br/>Track Resolution Trends]

    style Start fill:#e1f5ff
    style Prioritize fill:#fff4e1
    style Analyze fill:#e8f5e9
    style Reprocess fill:#f3e5f5
    style AutoResolve fill:#c8e6c9
    style Monitor fill:#4caf50,color:#fff
```

Step 1: Identify High-Priority Fallout

Prioritization Criteria:

1. **Volume**: High record count indicates systemic issue
2. **Age**: Old fallout suggests resolution challenges
3. **Business Impact**: Affects critical reports or KPIs
4. **Trend**: Growing fallout requires immediate attention

Prioritization Query:

```
SELECT
    f.ErrorCodeId,
    ec.ErrorDescription,
    ss.SourceDescription,
    COUNT(*) as RecordCount,
    AVG(DATEDIFF(DAY, f.InsertDate, GETDATE())) as AvgAgeDays,
    CASE
        WHEN COUNT(*) > 1000 THEN 'High'
        WHEN COUNT(*) > 100 THEN 'Medium'
        ELSE 'Low'
    END as VolumePriority,
    CASE
```

```
        WHEN AVG(DATEDIFF(DAY, f.InsertDate, GETDATE())) > 30 THEN 'High'
        WHEN AVG(DATEDIFF(DAY, f.InsertDate, GETDATE())) > 7 THEN 'Medium'
        ELSE 'Low'
    END as AgePriority
FROM Fallout.Fallout f
JOIN Fallout.ErrorCode ec ON f.ErrorCodeId = ec.ErrorCodeId
JOIN Reference.SourceSystem ss ON f.SourceSystemId = ss.SourceSystemId
WHERE f.SolveDate IS NULL
GROUP BY f.ErrorCodeId, ec.ErrorDescription, ss.SourceDescription
ORDER BY COUNT(*) DESC
```

Step 2: Root Cause Analysis

For Missing FK Errors (11-19):

```
-- Analyze missing dimension values
SELECT
    f.SourceSystemIdentifier,
    s.ColumnThatShouldMap,
    COUNT(*) as Occurrences
FROM Fallout.Fallout f
JOIN DS_Stage_Source.dbo.SourceTable s
    ON f.SourceSystemIdentifier = s.SourceId
WHERE f.ErrorCodeId = 15  -- Missing Customer FK
  AND f.SolveDate IS NULL
GROUP BY f.SourceSystemIdentifier, s.ColumnThatShouldMap
ORDER BY COUNT(*) DESC
```

Common Root Causes:

- Dimension data not yet loaded (timing issue)
- Source uses different identifier than DWH
- Data quality issue at source (invalid FKs)
- Mapping logic error in dataflow

For Business Rule Violations (30-39):

```
-- Review rule violations
SELECT TOP 100
    f.SourceSystemIdentifier,
```

```
    s.Amount,
    s.OrderDate,
    s.Status
FROM Fallout.Fallout f
JOIN DS_Stage_Source.dbo.Orders s
    ON f.SourceSystemIdentifier = s.OrderId
WHERE f.ErrorCodeId = 31  -- Negative amount
  AND f.SolveDate IS NULL
ORDER BY f.InsertDate DESC
```

Step 3: Remediation

Option A: Fix at Source

Best practice: Correct data in source system

- Sustainable long-term fix
- Improves overall data quality
- Prevents recurrence

Example:

```
-- After fixing in source, verify fix
SELECT * FROM DS_Stage_Source.dbo.Customers
WHERE CustomerId IN (
    SELECT SourceSystemIdentifier
    FROM Fallout.Fallout
    WHERE ErrorCodeId = 15 AND SolveDate IS NULL
)
```

Option B: Add Missing Dimension Data

For missing FKs: Load the missing dimension records

```
-- Insert missing customers
INSERT INTO Contact.Customer (
    CustomerNK,
    CustomerName,
    ValidFromDate,
    ValidToDate,
```

```
    SourceSystemId,
    SourceSystemIdentifier
)
SELECT DISTINCT
    MissingCustomerId,
    'Unknown Customer - ' + MissingCustomerId,
    '1900-01-01',
    '9999-12-31',
    99,  -- Special SourceSystemId for "Unknown" records
    MissingCustomerId
FROM (
    SELECT DISTINCT s.CustomerId as MissingCustomerId
    FROM Fallout.Fallout f
    JOIN DS_Stage_Source.dbo.Orders s
        ON f.SourceSystemIdentifier = s.OrderId
    WHERE f.ErrorCodeId = 15 AND f.SolveDate IS NULL
) missing
WHERE NOT EXISTS (
    SELECT 1 FROM Contact.Customer c
    WHERE c.CustomerNK = missing.MissingCustomerId
)
```

Option C: Adjust Business Rules

If rule is too strict or incorrect:

```
-- Document rule change
INSERT INTO [Logging].[BusinessRuleChanges] (
    RuleId,
    OldRule,
    NewRule,
    Reason,
    ApprovedBy,
    ChangeDate
) VALUES (
    'NEG_AMOUNT',
    'Amount must be positive',
    'Amount must be >= -1000 (allow small corrections)',
    'Business requested to allow small negative adjustments',
    'Data Governance Committee',
    GETDATE()
```

```
)

-- Update dataflow or validation logic accordingly
```

Step 4: Reprocessing

Manual Reprocessing:

```
-- Repopulate Flow table for affected records
DECLARE @SourceSystemId BIGINT = 42

-- Clear Flow table
EXEC('TRUNCATE TABLE Flow.Sales_Order_' + @SourceSystemId)

-- Re-extract fallout records specifically
INSERT INTO Flow.Sales_Order_42
SELECT
    -- Apply SourceColumnMapping
FROM DS_Stage_Source.dbo.Orders s
WHERE s.OrderId IN (
    SELECT SourceSystemIdentifier
    FROM Fallout.Fallout
    WHERE SourceSystemId = @SourceSystemId
      AND ErrorCodeId IN (11, 12, 13)  -- FK errors
      AND SolveDate IS NULL
)

-- Run ADF pipeline (manually or via trigger)
```

Automatic Resolution:

Records automatically resolve when they pass validation:

- Fallout record remains with SolveDate = NULL
- Pipeline re-runs (scheduled or manual)
- Record now passes validation
- Loads to DWH successfully
- Automated process updates `SolveDate`

```
-- Automated resolution trigger (runs after successful pipeline)
CREATE PROCEDURE [Fallout].[MarkResolvedFallout]
    @SourceSystemId BIGINT
AS
BEGIN
    UPDATE f
    SET SolveDate = GETDATE()
    FROM Fallout.Fallout f
    JOIN Reference.SourceSystem ss
        ON f.SourceSystemId = ss.SourceSystemId
    WHERE f.SourceSystemId = @SourceSystemId
      AND f.SolveDate IS NULL
      AND EXISTS (
          -- Record now exists in DWH
          SELECT 1
          FROM [Sales].[Order] o
          WHERE o.SourceSystemId = f.SourceSystemId
            AND o.SourceSystemIdentifier = f.SourceSystemIdentifier
      )
END
```

Source System Traceability

Complete Data Lineage

Every record tracks its origin:

```
-- Standard columns on all DWH tables
[SourceSystemId] [bigint] NOT NULL          -- Which source system
[SourceSystemIdentifier] [nvarchar](100)    -- PK in source system
```

Benefits:

1. **Impact Analysis**: Know which reports affected by source changes
2. **Data Reconciliation**: Trace discrepancies back to source
3. **Audit Compliance**: Prove data provenance
4. **Multi-Source Support**: Same entity from multiple sources

Lineage Queries

Find source of specific record:

```
SELECT
    o.OrderId,
    o.OrderDate,
    ss.SourceDescription,
    ss.SourceDatabase,
    ss.SourceSchema,
    ss.SourceTable,
    o.SourceSystemIdentifier as SourcePK
FROM Sales.Order o
JOIN Reference.SourceSystem ss ON o.SourceSystemId = ss.SourceSystemId
WHERE o.OrderId = 12345
```

Count records by source:

```
SELECT
    ss.SourceDescription,
    COUNT(*) as RecordCount,
    MIN(o.OrderDate) as EarliestOrder,
    MAX(o.OrderDate) as LatestOrder
FROM Sales.Order o
JOIN Reference.SourceSystem ss ON o.SourceSystemId = ss.SourceSystemId
GROUP BY ss.SourceDescription
ORDER BY RecordCount DESC
```

Trace back to raw source data:

```
-- Complete lineage
DECLARE @OrderId BIGINT = 12345

SELECT
    'DWH' as Layer,
    o.OrderDate,
    o.Amount,
    o.CustomerName
FROM DS_DWH.Sales.Order o
```

```
WHERE o.OrderId = @OrderId

UNION ALL

SELECT
    'Stage' as Layer,
    s.OrderDate,
    s.Amount,
    s.CustomerName
FROM DS_DWH.Sales.Order o
JOIN DS_DWH.Reference.SourceSystem ss ON o.SourceSystemId = ss.SourceSyst
JOIN DS_Stage_Source.dbo.Orders s
    ON s.OrderId = o.SourceSystemIdentifier
WHERE o.OrderId = @OrderId

-- Can extend to History tables for complete timeline
```

Audit Trails and Compliance

Timeslice-Based Audit

Point-in-time reconstruction:

```
-- What was Customer #123's data on 2023-06-15?
SELECT
    CustomerName,
    Email,
    Phone,
    Address
FROM Contact.Customer
WHERE CustomerNK = '123'
  AND '2023-06-15' BETWEEN ValidFromDate AND ValidToDate
```

Change history:

```
-- All changes to Customer #123
SELECT
    CustomerId,
```

```
    CustomerName,
    Email,
    ValidFromDate,
    ValidToDate,
    CASE
        WHEN ValidToDate = '9999-12-31' THEN 'Current'
        ELSE 'Historical'
    END as Status
FROM Contact.Customer
WHERE CustomerNK = '123'
ORDER BY ValidFromDate
```

History Tables in Stage

Complete change log:

```
-- All changes ever made to source record
SELECT
    ChangeType,
    ChangeTime,
    ProductName,
    Price,
    ModifiedBy
FROM DS_Stage_Source.History.Products
WHERE ProductId = 456
ORDER BY ChangeTime DESC
```

Regulatory Compliance:

- GDPR: Prove when data was received, modified, deleted
- SOX: Audit trail for financial data
- HIPAA: Track access and modifications to sensitive data

Retention Policies

Fallout Retention:

```
-- Archive old resolved fallout (keep for audit period)
INSERT INTO Fallout.Fallout_Archive
SELECT * FROM Fallout.Fallout
```

```
WHERE SolveDate IS NOT NULL
  AND SolveDate < DATEADD(YEAR, -2, GETDATE())

DELETE FROM Fallout.Fallout
WHERE SolveDate IS NOT NULL
  AND SolveDate < DATEADD(YEAR, -2, GETDATE())
```

Historical Data Retention:

```
-- Archive old timeslices (beyond regulatory requirement)
-- Move to partitioned archive table or compress
ALTER TABLE Contact.Customer REBUILD PARTITION = @OldPartition
WITH (DATA_COMPRESSION = PAGE)
```

Data Quality Metrics and KPIs

Quality Scorecard

Source System Quality Score:

```
CREATE VIEW [Fallout].[VW_SourceSystemQualityScore] AS
WITH SourceMetrics AS (
    SELECT
        ss.SourceSystemId,
        ss.SourceDescription,
        COALESCE(loaded.RecordCount, 0) as LoadedRecords,
        COALESCE(fallout.FalloutCount, 0) as FalloutRecords
    FROM Reference.SourceSystem ss
    LEFT JOIN (
        SELECT SourceSystemId, COUNT(*) as RecordCount
        FROM Sales.Order
        WHERE OrderDate >= DATEADD(MONTH, -1, GETDATE())
        GROUP BY SourceSystemId
    ) loaded ON ss.SourceSystemId = loaded.SourceSystemId
    LEFT JOIN (
        SELECT SourceSystemId, COUNT(*) as FalloutCount
        FROM Fallout.Fallout
        WHERE InsertDate >= DATEADD(MONTH, -1, GETDATE())
```

```
        GROUP BY SourceSystemId
    ) fallout ON ss.SourceSystemId = fallout.SourceSystemId
)
SELECT
    SourceSystemId,
    SourceDescription,
    LoadedRecords,
    FalloutRecords,
    LoadedRecords + FalloutRecords as TotalRecords,
    CAST(100.0 * LoadedRecords /
        NULLIF(LoadedRecords + FalloutRecords, 0) AS DECIMAL(5,2)) as Qua
    CASE
        WHEN CAST(100.0 * LoadedRecords / NULLIF(LoadedRecords + FalloutR
        WHEN CAST(100.0 * LoadedRecords / NULLIF(LoadedRecords + FalloutR
        WHEN CAST(100.0 * LoadedRecords / NULLIF(LoadedRecords + FalloutR
        ELSE 'Needs Improvement'
    END as QualityRating
FROM SourceMetrics
GO
```

Target KPIs:

- Quality Score > 95% (< 5% fallout rate)
- Fallout resolution time < 48 hours
- No fallout older than 30 days
- 100% source system traceability

Continuous Monitoring

Automated Alerts:

```
-- Daily quality check (run via scheduled job)
DECLARE @FalloutThreshold INT = 100
DECLARE @QualityThreshold DECIMAL(5,2) = 90.0

IF EXISTS (
    SELECT 1
    FROM Fallout.VW_SourceSystemQualityScore
    WHERE QualityScore < @QualityThreshold
       OR FalloutRecords > @FalloutThreshold
)
```

```
BEGIN
    -- Send alert email
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'DWH Alerts',
        @recipients = 'data-stewards@company.com',
        @subject = 'Data Quality Alert: Threshold Exceeded',
        @body = 'One or more sources have exceeded quality thresholds. Re
        @importance = 'High'
END
```

Governance Framework

Data Stewardship Roles

Data Steward Responsibilities:

1. Monitor daily fallout reports
2. Prioritize fallout resolution
3. Coordinate with source system owners
4. Approve business rule changes
5. Maintain data quality documentation

Source System Owner Responsibilities:

1. Maintain source data quality
2. Respond to fallout investigations
3. Implement fixes at source
4. Communicate schema changes
5. Provide subject matter expertise

Data Governance Committee:

1. Set quality standards and KPIs
2. Approve policy changes
3. Resolve escalated issues
4. Review quarterly metrics
5. Prioritize framework enhancements

Standard Operating Procedures

SOP 1: Daily Fallout Review

- Time: 9:00 AM daily
- Owner: Data Steward
- Process:
    1. Review overnight fallout report
    2. Identify new high-priority issues
    3. Assign to source system owners
    4. Update ticket system

SOP 2: Root Cause Analysis

- Trigger: Fallout volume > 100 records or age > 7 days
- Owner: Data Steward + Source System Owner
- Process:
    1. Extract sample records
    2. Analyze source data
    3. Identify root cause
    4. Document findings
    5. Propose remediation

SOP 3: Business Rule Change

- Trigger: Proposed rule modification
- Owner: Data Governance Committee
- Process:
    1. Submit change request with justification
    2. Impact analysis (affect how many records?)
    3. Committee review and approval
    4. Implementation
    5. Testing and validation
    6. Documentation update

Best Practices

Design-Time Best Practices

1. **Define Error Codes Early**: Standardize error taxonomy

2. **Implement Fallout for All FKs**: Don't let missing FKs block pipelines
3. **Deduplicate Fallout**: Check for existing fallout before inserting
4. **Meaningful Error Messages**: Include context in ErrorDescription

Runtime Best Practices

1. **Monitor Fallout Daily**: Don't let it accumulate
2. **Resolve Systematically**: Prioritize by volume and impact
3. **Fix at Source**: Sustainable long-term solution
4. **Document Patterns**: Create runbooks for common issues

Organizational Best Practices

1. **Assign Clear Ownership**: Every source has a designated owner
2. **Regular Review Meetings**: Weekly fallout review with stakeholders
3. **Quality Metrics in Dashboards**: Make quality visible to all
4. **Continuous Improvement**: Track and celebrate quality improvements

---

Conclusion

The UDM Framework's data quality and governance features provide:

1. **Visibility**: Complete transparency into data quality issues
2. **Resilience**: Graceful degradation when issues occur
3. **Traceability**: Full audit trail from source to consumption
4. **Accountability**: Clear ownership and resolution workflows
5. **Compliance**: Built-in audit capabilities for regulations
6. **Continuous Improvement**: Metrics to track quality over time

These features transform data quality from a reactive problem into a proactive, manageable process, enabling organizations to maintain trustworthy data at scale.

Next: 08_Case_Study_Pricewise.md - Real-world implementation example