Introduction
- Strategy 1 - Improve Quality of Testing
- Strategy 2 - Reduce the Prevalence of Errors
- Strategy 3 - Reduce the Need for Testing
- Strategy 4 - Automate Testing
- Strategy 5 - Integrate Continuous Testing
- Conclusion
Success of data warehousing projects is based on trust of the business consumers. It doesn’t matter if the data is “correct” because perception is everything. Compromising trust can permanently damage the viability of the initiative.

The risk of not doing proper testing:

1\textsuperscript{st} Consequence: Users don’t trust or can’t use the information the DW.
2\textsuperscript{nd} Consequence: Users revert to old methods of answering questions (report scraping, Excel)
3\textsuperscript{rd} Consequence: Business leaders lose interest and/or momentum.
4\textsuperscript{th} Consequence: Project stops getting resources, business support and funding.

*The initiative is effectively paralyzed until the next Data Warehouse reboot comes along.*

How many times have we heard: “This is the third attempt here. I hope you get it right this time.”

Pay any price (testing) to gain Business Consumer’s confidence.
Why So Difficult?

Testing and development of data warehousing differs significantly from traditional application development. Designers must often deal with ambiguous user requirements, complex and poor quality source data, lack of understanding of source data by IT. Testers must have excellent SQL skills to perform testing across data sets – an uncommon skillset, indeed.

**Data Warehouse development is fundamentally different from application development**

- Volatile source data cardinality (uniqueness assumption may be invalid)
- Volatile source structure
- Addresses complex integration
- Manages late-arriving data.
- Data set transformation vs. data entry

**Data Warehouse testing resources are hard to find!**

- Traditional QA pools are mostly useless.
- Expert SQL skills are a prized possession
- It takes a special kind of person...

**Business Consumers have difficulty articulating requirements**

- “Just give me all the data”
- “I’ll know what I need when I see it.”
- Often not aware of the complexities in their own business.
Agenda

- Introduction
- **Strategy 1 - Improve Quality of Testing**
- Strategy 2 - Reduce the Prevalence of Errors
- Strategy 3 - Reduce the Need for Testing
- Strategy 4 - Automate Testing
- Strategy 5 - Integrate Continuous Testing
- Conclusion
Who Is Our Audience

Testing should focus on delivering capability to the business consumer, first and foremost. This means that we need to get our eyes of data and onto business process. Too many IT managers believe that their company needs a data warehouse, when in fact what they need is trustworthy information on which to base decisions.

Fundamentally, are we asking the right questions?

It’s not about the code. It’s not about report layouts. It’s not about the data.

It’s about the users:

• Knowing that they can trust information as true. Their job and company success are on the line.
  — Reconciles with their understanding of the business.
  — Makes sense semantically.
  — Knowing that information is up-to-date (or at least how up-to-date)
• Knowing that they can access the system when they need to.
• Knowing that they can locate the information they need.
• Feeling that it can answer the questions they need to ask.

What users really want...

Users don’t want data warehouses. Users don’t want reports.

*They want trustworthy information on which to base decisions.*
The “Data Warehouse” must include all components that collectively deliver business capability. It’s possible to have correctly developed reports with bad data behind them and vice-versa. Both are separate components and must be tested independently to ensure a quality solution.

What is the “Data Warehouse”?

The BI ecosystem that delivers Business Consumer capability and experience...
Two Stakeholders

The Business Consumer should rightly be top priority in determining requirements. But the data warehouse can live without the business consumer and still must be correct in its management of data. This means that the BI Architect – who determines the structure and rules that govern the DW – is a de facto stakeholder and driver of requirements.

Business Consumer

The Business Consumer’s capability of decision-making is the top-line requirement. What questions must we ask?

What is the proper... Context? Application? Reconciliation? Business Rules? Other Criteria?

BI Architect

The BI Architect’s vision of how data (and data quality) is managed is a reflection of business consumer requirements. But they are different requirements and require another set of questions be asked:

How is the data... Extracted, Transformed, Persisted, Uniquely Identified, Integrated, Indexed, Purged, Backed-Up, Validated?
Introducing Risk

While the stakeholders drive requirements, risk to the ecosystem also drives requirements. “Risk-based testing” is more about testing the requirements introduced by unacceptable risk than it is testing only risky code.
Risk Based Testing

Assessing what risks are unacceptable requires us to ponder concepts such as risk horizon, risk impact and our definition of “failure”. The assumption should be that failure of any transformation in an ETL spec is unacceptable or it shouldn’t have been there in the first place. Yet there are other risks that drive unexpected requirements: how does the architecture change if you know a star developer might not be on the project in 6 months?

“...prioritizes the tests of features and functions based on the risk of their failure - a function of their importance and likelihood or impact of failure”

What is our threshold for Unacceptable Risk?

– What is “failure”...it’s about the whole BI ecosystem.
– What is the average life of a DW implementation? 5-7 years?
– What’s your number? 1%, 10%, 20% likelihood?

Considerations:

– How likely is it that a data element will be used?
– How likely is it that components failure will occur or even cascade? ...think Fukushima.
– How likely is it that a network connection will fail during a load?
– How likely is it that a key developer will leave the project?
– How likely is that the infrastructure will fail?
– How likely is it that an asteroid will hit the primary data center?
Action Items

1. Make the goal of BI to deliver capability, not functionality
   • Change the questions you ask users – understand decisions that want to make from the information.
   • Reports must be tested in context of how they will be used.

2. Recognize that the entire BI ecosystem is critical.
   • Non-Functional Requirements (latency, performance, availability, ease-of-use)
   • Relationship between information and business process.
   • …Every step in between

3. Accept that the BI Architect is a kind of stakeholder and that data is an asset. The architecture (data model, flow, non-functional requirements) drives it’s own risk and requirements.

4. Include BI ecosystem risks as requirements. Recognize that this is not just about ETL.

5. Gather requirements around all three: Capability, Ecosystem, Architecture.
Agenda

- Introduction
- Strategy 1 - Improve Quality of Testing
- **Strategy 2 - Reduce the Prevalence of Errors**
- Strategy 3 - Reduce the Need for Testing
- Strategy 4 - Automate Testing
- Strategy 5 - Integrate Continuous Testing
- Conclusion
Data Structure

Databases provide the single most powerful tool for preventing errors while still in development. This is because unique constraint violation in the target is so easy to induce – it happens often due to inadequate data profiling. Simplifying data structures also drives simpler ETL and reduces system risk.

Let the DW database do the work for you.

- Create **unique indexes/constraints** in the Data Warehouse – most powerful and effective way to avoid problems.
- Forces ETL failure in development when constraints are tested

Keep the data structure simple

- Persist data in a way that it can be most easily managed, reconciled and tested.
- Kimball is good for data marts but not for data management which addresses quality, integration, reconciliation, “facts” resulting from business process.
- Keep the following type of business rules out of the Data Warehouse data (instead put them in a Data Mart or Semantic Layer)
  - Interpretive business rules (e.g. Age Banding, Risk Level)
  - KPIs/metrics requiring aggregation or combination of multiple subjects (e.g. YTD Sales, Loss Ratio)
  - Formulas where inputs may change outside the formula grain due to updates or Type II revisions. (e.g. Days Since Previous Payment).
  - Reference to Current Time (e.g. Tenure, Current Age)

Architect to resolve referential integrity issues

- Have a good reason why columns must be Non-Nullable.
- Strategically apply dummy records.
Designers often use business keys (e.g. account number, sales order number) to tie data together in the Warehouse. This is an amazingly risky proposition because business keys are prone to change even when business rules say they should not. ETL should be developed on the rails of source system primary key and unique constraints. The source database acts a guardian of data quality on your behalf.

**Base transformation on Source System database constraints**

- Business keys are nice, but don’t bank on them never changing.
- Unique identification in the DW should be based on source uniqueness.
- Look for the PKs and Unique Indexes in source systems.

**Retain information that supports reconciliation.**

- Original source values feeding formulas and complex transformation.
- Source system unique identifiers.

**Data Profiling must influence data modeling and transformation**

- If the column is NULL-able, it will have a NULL value eventually.
- If the column can have 200 characters, it will eventually
- If the column value can be negative, it will be negative eventually.
- If the column text can break the expected format, it will break format eventually.
Pattern Based Architecture

Pattern based architecture is less about having the “perfect” architecture than it is about having a consistent architecture. People like consistency because they feel confident that it’s worth investing their time to embrace it. When the whole team does this, they will execute it competently and without hesitation.

Pattern-Based Data Management and Flow:

- Staging Data from Sources.
- Naming Conventions (content-oriented naming vs. use-oriented naming)
- Purpose-specific table structures (mapping, content, cross-reference, domain)
- Avoid pre-aggregation unless performance demands it.

Pattern-Based DW-Specific Metadata (In Every Table)

- Auditing Columns
- Update/Insert Datetimes
- Logical Delete Tracking
- Versioning (Type II)
Pattern-Based ETL is the younger sibling – and beneficiary – of Pattern-Based Architecture. Once a validated ETL template is prepared, developers increase their velocity of delivery and reduce errors because they don’t have to rebuild (and test) the ETL substrate each time.

**Pattern-Based ETL Specifications**
- Consistent ETL specification structure
- Consistent ETL instructions and notation. Do ETL developers know what to do?

**Persist Data More Often**
- Break ETL into smaller pieces.
- Store data in its more granular, normal form.

**Pattern-Based ETL Templates**
- First develop Pattern-Based ETL templates to match architecture.
- **Requires a Pattern-Based architecture!**
- **Requires a Pattern-Based naming convention!**
- **Requires a consistent ETL specification structure, instruction and notation.**
- Deviation from validated ETL templates creates risk.

_If the development team can’t explain the pattern, then you may as well not have one._
Remove all possible transformation in the Semantic Layer

- Semantic Layer is only about translating physical data into semantic business terms/concepts.
- Notable exception is business rules requiring aggregation (i.e. KPIs)

Let performance (ETL and query times) drive architecture and ETL complexity

- Pre-aggregation of fact tables is mostly a relic (powerful CPU, memory, etc.)
- Dimension require more complexity (risk) to maintain than more Normalized constructs.
- Views vs. Managed ETL – fewer moving parts
- Stored procedures vs. Managed ETL – who will maintain it and what is their skillset?

Eliminate Gold-Plating

- 50% of data elements never used.
  - Does this means 20-30% of time spent is wasted? Yes, if it’s being tested properly.
- Only implement what meets business requirements.
- Don’t build in “cool” features unless the business has asked for it.
  - Most “cool” features are rarely used.
  - There would have been a lower cost just to add onto existing ETL!
What is Virtualization?

- Creating an whole server environment that is stored as a file on the filesystem of another machine. Because it is stored as a file, the Virtual environment can be cloned with a simple file backup.
- Operating systems of Virtual environment and host may be completely different.
- Behaves exactly as a separate physical environment (with some caveats).
- Easily meets the need of restoring an environment to previous state.

Testing and Virtualization

- Source data may be staged into “clean” Virtual environment then backed-up.
- Testers may seed the Virtual environment with “dirty” data and test ETL.
- At any time, the Source Virtual environment may be replaced with the original backed-up copy.
- Eliminates complexity of continuous data massaging (esp. database restores) to ensure accurate testing.
- Frees the team to “go crazy” with testing bad data scenarios when it’s a simple matter to restore a viable Source dataset.

Ensuring Environment Integrity

- Providing that Production is a Virtual environment...
- Simple to clone Production to Development, Test, and Pre-Prod after each release.
- Ensures integrity of configuration and data quality between environments.
- Eliminates a key risk in the BI ecosystem.
What best practices can IT follow when testing in a virtualized environment?

IT should exploit the capabilities that virtualization provides to support best-practice guidelines for maximum testing rigor and maturity. I recommend IT:

• Use virtualization to establish development, unit test, QA, and pre-production sandboxes when developing systems for virtual production environments

• Use virtualization to establish x86-based staging environments and use existing import technologies to clone representative images of the physical production software infrastructure stack within the virtual staging environment

• Test all changes in the virtual staging and testing sandbox that are to be deployed to production

• Test the change across as much of the end-to-end infrastructure as can be established within the virtual sandbox and connected to across the physical test lab

• Schedule changes on a regular basis rather than making changes on an ad-hoc basis.

Much of the risk in Data Warehousing is created by complexity in data management, application of business rules, data and ETL architecture. The mantra should be “Reduce Complexity to Reduce Risk”.

Complexity = Risk

Architecture

Data Structure

Naming Conventions

ETL Specifications

Semantic Layer
Action Items

1. Develop a pattern-based approach
   - Does one exist today...is it comprehensive? Architecture, data management, ETL, etc.
   - Start today – everything developed from today-forward based on a pattern
   - Develop a set of ETL templates (e.g. Informatica Power Architect for Visio)
   - Educate your team. It’s only valuable if they understand it.

2. When planning ETL Transformations...
   - Base them on columns values backed up by unique constraints.
   - Only trust what the database says to be true, **not** the application or an app developer.
   - Expect the worst possible data values to occur in a column...because they will.

3. Reduce risk by reducing complexity
   - Keep a watchful eye out for unwarranted complexity.
   - Eliminate gold-plating.
   - Don’t do anything more complex than what is needed to support business requirements.
   - Recognize what adds value to the business vs. what adds ease-of-change for developers.
Introduction

Strategy 1 - Improve Quality of Testing
Strategy 2 - Reduce the Prevalence of Errors
**Strategy 3 - Reduce the Need for Testing**
Strategy 4 - Automate Testing
Strategy 5 - Integrate Continuous Testing
Conclusion
Unit Testing – Radical Change

At some point, we asked ourselves, why are there so many errors in system testing? Then we asked, why aren’t developers releasing code faster? The answers were the same – ETL developers were not 100% confident in the requirements or that their code performed as specified. This demanded a change in approach. We needed to get testing in front of the development process so that developers could unit test and release with confidence.

What if we pushed testing to the front of the process instead of the end?

– Requires more careful thought into business requirements. How do we write a test case for a capability?
– Required building of test cases – but we’re doing that anyway.
– Involve developers in the test-case development process.
– Leverage pattern-based architecture to automate testing.

What if we gave developers the ability to execute tests?

– Give developers the facility to repeatedly execute groups of similar tests in development environment
– Developers aren’t finished with development until their code passes all the tests!!!
– After the first several sprints, the team radically changed their habits
  • Data Analysts (ETL spec writers) became better – anticipated/mitigated data issues!
  • Developers became better – reduced bugs in system testing!

What’s left for the testers?

– Not much? Well...not as much as before.
– System testing – just a repeat of developer tests. Warranted, but expect almost zero defects.
– Integration testing still extremely valuable.
– Do we need a formal tester role?
Types of Data Tests

There are many classes of tests encountered in a data warehouse. Some of them involve testing ETL transformations. Others involve testing quality of data at rest (i.e. after an ETL process has completed).

Individual Column Testing of ETL

- Requires pattern-based ETL
- Management of source system meta-data to drive processing
- Seed all possible combination of “bad” values in source tables.
- Data quality in target table.

Structural Testing of Resting Data

- Uniqueness assumptions
- Referential Integrity Assumptions
- Validity of Type II date ranges

Reconciliation Rules of Resting Data

- Aggregate level reconciliation is good.
- Record-level reconciliation is best.

Business Rules of Resting Data

- Tests assumptions the business has about the data.
- Usually reveals source data issues – not necessarily ETL issues.
We also found that the ETL management itself could be tested – process control and error tables.

**Monitoring of Process Control tables**
- Record processing counts
- Failed or incomplete processes.

**Monitoring of Error Tables during ETL Processing**
- Did the ETL kick out errors?
Action Items

1. Change the SDLC to move test-cases up-front
   • Change the mindset of the team (and often management)
   • Change the cadence of development – more up-front thought is needed.
   • Involve the development team – get them to write test cases from requirements.
   • Development quality will skyrocket!

2. Developers aren’t “done” until their code passes all the tests.
   • Need to provide a facility for executing tests.
Introduction

Strategy 1 - Improve Quality of Testing

Strategy 2 - Reduce the Prevalence of Errors

Strategy 3 - Reduce the Need for Testing

Strategy 4 - Automate Testing

Strategy 5 - Integrate Continuous Testing

Conclusion
Automation of Tests

Using a library of SQL statements test, we found that most testing could be automated. Some tests require maintenance of metadata (i.e. structural and transformations), whereas others required custom scripting to match user requirements.

**Individual Column Testing of ETL**
- Requires pattern-based ETL
- Management of source system meta-data to drive processing
- Seed all possible combination of “bad” values in source tables.
- Data quality in target table.

**Structural Testing of Resting Data**
- Uniqueness assumptions
- Referential Integrity Assumptions
- Validity of Type II date ranges

**Reconciliation Rules of Resting Data**
- Aggregate level reconciliation is good.
- Record-level reconciliation is best.

**Business Rules of Resting Data**
- Tests assumptions the business has about the data.
- Usually reveals source data issues – not necessarily ETL issues.
Some tests could be fully automated since the tested specific tables built that support DW processing.

**Monitoring of Process Control tables**
- Record processing counts
- Failed or incomplete processes.

**Monitoring of Error Tables during ETL Processing**
- Did the ETL kick out errors?
Test harnesses exist for application development.

Test harnesses exist for application development.

Follows the Feature/Scenario/Step pattern (RSpec, Cucumber, etc.)
Test Harnessing for DW?

The question is: could this be done for data warehousing? In fact, many BI user stories bear a strong resemblance to their app-dev cousins. However, BI also demands addressing uncontrolled incoming data which is then manipulated and aggregated across large sets. What is the best way to write a Step? And how is this effectively executed through the harness.

How can this be implemented for data warehousing? LINQ?
Must address transformation across set.
Goal should be to code in a manner that is more semantic and does not require advanced SQL skills.

**Feature : Marketing Email Campaign Support**

As a Marketing Analyst
I want to Know the amount each customer has purchased (retail value) over the past year, banded by $1000 increments.
So that I can Email promotional discounts to each Customer who has purchased over $1000 in the past year.

**Scenario : Customer Retail Sales Banding**

Given a set of all Customers whose Retail Sales are between $1000.00 - $1999.99 for a given year.
When For each of those Customer’s Retail Sales Band for the same period
Then I will see “$1000 up to $2000” listed as the Retail Sales Band

**Step :**

SQL? .NET? Ruby?
Because there are multiple stakeholders and risks, the structure of Scenarios may vary widely.

**Non-Functional Requirement**

**Scenario**: AXM_CUST_ORDER_LINE Benchmark Time

**Given** that the workflow AXM_CUST_ORDER_LINE will execute

**When** the records processed are between 20k-30k

**Then** processing will take no longer than 4 minutes.

**ETL Transformation**

**Scenario**: Determination of Terminate Flag for a Customer

**Given** every Customer in the Warehouse

**When** where the Customer Status = ‘T’ and Date of Death exists

**Then** show their Terminated Flag as False

**Business Rule**

**Scenario**: Close Date Exists for Closed Accounts

**Given** every Account in the Warehouse

**When** the Account Status is Closed

**Then** Close Date must exist.

**Reconciliation**

**Scenario**: Sales Order Line Total Reconciliation

**Given** every a Sales Order Line in the warehouse

**When** the Sales Order Amount must match the corresponding SLS_LINE.SLS_LINE_TOT in the AXM system.

**Then** Close data must exist.

**Referential Integrity**

**Scenario**: Customer Has A Customer Status

**Given** every record in CUST

**When** looking up the CUST_STATUS

**Then** a current (open end date) CUST_STATUS must exist.
Agenda

- Introduction
- Strategy 1 - Improve Quality of Testing
- Strategy 2 - Reduce the Prevalence of Errors
- Strategy 3 - Reduce the Need for Testing
- Strategy 4 - Automate Testing
- **Strategy 5 – Continuous Integrated Testing**
- Conclusion
Automated tests (whether as SQL or through a test harness) have benefited us in development but are also an asset that can be leveraged in production to monitor data quality. This provides a method of ensuring that the team can catch unexpected errors before they compound into irreversible data corruption.

- **We recognize that data issues will still occur in production – it’s a fact of life...**
  - Most issues are not exposed as unhandled exceptions in ETL – they’re stealthy!
  - Wanted a way to monitor and assess those issues
  - Wanted a way to monitor growth of issues (not every data issue needs to be fixed immediately)
  - Wanted to communicate issues (and mitigation plan) before users found them.

- **What if we took the tests built for development and ran them on production?**
  - Comprehensive testing executed after every run.
  - Status notifications sent out every night.

*We were able to catch errors and troubleshoot quickly before compounding corruption of datasets occurred. The secret is to catch them on the first execution.*
Agenda

- Introduction
- Strategy 1 - Improve Quality of Testing
- Strategy 2 - Reduce the Prevalence of Errors
- Strategy 3 - Reduce the Need for Testing
- Strategy 4 - Automate Testing
- Strategy 5 – Continuous Integrated Testing
- Conclusion