ACC 424 Accounting Information System

Automating internal controls testing
Python notes for 6.1-6.4

Section 001 MW 9:30 AM – 10:45 AM at Rm. 257 over Jan 13 – May 07
Section 002 MW 11:00 AM – 12:15 AM at Rm. 127 over Jan 13 – May 07

6.1. Introduction to automating controls testing

Internal controls testing is a key process in ensuring data integrity, accuracy, and compliance within accounting information systems. Traditionally, such testing was **manual**—time-consuming, prone to human error, and limited in scope. Automation of internal controls testing can improve:

- **Efficiency**: For example, instead of manually checking 50 sales invoices each month, a lightweight program can verify all 10,000 invoices against control rules in seconds.
- **Consistency**: Automated tests apply the **same** criteria to every transaction, eliminating differences caused by individual judgments.
- **Coverage**: Automation enables **full-population** testing. For instance, every journal entry can be reviewed for proper approval and amount limits, not just a random sample for control testing.
- **Real-Time Monitoring**: Tools like SAP Governance, Risk, and Compliance can flag control violations (e.g., unauthorized payment approvals) as they occur. That is, automation can generate real-time alerts if a violation is detected.
- **Audit Trail**: In manual testing, audit trail records are often handwritten notes or audit memos. In automated testing, however, the system **automatically** creates and stores this documentation in a **structured, tamper-resistant format**.

Types of Controls Suitable for **Automation**

Not all controls are equally suited for automation. The best candidates include:

- **Data Validation Controls**: Ensure the data entered the system is accurate, complete, and follows expected formats or rules.

- Segregation of Duties: Ensures that no one individual has excessive control over a critical process (e.g., initiating and approving payments). Automated checks can verify if the same user has roles assigned that should be mutually exclusive.
- **Transaction Limit** Controls: Flags any transactions that exceed predefined thresholds without proper approval.
- Master Data Change Monitoring: Tracks changes to critical records (e.g., vendor bank accounts, employee salary rates). Automated checks can verify who made the change, when it was made, and what was changed.
- Reconciliation Procedures: Comparing balances between systems. **Automated checks can verify** that the general ledger cash balance matches the bank statement; sub-ledger totals (e.g., A/R) match the general ledger control account.
- Regulatory Compliance Checks: transactions and disclosures comply with legal or regulatory requirements (e.g., SOX Section 404).

6.2. Frameworks for automated controls testing

Automated controls testing should start with a **risk-based approach**. This means focusing your automation efforts where they matter most—on the business processes and controls that carry the highest risk to financial reporting.

The first step is to identify **critical processes**—such as revenue recognition, payroll, or vendor payments—that could significantly affect financial statements if something goes wrong. Once these processes are identified, the next task is to assess the **risk** levels of each process. This involves evaluating both the likelihood of a control failure and the potential impact such a failure would have on the business.

With risk levels defined, companies can then map **specific controls** to each identified risk. For instance, if the risk is unauthorized vendor payments, relevant controls might include approval workflows and segregation of duties. Finally, for each control, it's important to evaluate its **automation** potential. Controls that rely on **structured data** and **logical rules** are generally better candidates for automation than those requiring **human judgment**.

Implementation Considerations

When implementing automated controls testing, a common best practice is to start **small**, selecting a few to automate as a **pilot** project. This allows the team to test the effectiveness of automation before scaling up. Once a test is automated, it's essential to **validate** the results by comparing them against **manual** testing outcomes.

Equally important is **documentation**. Organizations must keep detailed **records** of how the automated tests are designed—including the logic used, the data sources involved, and how exceptions are handled.

Speaking of exceptions, a strong automated testing program must have clear **exception handling** procedures. When a control test fails, there should be a defined process for investigating, escalating, and resolving the issue.

Finally, **governance** is key. Automated controls testing should be overseen by a **cross-functional** team—often including internal audit, IT, and compliance—to ensure that automation supports business objectives while maintaining data **integrity** and regulatory **compliance.**

6.3. Hands-on automating internal controls testing with Python

Setting Up Python Environment

To begin automating internal controls testing, we need to import specialized libraries that support financial data analysis and statistical evaluation.

```
# Essential libraries for automating internal controls testing
import pandas as pd               # For structured data analysis of financial transactions
import numpy as np                # For numerical computations and statistical operations
import matplotlib.pyplot as plt   # For visualizing control testing results
import seaborn as sns             # For enhanced statistical visualizations
import datetime as dt             # For handling transaction dates and time periods
```

These libraries are useful for internal controls testing because:

- pandas efficiently processes large transaction datasets
- numpy performs statistical calculations to identify **anomalies**
- matplotlib and seaborn visualize **exceptions** and control breakdowns
- datetime properly handles **fiscal** periods and transaction **timing**

Working with Accounting Transaction Data

For internal controls testing, we'll analyze financial transaction data to identify control exceptions:

```
# Sample financial transaction data for controls testing
sample_transactions = [
    {'TransactionID': 1001, 'TransactionDate': '2023-01-15', 'UserID': 'U001',
     'VendorID': 'V100', 'InvoiceNumber': 'INV001', 'TransactionAmount': 5000.00,
     'ApproverID': 'U005', 'PaymentMethod': 'ACH', 'AccountCode': '5010'},
    {'TransactionID': 1002, 'TransactionDate': '2023-01-16', 'UserID': 'U002',
     'VendorID': 'V200', 'InvoiceNumber': 'INV002', 'TransactionAmount': 12500.00,
     'ApproverID': 'U006', 'PaymentMethod': 'Check', 'AccountCode': '5020'},
    {'TransactionID': 1003, 'TransactionDate': '2023-01-16', 'UserID': 'U003',
     'VendorID': 'V100', 'InvoiceNumber': 'INV003', 'TransactionAmount': 3200.50,
     'ApproverID': 'U005', 'PaymentMethod': 'ACH', 'AccountCode': '5010'},
    {'TransactionID': 1004, 'TransactionDate': '2023-01-17', 'UserID': 'U001',
     'VendorID': 'V300', 'InvoiceNumber': 'INV004', 'TransactionAmount': 7800.75,
     'ApproverID': 'U006', 'PaymentMethod': 'Wire', 'AccountCode': '5030'},
    {'TransactionID': 1005, 'TransactionDate': '2023-01-18', 'UserID': 'U004',
     'VendorID': 'V200', 'InvoiceNumber': 'INV005', 'TransactionAmount': 15000.00,
     'ApproverID': 'U007', 'PaymentMethod': 'Check', 'AccountCode': '5020'},
    {'TransactionID': 1006, 'TransactionDate': '2023-01-18', 'UserID': 'U002',
     'VendorID': 'V100', 'InvoiceNumber': 'INV001', 'TransactionAmount': 5000.00,
     'ApproverID': 'U002', 'PaymentMethod': 'ACH', 'AccountCode': '5010'}
]
```

This code creates a list of dictionaries, where each dictionary represents a single financial transaction with these important fields:

- TransactionID: Unique identifier for each transaction
- TransactionDate: When the transaction occurred
- UserID: Employee who initiated the transaction
- VendorID: Vendor or payee identifier
- InvoiceNumber: Reference to source document
- TransactionAmount: Monetary value of the transaction
- ApproverID: Employee who authorized the transaction
- PaymentMethod: How the payment was made
- AccountCode: General ledger account

Convert to DataFrame

```
# Create financial transaction DataFrame for control testing
transactions = pd.DataFrame(sample_transactions)
```

This line converts our list of transaction dictionaries into a pandas DataFrame, which is a table-like data structure with rows and columns.

```
transactions['TransactionDate'] = pd.to_datetime(transactions['TransactionDate'])
```

This line converts the text-based date strings ('2023-01-15') into proper datetime objects.

```
# System user activity data for segregation of duties testing
sample_user_activities = [
    {'UserID': 'U001', 'ActivityType': 'CreateVendor',
     'ActivityTimestamp': '2023-01-10 09:15:00', 'IPAddress': '192.168.1.45'},
    # Additional activities...
]

# Create user activities DataFrame for control testing
user_activities = pd.DataFrame(sample_user_activities)
user_activities['ActivityTimestamp'] = pd.to_datetime(user_activities['ActivityTimestamp'])
```

This code:

1. Creates a second dataset containing **user activities** in the system
2. Convert it to a DataFrame
3. Formats the timestamp field to datetime objects

This activity log is important for testing segregation of duties controls, as it records who performed which sensitive system actions and when.