Test Data Fabrication Use Case

Mark Moncelle – IT Architect

State Farm

YouTube Video:

State Farm: Speed application and product development with test data management

https://www.youtube.com/watch?v=C6EdxiNKoZA&feature=youtu.be

Disclosure: Views represented are those of the presenters and not necessarily State Farm

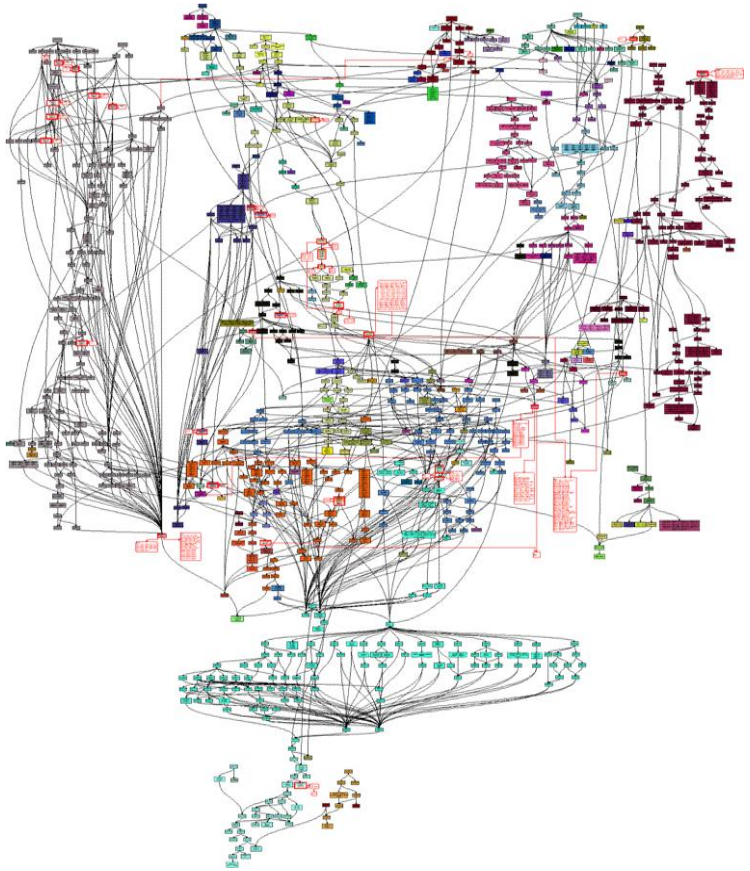
DevOps – Principles and practices designed to improve the delivery of high value, high quality changes to production.*

- Core Capabilities
  - Version Control
  - Comprehensive Test Automation
  - Deployment automation
  - Continuous integration
  - Shifting left on security
  - Short-live feature branches
  - Effective Test data Management

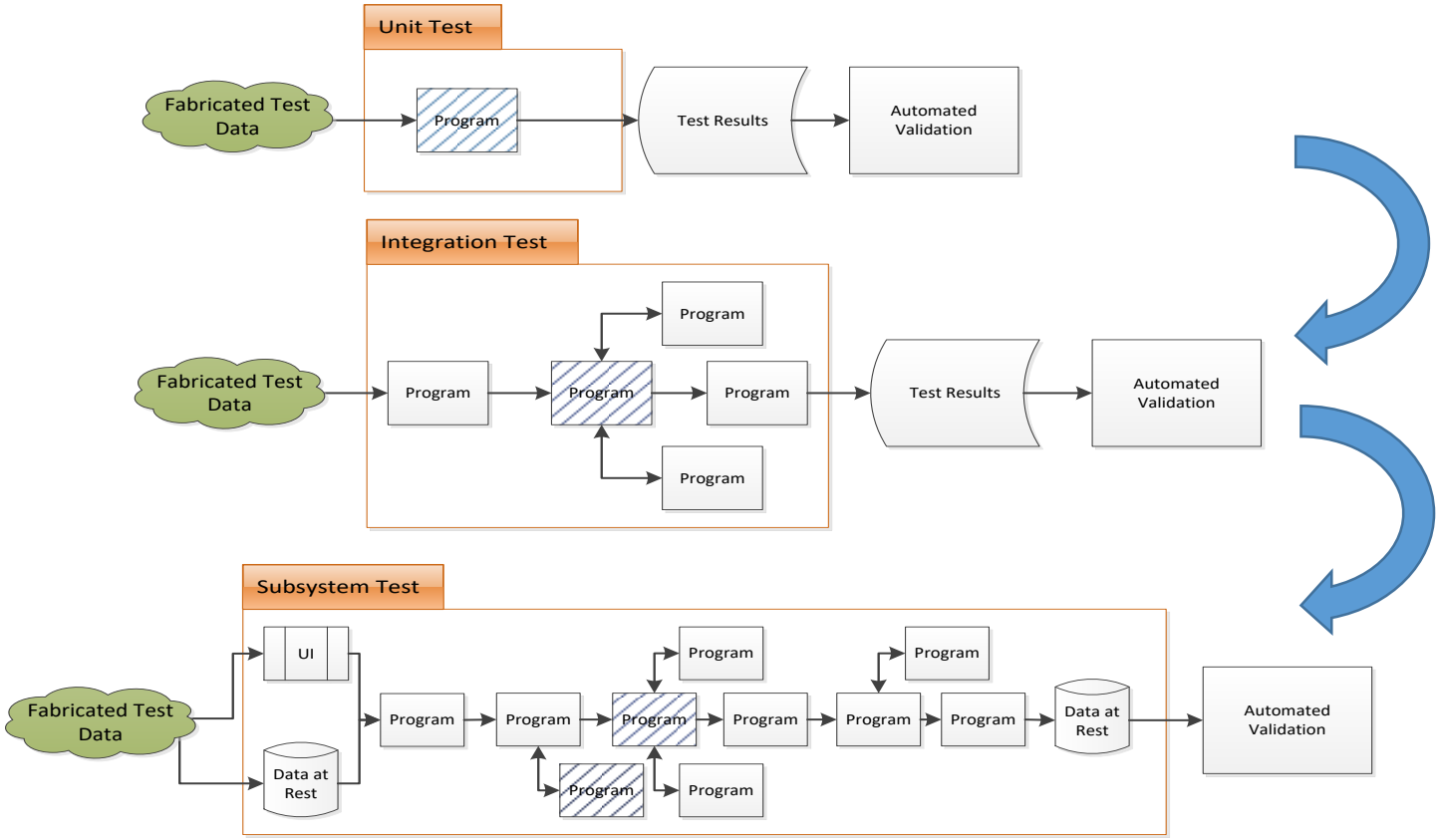
*Forsgren, Nicole, et al. *Accelerate: the Science behind DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution, 2018*

DevOps, Test Driven Development, and Continuous Delivery/Integration all drive smaller tests

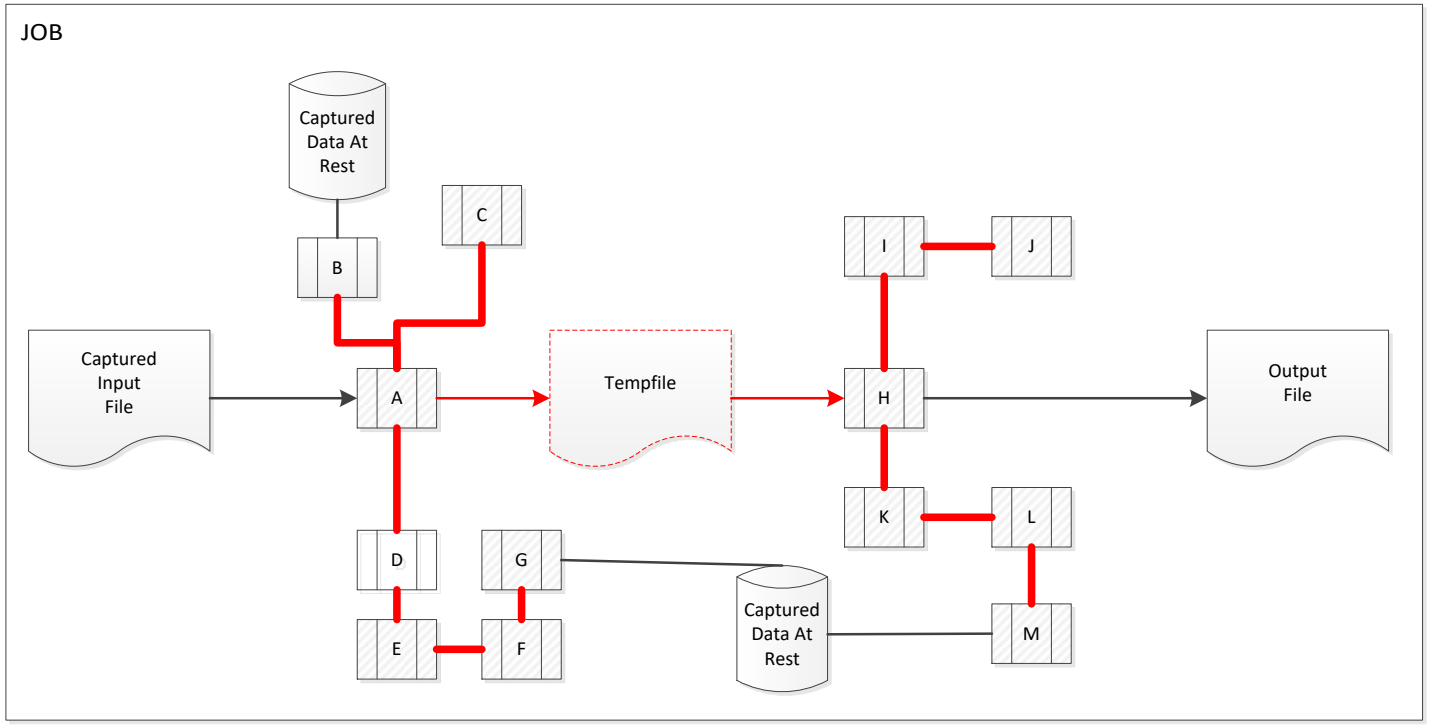
Less of this:



More of this:



Business Challenge: How to enable Unit test for complex processes

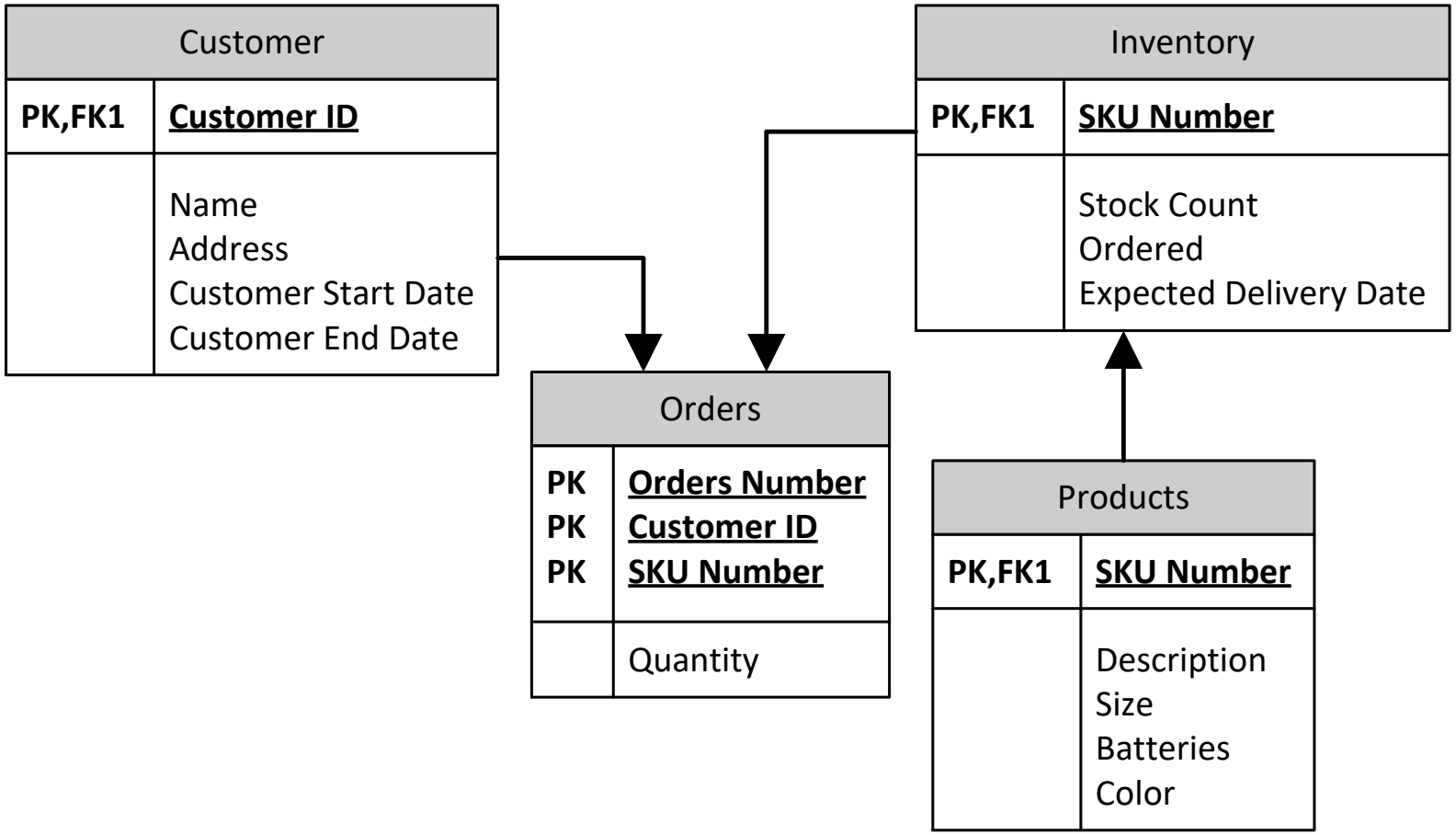


Data passed between programs only ever exists in memory. Temporary files disappear when the job completes.

Why/Why Not Capture or Fabricate?

Capture	Fabrication
Reliable source of realistic data	Readily available once set up
Scales in complexity*	Difficult to scale the complexity of the data*
Costs of capture and obfuscating production data	Cost to build and maintain fabrication routines
Availability of Edge and Fringe cases	Flexibility to create edge and fringe cases

A few words about Complexity



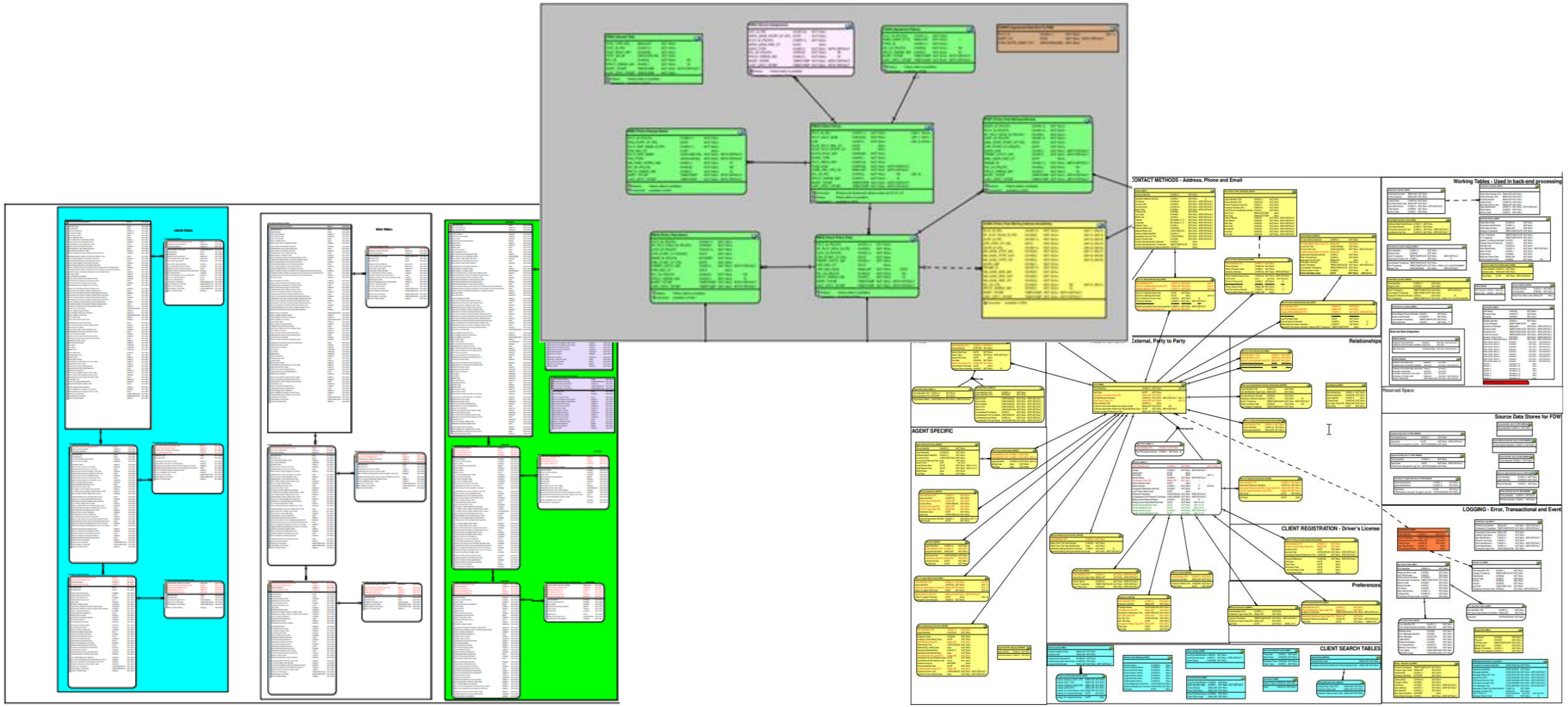
To Capture, you need to understand:

- 4 Tables (structures)
- 4 Table relationships

To Fabricate, you need to understand:

- 4 Tables (structures)
- 4 Table Relationships
- 15 field types (relevant content)
- 8+ Field Relationships (for Example)
  - Cust Start < Cust End
  - Cust Start < Current Date
  - Expected Delivery Date > Current Date
  - SKU Number ~ Size
  - Size ~ Batteries

Production environment may have very complex data models



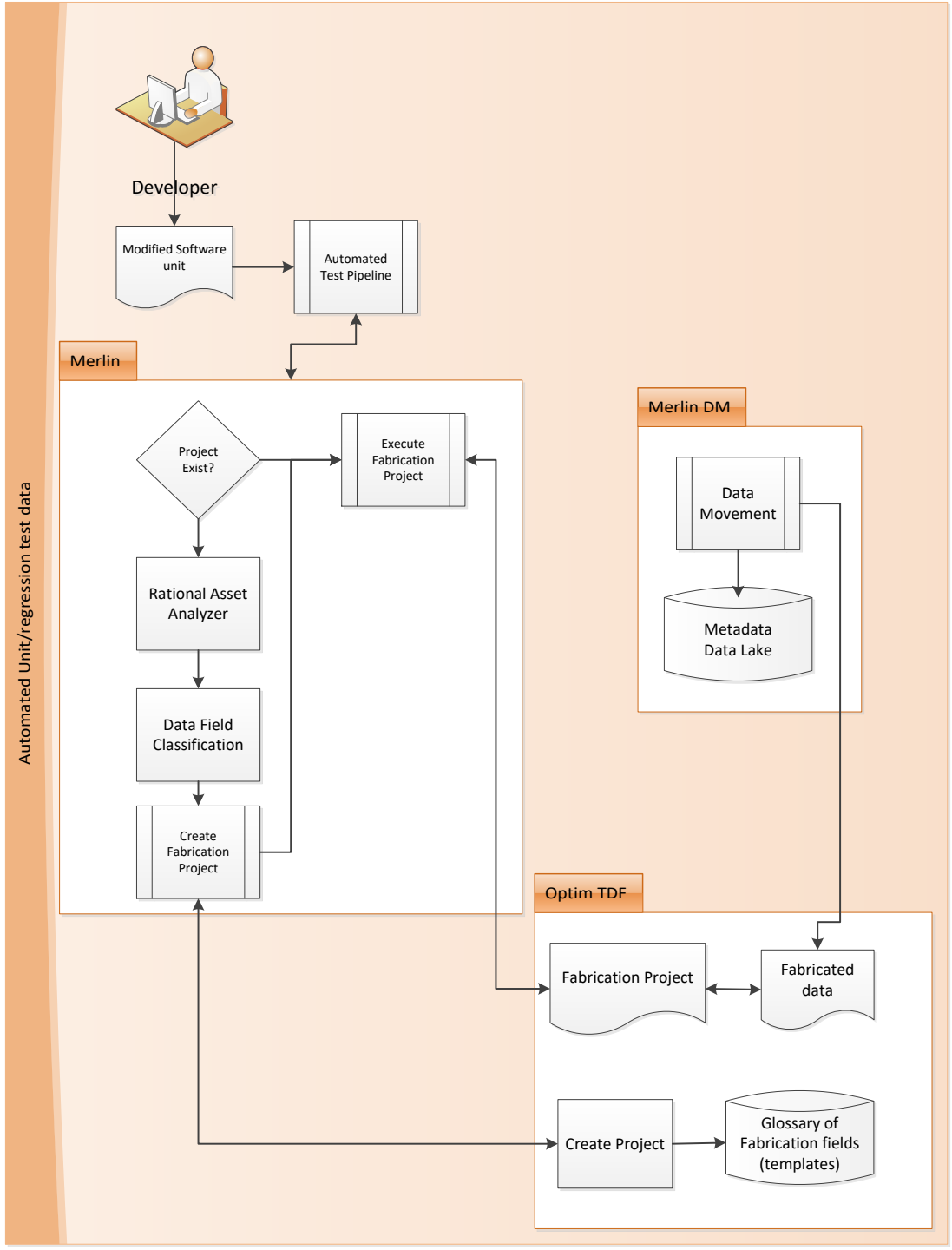
Legacy programs may be complex

And may not be well understood by the developer

Future State – Automated Unit Test Data

Use “stock” technology to:

- Evaluate source code for important fields
- Use Machine Learning Prediction Models to classify fields
- Build Fabrication rule templates for various classifications
- Automatically generate data for unit and integration testing
- Execute Automated Tests



Challenge #1: Large number of fields are read in by each program

- Use Static Analysis to detect “interesting” fields.
  - Input fields that are used in logic (IF, WHEN, CASE…) or calculations (ADD, EVALUATE, etc.)

This eliminates approximately 80% of the fields from further analysis. (on average, 25 “interesting fields” per program)

```
IF (EXPIR-DATE-8 = '99999999') OR
     (EXTNL-ERR-IND = 'Y') OR
     ((STATUS-CODE OF MASTER-RECORD (1) >
'09' AND < '19')
      AND ((EFFECTIVE-DATE-8
                        OF MASTER-RECORD
                        NOT = SPACES)
      AND (EFFECTIVE-DATE-8
                        OF MASTER-RECORD
                  < '20050101')))
     CONTINUE
  ELSE
PERFORM 001-OLD-TERM-DLET
     THRU 001-OLD-TERM-DLET-EXIT
     VARYING TERM-SUB FROM +1 BY +1
       UNTIL TERM-SUB > STATUS-SEGMENT-LMT
          OR TERM-DLTE-ROW (TERM-SUB) =
SPACES
  IF MSF05-ERR-IND = 'Y'
     GO TO 000-GOBACK
  END-IF
```

Challenge #2: Naming conventions for fields are inconsistent and documentation is scarce

ADDR_STATE, ADDRESS_STATE_CODE, AGENT_STATE, AGENT_STATE_CODE, AGT_STATE, ALPHA_STATE, ALPHA_STATE_CODE, ANNUAL_STATEMENT_LINE, ANNUAL-STATEMENT-LINE, AUTO_STATE_CODE, BRNDLRSTATE-TXT, CITY_CODE_ST, CITY_ST, CLAIM_STATE, CLM_STATE, CLM-STATE, CLTRL_STATE, COMBINED_STATE, CROSS-REFR-STATE-QT1500, CROSS-REFR-STATE-QT4010, CROSS-REFR-STATE-QT8110, CR-TO-STATE-QT2500, CR-TO-STATE-QT4010, CR-TO-STATE-QT8110, CR-TO-STATE-QTD021, CUST_STATE, DEFAULT_STATE, DEQUE-STATE, DRIVER_STATE, DRIVER-LICENSE-STATE-ID, DRVR_STATE, DSGTN-MED-PRVDR-POSTL-ST-CD, ECHO-STATE, EFF-DATE-STATE, EXCL_STATES, EXCLUDED_STATES, FINAL-STATE, FIRE-STATE-ID, GEO-STATE, GEO-ST-CD, H983_PAYEE_STATE, H983_TAX_STATE, HD001_PI_STATE, HD001_PROC_STATE, HD001_SERV_STATE, HD002_PROC_STATE, HD002_RATE_STATE, HD015_PEND_CHANGE_RATE_STATE, HD043_ASSIGN_STATE, HD049_WRT_STATE, HD060_PROC_STATE, HD060_UND_PI_STATE, HD090_PROC_STATE, HIST_AGENT_STATE, LOC_STATE, LOC-RISK-STATE, MAIL_MILITARY_STATE, MAIL-MILITARY-STATE, MAIL-ST-CD, MID_[illegible]_STATE, NAII-STATE-ALPHA, NAII-STATE, NCOA_STATE, NCOA-FINAL-STATE, NCOA-STATE, NEW_POSTAL_ST, NEW_STATE, NEXT_STATE, NO-BILL-FOLL-UP-STATE, OLD_POSTAL_ST, OTHER_STATES, PARM-016-STATE, PAYEE_STATE, POLICY_STATE, POSTAL_ABBREV, POSTALCODE, POSTAL-STATE-CODE-0001, POSTAL-STATE-CODE-001, POSTL-ST, POSTL-CD, POSTL-ST, POSTL-ST-CD, POSTL-ST-PRVNC-CD, PRIN_STATE, PROC_STATE, PROCESSING_STATE, PRVNC-POSTL-CD, PSTL-ST, PSTL-ST-CD, QOG14-STATE-CODE, QOG14-STATE-SUB, RATE_STATE, RATE-STATE, RATING_STATE, RECAP_STATE, REPL-STATE-QT0200, REPL-STATE-QT0300, REPL-STATE-QT0500, REPL-STATE-QT0600, REPL-STATE-QT1000, REPL-STATE-QT1500, REPL-STATE-QT1800, REPL-STATE-QT1900, REPL-STATE-QT4000, REPL-STATE-QT7200, REPL-STATE-QT7500, RES_STATE, RESIDENT_STATE, RESIDENT-STATE, RISK-MIS-STATE, SCO-STATE, SCR_STATE, SC-STATE, SELLER-STATE-ID, SERV_STATE_AGENT_CODE, SF_STATE, SF_STATE_CODE, SORT_STATE, ST_ABBREV, ST-ABBR-CD, STATE, STATE_AGENT_CODE, STATE_ALPHA, STATE_CODE, STATE_KEY, STATE_NUM, STATE-0001, STATE1, STATE3, STATE8, STATE-ADD-INT, STATE-AGENT-CODE, STATE-ALPHA, STATE-ALPHAI, STATE-ALPHAL, STATECD, STATE-CODE, STATEI, STATE-ID, STATE-IN, STATE-INITIALS, STATEL, STATE-NUMERIC, STATEO, STATE-OUT, STATE-QTSCOM, STATESORTCODE, STAT-STATE, SUM-MIS-STATE, SUP_STATE, TAX_RES_STATE, TAX_STATE, TAX-STATE, TEAM-STATE, TERM_AGENT_STATE, TERM_STATE, TITLE-STATE-ID, TOWN_ST, TOWNSHIP_ST, TRANSFER_TO_STATE, TRFTO_STATE, UNDERLY-STATE-QT7200, WC-DESCRIPTION-STATE, WC-DESCRIPTION-STATE-0155, WC-HOLD-STAT-STATE, WRITE_STATE, WRITING_STATE, WRITING-MIS-STATE, WRT_STATE, WRT-STATE, WS-PARM71-SAVE-STATE, XSTATE, ZQOGAC-GEO-STATE-P01, ZQOGAC-STATE-P01,

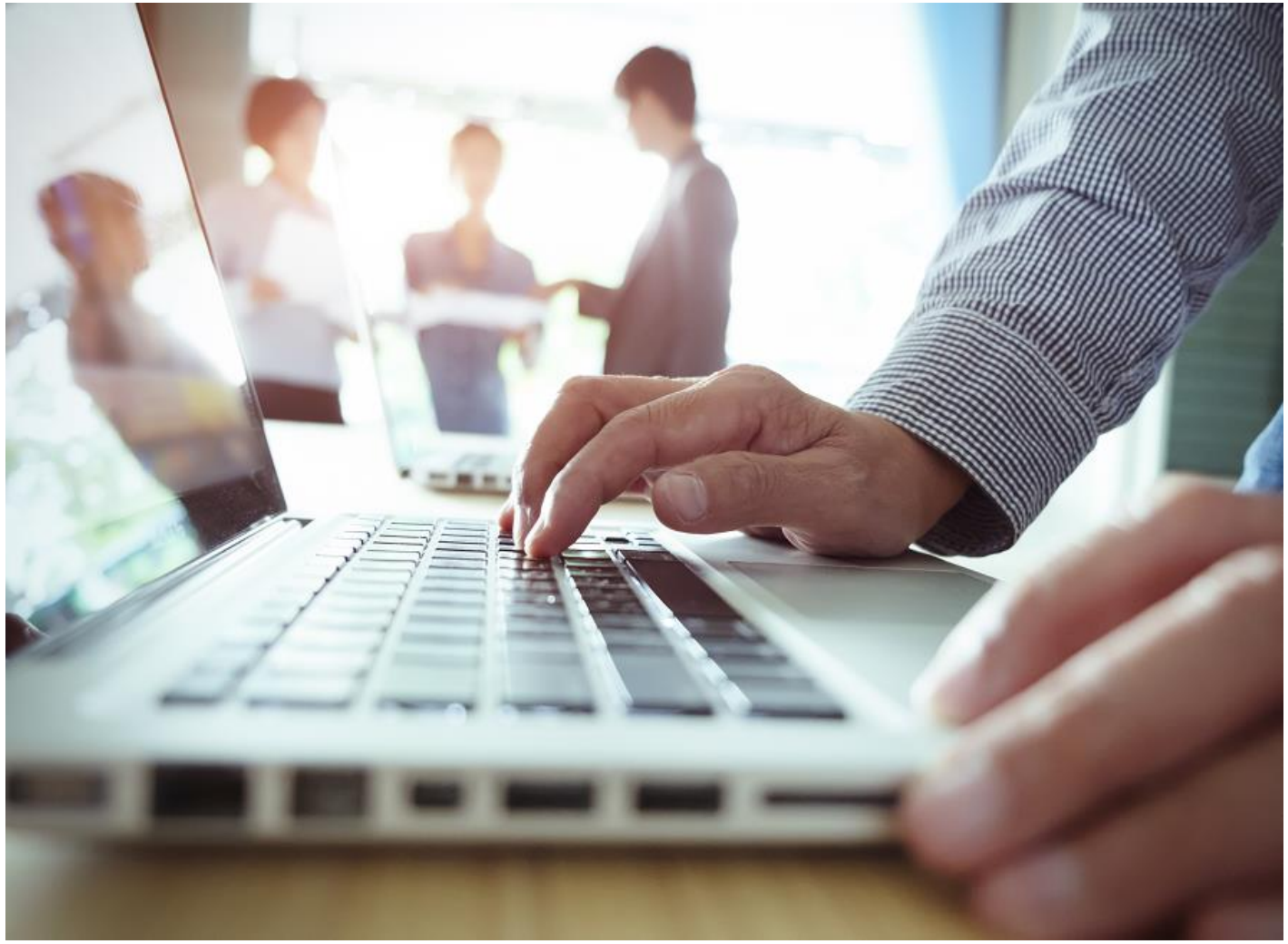
- Use Machine Learning models to predict field content types (e.g. State Code, SQL return Code, etc.)
  - Field Name, Size, Type, & comparison values as features for ML Classification

Initial set of training produced up to 95% accuracy for 100 different classifications

Challenge #3: Building a fabrication project is time-consuming

- Create Rule templates for each classification (estimate 200-300 total classifications)
- Leverage Optim – TDF API's to:
  - Import and/or build data structure
  - Assign rule templates to fields
  - Generate fabricated data

Challenge #4: Incorporate developer feedback to correct the data



- Develop simple UI to allow developer to:
  - Pick a different classification
  - Type in allowed values

  Capture developer corrections are used for future re-training of Classification model.

Challenge #5: Execute Unit tests with fabricated data

- Integrate output with zUnit testing tool

Current state

- Process has been proven “manually”, now operationalizing the solution

- Identify additional classifications
- On-going retraining for the model as new classifications are identified
- Building fabrication rules/templates for known classifications
- Creating automation to generate fabrication projects within Optim-TDF
- UI for Developer feedback
- Integration into Unit test tooling

Where do we go from here?

- Identify/Refine initial use cases – e.g. compiler upgrade
- Train classification model for Java
- Expand scope of analysis for Integration testing