Imagining Contract-Based Testing for Event-driven Architectures

Dave Copeland
Director of Engineering
Stitch Fix
@davetron5000
What Problem Are We Solving?

- Systems communicate to facilitate a process
- We need to know if that works
- We need to know if our changes will break it
- We need to know that without a bunch of manual clicking
Hi!

• I’m Dave Copeland, Director of Engineering @ Stitch Fix

• We are a personal styling service

• eCommerce-like business model

• All internal operations are via applications and services the engineering team writes.

• Lots of HTTP, but **lots more** messages (in our case, RabbitMQ)
Hi Emily,

Thanks for letting us style Fix #3 for you. We hope you love it.

Items handpicked for you by Arielle

<table>
<thead>
<tr>
<th>Item Code</th>
<th>Product Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>367-A</td>
<td>Burgundy Crew Neck Sweater</td>
<td>$88.00</td>
</tr>
<tr>
<td>194-367</td>
<td>Willow &amp; Clay</td>
<td></td>
</tr>
<tr>
<td>650-A</td>
<td>Black Dot-Navy Crew Neck Sweater</td>
<td>$48.00</td>
</tr>
<tr>
<td>205-659</td>
<td>LA MADE</td>
<td></td>
</tr>
<tr>
<td>912-A</td>
<td>Blue Jumper Striped Tab-Sleeve Knit Shirt</td>
<td>$84.00</td>
</tr>
<tr>
<td>281-912</td>
<td>Splendid</td>
<td></td>
</tr>
<tr>
<td>966-A</td>
<td>Black Rayon Hi Lo Peplum Knit Top</td>
<td>$48.00</td>
</tr>
<tr>
<td>279-969</td>
<td>Tart</td>
<td></td>
</tr>
<tr>
<td>092-A</td>
<td>Navy Denim Button Denim</td>
<td>$48.00</td>
</tr>
<tr>
<td>205-092</td>
<td>Wax Me</td>
<td></td>
</tr>
</tbody>
</table>

Don't lose your styling fee! It will be credited towards any items you keep in this Fix.

Subtotal: $316.00
Buy 5 discount 25%*: -$79.00
Styling fee: $20.00
Total: $217.00

* You must purchase all five items to receive a 25% discount.

Note: Additional sales tax may apply.
Hi Emily,

Thanks for letting us style Fix #3 for you. We hope you love it.

<table>
<thead>
<tr>
<th>Items handpicked for you by Arielle</th>
<th>Charging &amp; Discount Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>659-A Black 210-659 M. LA MADE Jemma Dot Trimmed Tie-Waist Cardigan</td>
<td>$48.00</td>
</tr>
<tr>
<td>912-A Blue 241-912 M. Splendid Louisa Striped Tab-Sleeve Knit Shirt</td>
<td>$84.00</td>
</tr>
<tr>
<td>969-A Black 215-969 M. Tart Peyment Hi-Lo Poplin Knit Top</td>
<td>$48.00</td>
</tr>
<tr>
<td>092-A Navy 209-092 d Miss Me Detachable Rhinestones Button Denim</td>
<td>$48.00</td>
</tr>
</tbody>
</table>

Don't lose your styling fee! It will be credited towards any items you keep in this Fix.

Subtotal: $316.00
Buy 5 discount: 25%* - $79.00
Styling fee: purchase credit: $20.00
Total: $217.00

* You must purchase all five items to receive a 25% discount.

Note: Additional Sales Tax may apply.
Team

Warehouse Operations

Styling

Merchandising

Consumer

Finance

Customer Service

dedicated engineering team
Consumer-Driven Contracts

Order ID

WMS

Inventory
Metadata

Test

Financial
Transactions

Warehouse
Engineering

Finance
Engineering

pack
slip

Finance Engineering

Test
The great thing about synchronous services...

- You know at deploy/test/CI-time who calls what
- You could codify that as contracts
- If all contracts are satisfied, end-to-end behavior is still good.
Item 4567’s Price Updated

Item 1234’s description changed

Item 9876 added to order 765
How might this work?
Guarantees

- Payload schema
- Metadata guarantees:
  - routing key
  - headers/metadata
- Some sort of identifier - “what guarantee might I expect?”
Expectations

- Id of the guarantee that is expected
- Schema that the payload must conform to
- Metadata expectations
- Ability to feed into several different test cases
Safe Consumer Changes

Central Authority

Producer Test Framework

Consumer Test Framework

Guarantee Definition

Consumer Knows if it’s been broken 👍
Safe Producer Changes

Producer Knows It’s broken someone 👍
## Failures

<table>
<thead>
<tr>
<th>CONSUMER</th>
<th>No Guarantee Exists</th>
<th>Code Might Never Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guarantee Exists, my Test Fails</td>
<td>Consumer Fails in Production</td>
<td></td>
</tr>
<tr>
<td>PRODUCER</td>
<td>Expectation Exists, my Schema/Examples Aren’t compatible</td>
<td>Consumer Fails in Production</td>
</tr>
</tbody>
</table>
Side Benefits

• Listens for messages in production

• Anything with no guarantee → alert/notify

• Guarantees for messages not sent after X days → alert/notify

• Could document actual realtime dependencies!
  • Understanding implementation of a business process becomes easier!
Verification Hand-waving

- Guarantee is a Schema
- Expectation is a Schema
- Isn’t this just “check that everyone’s schemas are the same?”
- Not necessarily:
  - Enforcing equivalence is tight coupling we want to avoid
  - Guarantee must *subsume* the Expectation
• Our consumer just needs `item_id` and `new_price`
Subsume Example
Expected Schema

• The guarantee schema subsumes this one—there’s nothing here we aren’t getting from the producer

```json
{
    "namespace": "item_events",
    "type": "record",
    "name": "ItemPriceChange",
    "fields": [
        {"name": "item_id", "type": "string" },
        {"name": "old_price", "type": "int" }
    ]
}
```
• Consumers don't care about user_id, so this still subsumes consumer's schema.
Subsume Example

Guarantee Schema Changes

- Consumers rely on `new_price`, so this no longer subsumes their schema's

```json
{
    "namespace": "item_events",
    "type": "record",
    "name": "ItemPriceChange",
    "fields": [
        {"name": "item_id", "type": "string" },
        {"name": "old_price", "type": "int" },
        {"name": "updated_price", "type": "int" }
    ]
}
```
Subsume Example
New Expected Schema

```
{
  "namespace": "item_events",
  "type": "record",
  "name": "ItemPriceChange",
  "fields": [
    {"name": "item_id",   "type": "string" },
    {"name": "old_price", "type": "int"    },
    {"name": "reason",    "type": "string" }
  ]
}
```

- The guarantee schema no longer subsumes this one!
Confounders

• Schemas are complex - can we programmatically check subsumption?
• How to uniquely identify guarantees w/out coupling?
  • styling_app_changes_order_items BAD
  • changes_order_items TOO GENERIC?
• Easily actually writing and managing these tests
• Oh, and actually building this :(
item_price_update

{ 
  :item => { 
    :id => 8387,
    :new_price => "70.12",
    :old_price => "5.38"
  }
}

ItemPriceUpdater

PriceCache

PackSlipUpdater
before do
  updater.update(item, new_price)
end

it "should update the item's price" do
  expect(item.price).to eq(new_price)
end

it "should send a message about it" do
  expect(Pwwka::Transmitter).to have_sent_message(
    matching_schema: :item_price_change,
    identified_by: :price_change,
    payload_including: {
      item: {
        id: item.id,
        new_price: new_price,
        old_price: original_price,
      }
    },
    on_routing_key: "sf.item_price_change"
  )
end
ItemPricerUpdater Spec

```ruby
expect(Pwwka::Transmitter).to have_sent_message(
  matching_schema: :item_price_change,
  identified_by: :price_change,
  payload_including: {
    item: {
      id: item.id,
      new_price: new_price,
      old_price: original_price,
    }
  }
)
```
ItemPricerUpdater Schema

```json
{
    "type": "object",
    "required": ["item"],
    "properties": {
        "item": {
            "type": "object",
            "required": ["id", "new_price", "old_price"],
            "properties": {
                "id": {"type": "integer"},
                "new_price": {"type": "string"},
                "old_price": {"type": "string"}
            }
        }
    }
}
```
ItemPricerUpdater Guarantee

```json
{
"id": "price_change",
"schema": {
"type": "object",
"required": ["item"],
"properties": {
"item": {
"type": "object",
"required": ["id", "new_price", "old_price"],
"properties": {
"id": {"type": "integer"},
"new_price": {"type": "string"},
"old_price": {"type": "string"}
}
}
},
"metadata": {
"routing_key": "sf.item_price_change"
}
};
```

"example_payload": {
"item": {
"id": 1,
"new_price": "34.45",
"old_price": "12.34"
}
}
it "updates the cache with the new price" do
    payload = receive_message(
        guaranteed_by: :price_change,
        expected_schema: :price_cache_price_change,
        app_name: "financial_data_warehouse",
        use_case: "cache_price"
    )

    cached_item = PriceCacheHandler.cache[payload["item"]["id"]]

    expect(cached_item).to eq(payload["item"]['new_price'])
end
ItemPricerUpdater Guarantee

```json
{
    "app_name": "wms",
    "use_case": "pack_slip_exists",
    "guarantee_id": "price_change",
    "schema": {
        "type": "object",
        "required": ["item"],
        "properties": {
            "item": {
                "type": "object",
                "required": ["id", "new_price"],
                "properties": {
                    "id": {
                        "type": "integer"
                    },
                    "new_price": {
                        "type": "string"
                    }
                }
            }
        }
    },
    "example_payload": {
        "item": {
            "id": 1234,
            "new_price": "34.12"
        }
    }
}
```
PackSlip Spec

```ruby
receive_message(
  guaranteed_by: :price_change,
  expected_schema: :pack_slip_new_price,
  app_name: "wms",
  use_case: "pack_slip_exists",
  override_sample: {
    "item" => { "id" => item_id, "new_price" => price }
  }
)
```

Override the published sample
(checks the overridden payload against schemas)
It Works!
It Works!

~/Projects/StitchFix/event-lawyer>
Let’s Break Something

```json
{
  "app_name": "wms",
  "use_case": "pack_slip_exists",
  "guarantee_id": "price_change",
  "schema": {
    "type": "object",
    "required": [ "item" ],
    "properties": {
      "item": {
        "type": "object",
        "required": [ "id", "reason", "new_price" ],
        "properties": {
          "id": { "type": "integer" },
          "reason": { "type": "string" },
          "new_price": { "type": "string" }
        }
      }
    }
  },
  "example_payload": {
    "item": {
      "id": 1234,
      "new_price": "34.12",
      "reason": "markdown"
    }
  }
}
```
It Catches It!

```
~/Projects/StitchFix/event-lawyer> make producer:verify
```
How Real is This?

- The code is on GitHub: https://github.com/davetron5000/event_lawyer
- It’s in Ruby (sorry not sorry)
- I think this has potential as a concept!
Thanks!!

Dave Copeland
@davetron5000

Get a job: http://multithreaded.stitchfix.com/careers/

Read my blog: http://naildrivin5.com