No exactamente. El Caso 1 que propuse demuestra **detección de bugs**, pero no captura la esencia completa de las pruebas de regresión.

¿Qué falta en mi propuesta inicial?

Las pruebas de regresión se centran en verificar que:

- **Código que ANTES funcionaba SIGUE funcionando** después de cambios
- Se prueban funcionalidades **ya validadas previamente**
- Se evita la introducción de **defectos en código existente**

Demo Corregida que SÍ demuestra Regression Testing:

Escenario mejorado (5-7 minutos):

1. **Estado inicial**: Aplicación de e-commerce funcionando con suite de pruebas completa (todas pasan ✅)

2. **Cambio legítimo**: Agregar una nueva funcionalidad (ej: "código de cupón de descuento")

3. **Ejecutar pruebas de regresión**:
    - Mostrar que las pruebas **viejas** (login, carrito, checkout) siguen pasando
    - Demostrar que el cambio NO rompió funcionalidades existentes
4. **Introducir regresión accidental**: Al agregar cupones, se rompió el cálculo de impuestos

5. **Las pruebas de regresión detectan el problema**:
    - Prueba de "cálculo de total" (que antes pasaba) ahora falla ❌
    - Esto demuestra cómo regresión testing protege código existente
6. **Mostrar optimización**:
    - Re-test all: 50 pruebas, 2 minutos
    - Selective: Solo 12 pruebas afectadas, 30 segundos
    - Ambas detectan la regresión

¿Qué hace que esto sea genuinamente Regression Testing?

✅ Verifica código **previamente funcional**
✅ Detecta **defectos introducidos por cambios**
✅ Demuestra estrategias (re-test all vs selective)
✅ Muestra el valor de optimización (tiempo/recursos)

```
regression-testing-demo/
│
├── README.md
├── package.json
│
├── src/
│   ├── index.html
│   ├── styles.css
│   └── app.js
│       ├── cart.js
│       ├── checkout.js
│       ├── auth.js
│       └── coupon.js          # Nueva funcionalidad (introduce regresión)
│
├── tests/
│   ├── setup.js
│   │
│   ├── existing-tests/        # Pruebas que ya existían (regresión)
│   │   ├── auth.test.js       # Login, registro
│   │   ├── cart.test.js       # Agregar/remover productos
│   │   ├── checkout.test.js   # Cálculo de totales, impuestos
│   │   └── product.test.js    # Búsqueda, filtros
│   │
│   ├── new-tests/             # Pruebas para nueva funcionalidad
│   │   └── coupon.test.js     # Validación de cupones
│   │
```

```
│   └── regression-suite.js    # Configuración de suites
│
├── scripts/
│   ├── run-all-tests.sh       # Re-test all (50 tests)
│   ├── run-selected-tests.sh  # Selective (12 tests afectados)
│   └── demo-scenario.sh       # Script automatizado para demo
│
├── demo/
│   ├── scenario-1-baseline.js    # Estado inicial (todo funciona)
│   ├── scenario-2-new-feature.js # Agregar cupones
│   ├── scenario-3-regression.js  # Introducir bug en impuestos
│   └── metrics-comparison.json   # Métricas para mostrar
│
├── coverage/                  # Reportes de cobertura
│   └── index.html
│
└── ci/
    └── jenkins-pipeline.groovy   # (Opcional) Para mostrar CI/CD
```

Explicación de la estructura:

src/ - La aplicación

- `cart.js`, `checkout.js`, `auth.js`: Código existente y funcional
- `coupon.js`: **Nueva funcionalidad** que introducirá la regresión

tests/existing-tests/ - El núcleo de la demo

Estas pruebas **ya pasaban antes** y deben seguir pasando. Aquí es donde se detecta la regresión:

- `checkout.test.js` fallará cuando agreguen cupones (regresión en cálculo de impuestos)

scripts/ - Para la presentación

- `run-all-tests.sh`: Ejecuta las 50 pruebas (2 minutos)
- `run-selected-tests.sh`: Ejecuta solo las 12 pruebas relacionadas (30 seg)
- `demo-scenario.sh`: Automatiza toda la demo

demo/ - Escenarios preparados

Estados guardados para cambiar rápidamente durante la presentación:

1. Todo funciona ✅
2. Nueva funcionalidad agregada ✅
3. Regresión introducida ❌ (pruebas viejas fallan)

Flujo de la demo:

1. Estado inicial

npm test # 50/50 pruebas pasan ✅

2. Agregar cupones

cp demo/scenario-2-new-feature.js src/coupon.js

npm test # 55/55 pruebas pasan ✅

3. Se introduce regresión (accidental)

cp demo/scenario-3-regression.js src/checkout.js

npm test # 53/55 fallan ❌

checkout.test.js detecta el problema

4. Comparar estrategias

npm run test:all # 2 minutos, detecta regresión

npm run test:selected # 30 segundos, detecta regresión

¿Les genero el código completo para esta estructura?