No pierdas el tiempo escribiendo tests

Juan María Hernández (@gulnor)

Quién soy

Juan María Hernández

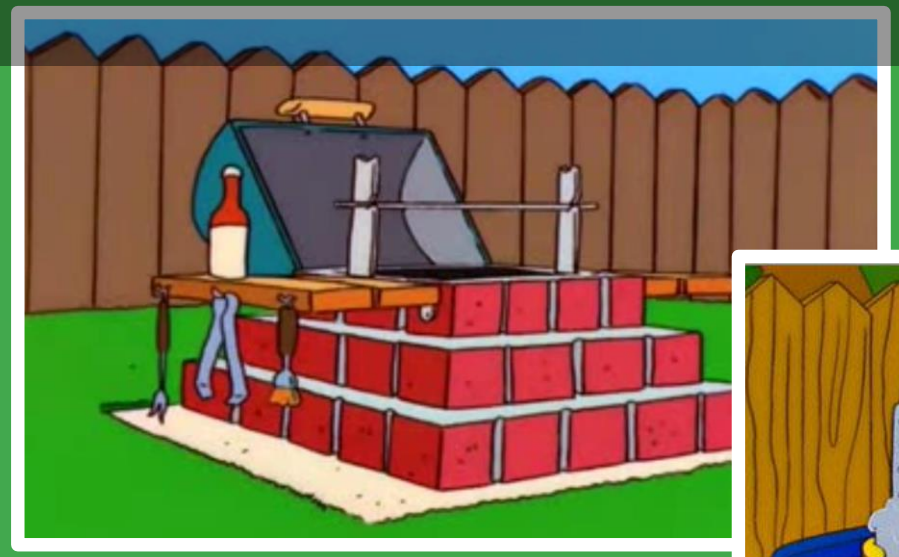
- Desarrollo software.
- Escribo un blog: http://blog.koalite.com
- Me quejo por todo en twitter: **@gulnor**

Qué vamos a ver

1. Problemas típicos al escribir tests
2. Para qué querría alguien escribir tests
3. Cómo escribir código testeable
4. Cómo escribir tests sólidos
5. Qué tipo de tests escribir
6. Para qué más podemos usar los tests

Disclaimer: Survivor Bias

Primer intento



Vi la luz

Tests **UNITARIOS**

Tests unitarios

```
class Person {
  public Person(string firstName, string lastName) {
    FirstName = firstName;
    LastName = lastName;
  }

  public string FirstName { get; }
  public string LastName { get; }
  public string FullName => $"{FirstName} ${LastName}";
}
```

Tests unitarios

```
[Test]
public void Person_has_first_name() {
  var person = new Person("Paco", "Porras");
  Assert.Equals(person.FirstName, "Paco")
}

[Test]
public void Person_has_last_name() {
  var person = new Person("Paco", "Porras");
  Assert.Equals(person.LastName, "Paco")
}

[Test]
public void Person_has_full_name() {
  var person = new Person("Paco", "Porras");
  Assert.Equals(person.FullName, "Paco Porras")
}
```

Vi la luz (otra vez)

INYECCIÓN DE DEPENDENCIAS

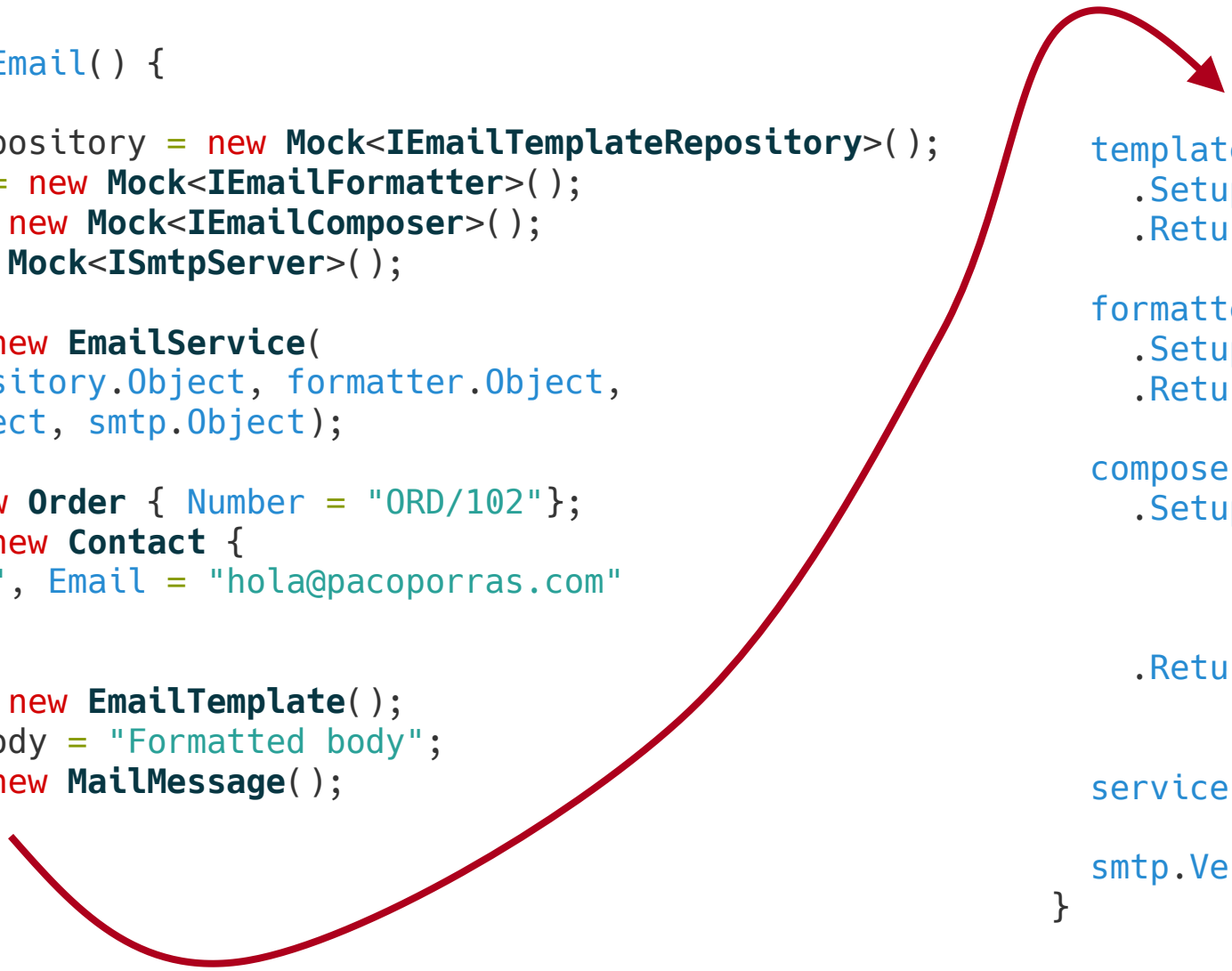
MOCKS
FAKES
STUBS

Así que DI y mocks. Ya, claro.

```
public void SendEmail(Contact contact, Order order) {

  var template = emailTemplateRepository.GetForOrder();
  var body = emailFormatter.Format(template, order);
  var message = emailComposer.Compose(
    contact.Name, contact.Email, $"Order #{order.Number}", body);

  smtpServer.Send(message);
}
```

Así que DI y mocks. Ya, claro.

```
[Test]
public void SendEmail() {

  var templateRepository = new Mock<IEmailTemplateRepository>();
  var formatter = new Mock<IEmailFormatter>();
  var composer = new Mock<IEmailComposer>();
  var smtp = new Mock<ISmtpServer>();

  var service = new EmailService(
    templateRepository.Object, formatter.Object,
    composer.Object, smtp.Object);

  var order = new Order { Number = "ORD/102"};
  var contact = new Contact {
    Name = "Paco", Email = "hola@pacoporras.com"
  };

  var template = new EmailTemplate();
  var formattedBody = "Formatted body";
  var message = new MailMessage();
```



```
  templateRepository
    .Setup(x => x.GetForOrder())
    .Returns(template);

  formatter
    .Setup(x => x.Format(template, order))
    .Returns(formattedBody);

  composer
    .Setup(x => x.Compose(contact.Name,
                          contact.Email,
                          "Order #ORD/102",
                          formattedBody))
    .Returns(message);


  service.SendEmail(contact, order);

  smtp.Verify(x => x.Send(message));
}
```

Sensación

Cuándo escribir tests

"A mi me pagan por escribir código **que funcione**, no por escribir tests."

Kent Beck

Creador de TDD

http://stackoverflow.com/questions/153234/how-deep-are-your-unit-tests

Qué espero de un test

- Debe aportar **confianza*** en el código
  - Validando algo útil
  - Siendo comprensible
  - Ejecutándose continuamente

- Fácil de mantener y extender
  - Cuando cambia el código (refactorización)
  - Cuando aparecen nuevos requisitos

* No podemos aspirar a seguridad. Son sólo tests, no demostraciones formales.


¿Cómo llego a eso?

Analizando dependencias

- Entre el código de producción…
  - … y sus colaboradores
  - … y su estado observable

- Entre el código del test…
  - … y el código que está testeando
  - … y el código que **NO** está testeando

Funciones puras

```
static int Add(IEnumerable<int> numbers) {
  return numbers.Aggregate(0, (a, b) => a + b);
}
```

Estado interno

```
public class Order {

  private readonly List<OrderItem> items = new List<OrderItem>();

  public void Add(string product, decimal quantity, decimal price) {
    items.Add(new OrderItem(product, quantity, price));
  }

  public decimal Total {
    get { return items.Sum(x => x.Quantity * x.Price); }
  }
}
```

Estado observable de otros

```
public class Catalogue {
  private readonly IProductRepository repository;
  public void DiscontinueProduct(int productId, DateTime from, string reason) {
    var product = repository.FindById(productId);
    product.Discontinue(from, reason);
  }
}
```

```
[Test]
public void Product_is_discountinued() {
  var repository = new Mock<IProductRepository>();
  var catalogue = new Catalogue(repository.Object);

  var product = new Product { Id = 5 };
  repository.Setup(x => x.FindById(5)).Returns(product);

  catalogue.DiscontinueProduct(5, DateTime.Today, "Fuera de Stock");

  Assert.True(product.IsDiscontinued)
}
```

Acciones sobre terceros

```
public class CustomerService {
  public void NotifyPreferredCustomers() {
    var customers = repository.FindPreferred();
    foreach (var customer in customers)
      notificationSender.SendNotification(customer);
  }
}

[Test]
public void EachPreferredCustomerIsNotified() {
  var repository = new Mock<ICustomerRepository>();
  var sender = new Mock<INotificationSender>();
  var service = new CustomerService(repository.Object, sender.Object);

  var customers = new [] { new Customer("Marta"), new Customer("Pedro") };

  repository.Setup(x => x.FindPreferred()).Returns(customers);

  service.NotifyPreferredCustomers();

  sender.Verify(x => x.SendNotification(customers[0]));
  sender.Verify(x => x.SendNotification(customers[1]));
}
```

Qué debemos buscar

"La simplicidad es prerrequisito de la fiabilidad."

Edsger Dijkstra

Semidios

Separar lógica de interacción

```
public class OrderStatsCalculator {

  private readonly IOrderRepository repository;

  public OrderStatsCalculator(
    IOrderRepository repository) {

    this.repository = repository;
  }

  public decimal GetAverageAmount() {
    var orders = repository.GetOrders();
    var total = 0;
    foreach(var order in orders)
      total += order.Total;
    return total / orders.Length;
  }
}
```

```
public decimal GetAverageAmount() {
  var orders = repository.GetOrders();
  return GetAverageAmount(orders);
}


// Función pura

public static decimal GetAverageAmount(
  Order[] orders) {

  var total = 0;

  foreach(var order in orders)
    total += order.Total;
  return total / orders.Length;
}
```

Separar lógica de interacción

```
public void Send(ISmsSender sender) {

  this.startTime = DateTime.Now;

  foreach (var recipient in recipients) {
    var message = new Sms(recipient, content);
    sender.Send(message);
  }

  this.endTime = DateTime.Now;
}
```

```
public void Send(ISmsSender sender) {
  send(sender.Send);
}

public void Send(Action<Sms> send) {
  this.startTime = DateTime.Now;

  foreach (var recipient in recipients) {
    var message = new Sms(recipient, content);
    this.send(message);
  }

  this.endTime = DateTime.Now;
}

// ... en el test
var sent = new List<Sms>();
smsCampaign.Send(sent.Add);
Assert.Equal(sent, ...);
```

Minimizar dependencias en tests

- Reducir dependencias sobre código testeado o **NO**
  - Más legibles (mayor confianza)
  - Más fáciles de extender
  - Más estables en el tiempo

ObjectMother

```
[Test]
public void Add_Product_To_Order_Increments_Total() {

  var address = new Address("C/ Lorca", "Parla", "28019", "Madrid");
  var customer = new Customer("Elena", address);

  var category = new ProductCategory("Electronics");
  var tax21 = new Tax("General", 0.21m);
  var laptop = new Product("Laptop", category, tax21);

  var order = new Order(customer);

  order.Add(laptop, quantity: 2, price: 400);

  Assert.Equals(order.Total, 800);
}
```

ObjectMother

```
using static My.App.Tests.ObjectMother;

[Test]
public void Add_Product_To_Order () {
  var order = new Order(Elena);

  order.Add(Laptop, quantity: 1, price: 699);

  Assert.Equals(order.Total, 699);
}
```

```
public class ObjectMother {
  public static Customer Elena =>
    new Customer("Elena", new Address(...));

  public static ProductCategory Electronics =>
    new ProductCategory("Electronics");

  public static Tax Tax21 =>
    new Tax('General', 0.21m);

  public static Product Laptop =>
    new Product("Laptop", Electronics, Tax21);

  // ...
}
```

Builder

```
[Test]
public void Cannot_Reassing_Completed_Task() {

  var paco = new User("Paco", UserProfile.User);
  var lucas = new User("Lucas", UserProfile.Admin);
  var location = new Location("C/ Salvador, 12", "28010", "Madrid");

  var task = new Task(location, "Cambiar filtro aire");
  task.Assign(paco);
  task.Complete("Filtro cambiado");

  Assert.Throws<InvalidOperationException>(() =>
    task.Assign(lucas));
}
```

Builder

```
[Test]
public void Cannot_Reassing_Completed_Task() {

  Task task = Build.Task().Completed();

  Assert.Throws<InvalidOperationException>(
      () => task.Assign(lucas));
}


public static class Build {
  public Task Task() => new TaskBuilder();
}
```

```
public class TaskBuilder {
  private readonly Task task;

  public TaskBuilder() {
    task = new Task(ObjectMother.Location,
                    "Hacer algo");
  }

  public TaskBuilder AssignedTo(User user) {
    task.Assign(user);
    return this;
  }

  public TaskBuilder Completed() {
    if (!task.IsAssigned)
      task.Assign(ObjectMother.Paco);
    task.Complete("Terminado con éxito")
    return this;
  }

  public static implicit Task(TaskBuilder b) {
    return b.task;
  }
}
```

Factory Methods

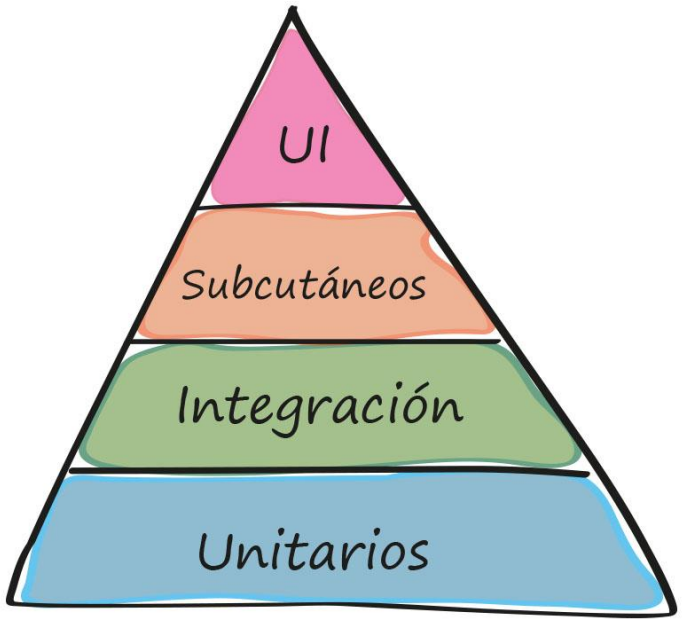
```
[Test]
public void Returns_VIP_Customers_Of_City() {
  var juan = new Customer("Juan", "A8910120",
    new Address("C/ Uno", "28019", "Madrid")) { IsVip = true };
  var mateo = new Customer("Mateo", "B8102901",
    new Address("C/ Dos", "28099", "Madrid")) { IsVip = false };
  var marcos = new Customer("Marcos", "A12312333",
    new Address("C/ Tres", "08800", "Barcelona")) { IsVip = true };
  var lucas = new Customer("Lucas", "B88281020",
    new Address("C/ Cuatro", "08016", "Barcelona")) { IsVip = false };

  var customers = CustomerQuery.Filter(city: "Madrid", custs: new[] {
    juan, mateo, marcos, lucas
  })

  Assert.Equal(customers.Single(), juan);
}
```

Factory Methods

```
[Test]
public void Returns_VIP_Customers_Of_City() {
  var juan = Customer(city: "Madrid", isVip: true);
  var mateo = Customer(city: "Madrid", isVip: false);
  var marcos = Customer(city: "Barcelona", isVip: true);
  var lucas = Customer(city: "Barcelona", isVip: false);

  var customers = Filter("Madrid", juan, mateo, marcos, lucas);

  Assert.Equal(customers.Single(), juan);
}

private Customer Customer(string city, bool isVip) {
  var address = new Address("Calle", "Código Postal", city);
  return new Customer("Name", "Cif", address) { IsVip = isVip };
}

private Customer[] Filter(string city,
                          params Customer[] customers) {
  return CustomerQuery.Filter(city, customers).ToArray();
}
```

La pirámide de los tests

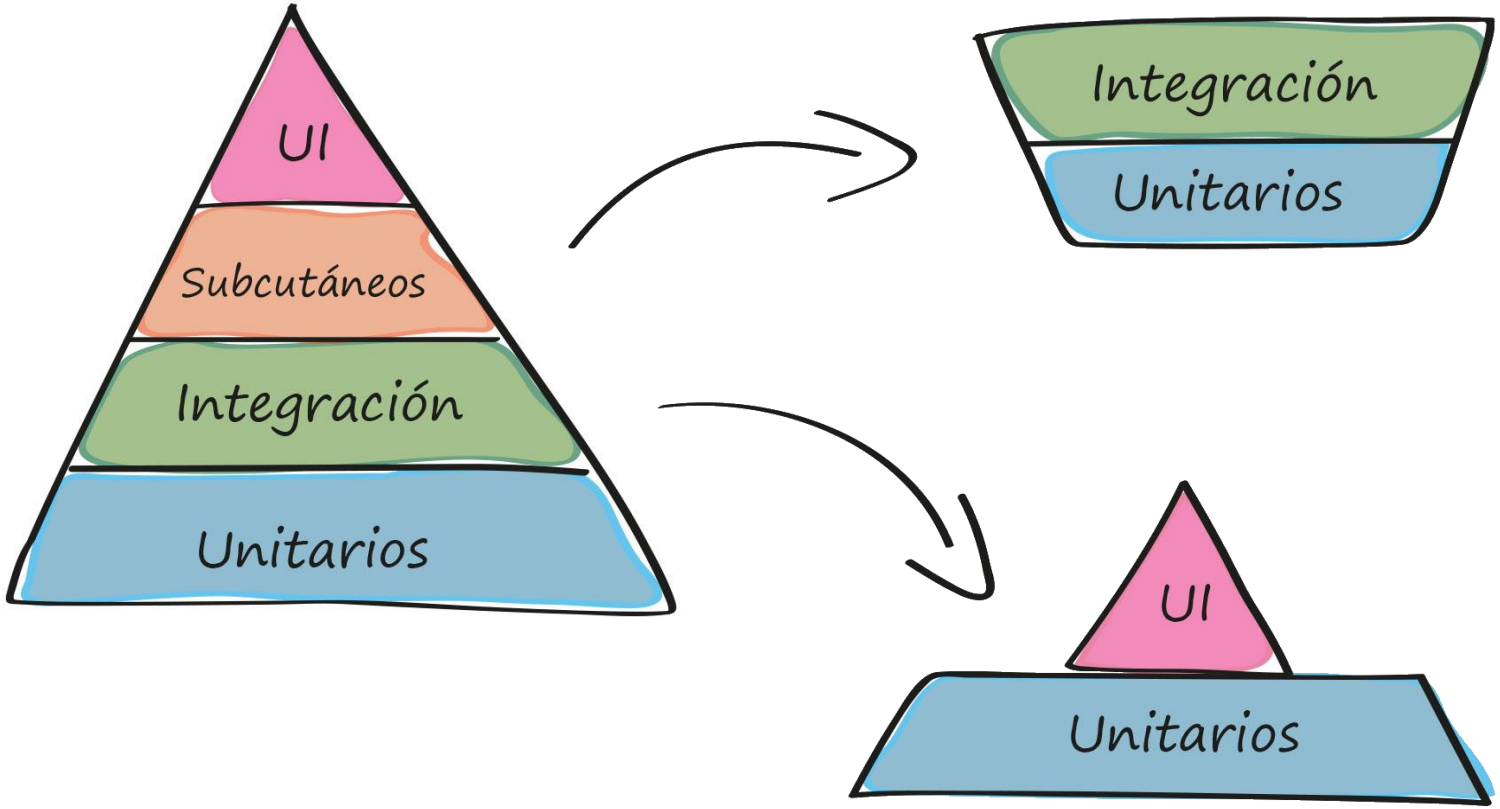
La pirámide de los tests



El secreto de la pirámide

- No todas las aplicaciones son iguales
- Los tests unitarios a veces no sirven
- **Acaba condicionando nuestra arquitectura**
- **Puedes acabar con diseños poco adecuados**

Deconstruyendo la pirámide



Integración BD

- Automatizar la creación de la base de datos
- Recuperar estado entre tests
- Simplificar al máximo el setup de cada test

Integración BD

```
public class TaxByCustomerDTO {
  public string Customer { get; set; }
  public DocumentType Document { get; set; }
  public decimal TaxRate { get; set; }
  public decimal Gross { get; set; }
}


public class ReportService {

  public IEnumerable<TaxByCustomerDTO> GetTaxesByCustomer(DateTime from, DateTime to) {
    // ... consulta SQL espantosa para cargar objetos TaxByCustomer
  }
}
```

Integración BD

```
[Test]
public void Invoices_Are_Aggregated_By_Customer() {
  AddInvoice("21/01/2016", db.Pedro, gross: 100);
  AddInvoice("21/01/2016", db.Pedro, gross: 50);
  AddInvoice("21/01/2016", db.Lucas, gross: 40);

  var taxes = GetTaxes("01/01/2016", "31/01/2016");
  Assert.Equal(taxes.Single(x => x.Customer == db.Pedro.FiscalName).Gross, 150);
  Assert.Equal(taxes.Single(x => x.Customer == db.Lucas.FiscalName).Gross, 40);
}

[Test]
public void Invoices_Are_Aggregated_By_Tax() {
  AddInvoice("21/01/2016", db.Pedro, gross: 100, taxRate: 0.10m);
  AddInvoice("21/01/2016", db.Pedro, gross: 50, taxRate: 0.10m);
  AddInvoice("21/01/2016", db.Pedro, gross: 40, taxRate: 0.05m);

  var taxes = GetTaxes("01/01/2016", "31/01/2016");
  Assert.Equal(taxes.Single(x => x.TaxRate == 0.10m).Gross, 150);
  Assert.Equal(taxes.Single(x => x.TaxRate == 0.05m).Gross, 40);
}
```

Extremo a extremo

- Difíciles de mantener
- Poder levantar la aplicación entera fácilmente
- Simplificar setup con DSLs

Extremo a extremo

Extremo a extremo

```
it('should sell product with formats and addins', function () {

  app.cleanDB();
  app.loginAsMauricio();
  app.selectTerrazaT2();

  screen.Whiskies.click();
  screen.JackDaniels.click();

  screen.titleShouldMatch(/Seleccionar Formato/);

  screen.CopaJackDaniels.click();

  screen.titleShouldMatch(/Añadir Complemento/);

  screen.ConCocaCola.click();
  screen.titleShouldMatch(/Productos/);

  screen.lastLineSummaryShouldContain(/Copa de Jack Daniels/);
  screen.lastLineSummaryShouldContain(/Coca Cola/);

});
```

Usos creativos de tests

Aprovecha la infraestrutura

- Validar convenciones
- Evitar despistes
- Verificar dependencias
- Documentar casos extraños

Validar convenciones

```
[Test]
public void All_Presenters_Have_RequiresPermission_Attribute() {

  var invalid = typeof(PosMainPresenter).Assembly.GetTypes()
    .Union(typeof(AdminMainPresenter).Assembly.GetTypes())
    .Where(x => x.IsConcreteType())
    .Where(x => x.Name.EndsWith("Presenter"))
    .Where(x => !x.HasAttribute<RequiresPermissionAttribute>())
    .Select(x => x.FullName)
    .ToList();

  if (invalid.Any())
    Assert.Fail(
      "Hace falta asignar un permiso a estos presenters:\r\n{0}",
      string.Join("\r\n", failed.ToArray()));
}
```

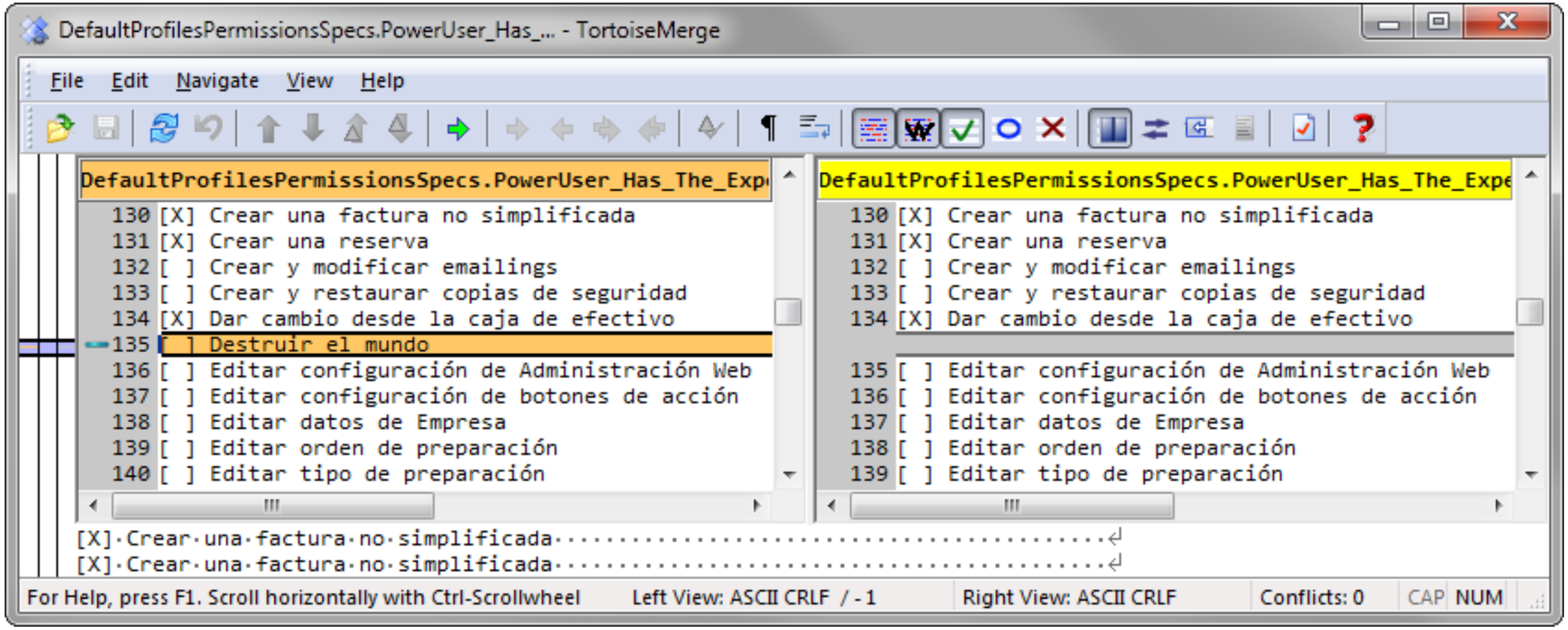
Evitar despistes

```
[Test]
[UseReporter(typeof(DiffReporter))]
public void PowerUser_Has_The_Expected_Permissions() {
  var info = BuildPermissionInformation("PowerUser", GetPowerUserPermissions());
  Approvals.Verify(new ApprovalTextWriterWithEncoding(info, Encoding.Default));
}

private string BuildPermissionInformation(string profileName, Permission[] permissions)
{
  var builder = new StringBuilder();
  builder.AppendFormat("Perfil: {0}", profileName).AppendLine()
    .Append(new string('-', 80)).AppendLine();

  foreach (var permission in GetAllPermissions().OrderBy(x => x.GetDescription())) {
    var isIncluded = permissions.Contains(permission);

    builder.AppendFormat("[{0}] ", isIncluded ? "X" : " ")
      .Append(permission.GetDescription().PadRight(76, ' '))
      .AppendLine();
  }

  return builder.ToString();
}
```

Evitar despistes

```
Perfil: PowerUser
--------------------------------------------------------------------------------
[ ] Abonar pagos previos de un albarán
[ ] Abonar pagos previos de un ticket
[X] Abrir cajón
[X] Acceder a tickets creados por otro usuario
[X] Acceder a tickets en uso por otro usuario
[ ] Activar módulos adicionales
[ ] Ajustar Márgenes
[X] Añadir notas a una línea de un ticket
[X] Añadir pagos a un albarán
[X] Añadir pagos a un ticket
[X] Añadir una línea a un ticket
[X] Añadir una línea de un producto inusual al ticket
[X] Aplicar descuento manual en moneda a un ticket
[X] Aplicar descuento manual en moneda sobre una línea
[X] Aplicar descuento manual en porcentaje a un ticket
[X] Aplicar descuento manual en porcentaje sobre una línea
[X] Aplicar nuevo precio a una línea
[X] Asignar un cliente a un ticket
[X] Borrar líneas de un ticket después de imprimirlas
[ ] Borrar tickets antiguos
[ ] Borrar un centro de venta
[ ] Borrar un cliente
[ ] Borrar un producto
...
```

Validar convenciones




En resumen

Conclusión

- Toda aplicación se testea. Intenta elegir cómo.
- Los tests deben generar confianza.
- Los tests no son gratis. Si no te aportan algo, no los hagas.
- No todas las aplicaciones ni todos los equipos son iguales.
- Busca una estrategia de testing adecuada para tu caso.

¿Preguntas?