Working Effectively with Legacy Code

Si en @cylicon_valley hacemos una jornada sobre libros (:P) estaría bien hablar sobre...

View translation

40% Working eff w Legacy Code

27% Becoming a ninja Angular2

13% Func programming in Scala

20% Software Craftsmanship

Legacy code is...

code that is hard to

change

a mess

legacy code **doesn’t**
need to be old

code without **tests**

Repeat with me...

- You can write very good code in a sea of legacy code

Repeat with me...

- You can write very good code in a sea of legacy code
- You can uglify code to be able to improve/test it

Repeat with me...

- You can write very good code in a sea of legacy code
- You can uglify code to be able to improve/test it
- This process is slow

Repeat with me...

- You can write very good code in a sea of legacy code
- You can uglify code to be able to improve/test it
- This process is slow
- Most of the time we are modifying code (and not adding)

1. Edit and Pray
2. Cover and Modify

Refactoring

Unit Tests

Strategies

1. Identify change points
2. Find test points
3. Break dependencies
4. Write tests
5. Make changes and refactor

Strategies

1. Identify **change points**
2. Find test points
3. **Break dependencies**
4. Write tests
5. Make changes and refactor

Seams

a place where you can **alter behavior** without modifying the code in that place

Seams

- Preprocessing seams: with macros or plugins

Seams

- Preprocessing seams: with macros or plugins
- Link seams: with different libraries

Seams

- Preprocessing seams: with macros or plugins
- Link seams: with different libraries
- **Object seams**

Seams

- Preprocessing seams: with macros or plugins
- Link seams: with different libraries
- **Object seams**

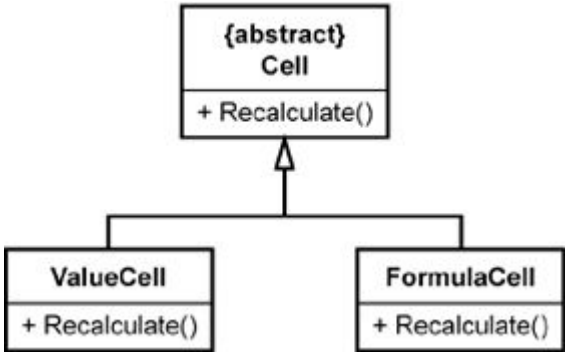
- Every seam has an **enabling point**, a place where you can make the decision to use one behavior or another.

Is this a seam?

```
cell.recalculate();
```

Seams

```
cell.recalculate();
```



This is not a seam

```
public void buildSheet() {
    Cell cell = new FormulaCell(this, "", "");
    cell.recalculate();
}
```

Seams

```
public void buildSheet(Cell cell) {

    cell.recalculate();
}
```

Is this a seam?

```
public void buildSheet(Cell cell) {

    recalculate(cell);
}

private static void recalculate(Cell cell) {
}
```

Break dependencies

1. Sensing: break dependencies to sense when we can’t access values our code computes.

Break dependencies

1. Sensing: break dependencies to sense when we can’t access values our code computes.

2. Separation: break dependencies to separate when we can’t even get a piece of code into a test harness to run.

What if I don’t want to write **tests**?

What I don’t want to write tests...

- **Sprout method**

Sprout method

```
public void postEntries(List<Entry> entries) {

    for (Entry entry : entries) {

            entry.postDate();


    }

    transaction.getListManager().addAll(entries);
}
```

Sprout method

```
public void postEntries(List<Entry> entries) {
    List entriesToAdd = new LinkedList();
    for (Entry entry : entries) {
        if(!transaction.getListManager().contains(entry)) {
            entry.postDate();
            entriesToAdd.add(entry);
        }
    }

    transaction.getListManager().addAll(entriesToAdd);
}
```

Sprout method

```
public void postEntries(List<Entry> entries) {

    for (Entry entry : entries) {

            entry.postDate();


    }

    transaction.getListManager().addAll(entries);
}
```

Sprout method

```
public void postEntries(List<Entry> entries) {
    List<Entry> filteredEntries = uniqueEntries(entries);
    for (Entry entry : filteredEntries) {

            entry.postDate();


    }

    transaction.getListManager().addAll(filteredEntries);
}
```

What I don’t want to write tests...

- Sprout method
- **Sprout class**

What I don’t want to write tests...

- Sprout method
- Sprout class
- **Wrap method**

Wrap method

```
public void pay() {
    Money amount = new Money();
    for (Timecard card : timecards) {
        if (payPeriod.contains(date)) {
            amount.add(card.getHours() * payRate);
        }
    }
    payDispatcher.pay(this, date, amount);
}
```

Wrap method

```
public void dispathPayment() {
    Money amount = new Money();
    for (Timecard card : timecards) {
        if (payPeriod.contains(date)) {
            amount.add(card.getHours() * payRate);
        }
    }
    payDispatcher.pay(this, date, amount);
}
```

Wrap method

```
public void pay() {
    logPayment();
    dispatchPayment();
}
```

What I don’t want to write tests...

- Sprout method
- Sprout class
- Wrap method
- **Wrap class**

What I don’t want to write tests...

- Sprout method
- Sprout class
- Wrap method
- Wrap class
- **Follow dependency tips**

FAQ

I don’t understand the code to change it

- Read it many times (d’oh!)
- Do sketches
- Extract small methods
- **Scratch Refactoring**

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods
- Look for decisions that can change together

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods
- Look for decisions that can change together
- Look for internal relationships

Class too big and I don't want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods
- Look for decisions that can change together
- Look for internal relationships
- Look for the primary responsibility of the class

Class too big and I don't want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods
- Look for decisions that can change together
- Look for internal relationships
- Look for the primary responsibility of the class
- Do some scratch refactoring

Class too big and I don’t want it to get any bigger

- Divide responsibilities (techniques!)
- Look for group methods
- Look at hidden methods
- Look for decisions that can change together
- Look for internal relationships
- Look for the primary responsibility of the class
- Do some scratch refactoring
- Focus on the current work

My application is all API calls

- Skin and wrap the API
- Responsibility-Based Extraction

My application has no structure

- Tell the story of the system.

What methods should I test?

- Do characterization tests

What methods should I test?

- Do characterization tests
- Sketches

What methods should I test?

- Do characterization tests
- Sketches
- Look for interception and pinch points

Tricks

Tricks

- Use DI, fakes, nulls, null objects, extract interfaces... to separate dependencies

- Preserve signatures

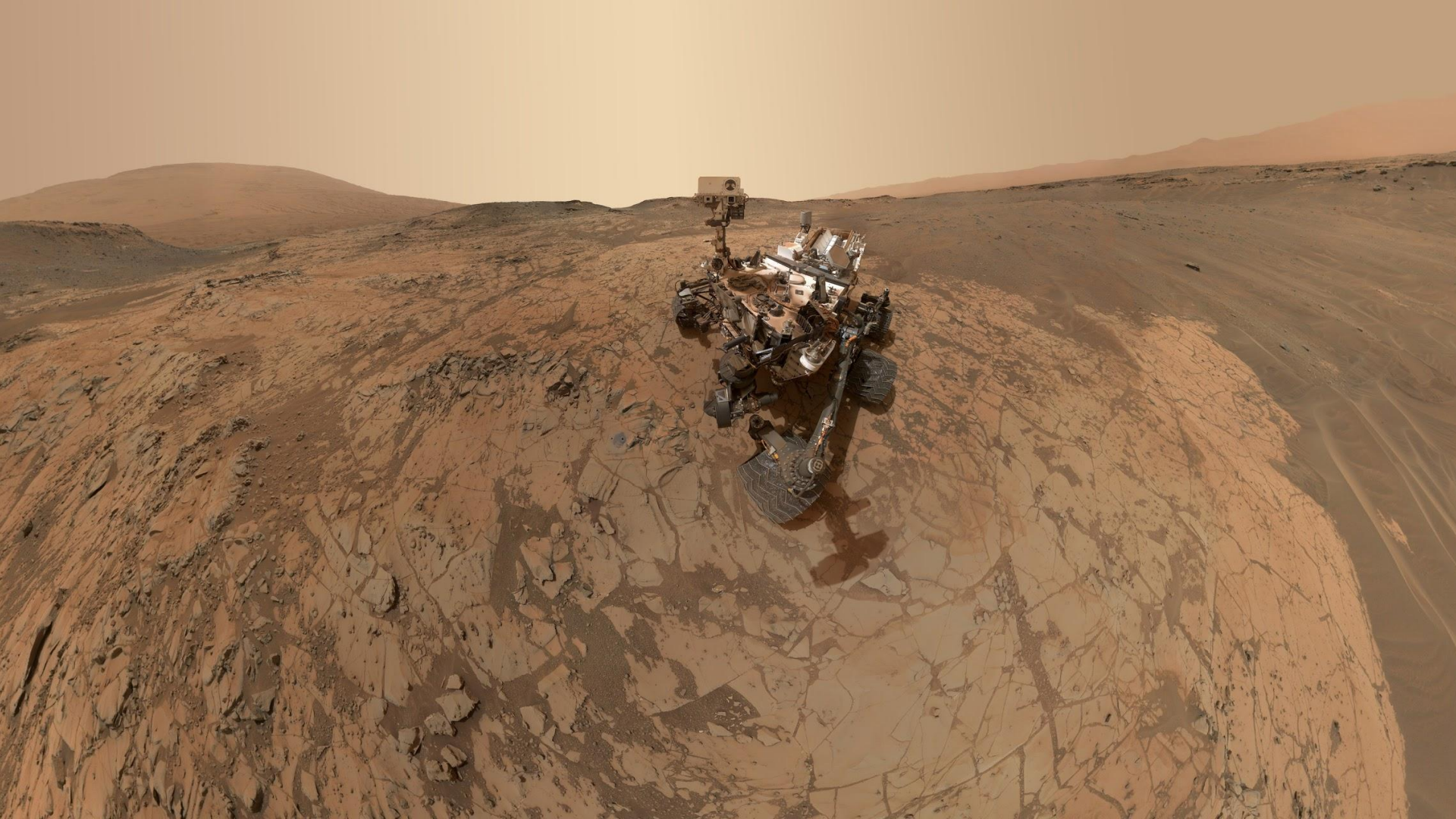
Tricks

- Minimize **coupling points** when extracting: the number of values that pass into and out of the method

- Do extractions with these goals:
  1. To separate logic from dependencies
  2. To introduce seams

Tricks

- **Do 1 thing at a time**



Techniques

Techniques

- Adapt Parameter: change signature to pass a parameter

```
public void populate(HttpServletRequest request) {
    String[] values = request.getParameterValues( pageStateName);
    if (values != null && values.length > 0) {
        marketBindings.put(pageStateName + getDateStamp(), values[ 0]);
    }
}
```

Techniques

- Adapt Parameter: change signature to pass a parameter

```
public void populate(ParameterSource source) {
    String[] values = source.getParameterForName(pageStateName);
    if (values != null && values.length > 0) {
        marketBindings.put(pageStateName + getDateStamp(), values[0]);
    }
}
```

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method
- Encapsulate Global References

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method
- Encapsulate Global References
- Expose Static Method

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method
- Encapsulate Global References
- Expose Static Method
- **Extract and Override Call / Factory Method / Getter**

Techniques

- **Extract and Override Call / Factory Method / Getter**

```
protected void rebindStyles() {
    styles = StyleMaster.formStyles(template, id);
}
```

Techniques

- **Extract and Override Call / Factory Method / Getter**

```
protected void rebindStyles() {
    styles = formStyles(template, id);
}

public class TestingPageLayout extends Hello {
    protected List formStyles(StyleTemplate template, int id) {
        return new ArrayList();
    }
}
```

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method
- Encapsulate Global References
- Expose Static Method
- Extract and Override Call / Factory Method / Getter
- **Extract interface**

Techniques

- Adapt Parameter: change signature to pass a parameter
- Break Out Method Object: create a class with the method
- Encapsulate Global References
- Expose Static Method
- Extract and Override Call / Factory Method / Getter
- Extract interface
- Introduce Instance Delegator

Techniques

- Skeletonize / Find Sequences

Techniques

- Skeletonize / Find Sequences
- Introduce Static Setter

Techniques

- Skeletonize / Find Sequences
- Introduce Static Setter
- Parameterize Constructor / method

Techniques

- Skeletonize / Find Sequences
- Introduce Static Setter
- Parameterize Constructor / method
- Replace Global Reference with Getter

Techniques

- Skeletonize / Find Sequences
- Introduce Static Setter
- Parameterize Constructor / method
- Replace Global Reference with Getter
- Supersede Instance Variable

Techniques

- Skeletonize / Find Sequences
- Introduce Static Setter
- Parameterize Constructor / method
- Replace Global Reference with Getter
- Supersede Instance Variable
- Text redefinition

What to draw from this book...

What to draw from this book...

- **Techniques** to allow safe changes to enable unit tests

- The concept of **Seams**

About the book itself (IMHO)

- FAQ & Catalogue
- Java/C/C++
- Long
- **Many** refactoring examples
- Very test focused

- I don’t **love** this book

questions?

@nhpatt
@cylicon_valley
@agilespain

Cylicon Valley

Cambiar la foto

Valladolid, España
Fundado el 10 de may de 2015

Acerca del grupo...

Invitar a tus amigos

Miembros 252

¡Bienvenido!

+ Programar un nuevo Meetup

Próximos 1 Pasados Calendario

Clean Code, SQL Performance Explained y Working effectively with Legacy Code

Agencia de Innovación
Calle de Vega Sicilia, 2, 47008 Valladolid, Valladolid (mapa)

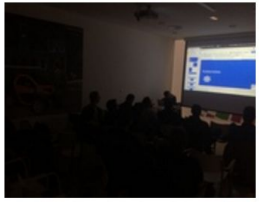
→

Volvemos el 4 de Junio a las 10:00 con 3 charlas sobre libros que nos han cambiado la forma de programar. Juan Ignacio Sánchez Lara nos contará todos los detalles de SQL...
Saber más

Organizado por: Javier Gamarra (Organizador), Alvaro Garcia (Coorganizador), and Juan Ignacio Sánchez Lara

sáb 4 de jun
10:00
✓ ASISTIRÉ
37 asistirán
13 lugares queda
0 comentarios

Novedades


MÁS

Cylicon Valley

Inicio | Miembros | Patrocinadores | Fotos | Páginas | Conversaciones | Más | Herramientas | Mi perfil

Cambiar la foto

Valladolid, España
Fundado el 10 de may de 2015

Acerca del grupo...

Invitar a tus amigos

Miembros 252

¡Bienvenido!

+ Programar un nuevo Meetup

Próximos 1 | Pasados | Calendario

Clean Code, SQL Performance Explained y Working effectively with Legacy Code

Agencia de Innovación
Calle de Vega Sicilia, 2, 47008 Valladolid, Valladolid (mapa)

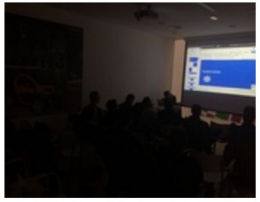
→

Volvemos el 4 de Junio a las 10:00 con 3 charlas sobre libros que nos han cambiado la forma de programar. Juan Ignacio Sánchez Lara nos contará todos los detalles de SQL...
Saber más

Organizado por: Javier Gamarra (Organizador), Alvaro Garcia (Coorganizador), and Juan Ignacio Sánchez Lara

sáb 4 de jun
10:00
✓ ASISTIRÉ
37 asistirán
13 lugares queda
0 comentarios

Novedades


MÁS

Cylicon Valley

Inicio | Miembros | Patrocinadores | Fotos | Páginas | Conversaciones | Más | Herramientas | Mi perfil

Cambiar la foto

Valladolid, España
Fundado el 10 de may de 015

erca del grupo...

Invitar a tus amigos

Miembros 252

¡Bienvenido!

+ Programar un nuevo Meetup

Próximos 1 | Pasados | Calendario

Clean Code, SQL Performance Explained y Working effectively with Legacy Code

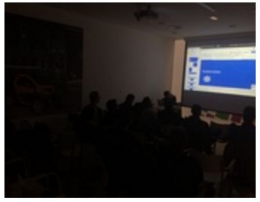
Agencia de Innovación
Calle de Vega Sicilia, 2, 47008 Valladolid, Valladolid (mapa)

sáb 4 de jun
10:00
✓ ASISTIRÉ
37 asistirán
13 lugares queda
0 comentarios

Volvemos el 4 de Junio a las 10:00 con 3 charlas sobre libros que nos han cambiado la forma de programar. Juan Ignacio Sánchez Lara nos contará todos los detalles de SQL...
Saber más

Organizado por: Javier Gamarra (Organizador), Alvaro Garcia (Coorganizador), and Juan Ignacio Sánchez Lara

Novedades


MÁS

Working Effectively with Legacy Code