ACTUALIZACIÓN INFORMÁTICA

CPAP

CURSO DE POSGRADO

Técnicas y Gestión de las Pruebas de Software

Darío Macchi

DOCENTE (invitado)

Pruebas estructurales y estáticas

Pruebas estructurales

ACTUALIZACIÓN INFORMÁTICA
CPAP

Pruebas estructurales

- Son pruebas de caja blanca
  - Basadas en estructura interna
    - código
    - arquitectura
    - flujos de trabajo
    - flujos de datos
- Vamos a manipular las entradas para comparar con salida esperada
- El artefacto a probar ¿Debe estar terminado de desarrollar?

Pruebas estructurales por nivel

Componente: sentencias, decisiones, ramas o incluso caminos distintos.

Integración: árbol de llamadas (un diagrama en el los módulos llaman a otros módulo).

Sistema: flujos / procesos de negocio, flujos de navegación en una página web.

Aceptación: …

Nivel componente

Quiero identificar:

- Instrucciones
- Condicionales
- Ramas de ejecución

Para:

- Construir casos que los ejerciten
- Chequear cuáles están siendo ejercitados
- Determinar si ya puedo parar de probar

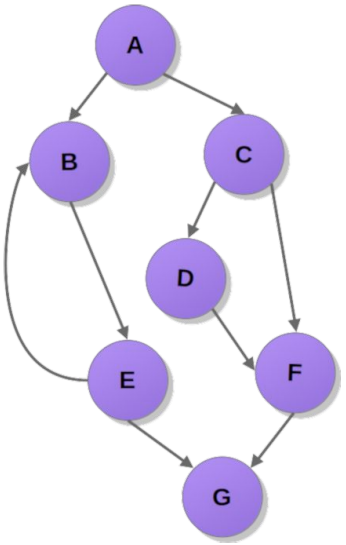
Cobertura de código

- Medida que describe el grado de comprobación del código fuente del programa.

- Forma de prueba de caja blanca para
  - Encontrar áreas del programa no ejercitadas.
  - Crear casos de prueba para aumentar la cobertura
  - Determinar una medida cuantitativa de la cobertura del código

Complejidad ciclomática (McCabe 1976)

```
A  import random
   num = random.randint(1, 10)
   print("Inicia el programa")
   if num > 3:
B      for i in range(num):
E          print("Hola")
C  else:
       if num == 2:
D          print("El número es 2")
F      print("El número es menor a 3")
G  print("Fin del programa")
```



Este valor permite determinar la cantidad de casos de pruebas máx. que serán necesarios para evaluar la totalidad de los posibles flujos del código.

Cobertura de sentencias

$$\text{cobertura de sentencias} = \frac{\text{sentencias ejecutadas}}{\text{total de sentencias}} \text{ X } 100$$

Ejemplo

```
1  Prints (int a, int b) {
2      int result = a+ b;
3      If (result > 0) {
4          Print ("Positive", result)
5      Else
6          Print ("Negative", result)
7      }
8  }
```

Si a=3, b=9

```
1  Prints (int a, int b) {
2      int result = a+ b;
3      If (result > 0) {
4          Print ("Positive", result)
5      Else
6          Print ("Negative", result)
7      }
8  }
```

sentencias ejecutadas = 5
total de sentencias = 7

cobertura de sentencias = (5/7)x100 = **71%**

Si a=-3, b=-9

```
1  Prints (int a, int b) {
2      int result = a+ b;
3      If (result > 0) {
4          Print ("Positive", result)
5      Else
6          Print ("Negative", result)
7      }
8  }
```

sentencias ejecutadas = 6
total de sentencias = 7

cobertura de sentencias = (6/7)x100 = **85%**

Ambos escenarios ejercitan todas las sentencias → cobertura de sentencias = **100%**

ACTUALIZACIÓN INFORMÁTICA
CPAP

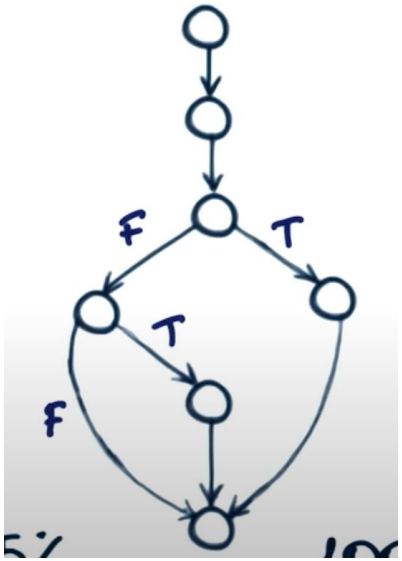
Cobertura de ramas

$$\text{cobertura de ramas} = \frac{\text{ramas ejecutadas}}{\text{total de ramas}} \text{ X } 100$$

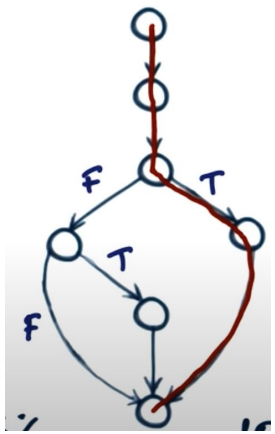
Ejemplo

```
1  function printSum(a, b) {
2    const result = a + b;
3    if (result > 0) {
4      console.log('red:', result);
5    } else if (result < 0) {
6      console.log('blue: ', result);
7    }
8    // else do nothing
9  }
```

En este ejemplo →2 puntos de decisión (IFs) → 4 ramas

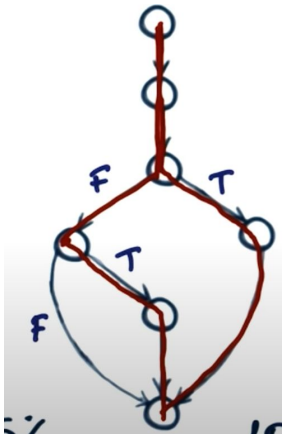


Si a=3, b=9



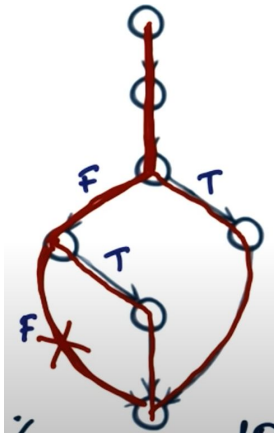
1/4 X 100 = 25%

Si a=-3, b=-9



2/4 X 100 = 50%

Si a=0, b=0



1/4 X 100 = 25%

Pruebas estructurales y estáticas

Pruebas estáticas

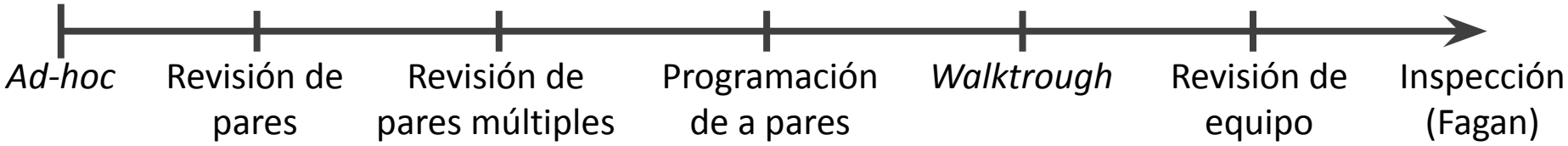
Pruebas estáticas - Revisiones

Definición: técnica que permite detectar defectos en artefactos de software analizando su estructura y contenido

Una revisión tiene 4 etapas

Planificación → Detección de defectos → Consolidación → Seguimiento

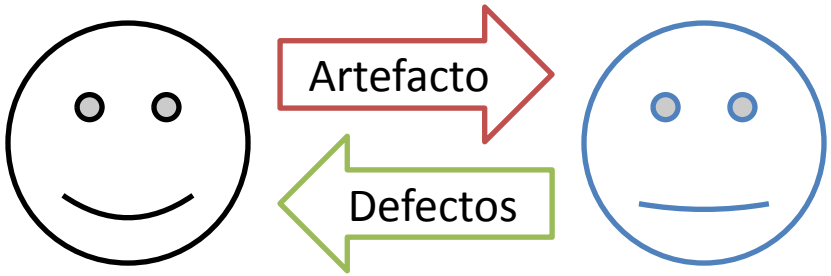
Revisiones según formalidad [Wiegers, 2001]



Formalidad depende de la sistematización de cada etapa

Revisión de pares

Procedimiento: envío copia de artefacto a un par; este la revisa y devuelve lista de defectos.



Adecuada para:

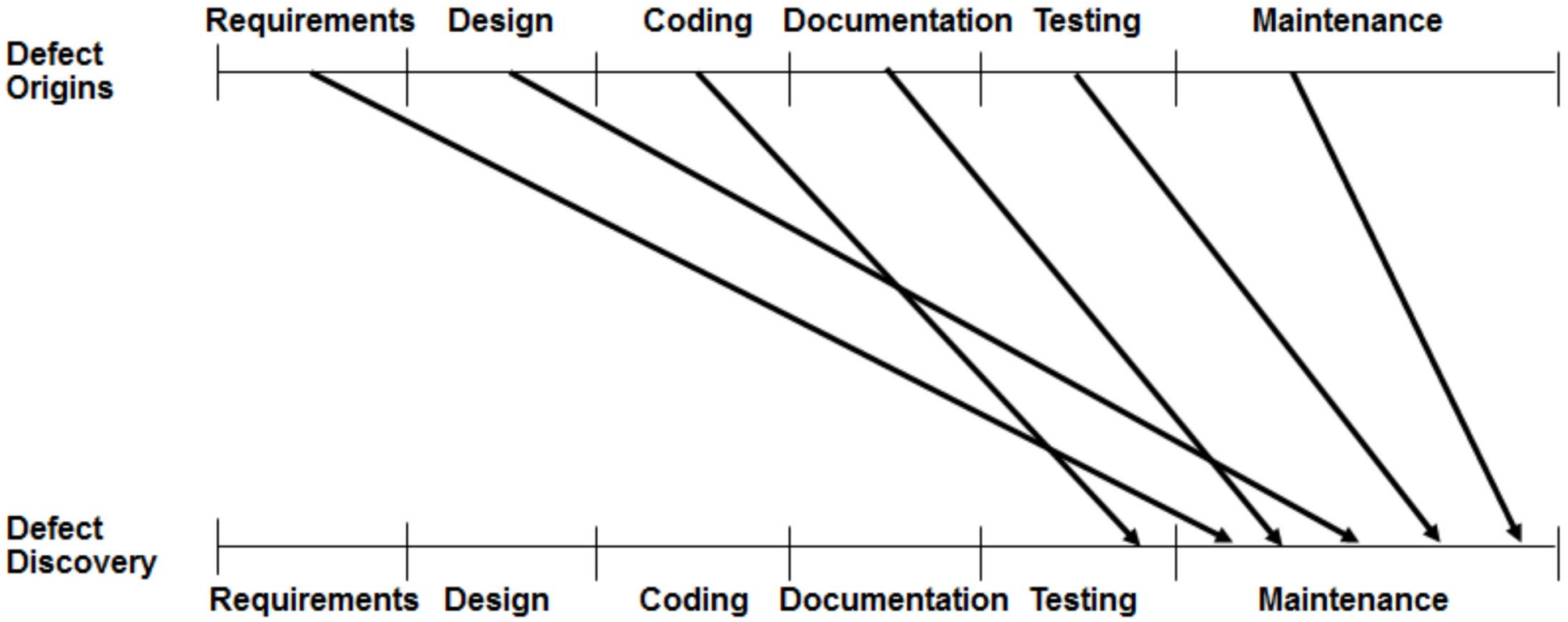
- artefactos de bajo riesgo
- personal hábil en la detección de defectos
- poco tiempo y recursos

[Weigers, 2001]

ACTUALIZACIÓN INFORMÁTICA

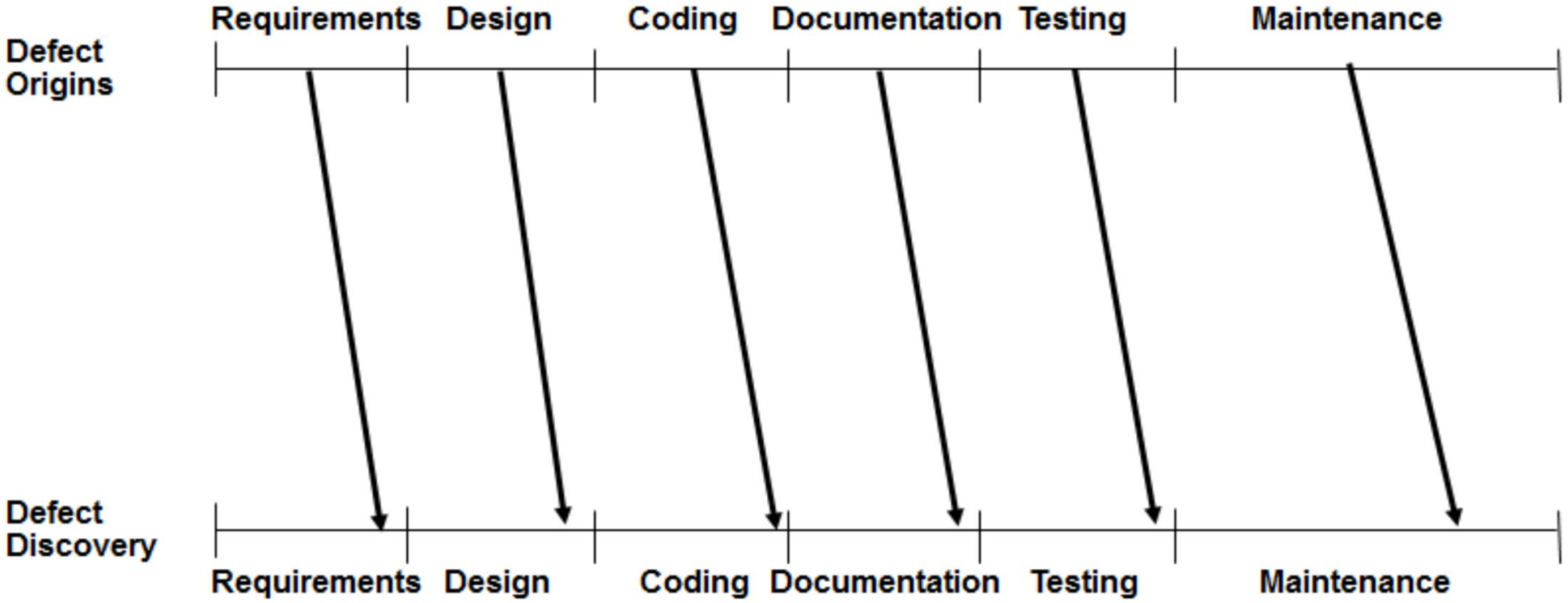
Shift Left para disminuir riesgo

Origen/descubrimiento de defectos



[Capers Jones, 2016]

Origen/descubrimiento de defectos (con revisiones)



[Capers Jones, 2016]

ACTUALIZACIÓN INFORMÁTICA
CPAP

Técnicas de lectura

¿De qué depende la variabilidad de resultados en los procesos de revisión?

Técnicas de lectura

¿De qué depende la variabilidad de resultados en los procesos de revisión?

Variabilidad de resultados de procesos de revisión son independientes del proceso utilizado [Land et al., 1997], [Porter and Johnson, 1997], [Votta, 1993].

- Entonces… ¿de qué depende?

Técnica de lectura:

Serie de pasos cuyo propósito es lograr un entendimiento profundo del artefacto revisado [Laitenberger & DeBaud, 2000].

Para que las necesitamos:

- Durante nuestra formación aprendemos a escribir artefactos pero no a leerlos ni analizarlos [Basili *et al.*, 1996].
- Capturar el conocimiento obtenido de buenas prácticas en detección de defectos [Melo & Shull, 2001].
- Independizar resultados de habilidades del revisor.

Defect Based Reading

- Sistemáticamente buscar cierta clase de defectos
  [Porter et al., 1995]
- Se elaboran escenarios en base a taxonomías de defectos para dirigir la lectura
- Escenarios pueden ser *checklists*
- Convierte al revisor en un lector activo, mejorando la efectividad.
- Aplicado con éxito sobre requerimientos.

Defect Based Reading

Escenario de ejemplo:

1. Por cada SRS identificar todos los datos de entrada y salida.
   a. Los valores escritos en cada salida son consistentes con la intención funcional del SRS?
   b. Identifique al menos una funcionalidad que use cada una de estas salidas.

[Porter et al., 1995]

Esto guía al revisor, dado que (1) indica el *dónde* encontrar defectos (a) y el *cómo* hacerlo (b).

ACTUALIZACIÓN INFORMÁTICA
CPAP

Defect Based Reading

Defectos en artefactos de software se pueden clasificar en:

- Omisiones
- Ambigüedades
- Inconsistencias
- No correctitud
- Información extraña

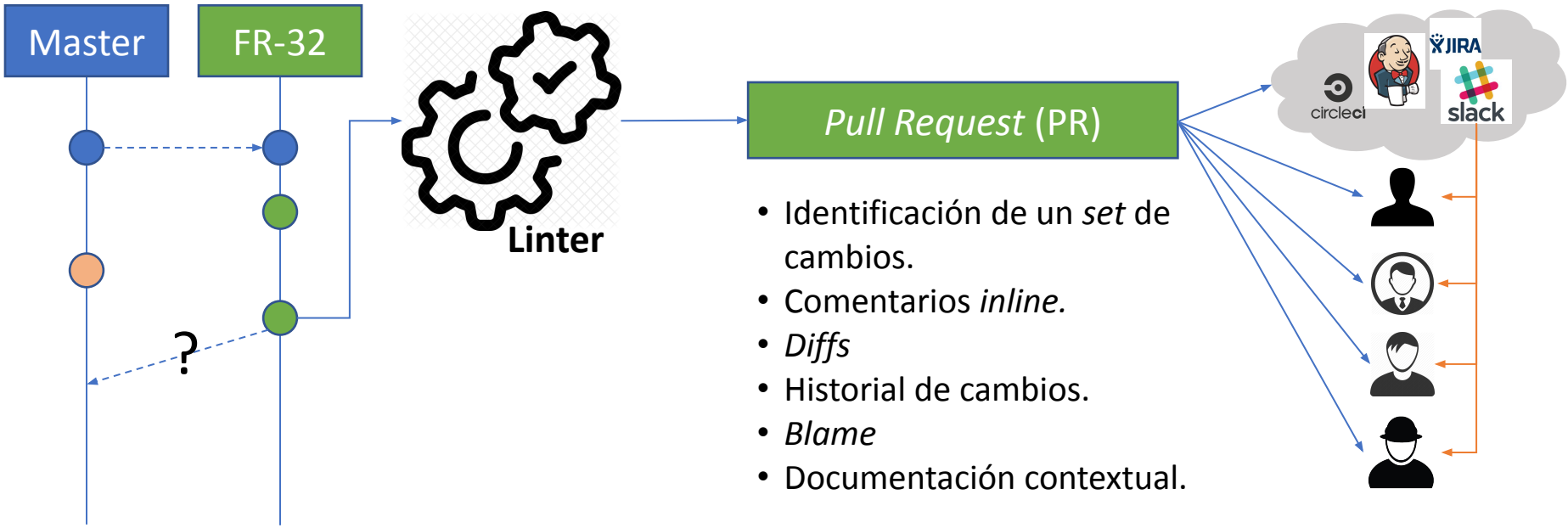
Test-Case Driven Reading

- Obj: revisar requerimientos minimizando el costo.
- *Testers* elaboran *test-cases* para
  - probar que sea testeable
  - detectar
    - faltas de funcionalidades
    - no completitud
    - inconsistencias
    - conflictos entre funcionalidades
- Casos de prueba se siguen usando en siguientes fases → costo de la revisión se amortiza (*lean*).

[Gorschek & Fogelstrom, 2005]

Pruebas estáticas - código

Asumiendo git, podemos usar *pull requests* en Github/Bitbucket para implementar **revisiones de pares múltiples**



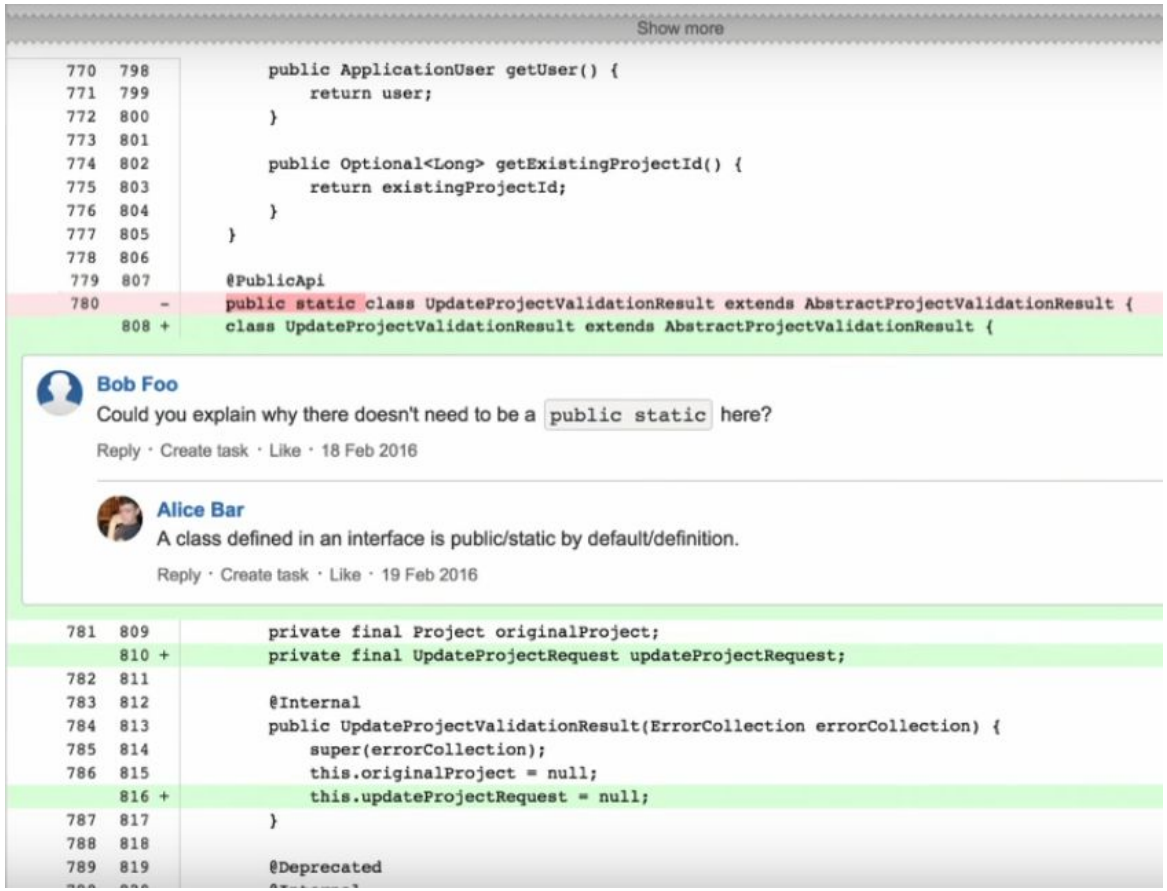
- Identificación de un *set* de cambios.
- Comentarios *inline.*
- *Diffs*
- Historial de cambios.
- *Blame*
- Documentación contextual.

ACTUALIZACIÓN INFORMÁTICA

CPAP

Pruebas estáticas - código

Revisión basada en *diffs* (valorar tiempo del revisor) dónde se da un intercambio entre revisores y autor del artefacto.



ACTUALIZACIÓN INFORMÁTICA

CPAP

Pruebas estáticas - código

Elementos que permitan asimilar mejor el *feedback*: informalidad, chistes, links a referencias bibliográficas, links a revisiones previas, etc.


```
127  +    private boolean canPlayVideo(final Mode mode) {
128  +        return !(playerMediator.isVideoPlayingOrLoading() || !settings.isA
```

dpreussler on Mar 6

is there a test for that? You need to be a vulcanian to read that?

Pruebas estáticas - código