# Confiabilidad y optimización del software.

Hernan Quintana

Universidad de Lima

# Introducción

El objetivo principal de este curso no es solamente escribir código que funcione, si no también que sea **bonito**, por lo tanto **fácil de mantener**.

¿Cómo sabes que cierto código es **poco bonito**? Lo podremos saber utilizando algunas técnicas que nos permitan analizar código.

# Métricas de Software

Ya hemos aprendido una ténica que nos permite analizar código, esta se llama Code Smells.

El análisis mediante **Code Smells** nos dan una visión **cualitativa** de qué tan bien está desarrollado nuestro software, pero muchas veces esto no es suficiente y queremos hacer análisis más detallados.

Por este motivo existen las técnicas **cuantitativas** que abordarán el problema utilizando teorías y técnicas en su mayoría matemáticas.

Estas métricas de software:

- Lines of Code (LOC)
- Complejidad ciclomática (Cyclomatic complexity)
- ABC Score

## LOC (Lines of Code)

Es la métrica más común y tradicional de poder medir que tan complejo (*feo*) puede estar cierto código.

Se basa en medir las líneas de código que un programa puede tener.

- **Beneficios**
  - Simplicidad.
  - Bajo costo (en tiempos)
- **Inconvenientes**
  - No refleja la realidad. Dos código que realizan lo mismo pueden tener diferentes líneas de código (según el estilo de codificación del programador).
  - No se toman en cuenta comentarios y sentencias que no aportan a la complejidad.

## Complejidad Ciclomática

Es una manera de visualizar cierta porción de código como un grafo dirigido. [1: Más información: https://es.wikipedia.org/wiki/Grafo_dirigido]

Fue creada por Thomas J. McCabe en 1976 en un paper que fue publicado por la revista IEEE Transaction on Software Engineering. [2: Más información: http://www.literateprogramming.com/mccabe.pdf]

Al ser un grafo, se le pueden aplicar varios operaciones según la teoría de grafos.

# Pasos para el cálculo de la complejidad ciclomática

## 1. Convertir el código en un grafo

Se van a aplicar las siguientes reglas:

**Código Lineal**

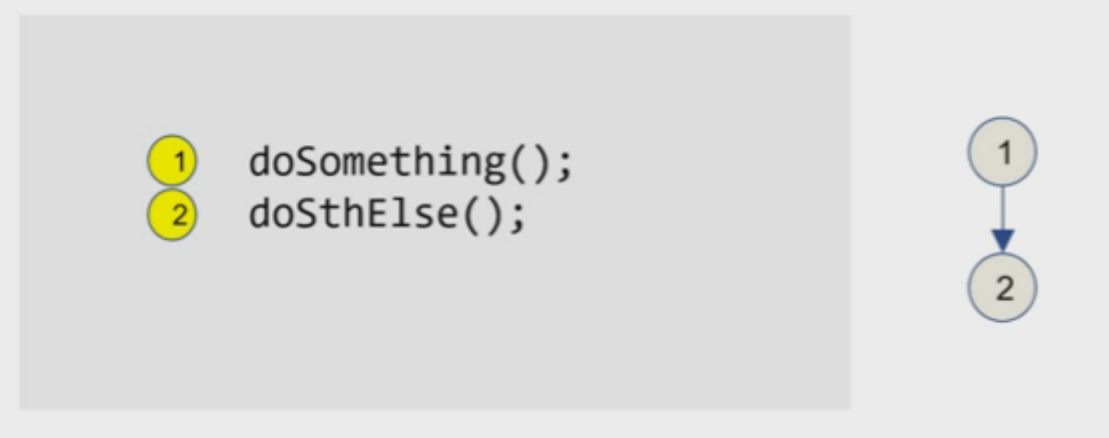



*Figure 1. Lineal*

**Sentencias if**

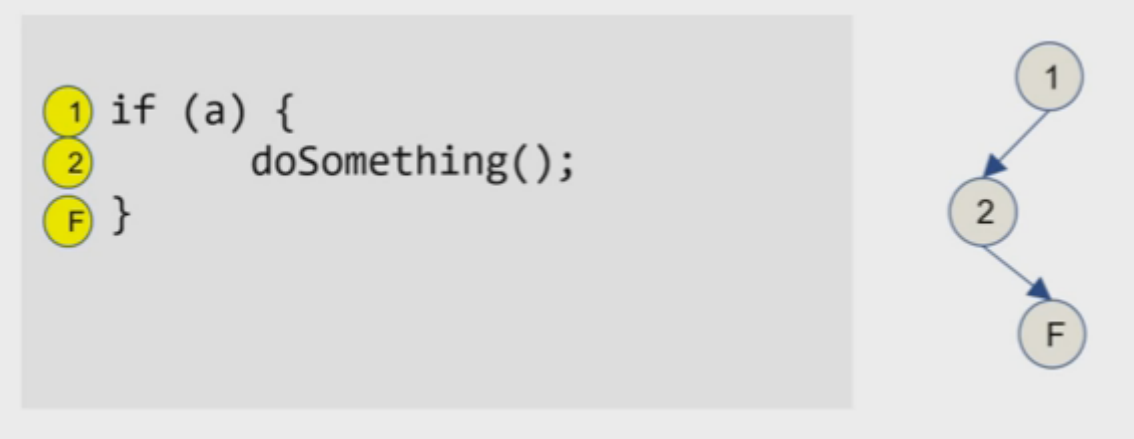



*Figure 2. Si*

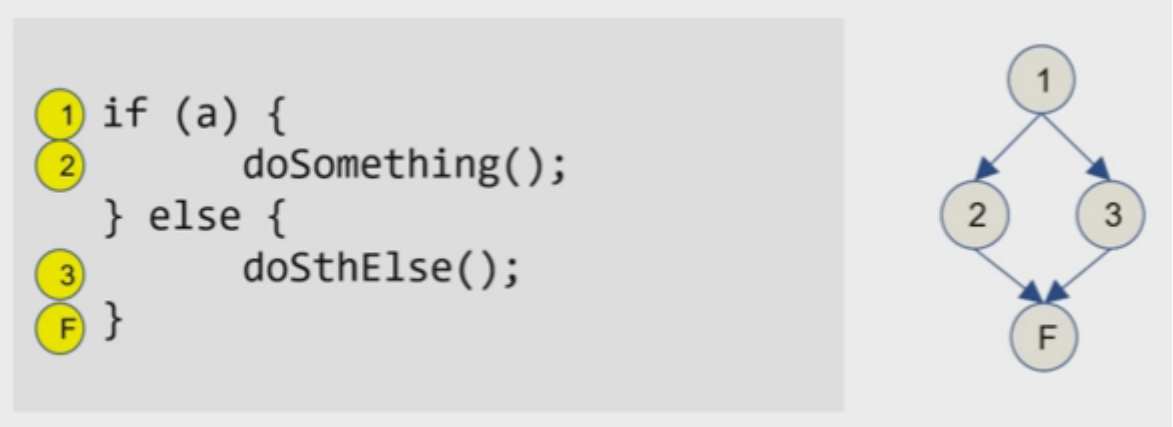



*Figure 3. Si No*

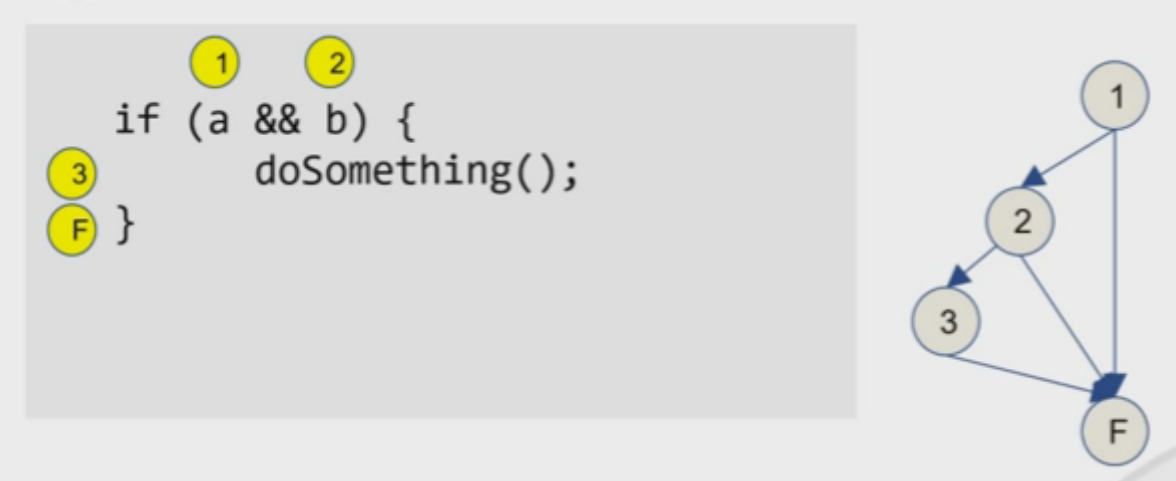



*Figure 4. Si (condicional AND)*

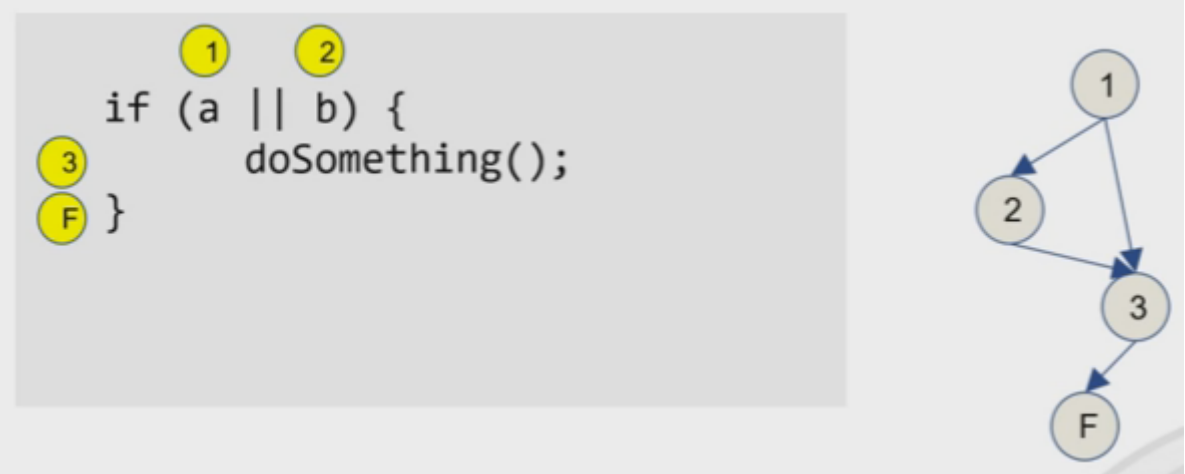



*Figure 5. Si (condicional OR)*

**Sentencias de bucle**


*Figure 6. while*

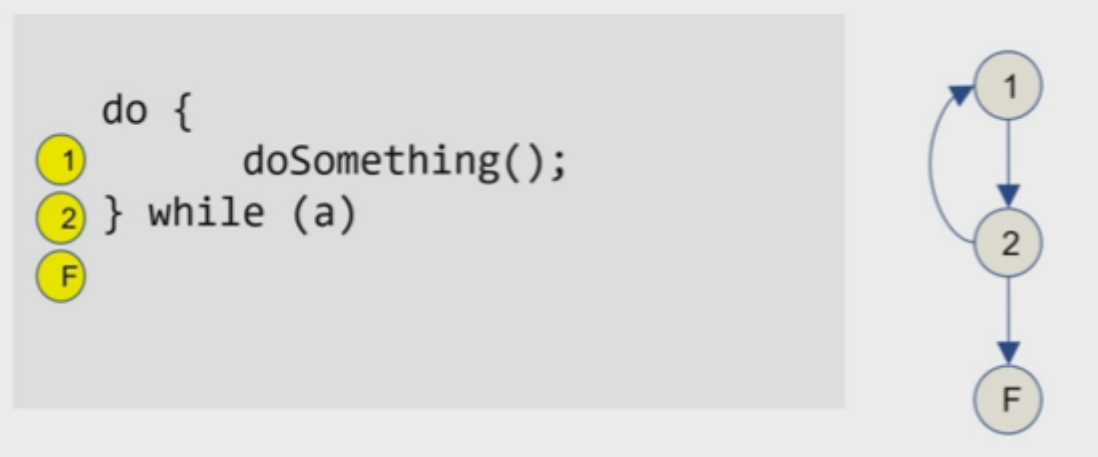



*Figure 7. do while*

**Sentencias switch**

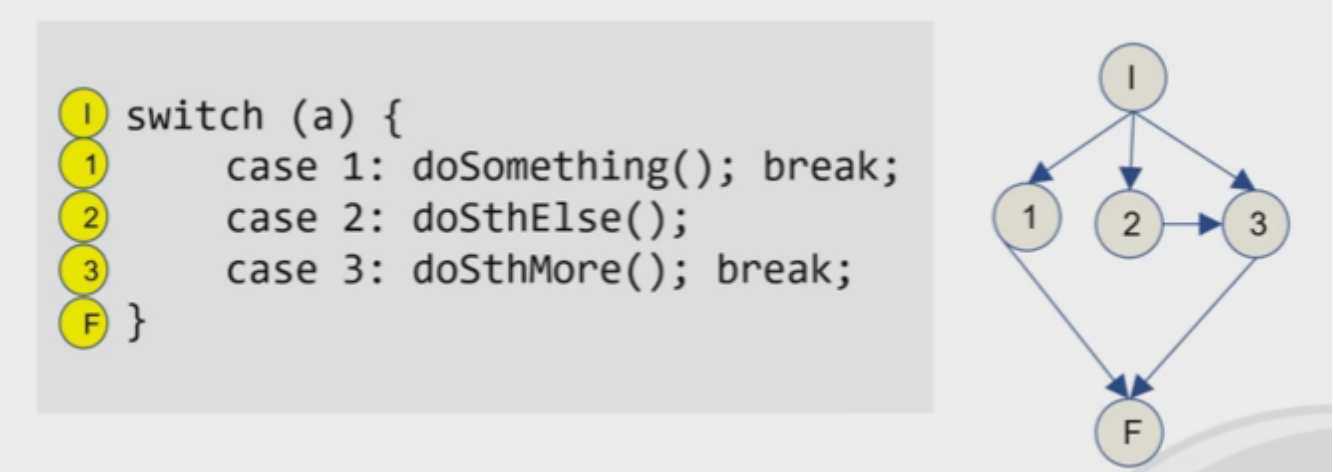



*Figure 8. switch*

## 2. Cálculo del número de caminos posibles

La complejidad ciclomática es el cálculo del número de caminos posibles que nuestro código puede tomar (**métrica de control de flujo**).

Denotaremos a nuestro código con la letra G y la complejidad ciclomática con V(G).

*Fórmula de la complejidad ciclomática*

```
V(G) = e - n + 2p
```

Siendo:

- e : nro de lados (edges)
- n : nro de vértices (nodes)
- p : nro de componentes conectados

Esta formula aplica correctamente cuando p es 1. Para el caso de tener dos o más componentes (p>1) no existe evidencia empírica que la fórmula funcione. [4: Umesh Tiwari and Santosh Kumar. 2014. Cyclomatic complexity metric for component based software. SIGSOFT Softw. Eng. Notes 39, 1 (February 2014), 1-6. DOI=http://dx.doi.org/10.1145/2557833.2557853]

## Ejemplo

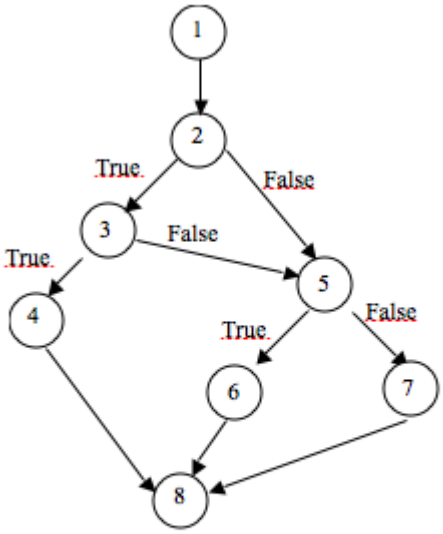
Calcular la complejidad ciclomática del siguiente código.

*Cálculo del máximo*

```
def max():
    x = input("Ingrese el primer número: ")
    y = input("Ingrese el segundo número: ")
    z = input("Ingrese el tercer número: ")

    maximo = 0
    if x > y and x > z:
        maximo =  x
    elif z > y:
        maximo = z
    else:
        maximo = y

    print(maximo)
```




Calculamos la complejidad ciclomática de McCabe :

$V(G) = a - n + 2 = 10 - 8 + 2 = 4$

$V(G) = r = 4$

$V(G) = c + 1 = 3 + 1 = 4$

*Figure 9. Cálculo de la complejidad ciclomática de un grafo.*

# Cuadro de riesgo

Según McCabe, debemos tener en cuenta el siguiente cuadro cuando tratemos de clasificar nuestros programas.

*Table 1. Según McCabe*

| **Complejidad ciclomática** | **Evaluación de riesgo** |
|---|---|
| 1-10 | Programa simple, sin mucho riesgo. |
| 11-20 | Ma s complejo, riesgo moderado. |
| 21-50 | Complejo, programa de alto riesgo. |
| 50 < | Programa no testeable. Muy alto riesgo. |

*Table 2. Según herramienta radon*

| **Complejidad ciclomática** | **Evaluación de riesgo** |
|---|---|
| 1-5 | A (Nada de riesgo. Programa muy simple) |
| 6-10 | B (Bajo riesgo) |
| 11-20 | C (Riesgo moderado) |
| 21-30 | D (Más que riesgo moderado) |
| 31- 40 | E (alto riesgo - bloque complejo). |
| 41+ | F (muy riesgoso, bloque inestable) |

**A menor complejidad el código tiene una mayor mantenibilidad por lo que tendrá una baja probabilidad de haber errores.**

# ABC Score

No mide complejidad, pero provee una manera de medir tamaños de código fuente sin prestar atención a distintos estilos de programación.

# Método de cálculo

Se calculará el vector ABC conformado por 3 componentes:

- **Assigments** : Transferencia explícita de datos a una variable
- **Branch** : Una llamada explicita a código fuera del alcance de la función analizada.
- **Conditionals** :Un test lógico que devuelve una variable booleana.

Se contarán cada una de las ocurrencias de estos componentes en una porción de código y se le asignará a un vector V.

*Vector ABC*

```
V = V<a, b, c>
```

- a : Cantidad de *assignments*.
- b : Cantidad de *branches*.
- c : Cantidad de *conditionals*.

Entonces, el cálculo de la métrica abc será dada por la siguiente fórmula:

*Fórmula cálculo métrica ABC*

```
V<a, b, c> = sqrt( ( a * a ) + ( b * b ) + ( c * c) )
```

# Reglas para Python

1. Se contará como un *assigment* cualquier ocurrencia de los siguientes operadores: = *= /= %= += <⇐ >>= &= |= ^=
2. Se contará como un *branch* cualquier llamada a un función o método de algún objeto (se incluye el constructor).
3. Se contará como *condición* según la ocurrencia de estos operadores condicionales: == != ⇐ >= < >
4. Se contará como *condición* según la ocurrencia de esta *keyword*: else
5. Para el caso de bloques try, estos se contarán como *conditionals*.

# Ejemplo

Calcular la métrica ABC del siguiente código:

```
def modificacion_matricula(request, idmatricula):
  if request.method == 'GET':
    matricula = Matricula.objects.get(id=idmatricula)

    matriculaForm = MatriculaForm(instance=matricula)

  return Render_to_response(
    'matricula-registrar.html',
    {'form':matriculaForm}, context_instance=RequestContext(request))
```

| | |
|---|---|
| Assignments: | 2 |
| Branches: | 4 |
| Conditionals | 1 |

Entonces, tendremos el vector V<2, 4, 1>. Por lo tanto, su métrica ABC es:

```
V<2,4,1> = 4.58
```

Es importante dejar el vector con los componentes calculados ya nos permiten identicar funciones más orientadas al almacenamiento (*assignments*), las que están más desacopladas, de alto nivel, modularizadas (*branches*) y las que tienen una mayor complejidad en la lógica (*conditionals*).

# Beneficios

- Nos permite tener una métrica que compara distintos códigos sin tomar en cuenta el estilo de programación del desarrollador.
- Nos da un análisis sencillo y rápido de porciones de código.
- Es basada en cero.

# Ejercicios

Debe completar el siguiente cuadro calculando la complejidad ciclomática y métrica ABC de las funciones del módulo que se le detalla.

| **Función** | **CC** | **ABC<a,b,c>** | **ABC** |
|---|---|---|---|
| loginUsuario | | | |
| guardar_matricula | | | |

| Función | CC | ABC<a,b,c> | ABC |
|---|---|---|---|
| listado_matricula | | | |
| listado_alumnos | | | |
| back_listar_matricula | | | |

*views.py*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from django.contrib.auth import authenticate, login, logout
from django.db import transaction
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import redirect
from django.template.loader import get_template
from django.template import RequestContext
from django.conf import settings
from django.shortcuts import render_to_response
from django.views.decorators.csrf import csrf_exempt
from django.contrib.auth.decorators import login_required
from django.contrib.auth.views import logout_then_login
import json
import logging
from django.core.exceptions import ObjectDoesNotExist
from models import Matricula, Alumno, Clase, Semestre, Taller, Nivel
from forms import MatriculaForm, FiltroAlumnosForm, AlumnoForm, LoginForm
from models import ABIERTO, CERRADO, ACTIVO, INACTIVO, MATRICULADO, NOMATRICULADO

@csrf_exempt
def loginUsuario(request):
    if request.method == 'GET':
        loginForm = LoginForm()
        return render_to_response('login.html', {'form':loginForm},
context_instance=RequestContext(request))
    else:
        loginForm = LoginForm(request.POST)
        if loginForm.is_valid():
            username = loginForm.cleaned_data['username']
            password = loginForm.cleaned_data['password']
            colegio = loginForm.cleaned_data['colegio']

            user = authenticate(username=username, password=password)
            if user is not None:
                # the password verified for the user
                if user.is_active:
                    print("User is valid, active and authenticated")
                    login(request, user)

                    request.session['colegio'] = colegio
```

```
                    listaGrupos = user.groups.all()
                    grupoStr = []
                    for grupo in listaGrupos:
                        grupoStr.append(grupo.name)


                    if "administrador" in grupoStr:
                        request.session['seguridad'] = "administrador"
                    elif "digitador" in grupoStr:
                        request.session['seguridad'] = "digitador"
                    elif "visualizador" in grupoStr:
                        request.session['seguridad'] = "visualizador"
                        return redirect('/matricula-seguimiento')

                    return redirect('/matricula-registrar')
                else:
                    print("The password is valid, but the account has been disabled!")
                    loginForm = LoginForm()
                    mensaje ='El password es valido pero el usuario ha sido
deshabilitado'
                    return render_to_response('login.html', {'form':loginForm,
'mensaje': mensaje}, context_instance=RequestContext(request))
            else:
                # the authentication system was unable to verify the username and
password
                print("The username and password were incorrect.")
                loginForm = LoginForm()
                mensaje ='El usuario y el password son incorrectos'
                return render_to_response('login.html', {'form':loginForm, 'mensaje':
mensaje}, context_instance=RequestContext(request))
        else:
            return render_to_response('login.html', {'form':loginForm},
context_instance=RequestContext(request))

@login_required
def logout_matricula(request):
    return logout_then_login(request, login_url='/login')

# Cambio de modificar matricula
@login_required
@csrf_exempt
def listado_matricula(request):
    #listado_alumnos(request):
    if request.method == 'GET':
        filtroAlumnosForm = FiltroAlumnosForm()
        return render_to_response('matricula-listar.html', {'form':filtroAlumnosForm},
context_instance=RequestContext(request))
    else:
        listadoAlumnosMatriculados = []
        filtroAlumnosForm = FiltroAlumnosForm(request.POST)
```

```
        if filtroAlumnosForm.is_valid():
            grado = filtroAlumnosForm.cleaned_data['grado']
            nombre = filtroAlumnosForm.cleaned_data['nombre']
            colegio = request.session['colegio']

            #listadoAlumnosMatriculados =
Matricula.objects.filter(alumno_nombreCompleto__icontains=nombre)

            if grado == None:
                listadoAlumnosMatriculados =
Matricula.objects.filter(alumno__nombreCompleto__icontains=nombre).filter(alumno__cole
gio__exact=colegio)
            else:
                listadoAlumnosMatriculados =
Matricula.objects.filter(alumno__grado__exact=grado).filter(alumno__nombreCompleto__ic
ontains=nombre).filter(alumno__colegio__exact=colegio)
        else:
            print "Formulario no es valido"

        return render_to_response('matricula-listar.html', {'form':filtroAlumnosForm,
'listado_alumnos_matriculados': listadoAlumnosMatriculados},
context_instance=RequestContext(request))

@login_required
@csrf_exempt
def listado_alumnos(request):
    if request.method == 'GET':
        filtroAlumnosForm = FiltroAlumnosForm()
        return render_to_response('matricula-listar-alumnos.html',
{'form':filtroAlumnosForm}, context_instance=RequestContext(request))
    else:
        listadoAlumnos = []
        filtroAlumnosForm = FiltroAlumnosForm(request.POST)

        if filtroAlumnosForm.is_valid():
            grado = filtroAlumnosForm.cleaned_data['grado']
            nombre = filtroAlumnosForm.cleaned_data['nombre']
            colegio = request.session['colegio']
            semestre = Semestre.objects.filter(estado__exact='A')[0]

            if grado == None:
                listadoAlumnos =
Alumno.objects.filter(nombreCompleto__icontains=nombre).filter(colegio__exact=colegio)
.filter(semestre__exact=semestre)
            else:
                listadoAlumnos =
Alumno.objects.filter(grado__exact=grado).filter(nombreCompleto__icontains=nombre).fil
ter(colegio__exact=colegio).filter(semestre__exact=semestre)


        return render_to_response('matricula-listar-alumnos.html',
```

```
{'form':filtroAlumnosForm, 'listado_alumnos': listadoAlumnos},
context_instance=RequestContext(request))

@login_required
@csrf_exempt
def registro_alumno(request):
    if request.method == 'GET':
        alumnoForm = AlumnoForm()
        return render_to_response('matricula-registrar-alumno.html',
{'form':alumnoForm}, context_instance=RequestContext(request))
    else:
        alumnoForm = AlumnoForm(request.POST)
        if alumnoForm.is_valid():
            nombreCompleto = alumnoForm.cleaned_data['nombreCompleto']
            grado = alumnoForm.cleaned_data['grado']
            seccion = alumnoForm.cleaned_data['seccion']
            #nivel = alumnoForm.cleaned_data['nivel']
            colegio = request.session['colegio']
            telefono = alumnoForm.cleaned_data['telefono']
            tutor = alumnoForm.cleaned_data['tutor']
            semestre = alumnoForm.cleaned_data['semestre']
            fechaNacimiento = alumnoForm.cleaned_data['fechaNacimiento']

            alumnoBean = Alumno()
            alumnoBean.nombreCompleto = nombreCompleto
            alumnoBean.grado = grado
            alumnoBean.seccion = seccion
            alumnoBean.nivel = grado.nivel
            alumnoBean.colegio = colegio
            alumnoBean.telefono = telefono
            alumnoBean.tutor = tutor
            alumnoBean.semestre = semestre
            alumnoBean.fechaNacimiento = fechaNacimiento
            alumnoBean.estado = NOMATRICULADO

            alumnoBean.save()

            return render_to_response('matricula-registrar-alumno.html',
{'alumno':alumnoBean}, context_instance=RequestContext(request))
        else:
            return render_to_response('matricula-registrar-alumno.html', {'form':
alumnoForm}, context_instance=RequestContext(request))

@login_required
def registro_matricula(request):
    seguridad = request.session['seguridad']
    colegio = request.session['colegio']
    semestre = Semestre.objects.filter(estado__exact='A')[0]
    matriculaForm = MatriculaForm(colegio, semestre)

    return render_to_response('matricula-registrar.html', {'form':matriculaForm,
```

```
'seguridad':seguridad}, context_instance=RequestContext(request))

@login_required
def modificacion_matricula(request, idmatricula):
    if request.method == 'GET':
        matricula = Matricula.objects.get(id=idmatricula)

        matriculaForm = MatriculaForm(instance=matricula)

    return render_to_response('matricula-registrar.html', {'form':matriculaForm},
context_instance=RequestContext(request))

# Metodos del backend

@csrf_exempt
def guardar_matricula(request):
    if request.method == 'POST':
        colegio = request.session['colegio']
        semestre = Semestre.objects.filter(estado__exact='A')[0]
        matriculaForm = MatriculaForm(colegio,semestre,request.POST)
        if matriculaForm.is_valid():

            semestre = matriculaForm.cleaned_data['semestre']
            alumno = matriculaForm.cleaned_data['alumno']
            taller = matriculaForm.cleaned_data['taller']
            idAlumno = matriculaForm.cleaned_data['idAlumno']
            tutor = matriculaForm.cleaned_data['tutor']
            telefono = matriculaForm.cleaned_data['telefono']
            fechaNacimiento = matriculaForm.cleaned_data['fechaNacimiento']

            # Comprobamos que no se hayan cubierto las vacantes para taller ni para
mate
            if taller.numeroVacantes <= taller.numeroInscritos:
                mensaje = "No hay cupos de taller disponibles"
                return render_to_response('matricula-registrar.html',
{'form':matriculaForm, 'error':mensaje, 'seguridad':request.session['seguridad']},
context_instance=RequestContext(request))

            alumnoBean = Alumno.objects.get(id=idAlumno)
            clase =
Clase.objects.filter(grado__exact=alumnoBean.grado).filter(colegio__exact=colegio).fil
ter(semestre__exact=semestre)[0]

            if clase.numeroVacantes <= clase.numeroInscritos:
                mensaje = "No hay cupos de clase de matematicas disponibles"
                return render_to_response('matricula-registrar.html',
{'form':matriculaForm, 'error':mensaje, 'seguridad':request.session['seguridad']},
context_instance=RequestContext(request))


            alumnoBean.tutor = tutor
```

```
            alumnoBean.telefono = telefono
            alumnoBean.fechaNacimiento = fechaNacimiento
            alumnoBean.estado = MATRICULADO
            alumnoBean.save()
            matricula = Matricula()
            matricula.semestre = semestre
            matricula.alumno = alumnoBean
            matricula.taller = taller

            matricula.save()

            #Debemos disminuir los disponibles para taller y para clase
            numInscritos = taller.numeroInscritos
            taller.numeroInscritos = numInscritos + 1
            print taller.numeroInscritos
            taller.save()

            numInscritos = clase.numeroInscritos
            clase.numeroInscritos = numInscritos + 1
            clase.save()


            matriculaForm = MatriculaForm(colegio, semestre)

            mensaje = "Matricula guardada exitosamente"
        else:
            mensaje = "Error procesando formulario, llene todos los campos"
            return render_to_response('matricula-registrar.html',
{'form':matriculaForm, 'error':mensaje, 'seguridad':request.session['seguridad']},
context_instance=RequestContext(request))

    return render_to_response('matricula-registrar.html', {'form':matriculaForm,
'mensaje':mensaje, 'seguridad':request.session['seguridad']},
context_instance=RequestContext(request))

@csrf_exempt
def modificacion_matricula(request, idmatricula):
    colegio = request.session['colegio']
    semestre = Semestre.objects.filter(estado__exact='A')[0]
    matricula = Matricula.objects.get(pk=idmatricula)
    if request.method == 'GET':

        print matricula.semestre.nombre
        matriculaForm = MatriculaForm(colegio, semestre)
        #matriculaForm.taller.value = matricula.taller
        return render_to_response('matricula-modificar.html',
{'matricula':matricula,'form':matriculaForm},
context_instance=RequestContext(request))
    else:
        #matriculaForm = MatriculaForm(colegio, semestre, request.POST)
```

```
        taller_id = request.POST['taller']
        taller_nuevo = Taller.objects.get(pk=taller_id)

        matricula = Matricula.objects.get(pk=idmatricula)
        taller_antiguo = matricula.taller

        #Disminuimos cuota a taller anterior

        if taller_nuevo.numeroInscritos <= taller_nuevo.numeroVacantes-1:
            print "Puede matricular porque " + str(taller_nuevo.numeroInscritos) +
"<=" + str(taller_nuevo.numeroVacantes)
            #Podemos realizar la matricula
            #Aumentamos cuota a taller nuevo
            taller_nuevo.numeroInscritos = taller_nuevo.numeroInscritos + 1
            taller_antiguo.numeroInscritos = taller_antiguo.numeroInscritos - 1

            matricula.taller = taller_nuevo
            matricula.save()

            taller_nuevo.save()
            taller_antiguo.save()
            return HttpResponseRedirect("/matricula-modificar-listar")
        else:
            #No puede realizar matricula porque taller esta lleno
            return render_to_response('matricula-modificar.html',
{'matricula':matricula,'form':matriculaForm},
context_instance=RequestContext(request))




@csrf_exempt
def eliminar_matricula(request, idmatricula):
    if request.method == 'GET':
        colegio = request.session['colegio']
        matricula = Matricula.objects.get(id=idmatricula)
        alumno = matricula.alumno
        semestre = Semestre.objects.filter(estado__exact='A')[0]

        # Se disminuiran los numero de inscritos del taller y de la clase a la que
pertenecian
        taller = matricula.taller
        clase =
Clase.objects.filter(grado__exact=alumno.grado).filter(colegio__exact=colegio).filter(
semestre__exact=semestre)[0]
        taller.numeroInscritos = taller.numeroInscritos - 1
        clase.numeroInscritos = clase.numeroInscritos - 1

        # Cambiamos el estado del alumno a NOMATRICULADO
```

```
        alumno.estado = NOMATRICULADO

        matricula.delete()
        taller.save()
        clase.save()
        alumno.save()

    return HttpResponseRedirect("/matricula-modificar-listar")

@csrf_exempt
def back_listar_matricula(request): #ok
    if request.method == 'POST':
        #inputStr = request.body
        #jsInput = json.loads(inputStr)
        lista = []
        try:
            #id_evento = jsInput['id']
            lista_matriculas = Matricula.objects.all()
            for matricula in lista_matriculas:
                nombre = matricula.alumno.nombreCompleto
                dict_matricula = {'idmatricula':matricula.id, 'alumno':nombre,
'grado': matricula.alumno.grado.nombre, 'seccion': matricula.alumno.seccion.nombre,
'taller': matricula.taller.nombre, 'email': matricula.alumno.email}
                lista.append(dict_matricula)


        except ObjectDoesNotExist:
            return HttpResponse(json.dumps({'mensaje' : 'No hay registros'}),
content_type="application/json")

        dict_output = {'mensaje' : '', 'lista_matriculas': lista}

        return HttpResponse(json.dumps(dict_output), content_type="application/json")

@login_required
def reporte_seguimiento(request):

    seguridad = request.session['seguridad']

    colegio = request.session['colegio']

    semestre = Semestre.objects.filter(estado__exact=ACTIVO)[0]

    niveles = Nivel.objects.all()

    talleres_total = {}
    for nivel in niveles:
        talleres =
Taller.objects.filter(nivel__exact=nivel).filter(semestre__exact=semestre).filter(cole
gio__exact=colegio)
        talleres_total[str(nivel.nombre)] = talleres
```

```
    clases_total = {}
    for nivel in niveles:
        clases =
Clase.objects.filter(nivel__exact=nivel).filter(semestre__exact=semestre).filter(coleg
io__exact=colegio)
        clases_total[str(nivel.nombre)] = clases

    return render_to_response('matricula-seguimiento.html',
{'talleres':talleres_total, 'clases':clases_total, 'seguridad':seguridad},
context_instance=RequestContext(request))

def reporte_matricula(request):

    seguridad = request.session['seguridad']

    #colegio = request.session['colegio']

    semestre = Semestre.objects.filter(estado__exact=ACTIVO)[0]

    lista_matricula = Matricula.objects.all();

    return render_to_response('matricula-reporte.html',
{'lista_matricula':lista_matricula, 'seguridad':seguridad},
context_instance=RequestContext(request))

def migrar_alumnos_semestre(request):
    from matricula.models import Semestre, Alumno, NOMATRICULADO
    semestre = Semestre.objects.filter(estado__exact='A')[0]
    nombre_semestre_anterior = "2013-I"

    semestre = Semestre.objects.filter(estado__exact='A')[0]
    semestre_ant = Semestre.objects.filter(nombre__exact=nombre_semestre_anterior)[0]

    listado_alumnos = Alumno.objects.filter(semestre__exact=semestre_ant)
    for alumno in listado_alumnos:
        alumno.pk = None
        alumno.semestre = semestre
        alumno.estado = NOMATRICULADO
        alumno.save()
```

# Referencias


- T. J. McCabe, "A Complexity Measure," in IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, Dec. 1976.
- Umesh Tiwari and Santosh Kumar. 2014. Cyclomatic complexity metric for component based software. SIGSOFT Softw. Eng. Notes 39, 1 (February 2014), 1-6.
- Jerry Fitzpatrick. 2000. Applying the ABC metric to C, C, and Java. In More C gems, Robert C.

Martin (Ed.). Cambridge University Press, New York, NY, USA 245-264.