PYTHON HASTA EN LOS RINCONES

o como una serpiente se puede comer un león

Uso de Python en el proyecto Meer para la generación automática de aplicaciones para terminales móviles inteligentes a partir de informes DICOM-SR

i3M
Instituto de Instrumentación para Imagen Molecular

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Grid y Computación de Altas Prestaciones
GRyCAP

BON DIA

Grid y Computación de Altas Prestaciones

GRyCAP

- Creado en 1986 por el Profesor Vicente Hernández García.
- Desarrollo y aplicación de las tecnologías Grid y la Computación de Altas Prestaciones en ámbitos como la ingeniería, la biomedicina, el e-Gobierno o la computación científica.
- 28 miembros y participamos en un número significativo de proyectos

Mayte Giménez

- Ingeniera Informática
- Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital (*)
- Grado en Bellas Artes (*)

Jon Nieve

- No sabe nada.

(*) Cursando

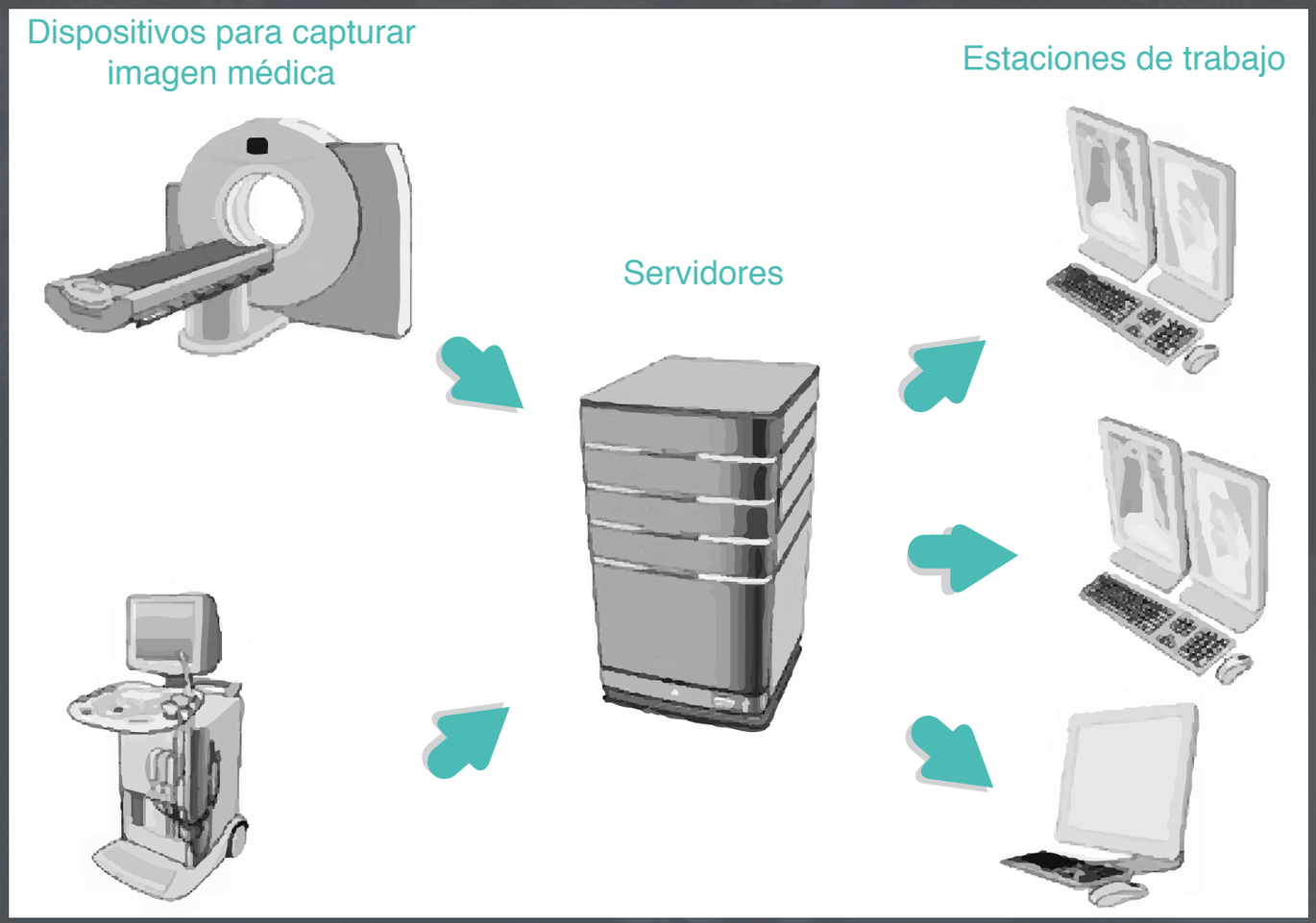
ÍNDICE

UNA SERPIENTE SE PUEDE COMER UN LEÓN

En trozos pequeños y con los dientes afilados

EL LEÓN EN SU SABANA

Definición del escenario del problema.



Componentes principales de los PACs

- Imágen médica digital revoluciona la práctica clínica.
- Sistemas de Adquisición y Procesado de Imagen Médica (PACs).
- Uso de estándar DICOM.

- Estándar para informes médicos estructurados DICOM-SR.
  - Información no ambigua y jerárquica.
  - Cuello de botella en la adqusición.

¿NO PUEDE LA SERPIENTE SER VEGETARIANA?

Ó porque nos decidimos a abordar este problema

- Trabajos previos del grupo de investigación, publicados en:

  Maestre C, Segrelles-Quilis JD, Torres E, Blanquer I, Medina R, Hernández V, Martí L. Assessing the Usability of a Science Gateway for Medical Knowledge Bases with TRENCADIS. J Grid Computing 2012; 10:665–688.

- Utilización de interfaz web para introducir informes médicos estructurados:
  - Más fácil e intuitivo.
  - Sencillo de aprender.
  - Pero menos productivo.

- Para mejorar estos resultados se solicitó el siguiente proyecto financiado por la UPV dentro del programa PAID:

  Diseño de Componentes Cloud Facilitadores del Despliegue y la Alta Disponibilidad de Servicios TRENCADIS, para compartir Imágenes Médicas DICOM e informes Asociados DICOM-SR

OBJETIVOS

Meer

- Desarrollar una aplicación que automá-ticamente genere una aplicación para Android que permitan la introducción de informes DICOM-SR.
- Interacción con el usuario simple.
- Aumentar la productividad al introducir informes médicos estructurados, apro-vechando las nuevas tecnologías.

Charla

- Analizadores sintácticos.
- Generación automática de código.
- Entornos virtuales.
- Configuración.
- Uso de plantillas.
- Código bello para aplicaciones bellas.
- Cómo comerse un león.

LA SERPIENTE TRAS EL ATRACÓN

Definición de la solución propuesta

Aplicación Android para gestión de informes médicos

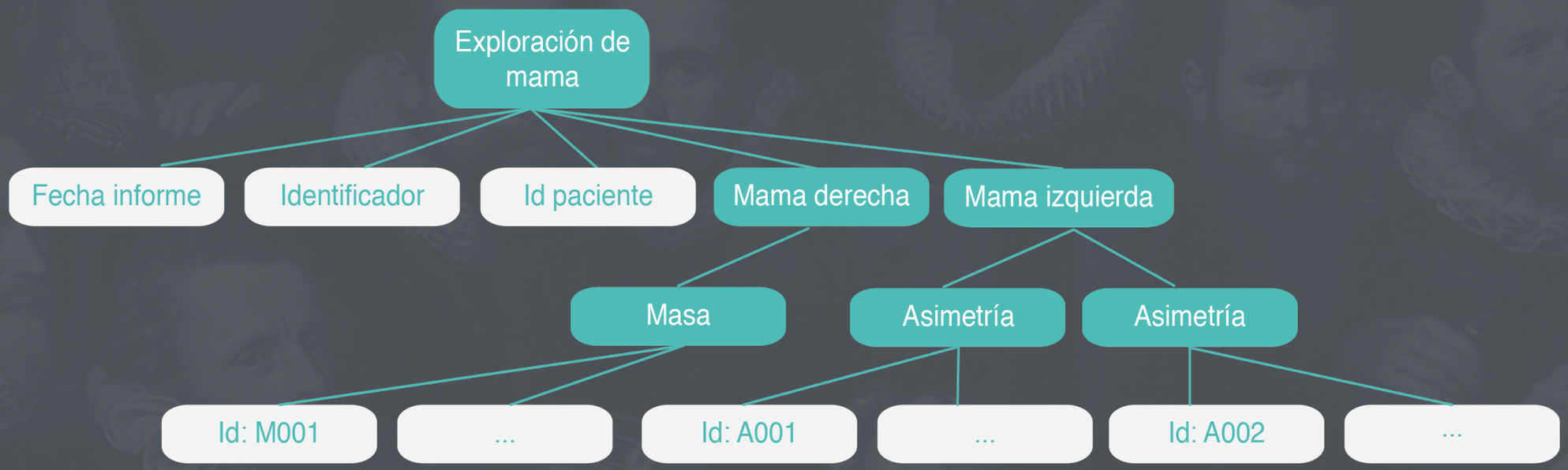
- Aplicación Android para rellenar informes médicos estructurados.
- Aplicación generada automáticamente.

- Ventajas de esta solución:
  - Interfaz de usuario intuitiva.
  - Fácil adopción.
  - Sigue el estándar DICOM-SR.
  - Solución genérica
  - Se adapta a distintos tipos de informes automáticamente.

DISECCIÓN DE LA SOLUCIÓN

ÍNDICE

DICOM-SR



- Múltiples beneficios derivados del uso de DICOM-SR:
  - Mejora la comunicación entre los profesionales
  - Los informes son más precisos y concisos.
  - Sistemas asistidos por computador para tomas de decisiones.
  - ...
- La adquisición de informes DICOM-SR de modo tradicional es un cuello de botella.

XML

```
...
<NUM>
   <CONCEPT_NAME>
      <CODE_VALUE>RID29929</CODE_VALUE>
      <CODE_SCHEMA>RADLEX</CODE_SCHEMA>
      <CODE_MEANING>Cuadrante Externo Superior de la Mama Derecha</CODE_MEANING>
      <CODE_MEANING2>Upper Outer Quadrant of Right Female Breast</CODE_MEANING>
   </CONCEPT_NAME>
   <PROPERTIES>
      <CARDINALITY max="1" min="1"/>
      <CONDITION_TYPE type="M"/>
      <EXPRESION_CONDITION xquery=""/>
      <DEFAULT_VALUE value="0"/>
      <UNIT_MEASUREMENT>
         <CONCEPT_NAME>
            <CODE_VALUE>000000001</CODE_VALUE>
            <CODE_SCHEMA>UNIT_MEASUREMENT</CODE_SCHEMA>
            <CODE_MEANING>Unidades Boleanas</CODE_MEANING>
            <CODE_MEANING2>Boolean Units</CODE_MEANING>
         </CONCEPT_NAME>
      </UNIT_MEASUREMENT>
   </PROPERTIES>
</NUM>
...
```

Fragmento de un fichero xml que contiene una plantilla basada en DICOM-SR

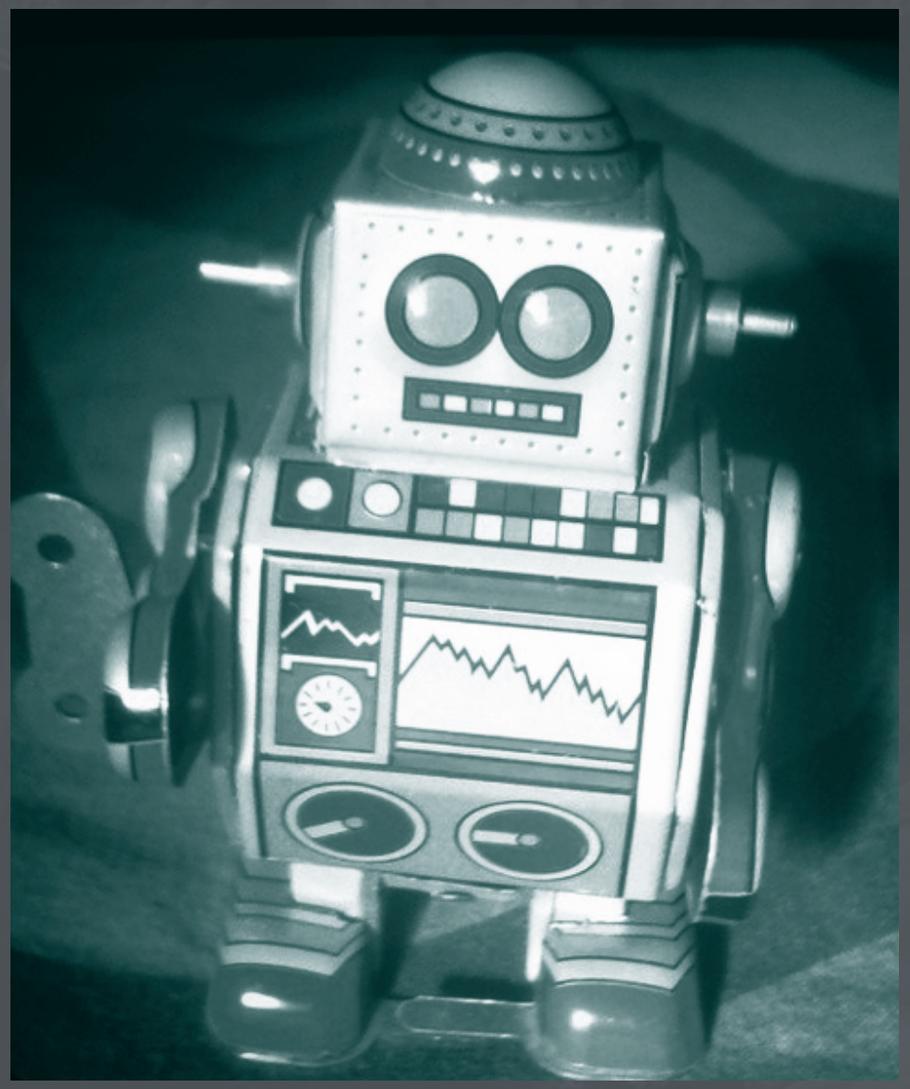
ANDROID

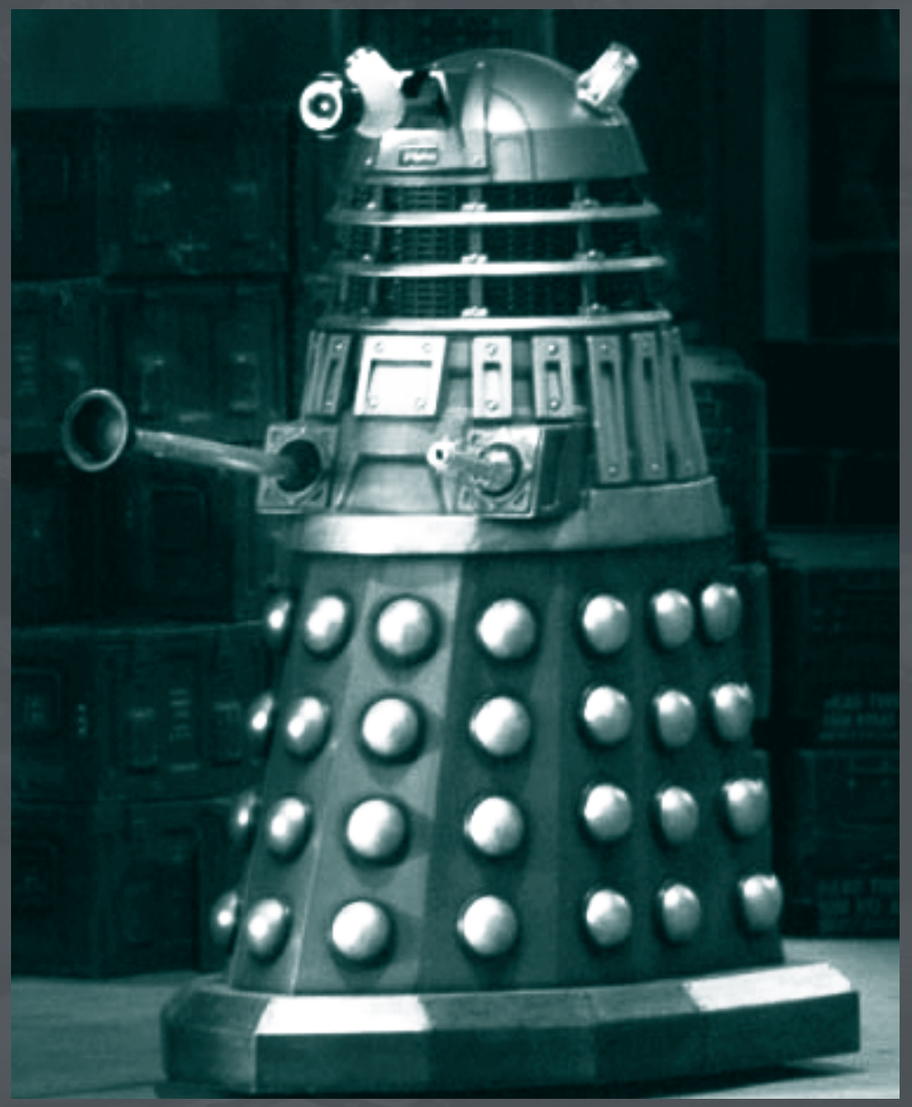
ÍNDICE

" LOS ORDENADORES SON BUENOS SIGUIENDO INSTRUCCIONES, PERO NO LEYENDOTE LA MENTE "

Donal Knuth

GENERACIÓN AUTOMÁTICA DE CÓDIGO





PYTHON GENERA CÓDIGO

Lenguaje	Ventajas	Inconvenientes
C/ C++	Una vez compilado el usuario no podrá leer el código fuente (?) Es un lenguaje rápido. (?)	Está orientado a la gestión de texto plano. La E/S no es portable. XML y las expresiones regulares son difíciles de utilizar. Fuerte tipado.
Java	Una vez compilado el usuario no podrá leer el código fuente (?)	No aconsejado para el análisis sintáctico. Fuerte tipado.
Python Ruby Perl	El código es portable. Los generadores son fácilmente escalables. El análisis de texto es simple. Existen APIs de XML son simples	Otros ingenieros pueden necesitar aprender el lenguaje. (?)

Code generation in action (Jack Herrington)

PYTHON

BELLO ES MEJOR QUE FEO.

Python zen.

PEP-8

PEP-256

APLICACIONES BELLAS PARA CÓDIGO BELLO

Amarás el estilo internacional.

Roboto Regular

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz

#50BAB5 #A1D1D4

#686868 #A7A6A7

#EA6D50 #DCBFB2

Colores primarios

Colores secundarios

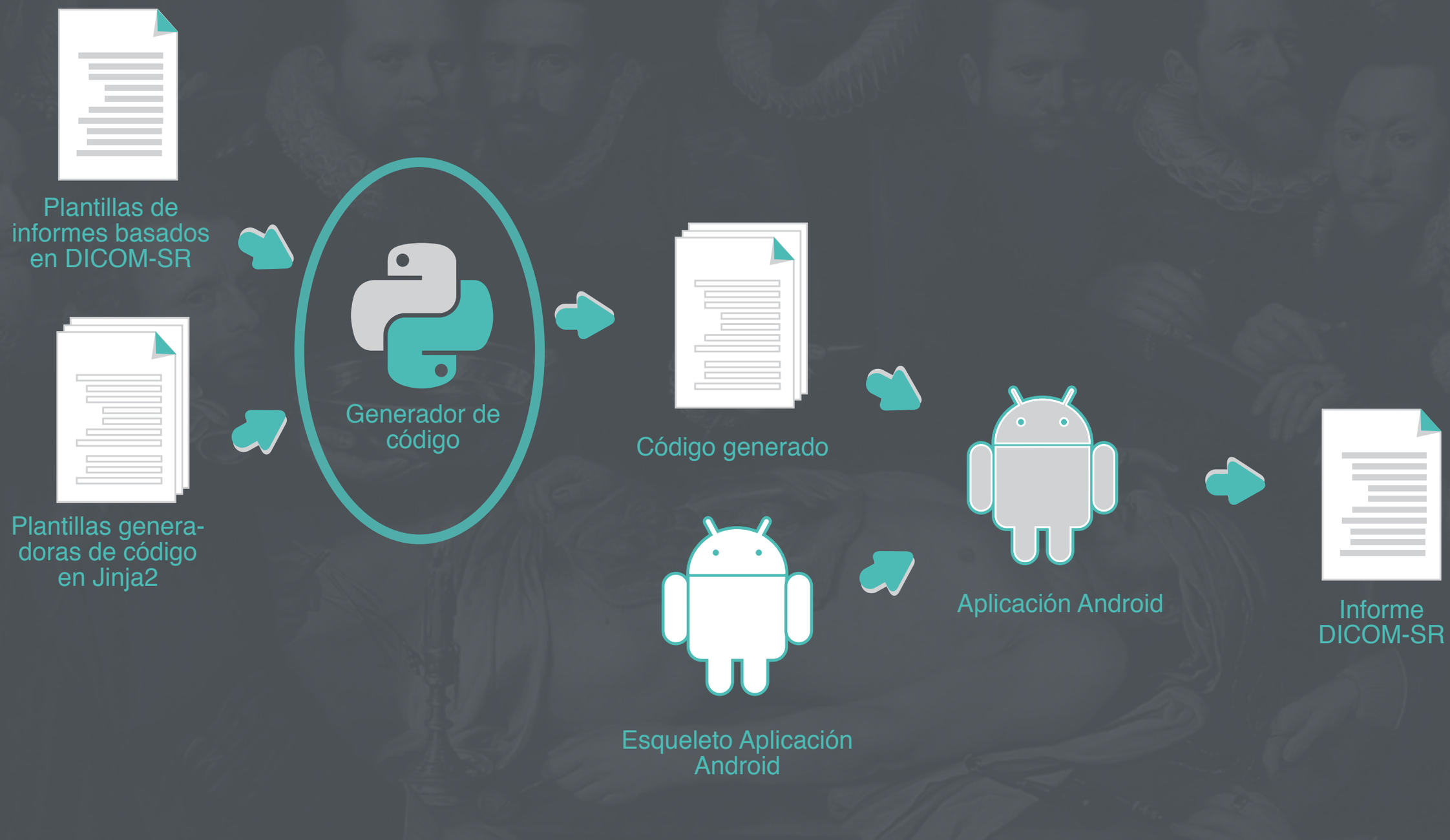
Destacado
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz

Destacado
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz

Texto común
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz

ÍNDICE

DISECCIÓN DE LA SOLUCIÓN

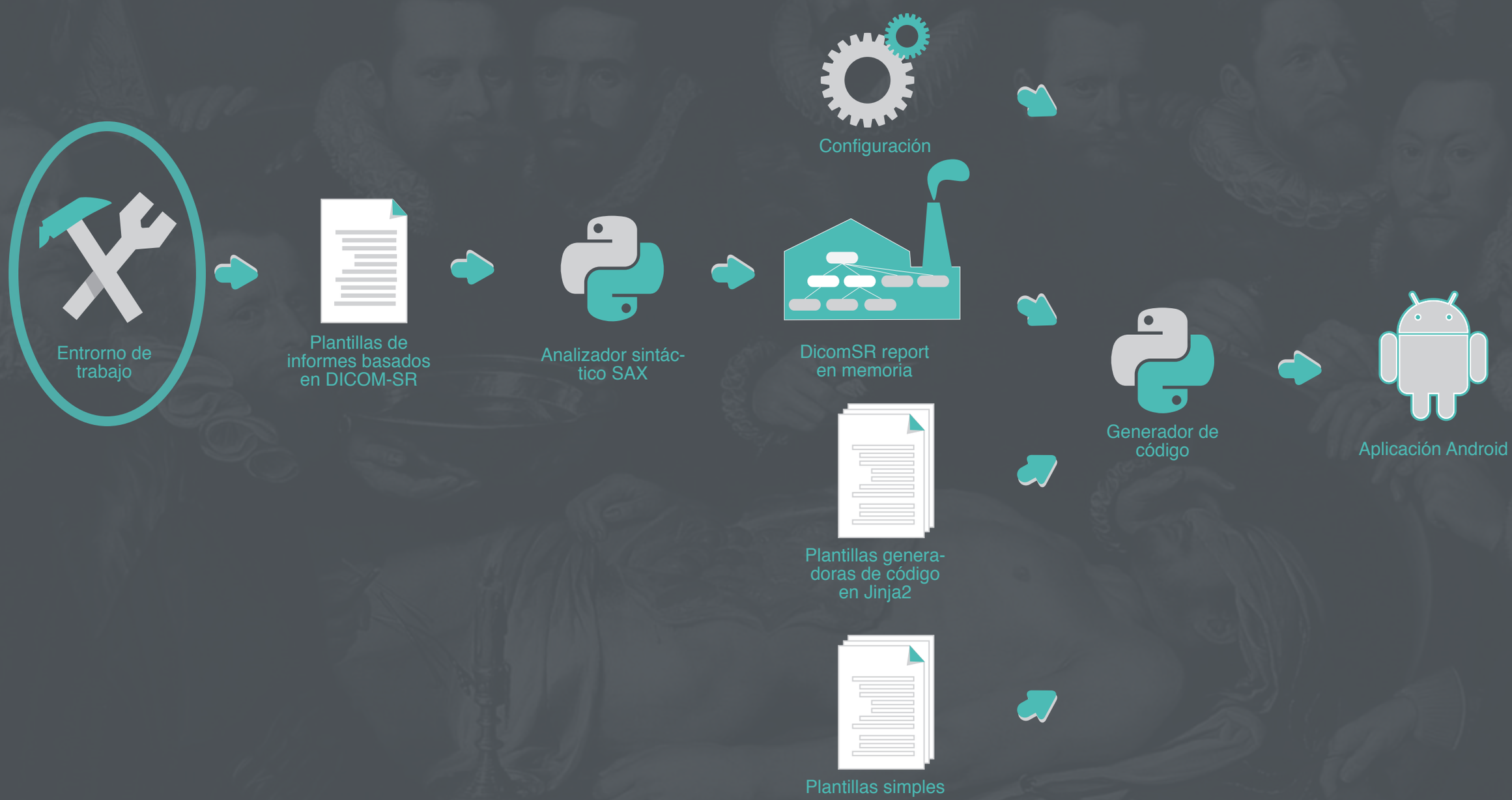


CÓMO SE UNEN TODAS LAS PIEZAS DEL PUZZLE

Arquitectura de una aplicación Android

- Generación automática de código guiada por modelos.
- Sistemas de plantillas:
  - strigs.Template
  - Jinja2
- Configurable en función de las necesidades del usuario.
- Prototipo evolucionado desde la fase de diseño para ser el esqueleto de la aplicación Android.
- Generación del código que construirá:
  - Vista (xml)
  - Modelo (java)
  - Controlador (java)
- Código generado se integra en la aplicación Android esqueleto.

DISECCIÓN DEL GENERADOR DE CÓDIGO



ENTORNOS VIRTUALES

pip +virtualenv + virtualevwrapper

requirements.txt

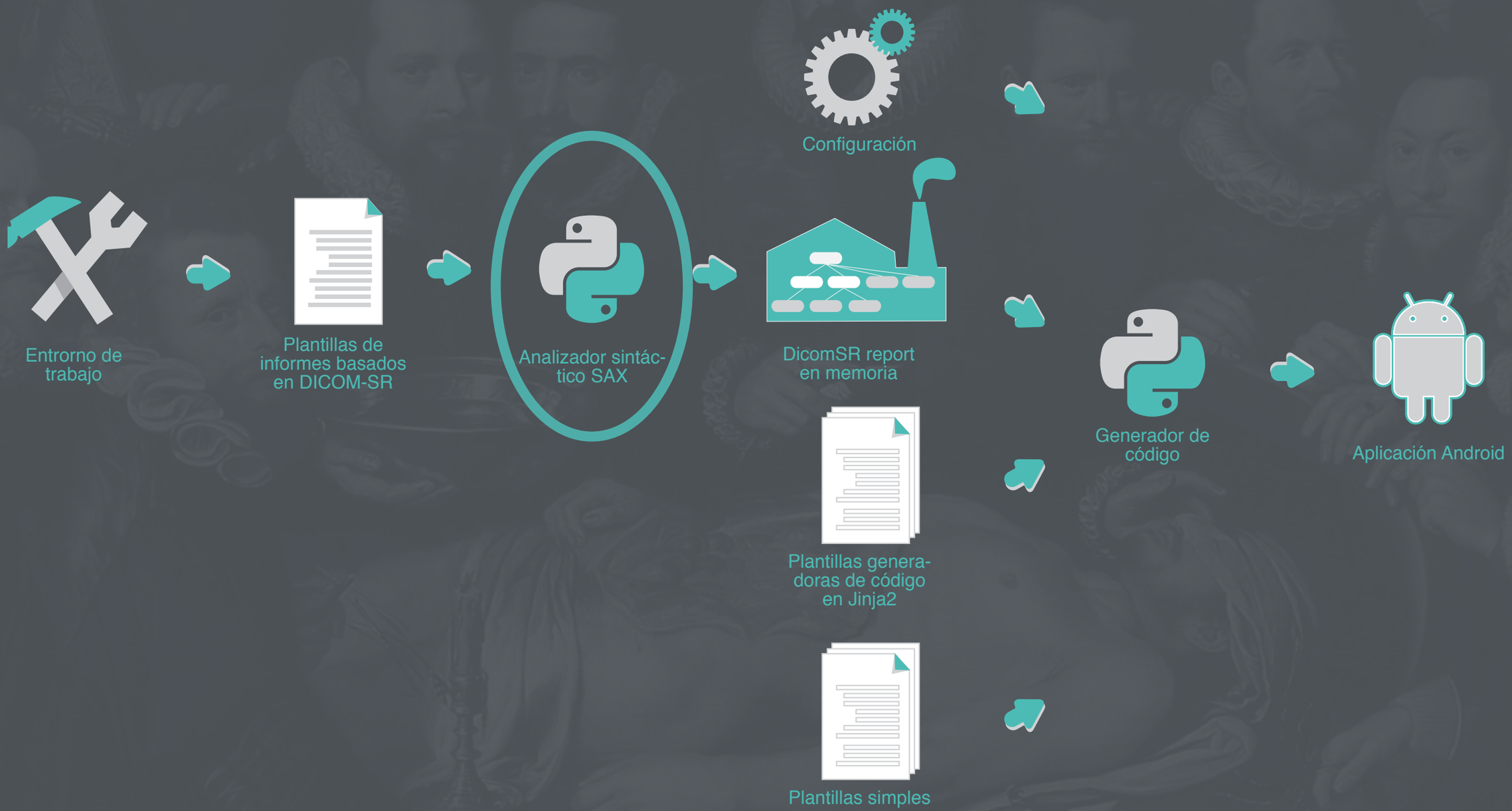
```
Jinja2==2.7
MarkupSafe==0.18
wsgiref==0.1.2
```

- Gestión de dependencias aséptica (de verdad).
- Ficheros bajo el sistema de control de versiones.
- Tantos ficheros de requisitos como entornos de trabajo:
  - Desarrollo
  - Producción
  - ...

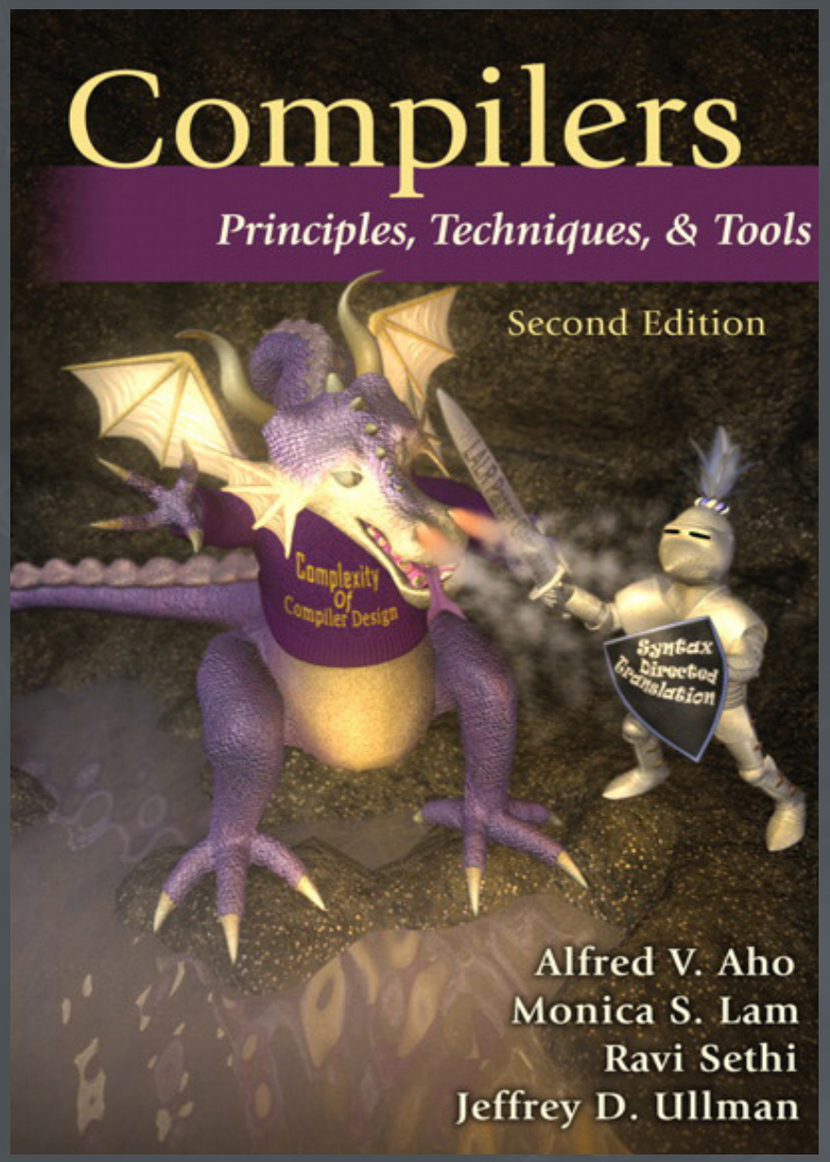
```
$ pip install requirements.txt
```

Migrar es complejo

DISECCIÓN DEL GENERADOR DE CÓDIGO



ANALIZADORES SINTÁCTICOS



EL libro del dragón

```c
void
xmlSAX2ExternalSubset(void *ctx, const xmlChar *name,
                      const xmlChar *ExternalID, const xmlChar *SystemID)
{
    xmlParserCtxtPtr ctxt = (xmlParserCtxtPtr) ctx;
    if (ctx == NULL) return;
#ifdef DEBUG_SAX
    xmlGenericError(xmlGenericErrorContext,
        "SAX.xmlSAX2ExternalSubset(%s, %s, %s)\n",
            name, ExternalID, SystemID);
#endif
    if (((ExternalID != NULL) || (SystemID != NULL)) &&
        (((ctxt->validate) || (ctxt->loadsubset != 0)) &&
     (ctxt->wellFormed && ctxt->myDoc))) {
    /*
     * Try to fetch and parse the external subset.
     */
    xmlParserInputPtr oldinput;
    int oldinputNr;
    int oldinputMax;
    xmlParserInputPtr *oldinputTab;
    xmlParserInputPtr input = NULL;
    xmlCharEncoding enc;
    int oldcharset;
    const xmlChar *oldencoding;

    /*
     * Ask the Entity resolver to load the damn thing
     */
    if ((ctxt->sax != NULL) && (ctxt->sax->resolveEntity != NULL))
        input = ctxt->sax->resolveEntity(ctxt->userData, ExternalID,
                                         SystemID);
    if (input == NULL) {
        return;
    }
```

libxml2: analizador sintáctico en C++

ANALIZADORES SINTÁCTICOS EN PYTHON

```
54      def startElement(self, name, attrs):
55          """ Handles start of a tag """
56          #XML root: stores the odontology
57          if (name == "DICOM_SR"):
58              try:
59                  self._report.report_type = attrs['Description']
60                  self._report.id_odontology = attrs['IDOntology']
61              except KeyError:
62                  #report_type it's an old specification
63                  self._report.report_type = attrs['reportType']
64          #Begin of a container tag
65          if (name == "CONTAINER"):
66              # We are in a new (deeper) tree level
67              self._tree_level += 1
68              if (self._tree_level > self._deepest_level):
69                  self._deepest_level = self._tree_level
70              self._in_level = True
71              logging.info('* Tree level {0}'.format(self._tree_level))
72          #Begin of child tag
73          if (name == "CHILDS"):
74              # We are in a new (deeper) child level
75              self._child_level += 1
76              self._in_level = False
77              logging.info('* Child level {0}'.format(self._child_level))
78          if (name == "CONCEPT_NAME"):
79              self._in_concept = True
80              self._repeated = False
81              #Unit measurement tag also has a concept name
82              #It explains the unit measurement type (boolean units basically)
83              if (self._in_unit_measurement):
84                  self._unit_measurement = Concept(-1, "", {})
85              else:
86                  self._concept = Concept(-1, "", {})
87          if (name == "DATE"):
88              self._in_type = True
89              self._current_attribute = Date()
```

Analizador sintáctico SAX en python.

xml.sax

- Acceso secuencial basada en eventos.
- Sax API que define 4 tipos de manejadores:
  - Clase `xml.sax.handler.ContentHandler`: análisis sintáctico de los eventos principales.
  - Clase `xml.sax.handler.ErrorHandler`: gestiona los errores encontrados en tiempo de ejecución.
  - Clase `xml.sax.handler.EntityResolver`: resuelve los conflictos entre las entidades.
  - Clase `xml.sax.handler.DTDHandler`: análisis sintáctico de los eventos relacionados con la definición del tipo de documento.
  - ...

CONTENT HANDLER

```
import xml.sax

class DicomParser(xml.sax.handler.ContentHandler):
    startDocument(self):
    endDocument(self):
    startElement(self, name, attrs):
    endElement(self, name):
    startElementNS(self, name, qname, attrs):
    endElementNS(self, name, qname):
    characters(self, content):
    processingInstruction(self, target, data):
    ignorableWhitespace(self, whitespace):
    skippedEntity(self, name):
    startPrefixMapping(self, prefix, uri):
    endPrefixMapping(self, prefix):
    setDocumentLocator(self, locator):
```

I SHOULD BE SLEEPING LIKE A LOG

Registros en Python

```
        if (name == "DICOM_SR"):
            try:
                self._report.report_type = attrs['Description']
                self._report.id_odontology = attrs['IDOntology']
            except KeyError:
                #report_type it's an old specification
                self._report.report_type = attrs['reportType']
        #Begin of a container tag
        if (name == "CONTAINER"):
            # We are in a new (deeper) tree level
            self._tree_level += 1
            if (self._tree_level > self._deepest_level):
                self._deepest_level = self._tree_level
            self._in_level = True
            logging.info('* Tree level {0}'.format(self._tree_level))
        #Begin of child tag
        if (name == "CHILDS"):
            # We are in a new (deeper) child level
            self._child_level += 1
            self._in_level = False
            logging.info('* Child level {0}'.format(self._child_level))
        if (name == "CONCEPT_NAME"):
            self._in_concept = True
            self._repeated = False
            #Unit measurement tag also has a concept name
            #It explains the unit measurement type (boolean units basically
            if (self._in_unit_measurement):
                self._unit_measurement = Concept(-1, "", {})
            else:
                self._concept = Concept(-1, "", {})
        if (name == "DATE"):
            self._in_type = True
            self._current_attribute = Date()
```

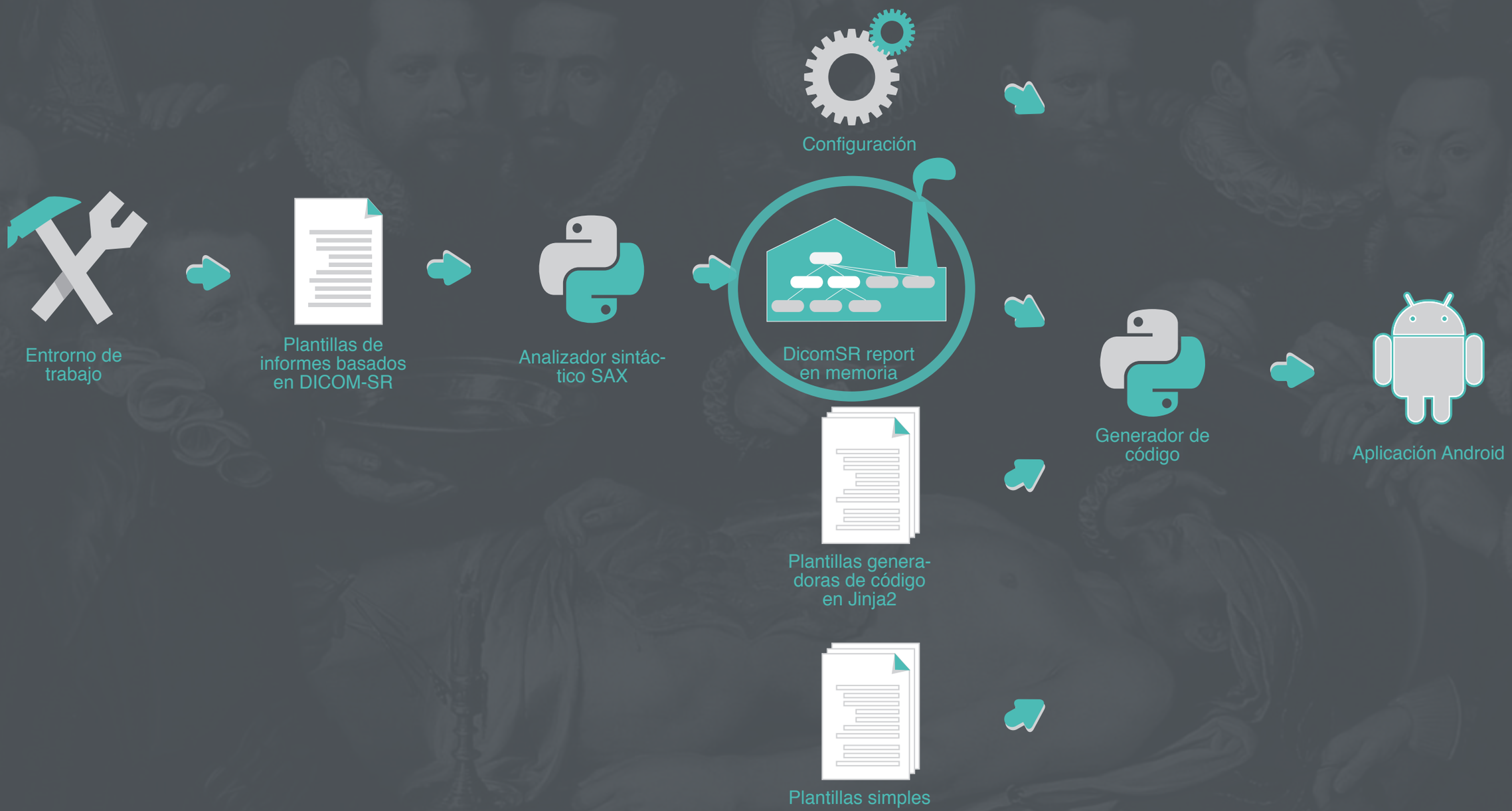
Analizador sintáctico SAX en python.

- No sys.stderr.write/print !
- Configurable:
  - Salida del log.
  - Formato.
- Distints niveles de log:
  - Debug.
  - Info.
  - Warning.
  - Error.
  - Critical.
- Sobrecarga de los manejadores.
- Filtros.

LOGGING

```
import loging

logging.basicConfig([**Kwargs])
                    {filename, filemode, format, datefmt, level, stream}
logging.propagate
                  {= True | False}

logging.debug(msg, *args, **kwargs)
logging.info(msg, *args, **kwargs)
logging.warning(msg, *args, **kwargs)
logging.error(msg, *args, **kwargs)
logging.critical(msg, *args, **kwargs)
logging.log(level, msg, *args, **Kwargs)

logging.addFilter(filt)
logging.filter(record)
```

DISECCIÓN DEL GENERADOR DE CÓDIGO



UNA EDA PARA DOMINARLOS A TODOS

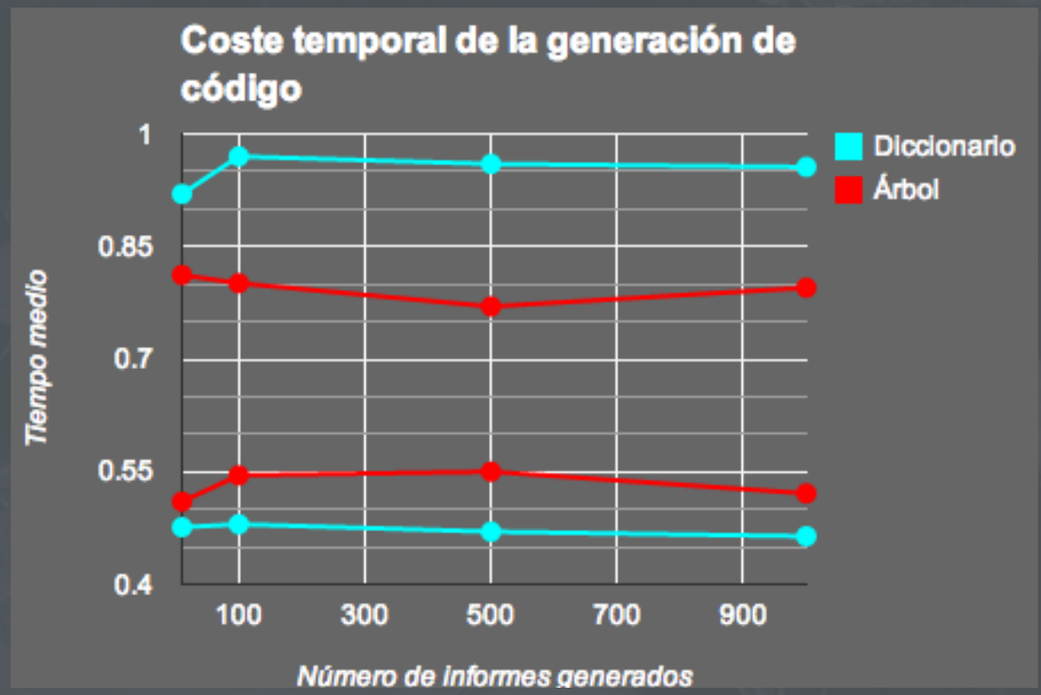
Los diccionarios.

```
# Un árbol en una línea y además rápido!
def tree(): return defaultdict(tree)
a = {40 : {30:{}, 50:{}}}
```

```
# Un árbol en una línea
r =report{(1, ‘Exploration of Breast’, [‘Date’, ‘Report ID’, ...]) : {
              (2, ‘Right Breast’, [ ]) : {} ,
              (2, ‘Left Breast’, []: {}}}
# Acceso!
for k,v in r.iteritems():
    k[2].append(new_attribute)
```

COMPLEJIDAD TEMPORAL VS VIDA REAL

Operación	Coste medio	Peor caso
Copia	O(n)	O(n)
Acceso	O(1)	O(n)
Asignación	O(1)	O(n)
Eliminación	O(1)	O(n)
Iteración	O(n)	O(n)



SIEMPRE HAN HABIDO CLASES

Definición de las EDAs.

```
# Un árbol en la manga
class Tree():
    “””Tree Object, contains value and child references”””
    def __init__(self, value=None, children=None):
        self.value = value
        if children is None:
            self.children = []
        else:
            self.children = children
```

```
class DicomSR():
    “””DICOM SR report”””
    def __init__(self, report_type=“”, id_ontology=-1):
        self.report_type = report_type
        self.id_ontology = id_ontology
        self.report = Tree()
```

NI LASAÑA NI ESPAGUETIS

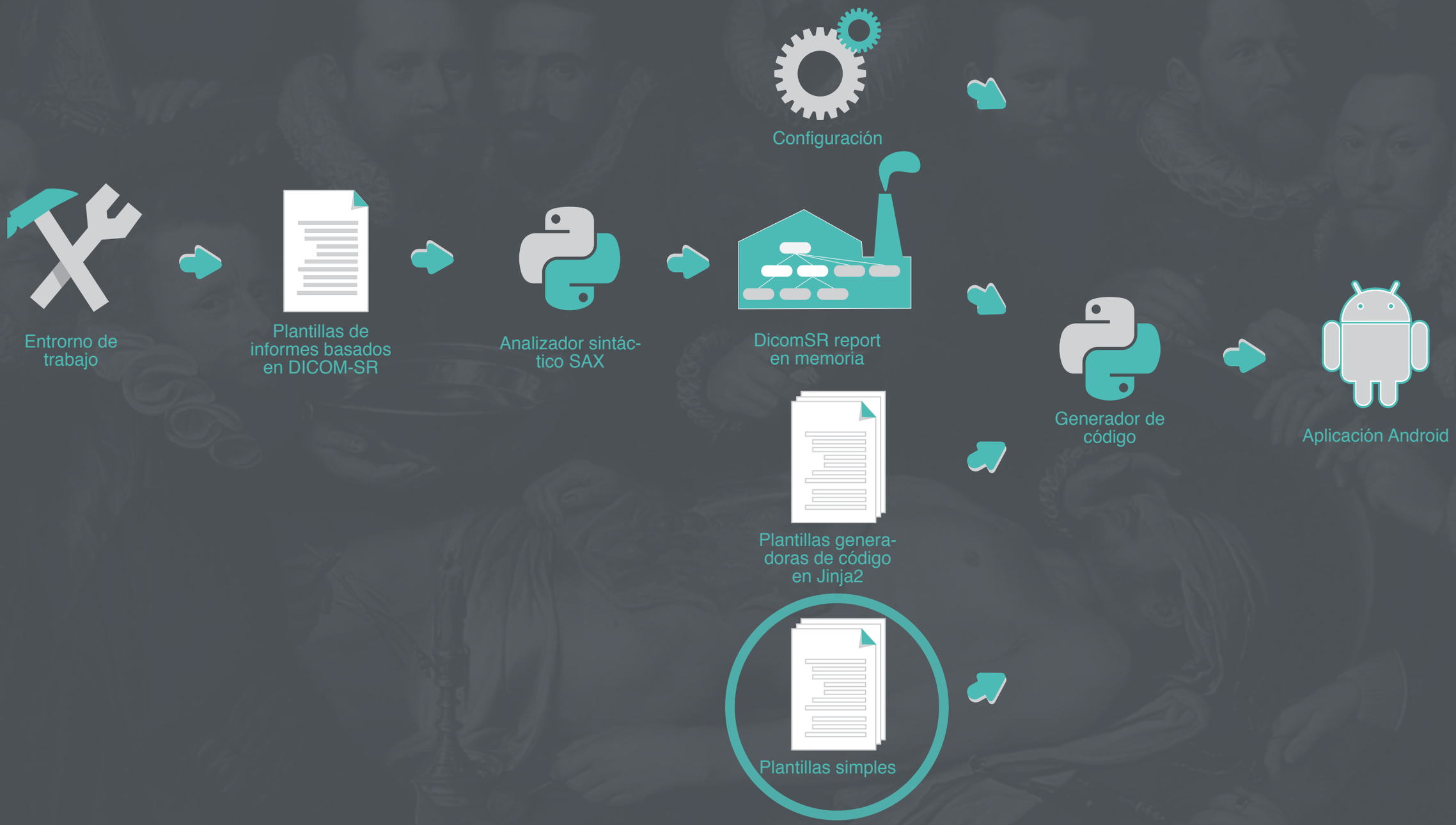
http://www.flickr.com/photos/moverelbigote/

Demasiado lío

http://www.flickr.com/photos/jeffreyww

Demasiadas capas

DISECCIÓN DEL GENERADOR DE CÓDIGO



PYTHON COMO UNA NAVAJA SUIZA

Probablemente esa librería que buscas ya existe. Y está en el nucleo de Python.

"NO INVENTES LA RUEDA"

KISS

STRINGS.TEMPLATE

- Sustitución simple de cadenas (PEP-292)
  - $identificador ≡ ${identificador}
- Uso:
  - Clase `string.Template(template)`
  - substitute(mapping,[,**kws])
  - safe_substitute(mapping,[,**kws])

STRINGS.TEMPLATE

```
from string import Template

xml_filename = 'tree_${PARENT}_${CODE}'
# Instanciamos la plantilla
xml_template = Template(xml_filename)
# Sustituimos por los valores concretos
filename = xml_template.safe_substitute(CODE=concept, PARENT=parent)
```

Uso de las plantillas de strings

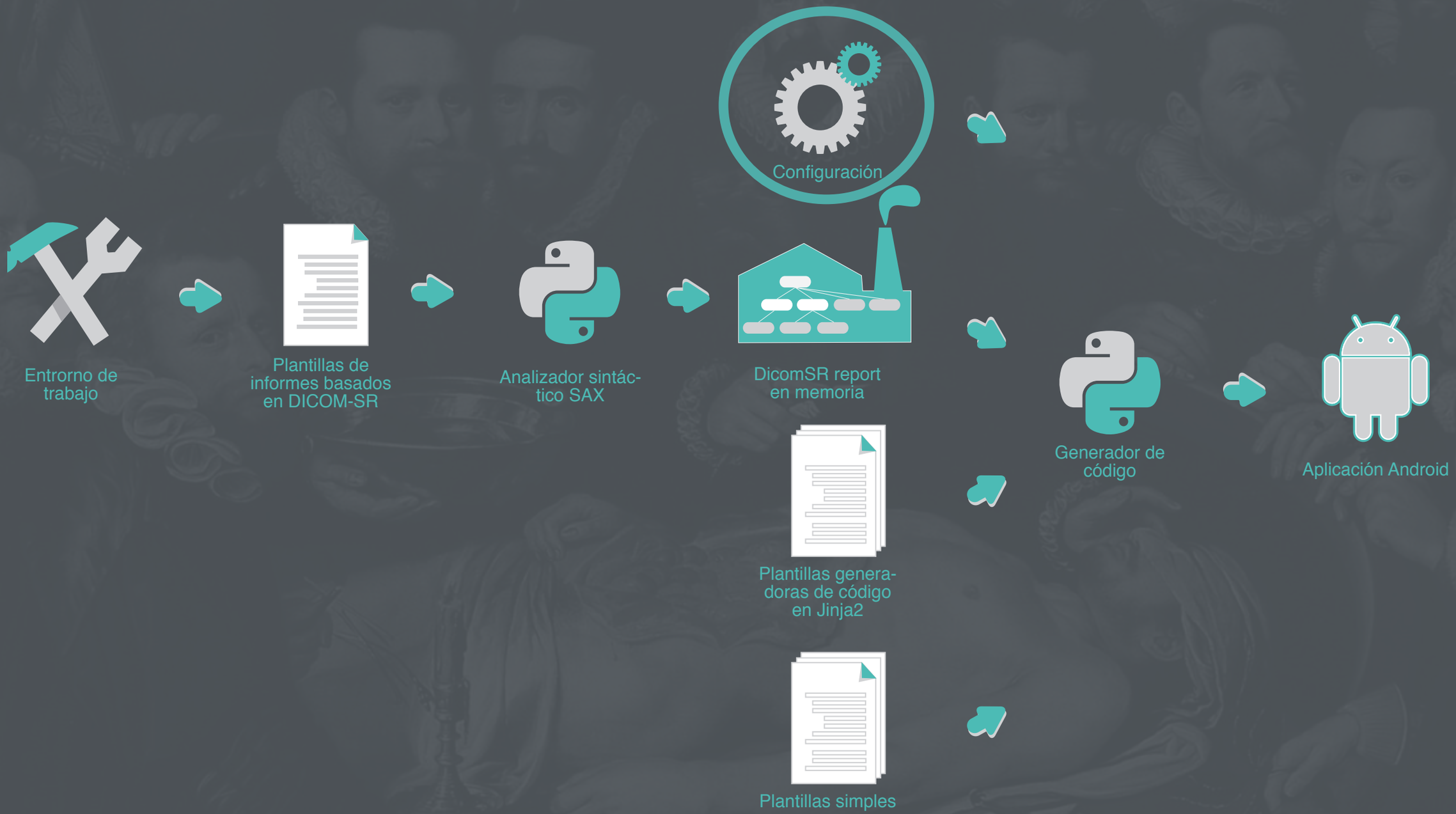
RE

expresiones regulares

```
>>> import re
>>> m = re.search(‘(?<=abc)def’, ‘abcdef’)
>>> m.group(0)
‘def’
```

Ejemplo básico del uso de las expresiones regulares

DISECCIÓN DEL GENERADOR DE CÓDIGO



CONFIGPARSER

- Características de la configuración:
  - Simple.
  - Fácil de crear
  - Fácil de intercambiar.
  - Fácil para la lectura.

- Formato simple

```
[Nombre de la sección]
atributo base: base
atributo 1 : %(atributo base)s/atributo.txt
```

CONFIGPARSER

Ficheros de configuración.

```
[Filenames]
level_1: summary_${CODE}
level_2: tree_${PARENT}_${CODE}
level_3: edit_${PARENT}_${CODE}

[Layouts Settings]
level_1: 2 Columns
level_2: 1 Column
level_3: 2 Columns
level_1_children: ListView
level_2_children: Expandable ListView
```

Configuración de usuario

```
[Output Directories]
MainDir: ./outputs
Strings: %(MainDir)s/strings
...
[Layout Templates]
#Layout
1 Column: one_column.xml
2 Columns: two_columns.xml
Main and Left: main_left.xml
Right : right_layout.xml
...
#UI
Generic title: generic_title.xml
Tree title: tree_title.xml
Code: spinner.xml
ListView: listview.xml
...
```

Configuración de sistema

CONFIGPARSER

- Implementación:
  - class ConfigParser.ConfigParser([defaults[, dict_type[, allow_no_value]]])
  - class ConfigParser.SafeConfigParser([defaults[, dict_type[, allow_no_value]]])
  - exception ConfigParser.NoSectionError
  - ...
- Objetos de ConfigParser
  - ConfigParser.get(section, option[, raw[, vars]])
  - ConfigParser.items(section[, raw[, vars]])

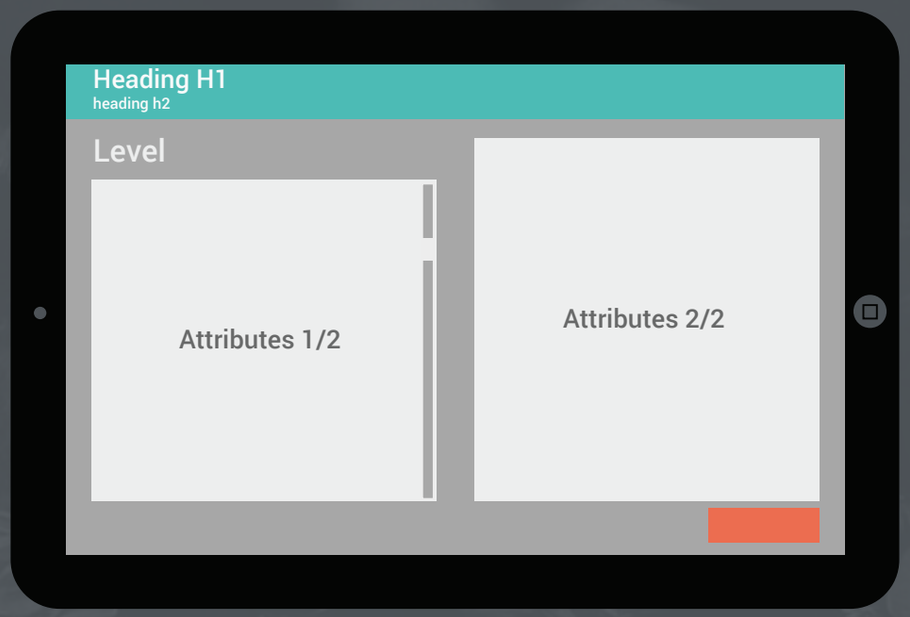
CONFIGPARSER

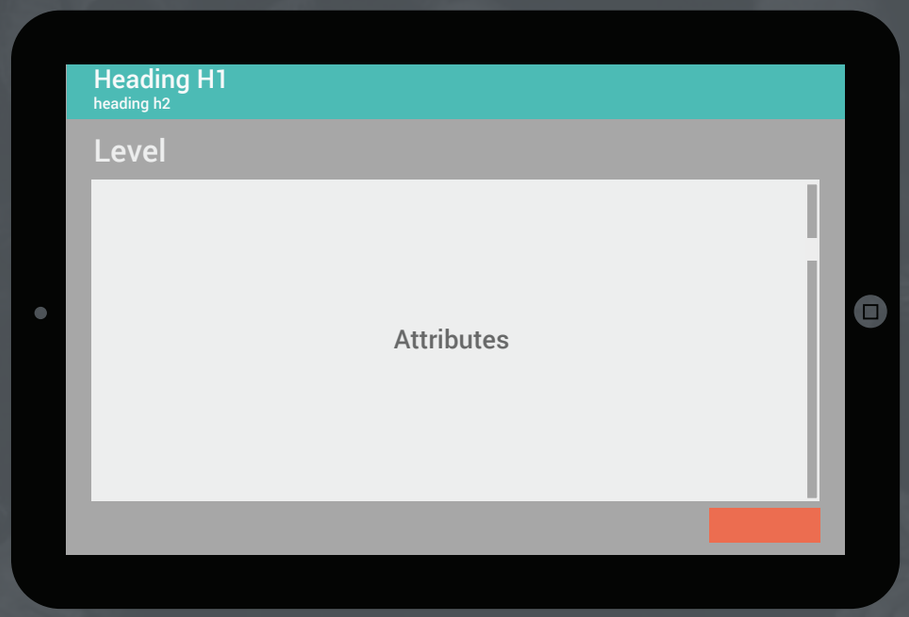
ejemplo de uso

```
def get_filepath(filetype, log):
    “”” Return the output path for the file type (section) given “””
   config = ConfigParser.ConfigParser()
    try:
        # Return user configuration
        return config.get(OUTPUT_DIRECTORIES_SECTION, filetype)
 # Handle exception when the filetype is not found.
except ConfigParser.NoOptionError:
    log.error(“Path not found”)
    return -1
```

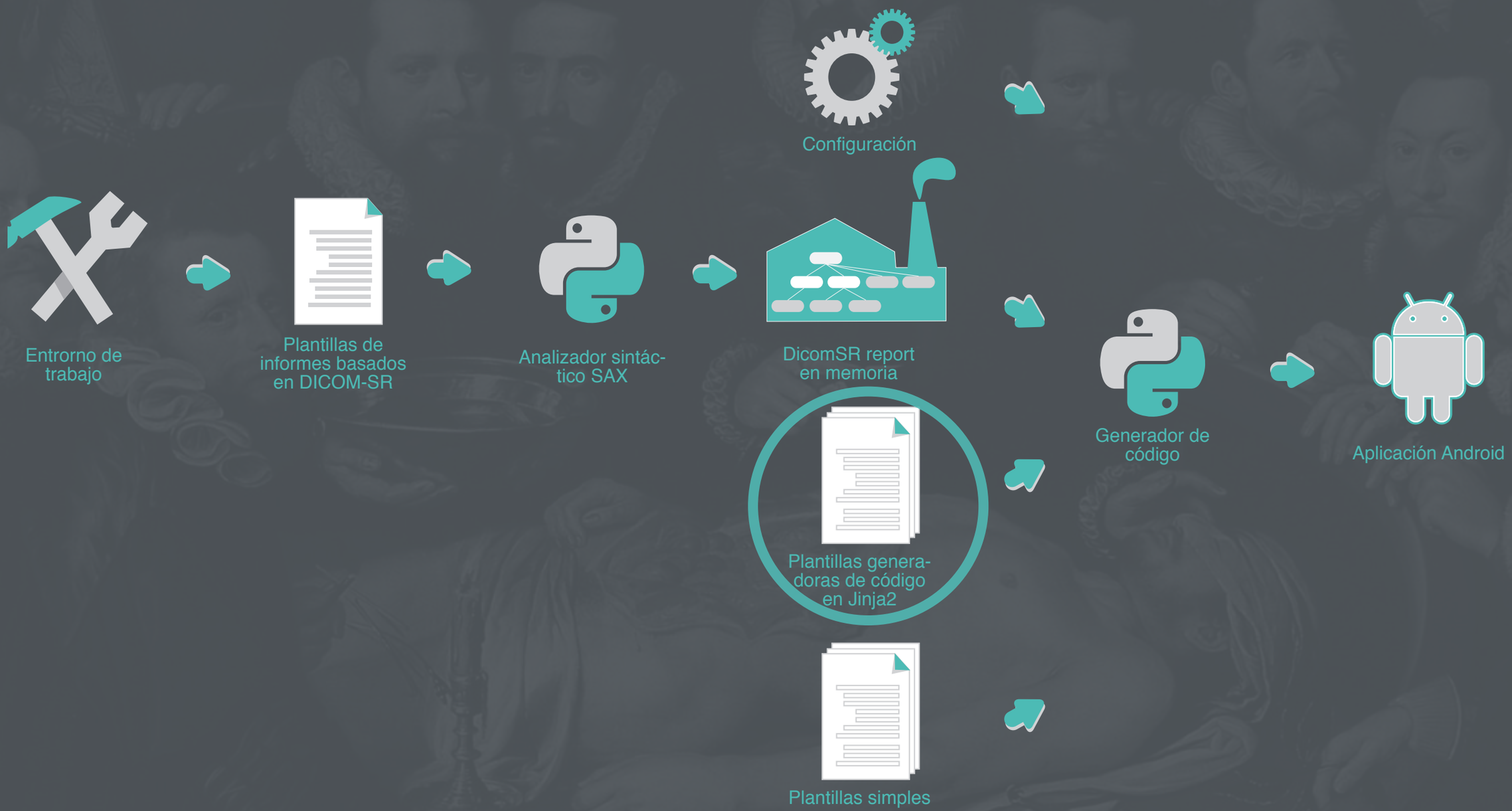
Instanciación del configparser

HOUSTON, NECESITAMOS PLANTILLAS MÁS COMPLEJAS





DISECCIÓN DEL GENERADOR DE CÓDIGO



JINJA 2

de Armin Ronacher

- Motor de plantillas para Python :
  - Simple.
  - Potente.
  - Rápido.
- Inspirado en el sistema de plantillas de Django.
- Sorporte unicode!
- Escrito en Python.
- BSD.

DEFINIENDO PLANTILLAS

de lo general a lo particular

```
{% extends “main.xml” %}
{% block content %}

        <!-- Main layout -->
        <RelativeLayout android:id=”@+id/center_layout” ... >
            <TextView android:id=”@+id/code_{{ level_code }}”... />
                {% block center_content%}
                {% endblock -%}

                {{content}}

            </RelativeLayout>
{% endblock %}
```

Ejemplo de una plantilla con una columna

DISEÑANDO PLANTILLAS

- Variables
  `{{ foo.bar }}`
- Filtros
  `{{ variable | filtro1 | filtro2 }}`
- Tests
  `{% if variable is defined %}`
- Comentarios
  `{# ... #}`
- Control de flujo
  ```
  {% for item in items %}
      ...
  {% endfor %}
  ```
- Herencia:
  - Padre:
    `{% block nombre %} {% endblock %}`
  - Hijo:
    ```
    {% extends plantillaBase %}
    ...
    {%block nombre%}...{% endblock %}
    ```
- Incluir otras plantillas:
  ```
  {% include ‘header.html’ %}
      Body
  {% include ‘footer.html’ %}
  ```

USANDO LAS PLANTILLAS

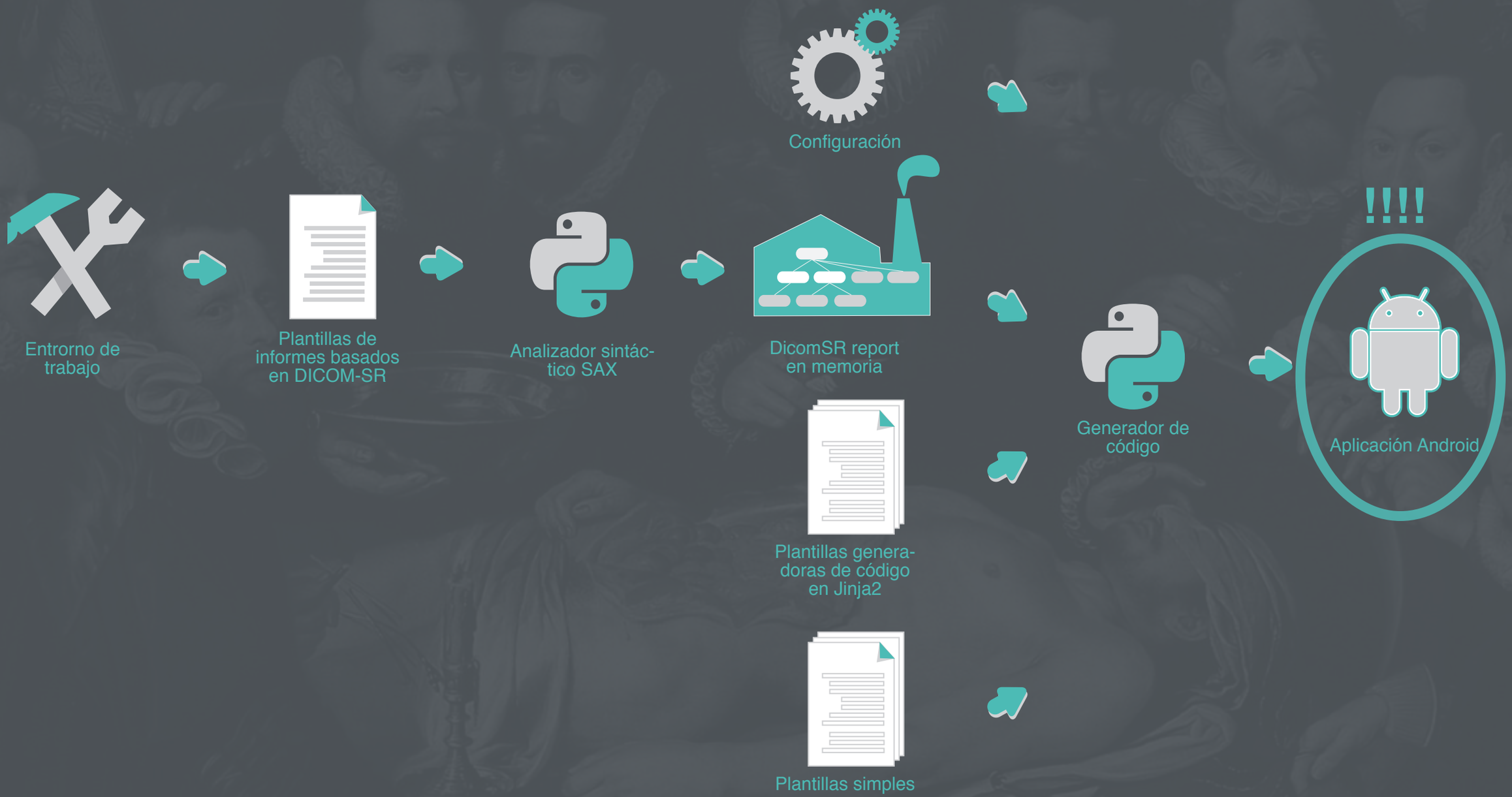
```
from jinja2 import Environment, PackageLoader, TemplateNotFound
...

def set_environment(template_type):
    “”” Set the jinja2 environment given a template type
    (string,layouts,activities or java classe).

    “””
    #Get the template_type path
    template_root = get_property(TEMPLATES_SECTION, TEMPLATES_ROOT_PATH)
    template_folder = get_property(TEMPLATES_SECTION, template_type)
    path = join(template_root, template_folder)

    ...
```

```
render_template = template.render(concept_value=concept.value,
                                  previous_item=previous_item)
```

Ejemplo básico de carga y uso de plantillas Jinja2

ÍNDICE

DISECCIÓN DEL GENERADOR DE CÓDIGO



LA SERPIENTE SE COMIÓ UN LEÓN

No hay problema demasiado grande para una pitón.

42

GRACIAS