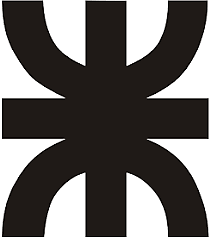
**Universidad tecnológica nacional**

Facultad Regional de Córdoba



Ingeniería en Sistemas de Información

Ingeniería de Software

***Docentes***

ING COVARO, Laura Inés (Adjunto);

ING. MASSANO, María Cecilia (JTP)

ING CRESPO, Mickaela (Ayudante 1ra)

***Integrantes***

ANGLADA, Martín Legajo: 58159 martinanglada@gmail.com;

CHECA, Nicolás Legajo: 58351;

MARRO, Yanina Legajo: 47594 yani\_marro@hotmail.com;

PICOSSI, Matias Legajo: 54941;

TOLEDO, Rodrigo Legajo: 75441.

**Grupo N°**: 7

***Curso:*** 4K3

2020

ÍNDICE

[PRÁCTICO 6 - Implementación de user stories (Evaluable) 2](#_Toc50574259)

[Unidad Nro. 3: Gestión del Software como producto 2](#_Toc50574260)

[Consigna: 2](#_Toc50574261)

[Objetivo: 2](#_Toc50574264)

[Propósito: 2](#_Toc50574265)

[Entradas: 2](#_Toc50574267)

[Salida: 2](#_Toc50574270)

[Instrucciones: 2](#_Toc50574273)

[Buenas prácticas y reglas de estilo de código Genexus 3](#_Toc50574280)

[Nomenclatura GIK 3](#_Toc50574282)

[Reglas 4](#_Toc50574283)

[Recomendaciones 6](#_Toc50574284)

PRÁCTICO 6 - Implementación de user stories (Evaluable)

Unidad Nro. 3: Gestión del Software como producto

Consigna:

Implementar una User Story determinada usando un lenguaje de programación elegido

por el grupo respetando un documento de reglas de estilo.

Objetivo:

Que el estudiante comprenda la implementación de una User Story como una porción

transversal de funcionalidad que requiere la colaboración de un equipo multidisciplinario.

Propósito:

Familiarizarse con los conceptos de requerimientos ágiles y en particular con User Stories en conjunto con la aplicación de las actividades de SCM correspondientes.

Entradas:

Conceptos teóricos sobre el tema desarrollados en clase. Definición completa de las User Stories correspondientes al Trabajo Práctico 2 “Requerimientos Ágiles – User Stories y

Estimaciones”

Salida:

Implementación de la User Story correspondiente en un programa ejecutable

Documento de estilo de código

Instrucciones:

* Seleccionar una User Story a implementar de entre las siguientes opciones:
  + Realizar Pedido a Comercio adherido (grupos pares)
  + Realizar un Pedido de "lo que sea" (grupos impares)
* Seleccionar el conjunto de tecnologías para implementar la funcionalidad elegida.
* Buscar y seleccionar un documento de buenas prácticas y/o reglas de estilo de código para el lenguaje de programación a utilizar.
* Implementar la US siguiendo las reglas de estilo determinadas.

BUENAS PRÁCTICAS Y REGLAS DE ESTILO DE CÓDIGO GENEXUS

En el presente documento se listan un conjunto de recomendaciones enunciadas por el fabricante de la herramienta GeneXus tendiendo a una mejor comprensión del conocimiento encapsulado en una KB.

Nomenclatura GIK

El Desafío: Compartir el conocimiento independientemente del criterio del Analista

GIK (GeneXus Incremental Knowledge Base)

Nombre de atributo > Objeto + Categoría + Calificador + Complemento

Objeto: Es el nombre de la transacción a la que pertenece el atributo. (1 a 6)   
Categoría: Es la categoría semántica del atributo. (1 a 3)   
Calificador: Puede existir uno o dos calificadores (1 a 3)   
Complemento: Texto libre

Ejemplos:

Nombre Trn: Cliente

CliCod\* Código de Cliente

CliNom Nombre de Cliente

CliApe Apellido de Cliente

CliFchNac Fecha Nacimiento de Cliente

CliTel Teléfono de Cliente

CliTelOficina Teléfono de Cliente en la Oficina

Recomendación de abreviar lo menos posible, es aceptable un largo en el nombre de un atributo de 10/12 Caracteres

¿Que Buscamos?

* Mejorar la comprensión
* Calidad en el Código
* Unificar criterios de programación
* Fácil integración y reutilización
* Eliminación de zonas oscuras de código
* Fácil comunicación entre programadores
* Claridad y correctitud en el código
* Incremento significable en la mantención del Software

Las Buenas prácticas de programación en GeneXus están compuestas por reglas y recomendaciones

Reglas

* Al nombrar atributos se debe utilizar la nomenclatura [GIK](http://wiki.genexus.com/commwiki/servlet/hwiki?GIK,)
* Los atributos deben tener descripción

Nombre: CliCod

Descripción: Código de Cliente

* Las Tablas deben tener nombres que representen la realidad y no el nombre heredado por la transacción que las crea.

Trn: Cli2SisPro

Tabla: Clientes

* Las variables que hagan referencia a un atributo deben ser basadas en el mismo y tener el mismo nombre del Atributo, si la lógica lo permite.

Atributo: CliCod - C(20) - Código de Cliente

Variable: &CliCod - C(20) - Código de Cliente

* Las variables que hagan referencia a un atributo deben ser basadas en el mismo y tener el mismo nombre del Atributo, agregando uno más sufijos para calificarla en el caso que sea necesario   
       Atributo: CliCod - C(20) - Código de Cliente  
       Variable: &CliCodOri - C(20) - Código de Cliente de Origen      <-----  Correcto  
       Variable: &CliCodDst - C(20) - Código de Cliente de Destino     <-----  Correcto  
       Variable: &CliDstCod - C(20) - Código de Cliente de Destino     <-----  Incorrecto
* Reporte de Especificación

Se considera una buena práctica que el Analista GeneXus revise con atención el reporte de especificación ya que está es la principal herramienta que tiene para detectar errores en el código.

* Definición de Reglas

Parm(in:EmpCod, in:&Usuario, in:&CliCod, in:&Mode);

noaccept(CiuCod);

&CliSaldoAux = udp(PcalcSaldo, EmpCod, &CliCod, CliSaldo);

error('Mensaje') If Null(&Usuario);

allownulls(EmpCod, LocCod ) ;

Call(PActInfo, EmpCod, CliCod) if <cond>;

error('Mensaje') IF CliDir = nullvalue(CliDir ) and after(CliDir) ;

prompt(Wclientes, EmpCod, CliCod);

default(CliFchCrea, Now() ) ;

Call(PInfoUsr, EmpCod,&Usuario) if <cond>;

nocheck(EmpCod, LocCod);

msg('Saldo menor que cero') if CliSaldo < 0;

Refcall(Wclientes, EmpCod, CliCod);

Call(Pprocedure, EmpCod, CliCod) if <cond>;

Si observamos este código nos damos cuenta de que para buscar algo tenemos que recorrer hasta el final ya que no muestra ningún criterio a seguir. Existen muchas maneras de definir las rules para que sean fáciles de entender, pero vamos a tomar dos criterios que serán vistos como buenas prácticas.

**Criterio Nro. 1**  
  
Definir las rules agrupadas por atributo:  
De esta manera podemos seguir el comportamiento que se programó para un atributo o variable en particular.

Ej:

Parm(in:EmpCod, in:&Usuario, in:&CliCod, in:&Mode);

error('Mensaje') If Null(&Usuario);

Call(PInfoUsr, EmpCod,&Usuario) if <cond>;

allownulls(EmpCod, LocCod ) ;

nocheck(EmpCod, LocCod);

Call(PActInfo, EmpCod, CliCod) if <cond>;

Call(Pprocedure, EmpCod, CliCod) if <cond>;

prompt(Wclientes, EmpCod, CliCod);

Refcall(Wclientes, EmpCod, CliCod);

error('Mensaje') IF CliDir = nullvalue(CliDir ) and after(CliDir) ;

default(CliFchCrea, Now() ) ;

default(CliArea, 'A' ) ;

noaccept(EmpCod);

noaccept(CiuCod);

msg('Saldo menor que cero') if CliSaldo < 0;

&CliSaldoAux = udp(PcalcSaldo, EmpCod, &CliCod, CliSaldo);

**Criterio Nro. 2 (Recomendado)**  
Agrupar las Rules por comportamiento  
  
De esta manera podemos ir directamente al sector del código en donde detectamos que está el error o para agregar comportamiento.

Parm(in:EmpCod, in:&Usuario, in:&CliCod, in:&Mode);

noaccept(EmpCod);

noaccept(CiuCod);

allownulls(EmpCod, LocCod ) ;

nocheck(EmpCod, LocCod);

default(CliFchCrea, Now() ) ;

default(CliArea, 'A' ) ;

error('Mensaje') If Null(&Usuario);

error('Mensaje') IF CliDir = nullvalue(CliDir ) and after(CliDir) ;

msg('Saldo menor que cero') if CliSaldo < 0;

Call(PInfoUsr, EmpCod,&Usuario) if <cond>;

Call(PActInfo, EmpCod, CliCod) if <cond>;

Call(Pprocedure, EmpCod, CliCod) if <cond>;

&CliSaldoAux = udp(PcalcSaldo, EmpCod, &CliCod, CliSaldo);

prompt(Wclientes, EmpCod, CliCod);

Refcall(Wclientes, EmpCod, CliCod);

Recomendaciones

* La descripción de los objetos de la KB debe ser clara independientemente del nombre del mismo.

Nombre: ModCliDeuda

Descripción: Modificación de Cliente con Deuda

* Se debe tener un método para dar nombre a los objetos. Esto depende mucho del tamaño de la(s) KB(s), tipo de instalación, criterios heredados y otra cantidad de factores, pero algunas cosas para tener en cuenta serían:
  + Al principio del objeto, un par de letras que identificara la aplicación (p.e. 'PE' para una aplicación de Personal, 'CO' para una aplicación de Compras). Esto podría ser opcional, dependiendo del ambiente.
  + Un identificador de la Entidad sobre la cual trabaja el objeto (p.e. Clientes, Facturas, Impuestos), la cual también podría ser nemotécnica (p.e. Cli, Fact, Imp)
  + Uno o más verbos, combinados con palabras o abreviaturas que identifiquen solamente qué hace el objeto (Calcular, Borrar, Modificar, Crear).
* La descripción de los objetos debe contener primero el objeto y después la acción que se ejecuta.

Nombre Descripción

CliDeudaModificar Cliente con Deuda - Modificar

FactIngresar Factura - Ingresar

COOrdenCompraImpCalcular Orden de Compra - Impuestos - Calcular (este sería un objeto de la KB de Compras, p.e. en un ambiente de desarrollo distribuido)

VEClientesTrabCon Clientes - Trabajar con (este sería un objeto de la KB de Ventas, p.e. en un ambiente de desarrollo distribuido)

* Utilizar nombres nemotécnicos para las variables que no correspondan a ningún atributo del sistema.

Se quiere cargar en una variable la existencia de un cliente.

Forma correcta: ExisteCliente

Forma Incorrecta: Flag

* En el encabezado de los programas se debe hacer un cuadro con una pequeña descripción del propósito del mismo y datos útiles.

/\*

Autor: Andres Esteban (andresesteban@gmail.com)

Fecha de Creación: 26-06-2014

Ultima modificación: 27-06-2014

Versión: 1.2

Descripción: Cambia el estado de los movimientos luego de la autorización del Usuario

\*/

* Colocar una línea en blanco entre las definiciones de eventos o subrutinas para separar los mismos y hacer más comprensibles los programas.

* Dentro de los eventos se debe comenzar a escribir código luego de hacer un “tab”, esto facilita la visualización del código

// Forma incorrecta:

Event 'NuevoCli'

If &CliCod = &Cliente

//Codigo

Endif

EndEvent

// Forma Correcta:

Event 'NuevoCli'

If &CliCod = &Cliente

//Codigo

Endif

EndEvent

* Para que los ForEach queden más claros y fáciles de identificar dentro de los eventos o del código en general, se recomienda que se escriban de la siguiente manera:

// Forma Incorrecta:

Event 'NuevoCli'

For Each

where CliCod = &CliCod

//Còdigo

EndFor

EndEvent

// Forma correcta:

Event 'NuevoCli'

For Each

where CliCod = &CliCod

//Còdigo

EndFor

EndEvent

* Para que los filtros de los ForEach queden más claros se recomienda tener un where para cada condición y no utilizar AND.

//Forma incorrecta:

For Each

where CliCod = &CliCod and CliStatus = &CliStatus and CliTipo = &CliTipo

//Còdigo

EndFor

// Forma correcta:

For Each

where CliCod = &CliCod

where CliStatus = &CliStatus

where CliTipo = &CliTipo

//Código

EndFor

* Evitar abreviar excesivamente

Los nombres de las variables, subrutinas, objetos, etc deben de ser lo más claro posibles ya que si alguien externo debe trabajar con el código deberá, además de entender el código en general, deberá estar descifrando los nombres de cada variable, etc.  
Ej: Se quiere cargar en una variable el cliente por proveedor

//Forma Incorrecta:

&CPProv

//Forma correcta:

&ClientePorProveedor

* Claridad en el código

La claridad en el código también se considera una buena práctica de programación, en muchos casos por costumbre los programadores abusan del uso del "if" olvidando que existe el comando "Do Case". no soportaban este comando y la costumbre es más fuerte que el cambio.

* Los atributos deben estar basados en Domains

Hay que tratar que todos los atributos estén siempre basados en algún Domain, de esta manera es más fácil adaptarse a los cambios de tipos o largos.

* Utilización de Patterns

Se recomienda que toda aplicación web utilice Patterns, los patterns de GeneXus nos ofrecen una herramienta ideal para crear aplicaciones web. Facilitan la migración de ambiente win a web y nos ofrecen una forma práctica de solucionar problemas que antes nos llevaban mucho tiempo. Por más información leer Patterns.

* Evitar constante en el codigo

Usar los [enumerators](http://wiki.genexus.com/commwiki/servlet/hwiki?GeneXus+Enumerator,) en vez de constantes en el código. De esa manera si se cambia la constante no es necesario cambiar en todos los lados que se usa.

&Type = "CR" // MAL

&Type = BalanceType.Credit // BIEN

*Fuente: https://wiki.genexus.com/commwiki/servlet/wiki?1755,Wiki+Home,*