Nombre y apellido: Tomás Santucci Legajo: S-5512/3

Ingeniería de Software 2
Lic. en Ciencias de la Computación
F.C.E.I.A. - U. N. R.

Cristiá, Krapf, Scandolo
Viernes 29 de septiembre de 2023
Duración aproximada 4 hs.

PRIMER PARCIAL

Nota: La interpretación de las consignas es parte del examen. El parcial se aprueba con al menos dos problemas bien y no menos del 65% del puntaje. Problemas parcialmente correctos no necesariamente suman puntaje.

Se evalúa: DBOI, DTAD, DOO, documentación de diseño.

Problemas

1. Explique el método propuesto por Parnas para diseñar un sistema de software.

2. Documente, como se indica más abajo, un diseño que tenga en cuenta los siguientes requerimientos e ítem de cambio.

   Se espera que los componentes de un sistema puedan comunicarse entre sí pero sin que se conozcan entre sí. En otras palabras, un componente del sistema no sabe de la existencia de otros componentes del sistema. Por lo tanto, se recurre al concepto de *evento*. Los componentes anuncian o *publican* eventos (sin saber quién los está esperando) y a su vez se anotan o *suscriben* a eventos (que tal vez nunca aparezcan).

   Un componente especial, llamado *administrador de eventos* es el encargado de intermediar esta forma de comunicación. Cada suscriptor manifiesta interés en un evento suscribiendo una subrutina en su interfaz a ese evento. La suscripción se hace ante el administrador de eventos. Los componentes publican eventos llamando al administrador de eventos. La tarea del administrador de eventos es, entonces, invocar a cada una de las subrutinas suscritas a un evento cada vez que este es publicado (por cualquier componente del sistema).

   Cada evento tiene un nombre y una serie de parámetros fija por tipo de evento. El administrador de eventos pasa los parámetros del evento al suscriptor.

   ### Ítem de Cambio

   - La política de invocación del administrador de eventos.
   - Agregar nuevos tipos de eventos sin tener que modificar el administrador.
   - Alterar la lista de parámetros de un evento sin tener que modificar el administrador.

   ### Documentación

   Documente un sistema que incluye el administrador de eventos, un componente que anuncia un evento, otro que se suscribe a ese evento y dos tipos de eventos diferentes.

   (a) Módulos 2MIL.

   (b) Guía de módulos.

3. Cierta aplicación tendrá que ejecutar en varios países. Suponga que una operación compleja de la aplicación tiene partes que dependen del país en el cual ejecute y otras que no dependen de eso.

   (a) Describa un diseño que utilice solo banderas para resolver el problema.

   (b) Describa un diseño que utilice solo herencia para resolver el problema.

   (c) Describa un diseño que combine herencia con composición para resolverlo.

   (d) Analice las ventajas y desventajas de las tres soluciones.

1) El método propuesto por Parnas consta de 5 pasos:
- Identifican los ítems de cambio presentes en los requerimientos.
- Para cada uno de ellos analizan cuales serían algunos cambios posibles (variantes).
- Establecen una probabilidad para las variaciones analizadas en el ítem anterior (baja/media/alta)
- Ocultan cada ítem con alta probabilidad en un módulo. Es decir, en cada módulo se oculta un único ítem de cambio.
- Diseñan interfaces para cada módulo que sean insensibles a los cambios anticipados.

3) Supongamos que esta operación compleja es parte de un módulo M.

a)
```
MODULE M
IMPORTS PAIS
EXPORTS
    establecerPais(i PAIS)
    operacion()
```

En cada parte donde operacion() dependa del país habrá que usar condicionales.

b)

```
MODULE M
EXPORTS
  operacion()
```

```
MODULE MArg INHERITS FROM M
```

```
MODULE MBra INHERITS FROM M
```

c)

```
MODULE ComponenteInt
EXPORTS
  op1()
  op2()
  ...
COMMENTS Una subrutina por cada parte que dependa
del país
```

```
MODULE CompArg INHERITS FROM ComponenteInt
```

```
MODULE CompBra INHERITS FROM ComponenteInt
```

```
MODULE M
IMPORTS ComponenteInt
EXPORTS
  setCompPais(i:ComponenteInt)
  operacion()
```

d) En cuanto al primer diseño:

Ventaja:

- Si necesito algo simple, chico y que no va a necesitar cambios

Desventaja:

- No se adapta a los cambios.
- No cumple el POI. Más de un ítem de cambio en el mismo módulo.
- Siempre entrego todo el código. Aunque sólo necesita la versión de 1 país.
- El código se tornará muy complejo debido a la anidación de condicionales.
- Si hago un cambio para un país tengo que retestear todo.

Segundo diseño:

Ventaja:

- Se pueden entregar distintas versiones del proyecto.
- Mejor separación en módulos para adaptarme a cambios.

Desventajas:

- Tendré duplicación del código que es común a los países.
- Si por ejemplo quiero cambiar el idioma dinámi-

nicamente, no podrá pues los relaciones son estáticas.

Ventajas del último diseño:

- Se entregan sólo los módulos requeridos para un proyecto.
- Conserva el POI.
- No se duplica código.
- Es fácil agregar países y no tengo que testear todo.
- El país se setea dinámicamente.

Desventaja:

- Como se compone el país dinámicamente podría ser más difícil la verificación del programa en tiempo de compilación.

2)

```
MODULE AdminEv
IMPORTS Eventos, Evento, Args, PoliticaInvocacion,
        RepoRutinas
EXPORTS
    suscribir(i IDEvento, i *F)
    desuscribir(i IDEvento, i *F)
    anunciar(i IDEvento, i Args)
```

```
MODULE PoliticaInvocacion
IMPORTS RepoRutinas, Evento, Args

EXPORTS
    setearRutinas(i RepoRutinas)
    invocar(i IDEvento, i Args)
COMMENTS setearRutinas recibe las rutinas que
coinciden con el ID anunciado.
        Luego invocar ejecuta cada una de ellas
según alguna política pasando los argumentos.
```

MODULE RepoRutinas
IMPORTS Evento
EXPORTS
  agregarID(i IDEvento)
  agregarFun(i *F)
  getID(): IDEvento
  getFun(): *F
  primero()
  siguiente()
  hayMas(): Bool

MODULE Evento
IMPORTS Params
EXPORTS
  setID(i IDEvento)
  getID(): IDEvento
  setParams(i Params)
  getParams(): Params

MODULE Eventos IS List<Evento>

MODULE Params IS List<TYPE>
COMMENTS asumimos TYPE es un tipo básico
  para representar los tipos de parámetros

MODULE Args IS List<Value>

Guía de Módulos:

1 SistemaEventos

Módulo lógico que agrupa todo el sistema

1.1 AdminEv

Provee una interfaz para que los componentes puedan suscribirse e invocar eventos.

Oculta los procesos de suscripción con el repositorio y la invocación con sus posibles políticas

1.2 Invocación

Módulo lógico que agrupa los módulos usados para la invocación de rutinas.

1.2.1 PoliticaInvocación

Recibe un conjunto de rutinas y los ejecuta pasando los argumentos.

Oculta los políticas para invocarlas.

1.2.2 RepoRutinas

Guarda ids de eventos junto con callbacks a invocar luego

Oculta la forma en que se guarda esta información. Puede ser bases de datos, archivos, etc.

1.3 AbstraccionEvento

Módulo lógico que agrupa los módulos relacionados con la información que guarda un evento.

1.3.1 Eventos

Guarda una lista de eventos y provee una interfaz para accederla.

Oculta qué forma tiene esta lista y la inclusión de eventos nuevos

1.3.2 Evento

Guardan el ID y los tipos de parámetros que un evento recibe.

Oculta la estructura de datos que tendrá este objeto.

1.3.3 Args

Guarda una lista de valores de ciertos tipos que pueden ser distintos.

El secreto es la forma en que se guardan/pasan los mismos.

1.3.4 Params

Guarda una lista de tipos.

Oculta la estructura de datos a utilizar y la inclusión de nuevos parámetros a un evento.

Un componente que anuncia un evento podría tener la siguiente forma:

```
MODULE Anunciador
IMPORTS AdminEv, Value, Evento
EXPORTS
    agregarArg(i Value)) No público
    setID(i IDEvento)
    anunciar()
COMMENTS anunciar() usa el ID y argumentos seteados
    y con ellos llama a anunciar del módulo
    AdminEv.
```

```
MODULE Suscriptor
IMPORTS AdminEv, Evento
EXPORTS
    op()
    setID(i IDEvento)  } ) No público
    suscribir()
COMMENTS suscribir() llama a suscribir de AdminEv
    con op() como callback.
```

Para definir dos tipos de eventos habría que crear 2 objetos. Ejemplo:

evento1, con evento1.getID() es "boot"
evento1.getParams() es <Int, String>

evento2, con evento2.getID() es "tick"
evento2.getParams() es <Int>

Un módulo que los crea podría tener la siguiente forma:

```
MODULE CreadorEvento
IMPORTS Evento
EXPORTS
    putID(i DEvento)
    putParam(i TYPE)
    create()
COMMENTS donde create() usa los subrutinos
    setID y setParams del módulo Evento
```