L^3(Lcubo)

December 23, 2010

Resumen Ingeniería I

Abstract

Contents

1 Ingenieria de Requerimientos **8**

1.1 Que es Ingeneria de Requerimientos? . . . 8

1.2 Alcanze de los requerimientos de ingenieria . . . 9

1.3 Factores de calidad . . . 9

1.3.1 Completitud . . . 9

1.3.2 Consistencia . . . 9

1.3.3 Adecuacion . . . 9

1.3.4 Trazabilidad . . . 9

1.3.5 Pertinencia . . . 9

1.3.6 Dimension PORQUE . . . 10

1.3.7 Dimension QUE . . . 10

1.3.8 Dimension QUIEN . . . 10

1.4 Ciclos de vida del desarollo . . . 10

1.4.1 Cascada . . . 10

1.5 Proceso en espiral (Boehm, 1988) . . . 11

1.5.1 Unified SW development . . . 11

1.5.2 Modelo Twin Peaks . . . 11

1.6 Relacion entre categorias de requerimientos . . . 12

1.7 Economia . . . 12

2 Modelo Jackson **12**

2.1 Definiciones . . . 12

2.2 Proposito de la maquina en el mundo . . . 13

2.3 Requerimientos . . . 13

2.4 Taxonomias . . . 14

3 Modelo de 4 variables **15**

4 Modelo Objetivos **15**

4.1 Definiciones . . . 15

4.2 Taxonomia de objetivos . . . 16

4.2.1 Objetivos Lograr . . . 16

4.2.2 Objetivos Mantener . . . 16

4.2.3 Objetivos Blandos . . . 16

4.2.4 Objetivos funcionales vs. no-funcionales . . . 17

4.3 Refinamientos de objetivos . . . 17

4.4 Asignacion de responsabilidades . . . 17

4.5 Patrones de Refinamientos . . . 17

4.5.1 Y-refinamiento . . . 17

4.5.2 O-refinamiento . . . 17

4.5.3 Refinamiento por casos . . . 17

4.5.4 Refinamiento por hitos . . . 18

4.5.5 Refinamiento Introduccion de guarda . . . 19

4.5.6 Refinamiento Divide and Conquer . . . 19

4.5.7 Refinamiento para lograr objetivos asignables . . . 20

4.6 Obstaculos . . . 20

4.7 Semantica . . . 20

5 Modelo Agentes **20**

5.1 Relacion con otros modelos . . . 20

5.1.1 Objetivos . . . 20

6 Modelo operacional **21**

6.1 Casos de uso . . . 21
6.1.1 Semantica CU . . . 21
6.1.2 Actores . . . 22
6.1.3 Relacion Participa En . . . 22
6.1.4 Generalizacion . . . 22
6.1.5 Entes Abstractos . . . 22
6.1.6 Inclusion o Usa a . . . 22
6.1.7 Elementos Foraneos . . . 22
6.1.8 Extensiones . . . 22
6.2 Relacion con el modelo de objetivos (operacionalizacion de objs) . . . 22
6.3 Semantica : Objetivos, agentes,objetos y operaciones . . . 23
6.4 Modelo Objetos . . . 23

7 Modelo Conceptual **23**

7.1 Propiedades : Atributos y enlaces . . . 23
7.2 Diagramas de clases . . . 24
7.2.1 Estrategias para identificar clases . . . 24
7.2.2 Asociaciones . . . 24
7.2.3 Visión Semántica . . . 24
7.2.4 Roles . . . 24
7.2.5 Agregación . . . 24
7.2.6 Composición . . . 24
7.2.7 Asociación n-arias . . . 25
7.2.8 Clases de asociación . . . 25
7.2.9 Generalización . . . 25
7.2.10 Clase Abstracta . . . 25
7.3 Diagramas de objetos . . . 25
7.4 OCL . . . 26
7.4.1 Colecciones . . . 26
7.5 LTS . . . 26
7.5.1 Modelado de procesos . . . 27

7.5.2 Modelado : Concurrencia . . . 27
7.5.3 Modelando : Eleccion . . . 27
7.5.4 Modelando : Eleccion no deterministica . . . 27
7.5.5 Equivalencias . . . 28
7.5.6 Equivalencia por Trazas . . . 28
7.5.7 Equivalencia isomorfismo . . . 28
7.5.8 Bisimulacion . . . 28
7.5.9 Chequeo bisimulación . . . 29
7.5.10 Mostrar no-bisimilitud . . . 29
7.5.11 Bisimiliridad Debil . . . 29
7.5.12 Congruencia . . . 30
7.5.13 Semanticas LTS . . . 30
7.5.14 Relación con otros modelos . . . 30
7.6 State Charts . . . 30
7.7 SDL flowchart . . . 30
7.8 Redes Petri . . . 30
7.8.1 Semantica . . . 30
7.9 MSC . . . 30
7.10 Diagramas de actividad . . . 30
7.10.1 Particiones . . . 31

8 Modelo Comportamiento 31

8.1 Maquinas de estado . . . 31
8.1.1 Definiciones . . . 31

9 Testing 31

9.1 Validacion vs. Verificacion . . . 32
9.2 Aspectos de los errores . . . 32
9.2.1 Verificacion : Estatica vs. Dinamica . . . 32
9.3 Test : Requerimientos no funcionales . . . 32
9.4 Testing Random . . . 33

9.5 Niveles de test . . . 33
9.6 Test Integracion . . . 33
9.6.1 Estrategias de test de integracion . . . 33
9.6.2 Modelo de ciclo de vida en V . . . 34
9.7 Testing Funcional . . . 34
9.7.1 Particion Categorias . . . 34
9.8 Criterio . . . 34
9.8.1 Criterios Ideales . . . 34
9.9 Tecnica particion de dominio (Ostrand y M. Balcer) . . . 35
9.10 Identificacion Categorias . . . 35
9.10.1 Heuristica 1 . . . 35
9.10.2 Heuristica 2 . . . 35
9.10.3 Heuristica 3 . . . 35
9.10.4 Heuristica 4 . . . 35
9.10.5 Heuristica 5 . . . 35
9.10.6 Heuristica 6 . . . 35
9.10.7 Heuristica 7 . . . 35
9.10.8 Heuristica 8 . . . 35
9.11 Notacion arbórea . . . 36
9.12 Tecnicas Combinatorias . . . 36
9.12.1 Grafo Causa-Efecto . . . 36
9.12.2 2-wise partition y arreglos ortogonales . . . 36
9.13 Testing Estructural . . . 37
9.14 Criterios . . . 37
9.14.1 Cubrimientos de sentencias . . . 37
9.14.2 Cubrimiento de deciciones (o branches) . . . 37
9.14.3 Cubrimiento de condiciones . . . 37
9.14.4 MC/DC . . . 37
9.14.5 def-use flowgraph . . . 37
9.14.6 Criterio : Cubrimiento all-uses . . . 38

9.14.7 Problema testing estructural . . . 38
9.15 Comparacion de criterios . . . 38
9.15.1 Subsumicion . . . 38
9.16 Medidas probabilisticas . . . 38
9.16.1 Properly Covers . . . 38

10 Testing Sistemas Reactivos **39**
10.1 Testing maquinas de estados . . . 39
10.2 Mealy Machines . . . 39
10.2.1 Algoritmo TS Chow . . . 40

11 Testing LTS **41**
11.1 Conformance trazas . . . 41

12 IOLTS **41**
12.1 Conformance trazas (IOTR) . . . 42
12.2 Limitaciones $\leq iotr$ . . . 43
12.3 IOCONF . . . 43
12.4 IOCO . . . 43

13 Preguntas de finales **43**
13.1 Preguntas Sueltas . . . 43
13.2 20 de nov de 2007 . . . 43
13.3 27 de nov de 2007 . . . 44
13.4 25 de febrero de 2008 . . . 45
13.5 4 de marzo de 2008 . . . 46
13.6 7 de abril de 2008 . . . 48
13.7 10 de junio de 2008 . . . 48
13.8 24 de julio de 2008 . . . 49
13.9 15 octubre 2008 . . . 50
13.1022 de diciembre de 2008 . . . 51
13.112 de marzo de 2009 . . . 51
13.122 de marzo de 2010 . . . 51

14 Casos de test en testing de sistemas reactivos **53**

1 Ingenieria de Requerimientos

Se ocupa de construir un producto de software de alta calidad bajo restricciones de tiempo y presupuesto.

1.1 Que es Ingeneria de Requerimientos?

Ingeniria de requerimientos Es un conjunto de actividades que intentan identificar y comunicar el proposito de un sistema de software y el contexto en que sera usado. RE actua como el puente entre las necesidades del mundo real de los usuarios, clientes y otras circunscripciones afectadas por el sistema de software, y las capacidades y oportunidades que ofrecen las tecnologias de software.

RE (Requirement Engineering) trata sobre explorar el espacio del problema. De la definicion de ingeniera de Requerimiento se desprenden varias cosas importantes.

- RE no es una etapa, es un conjunto de actividades y suele estar durante todo el desarrollo del sistema.
- Sale la palabra proposito, muy importante en la definicion y es la clave de requerimientos. El proposito esta relacionado con la calidad, cuando mejor se capture el proposito mejor es la calidad.
- Contexto : concepto muy importante para garantizar un buen sistema es que se conoce bien donde este se usara. Parece trivial, pero muchas veces el contexto no se conoce completamente (problemas de seguridad?).
-

Sistema Es un conjunto de componentes o procesos que interactuan con el mundo real para satisfacer objetivos de alto nivel. Estos objetivos de alto nivel suelen ser el proposito del sistema.

Software Es la maquina que se suele construir para satisfacer ciertos objetivos de bajo nivel (requerimientos en el mundo de objetivos). Es un componente del sistema, pero no el unico.

Consecuencias de la definicion de ingenieria de software.

- La ingenieria de requerimientos no es una etapa ni fase.
- Tanto el proposito del sistema como el contexto son importantes. El contexto es donde el sistema (sistema != maquina) sera usado.
- La identificacion y comunicacion del proposito es clave para el exito y calidad
- Necesidad de identificar todas las partes involucradas, no solo el usuario y cliente.

Validación Proceso cuyo objetivo es incrementar la confianza de que una descripcion formal se corresponde con la realidad.

Verificación Proceso cuyo objetivo es garantiza que una descripción formal es correcta con respecto a otra.

stakeholder Es un grupo o un individuo affectado por el sistema, quien puede influenciar la forma en que el sistema es construido y tiene cierta responsabilidad en la aceptacion. Son una fuente de informacion clave en RE.

1.2 Alcanze de los requerimientos de ingenieria

El $sistema - asi - como - esta$ ($system - as - is$) es el sistema en funcionamiento cuando el proyecto comienza. Consiste en un conjunto de componenetes (personas,hardware,software,etc). Este sistema suele tener problemas, deficiencias y limitaciones.

El $sistema - como - sera$ ($system - to - be$) es el sistema que se espera desarrollar, el cual sera nuevo o modificado para solucionar los problemas y limitaciones del $sistema - asi - como - esta$. Este no solo incluye el software, ademas debera contemplar el enterno en donde se usara.

El espacio del problema del $sistema - como - sera$ tiene tres dimensiones : $PORQUE$,QUE y $QUIEN$.

1.3 Factores de calidad

Los factores de caidad definen objetivos del proceso de RE. Tambien proveen una forma de evaluar sucesivas versiones de requerimientos

1.3.1 Completitud

Globalmente , los requerimientos, las aserciones y las propiedades de dominio juntas deben ser suficiente para garantizar que el $sistema - como - sera$ cumpla con sus objetivos. Estos objetivos tienen que estar bien definidos. En particular, se debe anticipar incidentes o comportamientos maliciosos del entorno.

Los requerimientos son completos si se verifica $R, D \vdash G$, en otras palabras que las presunciones son validas y que los requerimientos capturan las necesidades de los stakeholders. Los requerimientos son completos si :

- R garantiza G presumiendo D. $R, D \vdash G$
- G captura adecuadamente todas las necesidades de los stakeholders
- D representa presunciones validas acerca del mundo.

1.3.2 Consistencia

Los requerimientos, aserciones y propiedades del dominio deben ser compatibles entre ellas. $R, D \nvdash false$

1.3.3 Adecuacion

1.3.4 Trazabilidad

El contexto en el que un item de los documentos de requerimientos fueron creados, modificados o usados deben ser faciles de recuperar. Ese contexto debe incluir justificacion de la creacion,modificacion y uso. El impacto de crear, modificar o borrar un item debe ser facil de evaluar. Ese impacto puede referirse a items en el documento de requerimientos (trazabilidad horizontal) y depender de artefactos despues desarrollados, test data, manuales,etc.

1.3.5 Pertinencia

Los objetivos proveen un criterio preciso para la pertinencia de requerimientos. Un requerimiento es pertinente con respecto a un conjunto de objetivos si es usado como argumento de la satisfaccion de almenos un objetivo. Esto ultimo es muy importante, ya que habla de requerimientos irrelevantes. Esta de mas decir que los requerimientos irrelevantes no es algo que se deberia tener. Es tan importante como la consistencia o completitud.

1.3.6 Dimension PORQUE

1.3.7 Dimension QUE

1.3.8 Dimension QUIEN

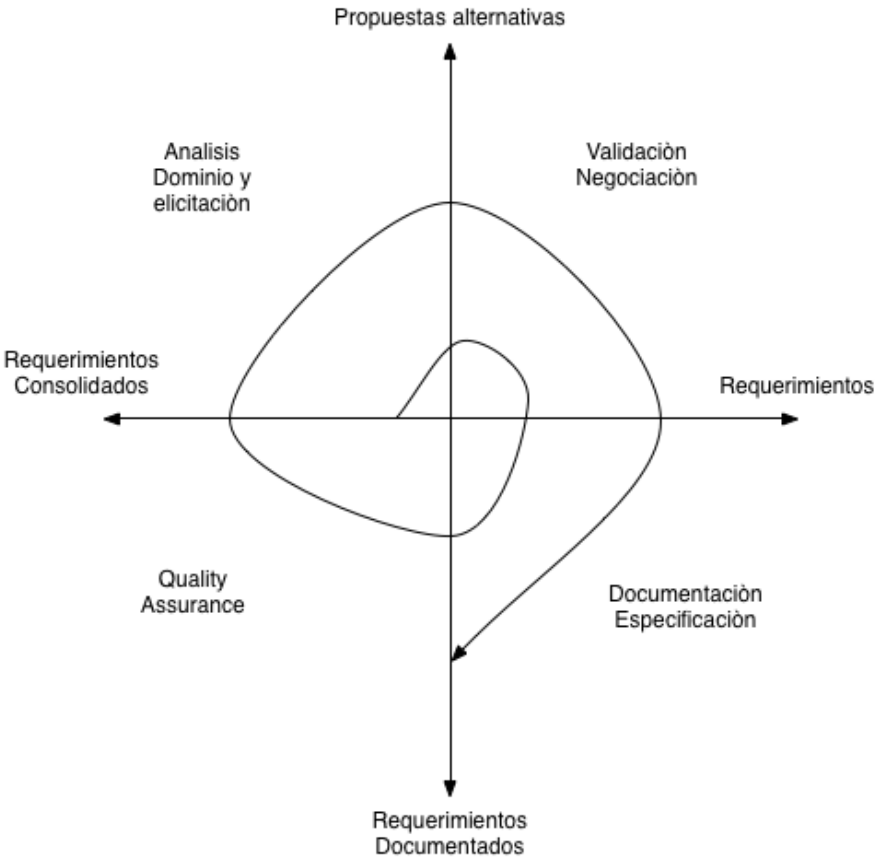
1.4 Ciclos de vida del desarollo

Refinamiento Es la transformacion de un dise;o de alto nivel o abstracto a uno de mas bajo nivel, mas concreto. Usualmente lo de mas alto nivel se lo llama especificacion y lo de mas bajo nivel implementacion

1.4.1 Cascada

- Elicitación y analisis de dominio Evocar una constestacion, respuesta, dato de alguien como reaccion a preguntas o acciones
- Modelado Consiste en abstraer y estructurar los elicitado. Se documenta de manera rigurosa
- Analisis Verificar inter e intra de los modelos.
- Validación Es el proceso que tiene como objetivo incrementar la confianza de los que se esta haciendo. Es el producto correcto?
- Priorización Se
- Negociación
- Especificación Se genera documentacion entregable, completa y detallada.

1.5 Proceso en espiral (Boehm, 1988)



En este tipo de proceso, las etapas son las mismas que las anteriores (elicitacion, modelado,etc). Se toma una idea de iteración (varias veces se pasa por elicitación). Cuatro variables se comparar (no tiene nada que ver con el modelo de 4 variables),estas son :

- Propuestas alternativas
- Requerimientos acordados
- Requerimientos documentados
- Requerimientos consolidados

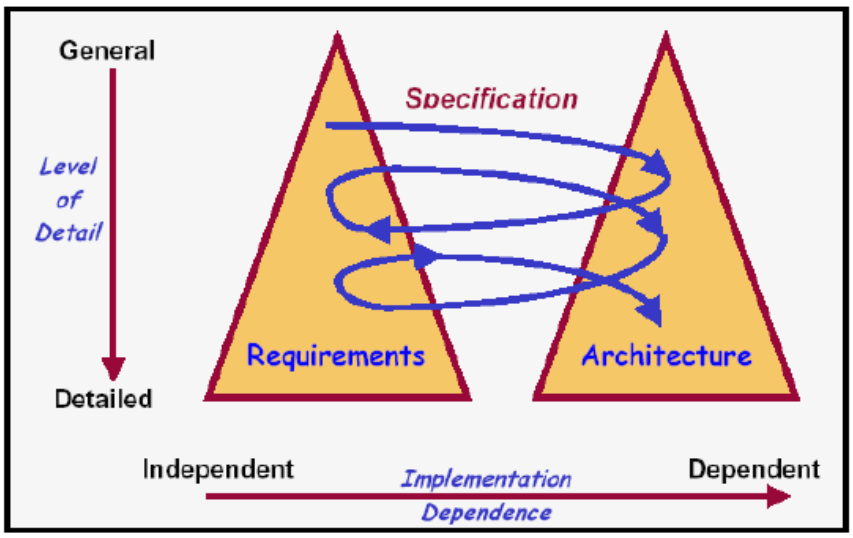
A medida que se pasan por las iteracion, el valor de cada variable se incrementa y de alguna forma se puede confiar en que se esta en algo mejor que antes.

DIAGRAMITA

1.5.1 Unified SW development

1.5.2 Modelo Twin Peaks

Si a esta altura de la carrera no viste la serie Twin Peaks, deberias empezar a bajarla (pero no tiene nada que ver con esto).



El modelo twin peaks es similar al modelo espiral, pero solo se focaliza a los requerimientos y arquitectura. La idea consiste en que hay una ida y vuelta de los requerimientos y la arquitectura. Inicialmente se arracan con los requerimientos, luego con estos requerimientos iniciales se comienza a pensar la arquitectura que seguramente proveera un feedback de los requerimientos. Este feedback es comun que requiera cambios de los requerimientos,que los questione o incluso que proponga nuevos requerimientos (una mala idea proponet nuevos requerimientos al cliente, salvo que haga las cosas mas faciles). A medida que se itera en el modelo twin peaks se genera un nivel de detalle mas rico y se genera un dependencia de la arquitectura que se fue adaptando.

1.6 Relacion entre categorias de requerimientos

La distintcion entre requerimiento funcional y no-funcional no se tiene que tomar en el sentido literal. El limite entre estas dos categorias no siempre es clara. Por ejemplo, muchos requerimientos en un software firewall tienden a ser requerimientos de seguridad.

1.7 Economia

En general los problemas de ingenieria de software son complejos y tienen muchos criterios tanto funcionales (capacidad funcional) como no-funcionales.

2 Modelo Jackson

Algunas definiciones que se tienen que saber para poder entender el modelo. *Recordar* Jackson dice que requerimientos son cosas del mundo y specificacion a las cosas de la interfaz. En la materia eso es a la inversa.

2.1 Definiciones

Fenomeno Es un evento o situacion cuya existencia puede observarse.

Maquina Porcion del sistema a desarrollar o modificar (no necesariamente el soft o hard).
El propositode la maquina esta en el mundo.

Mundo Porcion del mundo afectado por la maquinas

Interfaz Son fenomenos compartidos por el mundo y la maquina.

Asercion Descriptiva Son cosas que son o presumimos verdaderas en el mundo. Dentro de este tipo de asersiones esta el dominio.

Propiedad sobre el sistema que se mantiene apesar de como el sistema se comporta. Tipicamente son leyes naturales o fisicas.

Asercion prescriptiva Cosas que esperamos que sean verdaderas en el mundo.
Dentro de este tipo de asersiones esta los objetivos y los requirimientos.

Propiedad sobre el sistema que se mantienen o no dependiendo como el sistema se comporta.

Una asercion se refiere a un fenomeno que ocurre en el software, en el enterno o en la interfaz. En el ultimo caso, un fenomeno compartido puede ser controlado por el software u observado por el enterno, controlado por el entorno y observado por el software.

Requerimiento Toda asercion prescriptiva que trata sobre fenomenos que afectan la interfaz.

Especificación Toda asersion prescriptiva que trata sobre fenomenos del mundo.

*Jackson:*En el libro de jackson especificación se llama a todo lo de la interfaz y requerimientos todo lo que es del mundo.

Para encontrar requisitos es util revisar la lista de eventos que viven en la interfaz.

2.2 Proposito de la maquina en el mundo

Todo el siguiente chamuyo es para hablar sobre todo el modelo de Jackson en un final. Es ideal leer el paper The World and the Machine , Michael Jackson.

Jackson dice que el proposito de la maquina, que define su valor practico, esta ubicado en el mundo en el cual la maquina sera instalada y usada.
El requerimiento (esto es el problema), esta en el mundo. La maquina es la solucion que se construye.
Es importante saber que el problema no esta en la maquina, la maquina es quien posibilita solucionar el problema. Esto solo es posible porque la maquina interactua con el mundo, y esta interaccion es por medio de la interfaz (osea los fenomenos compartidos).
Es muy importante saber que los requerimientos deben tratar sobre cosas del mundo y usando vocabulario del mundo.

2.3 Requerimientos

Requerimiento del sistema Es una asercion prescriptiva,formulada en terminos del entorno. Deben ser expresados con vocabulario de los stakeholders

Requerimiento del software Son aserciones prescriptivas y deben ser formuladas en terminos de fenomenos entre el software y el entorno. Son lo requerimientos que van a ser usados por los programadores.

Observacion : Un requerimiento de software es por definicion un requerimiento del sistema, pero la vuelta no vale.

Propiedad del dominio Es una asercion sobre el dominio. Se espera que se mantendra invariante sin importar como el sistema se comporte.

Expectativa Es una asercion que va a ser satisfecha por el entorno, y esta formulada con terminos del entorno usando vocabulario de los stakeholders.

Definiciones Son aserciones provistas por el documento de requerimientos. Se utilizan para dar definiciones sobre conceptos y terminos auxiliares para acordar en forma precisa y completa el significado de ciertos terminos.

2.4 Taxonomias

Los requerimientos funcionales y no-funcionales en la taxonomia de requerimientos son los padres de muchas otras. Las taxonomias son utiles como heuristica para encontrar nuevos requerimientos.

Requerimieno Funcional Define los efectos funcionales requeridos por $software - to - be$ debe tener sobre el enterno . Tratan del QUE. Ejemplos, la busqueda de la bibilioteca proveera una lista de libros encontrados en la biblioteca sobre algun titulo.

Requerimieno No-Funcional Definen mas restricciones sobre la forma en el que el software debe satisfacer sus requerimientos funcionales o en la forma que deben ser desarollados. Ejemplos, Comando de aceleracion deben ser enviados cada 3 segundos al tren.

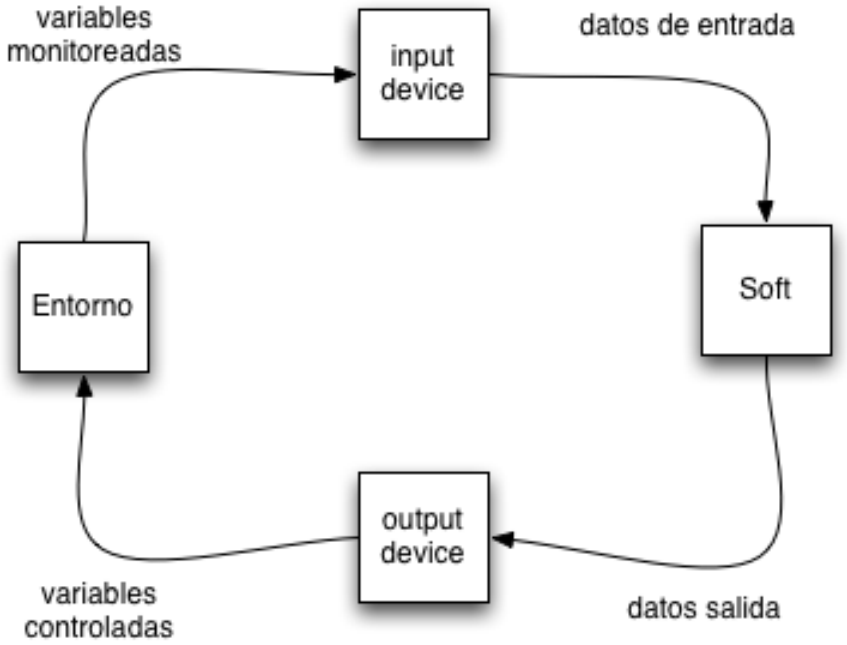
Requerimieno de Calidad Restricciones sobre los efectos del software sobre el entorno para tener adicionales caracteristicas (referidas a calidad).

Algunos problemas del analisis de requisitos:

- Requirimiento faltante
- Presuncion del dominio incorrecta
- Propiedad del dominio cambiante

Objetivos Los objetivos son acerca de los fenomenos en el dominio de la aplicacion.
Un objetivo es una ascersion prescriptiva que el sistema debera satisfacer a traves de la cooperacion de sus agentes.

3 Modelo de 4 variables



El modelo de 4 variables es muy similar el modelo de Jackson pero con la diferencia de que ademas de distinguir lo monitoriable y controlable, tambien agrega los inputs y outputs.

El vinculo entre los conceptos requerimientos de sistema y requerimientos de software puede hacerse mas precisos con las siguientes 4 tipos de variables :

- Variables de monitoreo : Son cantidad del ambiente que el software monitorea por medio de entradas, como sensores
- Variables de control : Son environmental quantitities que el software controla por medio de dispositivos como actuators
- Variables de entrada : Se trata de todos los datos que necesita el software como entrada
- Variables de salida : Son quantities que el software produce como salida

El modelo de las 4 variables define los requerimientos de sistema y software como distintas relaciones matematicas. Sea $\subseteq$ la inclusion de conjuntos, x el producto cartesiano, M las variables de monitoreo, C las variables de control, I las variables de input y O las variables de salida.

Requerimientos de software $SoftReq \subseteq IxO$

Requerimientos de software $SysReq \subseteq MxC$

4 Modelo Objetivos

4.1 Definiciones

Objetivo Es una asercion prescriptiva que el sistema debe satisfacer atravez de la cooperacion de sus agentes.

Agente Un agente es un componente activo que tiene un rol para satisfacer el objetivo.

Requirimiento Son aquellos objetivos que son asignados a solo un agente, y ese agente es el software.

Expectativa Son aquellos objetivos que son asignados a solo un agente, y ese agente es distinto al software o a un agente del entorno.

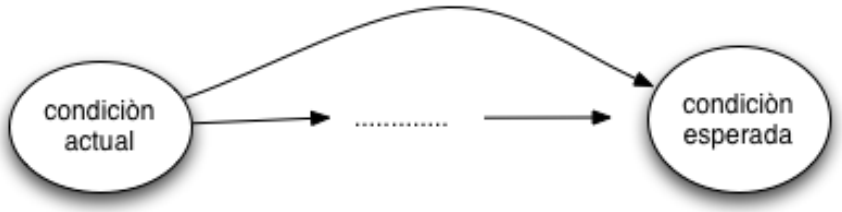
Propiedad de dominio Igual que en el modelo Jackson. asercion descritiva que es satisfecha por el entorno y que son invariantes.

Hipotesis de dominio Asercion descriptiva que sera satisfecha por el entorno y sujeta a cambio. Por ejemplo, Los domingos no son dias para reuniones.

4.2 Taxonomia de objetivos

Existen dos tipos de objetivos, y un objetivo solo puede ser de un tipo. Los tipos de objetivos son los de comportamiento y los blandos. La diferencia es que el de comportamiento tiene algun criterio de aceptacion, mientras que los blandos sirven para comparaciones. Los objetivos de comportamiento suelen tener subtipos : lograr,mantener/evitar.

4.2.1 Objetivos Lograr

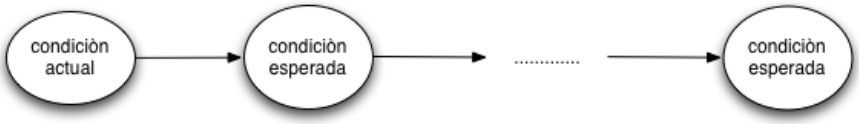


Prescriben comportamiento esperarados cuando alguna condicion esperada que tarde o temprano se cumple si otras condiciones se mantienen en el sistema.

$$si condicion Actual entonces$$

¡¿ condicion esperada

4.2.2 Objetivos Mantener



Prescriben comportamiento esperado cuando alguna condicion esperada siempre debe mantenerse [si condicionActual entonces] siempre buenaCondicion

4.2.3 Objetivos Blandos

Prescriben preferencias sobre alternativas sobre comportamientos del sistema. Estos objetivos suelen usarse para seleccionar opciones del sistema entre multiples opciones. Los objetivos blandos no $no - funcionales$, los nofuncionales suelen ser mas explicitos y responden si o no.

4.2.4 Objetivos funcionales vs. no-funcionales

funcional states the intent underpinning some system service

no-funcional states some quality or constraint on service provision or system development.

4.3 Refinamientos de objetivos

Una vez que se tienen los objetivos, se puede armar el grafo de y/o refinamientos. La forma de armarlo, es desde lo mas general hasta que el objetivo pueda ser asignado a un solo agente. (si es trivial se pueden dejar juntos). Cuando un objetivo es asignado a un agente que es la maquina eso es un requerimiento. Una expectativa es cuando el objetivo es asignado a un agente que no es la maquina. La forma de leer los nives es : cuando se esta en el nivel n: ver el nivel n+1 es ver porque y cuando se mira el n-1 es como.

4.4 Asignacion de responsabilidades

Un objetivo es realizable por un agente si el agente:

- puede monitorear los fenomenos
- el agente puede controlodar los fenomenos
- no es necesario conocer el futuro para garantizar el objetivo en el presente

4.5 Patrones de Refinamientos

4.5.1 Y-refinamiento

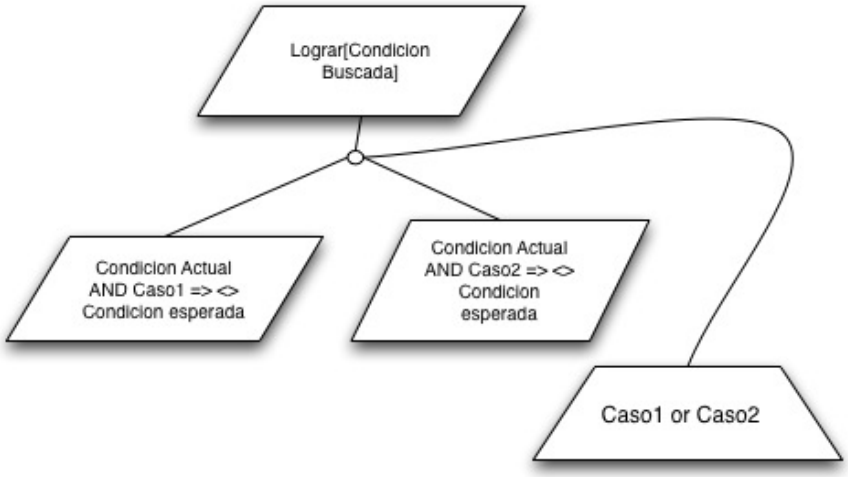
Los $Y - refinamientos$ deben cumplir :

- Completitud : $G_1 \wedge G_2 \wedge \ldots \wedge G_n \wedge Dominio \vdash G$
- Minimilidad : $G_1 \wedge G_2 \wedge \ldots \wedge G_{i-1} \wedge G_{i+1} \wedge \ldots \wedge G_n \wedge Dominio \nvdash G$
- Consistencia : $G_1 \wedge G_2 \wedge \ldots \wedge G_n \wedge Dominio \nvdash false$

4.5.2 O-refinamiento

4.5.3 Refinamiento por casos

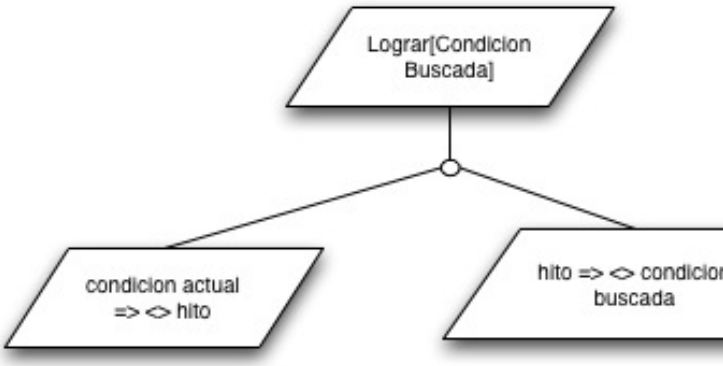
Consiste en refinar un objetivo cuando hay propiedades del dominio disjuntas. Estas propiedades son justamente los casos y cada caso debe ser completo. Que un caso sea completo significa que implica la condicion esperada del



objetivo de mas alto nivel. Es muy facil confundir este refinamiento con un $o-refinamiento$. Los $o-refinamientos$ introducen sistemas alternativos. Si uno usa un $O-refinamiento$ para cada caso lo que estaria logrando es un refinamiento incompleto de objetivos. Ejemplo :

4.5.4 Refinamiento por hitos

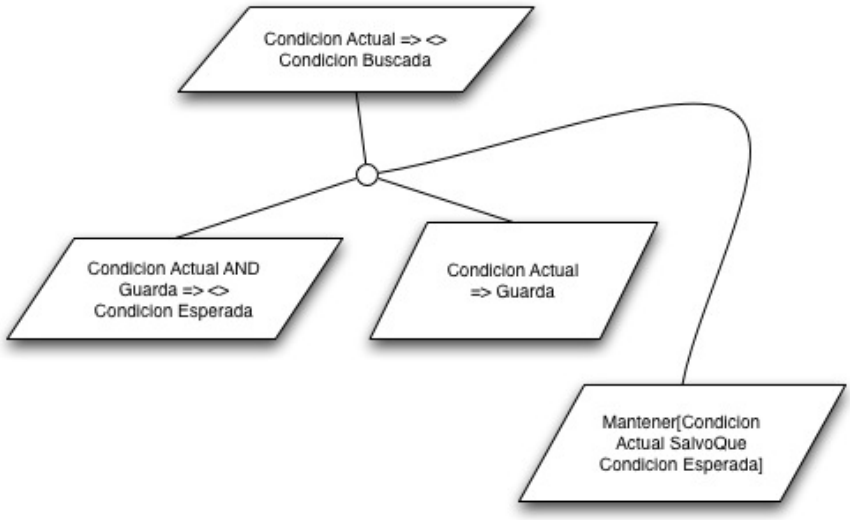
Es un refinamiento aplicable a objetivos donde una condicion intermedia puede ser identificada como un hito para lograr la condicion final. Ejemplo : Lograr[Ambulancia movilizada] se refina con Lograr[Ambulancia Asignada] y por



Lograr[Ambulancia asignada =¿ ¡¿ Ambulancia movilizada]

4.5.5 Refinamiento Introduccion de guarda

Es un refinamiento que consiste en refinar un objetivo del tipo Lograr[CondicionEsperada si CondicionActual] con



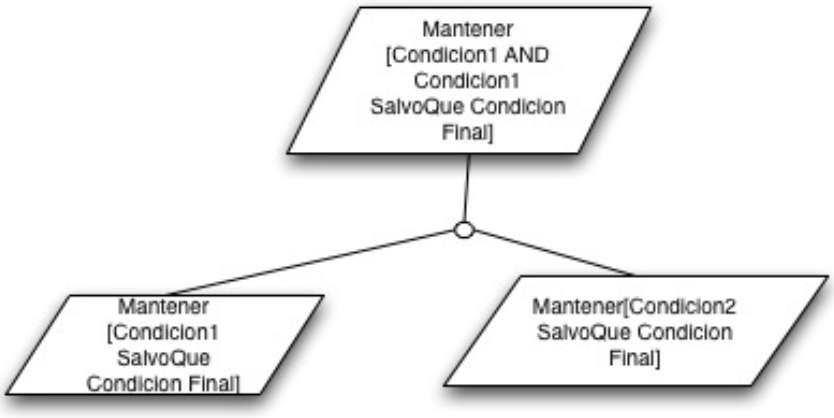
los siguientes tres subobjetvios :

- Lograr[CondicionEsperada si CondicionActual y Guarda]
- Lograr[Guarda si CondicionActual]
- Mantener[CondicionActual SalvoQue CondicionEsperada]

4.5.6 Refinamiento Divide and Conquer

Es un refinamiento para objetivos Mantener bastante simple Ejemplo : Mantener[Condicion1 y Condicion2 SalvoQue CondicionFinal] se refina en :

- Mantener[Condicion1 SalvoQue CondicionFinal]
- Mantener[Condicion2 SalvoQue CondicionFinal]



4.5.7 Refinamiento para lograr objetivos asignables

Hay dos tipos de monitoriabilidad y contrabilidad. Para las dos variantes se trata de forma similar. Ejemplo : Objetivo En CondicionUnMonitoriable se refina en Objetivo En CondicionMonitoriable y en CondicionMonitoriable sii CondicionUnMonitoriable. El segundo refinamiento que parece una Contradiccion suele ser una hipotesis de dominio y en un ejemplo se entiende que quiere decir. Por ejemplo VelocidadDistintaDeCeroMedida sii VelocidadDistintaDeCero.

4.6 Obstaculos

Los procesos de ingenieria de requerimientos usualmente caen bajo el problema de que los objetivos, requerimientos o asunciones son demasiado ideales. Como son demasiado ideales no son satisfechos por el sistema por comportamientos inesperados de los agentes . Como consecuencia el software desarollado no cumplira correctamente con el proposito y no sera lo suficientemente robusto. El modelo de objetivos tiene o se propuso algunas tecnicas para desidealizar los objetivos, esto es o quiere decir que de alguna forma se encuentran propiedades , asunciones, etc que hacen notar que el objetivo es demasiado ideal y por consecuencia se deberia buscar un objetivos, requerimiento, etc menos ideal y mas realista. Al tener un objetivo menos ideal (desidelizacion del objetivo) se llegaria a un software mas robusto. Con los objetivos ideales suele ocurrir que existen situaciones en donde no puede ser satisfechos por alguna expectativa ideal, requerimiento ideal,etc. Ademas la idealizacion de objetivos suele traer inconsistencias entre la especificacion y su actual comportamiento.

Asi como la completitud se puede dar en terminos de subobjetivos, tambien se la puede definir como la negacion de obstaculos. Sean $O_1, O_2, \ldots, O_n$ obstaculos :
$\{\neg O_1, \neg O_2, \ldots, Dom \vdash G\}$

4.7 Semantica

Un objetivo define un conjunto de comportamientos, donde los comportamientos son una secuencia temporal de estados.

5 Modelo Agentes

Agente Componente activo del sistema con algun rol en la satisfaccion de un objetivo.

En este modelo se suele utilizar como herramienta el diagrama de contexto. El diagrama de contexto esta relacionado con los fenomenos de Jackson. Es muy util para mostrar las relaciones entre los agentes, la interaccion de los agente con la maquina y tambien para conocer algunos fenomenos relevantes del mundo que son parte del sistema (pero no interactuan con la maquina).

Para armar el diagrama de contexto es util :

- primero buscar fenomenos que se cree relevantes
- cuando se tienen los fenomenos categorizar cada uno de estos segun sea maquina, interfaz o mundo (segun Jackson).
- despues se arma el diagrama con los fenomenos segun la interaccion que tengan los agentes.

5.1 Relacion con otros modelos

5.1.1 Objetivos

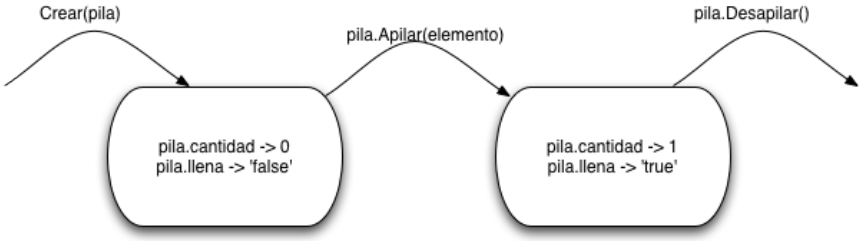
En cuanto al modelo de objetivos, lo que se hace es refinar los de alto nivel hasta llegar a los objetivos hojas que son aquellos que pueden ser asginados a un solo agente. En objetivos cuando una hoja es asignada a un agente

distinto de la maquina , se dice que el objetivo es una expectativa. Cuando es asignado al software , se dice que es un requerimiento.

6 Modelo operacional

Este modelo cubre el aspecto del QUE. El foco del modelo es sobre los servicios que van a brindar la funcionalidad requeridad para satisfacer los objetivos.

Operacion Una operacion es una relacion binario sobre los estados del sistema. Cada operacion tiene una tupla de variables de entrada y de salida,estaspuede solaparse.
Las variables de entrada son aquellas en las que las operacion aplican a. Las variables de salida son aquellas donde las operacion actuan en. Formalmente se puede definir como un conjunto Op de pares de $input - output$ tal que $OpincluidoenInputStatexOutputState$.



6.1 Casos de uso

Definicion Un caso de uso especifica una o varias secuencias de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.

No todos los actores que participan del caso de uso estan en todos los escenarios.
Los casos de uso son de suma utilidad para definir el alcance del software/máquina. Facilita la validacion con los stakeholders.

- Estructura el conjunto de operaciones atendiendo a la categoria de usuario
- que participan en el mismo. Describen bajo la forma de acciones y reacciones
- las operaciones provistas por una maquina desde el punto de viste del usuario. Solo se concentran en funcionalidad proviste por la maquina a construir.
- Solo interesan interacciones directas Maquina-Agente

Los nombres se expresan en gerundio. Sirven para definir el alcance del software/maquina a construir. Facilita la validacion con los stakeholders. Un caso de uso describe un conjunto de escenarios posibles. hasta cuanto detallar?

- Hasta cubrir la funcionalidad mas relevante/critica
- hasta que ya no es efectivo (cost vs. beneficio)

6.1.1 Semantica CU

Un Caso de uso denota un conjunto de escenarios.

6.1.2 Actores

Un actor representa un tipo de usuario,pero no necesariamente es una relacion 1 a 1. Abstrae el usuario real. El nombre describe el rol desempa;ado.

Vision Semantica Un actor denota un conjunto de agentes concretos.Un agente concreto puede ser modelado por varios actores distintos. La clave de entender la semantica consiste en que un agente puede tener varios roles, osea un empleado (el agente) puede ser un vendedor o un administrador, esto no tiene que ver con herencia en casos de uso (o por ahi si) tiene que ver mas con la trazabilidad del modelo de agentes y el modelo operacional.

6.1.3 Relacion Participa En

A participa en U $/leftrightarrow$ la descripcion detallada de U hace referencia explicita al actor A como participante de la interaccion.

6.1.4 Generalizacion

La semantica mas abstracta de herencia es subconjunto.
Si X hereda de Y entonces $X \subseteq Y$ Ademas,

- X es una especializacion de Y.
- Y es una generalizacion de X.
- X es hijo de Y, Y es padre de X
- X es sub-... de Y, Y es super-... de X

6.1.5 Entes Abstractos

6.1.6 Inclusion o Usa a

Un escenario del caso de uso A incluye tambien el comportamiento descrito por el caso de uso B.
Ejemplo , comiendo chocolate —incluye—¿ abriend chocolate. Vision Semantica :
Si un escenario s es donatado por A, entonces existe una porcion de s que contiene un escenario denotado por B.
Puede existir escenarios denotados por B que no aparecen en escenarios denotados por A

6.1.7 Elementos Foraneos

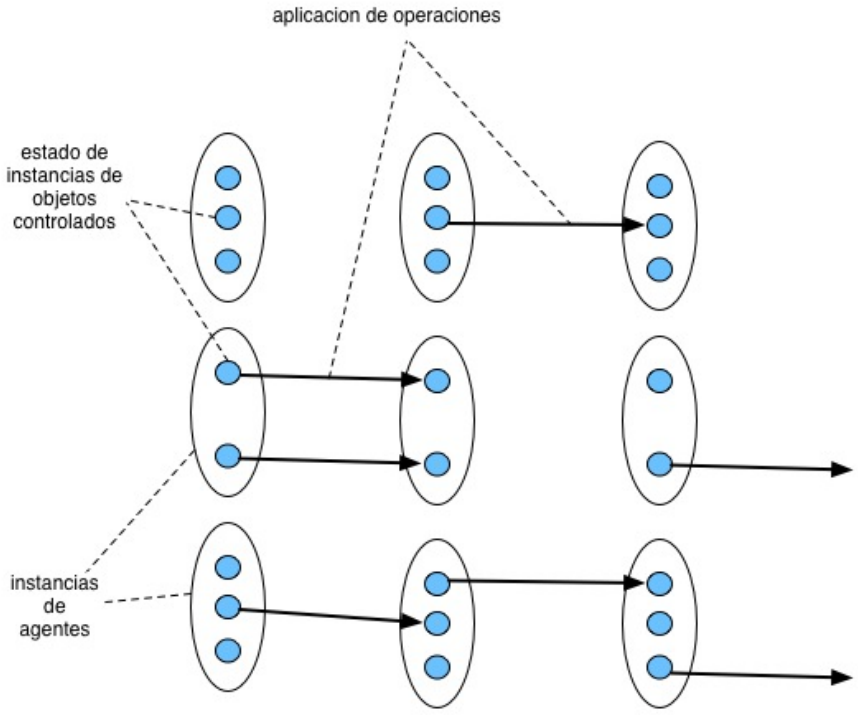
6.1.8 Extensiones

Una instancia del caso de uso A incluye,a veces, el comportamiento descrito por el caso de uso B.
Ejemplo Cenando ¡ — extiende — tomando cafe. Visión semantica:
Existe almenos un escenario s denotado por A, que contiene un escenario denotado por B.

6.2 Relacion con el modelo de objetivos (operacionalizacion de objs)

Operacionalización refiere el proceso de mapear hojas del arbol de objetivos a operaciones que aseguran estos objetivos. Una hoja de un arbol se dice operacionalidad por un conjunto de operaciones si la especificacion de estas operaciones aseguran que el objetivo es satisfecho. Una buena forma de comprobar trazabilidad con el modelo de objetivos es ver si se cubrieron todas las hojas con el modelo operacional. Si alguna hoja no se contemplo, algo seguramente esta faltando.

6.3 Semantica : Objetivos, agentes,objetos y operaciones



- Los agentes corren concurrentemente.
- No siempre se aplican operacion.
- Las operaciones son atomicas.
- Las operaciones se pueden aplicar concurrentemente, tanto dentro como fuera de los agentes en los objetos.

6.4 Modelo Objetos

7 Modelo Conceptual

El modelo conceptual explica cuales y como se relacion los conceptos relevantes en la descripcion del problema.

Objeto conceptual Denota una entidad o concepto del dominio del problema

Clase conceptual Denota un conjunto de objetos conceptuales que comparten caracteristicas comunes. Estas caracteristicas pueden ser atributos o relaciones.

7.1 Propiedades : Atributos y enlaces

Propiedad Representan caracteristicas estructurales de una clase. Las propiedades son un solo concepto,pero aparecen en distintas notaciones : atributos y asociaciones. Aunque en el diagrama se representan de forma distinta, son en realidad casi lo mismo.

Atributo Es una caracteristica de un objeto,es independiente de otros objetos. Tiene un nombre y un rango posible de valores.

En cada instante de tiempo, cada atributo de un objeto tiene un valor unico. Puede ser mutables o inmutables y describen el estado del objeto. *Enlace* Es una caracteristica que vincula conceptualmente a varios objetos. Casa objeto juega un rol conceptual en ese vinculo.

7.2 Diagramas de clases

7.2.1 Estrategias para identificar clases

- Identificar frases nominales en descripciones del dominio (sustantivo o un conjunto de palabras que actual como tal).
- Utilizar lista de categorias de clases conceptuales .

7.2.2 Asociaciones

La asociacion expresa una conexión bidireccional entre objetos. Una asociación es una abstracción de la relacioón existente en los enlaces entre los objetos. Multiplicidades :

- 1 Un elemento relacionado
- $0 \dots 1$ Uno o ningun elemento relacionado
- $0 \dots *$ Varios elementos relacionados o ninguno
- $1 \dots *$ Varios elementos relacionados, pero almenos uno
- $M \dots N$ entre M y N elementos relacionados

7.2.3 Visión Semántica

Un diagrama de clase define : Conjuntos de objetos, Relaciones entre elementos de conjuntos, restricciones sobre conjuntos y relaciones.

7.2.4 Roles

Son los labels que se ponen al extremo de una asociacion que explica la relación entre conceptos en un sentido particular. Cada asociación tiene dos roles; cada rol es una dirección en la asociación. Permiten navegar entre conceptos, por ejemplo Todo los automoviles que conduce el piloto X.

7.2.5 Agregación

Es una asociación especial, una relación del tipo $parte - de$ dentro de la cual una o más clases son partes de un conjunto. Es una forma de decir que algo partenece a un conjunto. Una buena forma de ver si esta bien usar agregación es preguntando, esta bien que viva el objecto cuando el otro no existe?. Por ejemplo en una recion Club/Persona. Cuando el club se destruye esta bien destruir a la persona o la dejamos? Lo mas dificil es establecer la diferencia entre asociacion y agregación. El rombo se ubica del lado del conjunto.Por ejemplo jugador/equipo, el rombo se pone de lado del equipo

7.2.6 Composición

La composición es una forma *fuerte* de agregación. Se diferencian en :

- En la composición tanto el tyodo como las partes tienen el mismo ciclo de vida.

- Un objeto puede pertenecer solamente a una composición.

Sirve para modelar que cuando el objeto relacionado muere , el otro tambien muere (depende donde este el rombo). Tambien sirve que el componente tenga un solo due;o. El rombo va del lado que no es componente. Por ejemplo un parrafo es un componente de un capitulo, el rombo se pone en el capitulo

7.2.7 Asociación n-arias

Son asociaciónes que se establecen entre más de dos clases.
Una clase puede aparecer varias veces desepanando distintos roles. Semanticamente introducen relaciones n-arias : R incluido en AvionesxPasajerosxPilotos.

7.2.8 Clases de asociación

Semantica : Modelan caracteristica de una asociacion que son independientes de las clases que asocia. Se C es una clase de asociacion para la asociacion R en (AxB), entonces introduce funcion $f : R \leftarrow G$. Esto garantiza que no hay dos c, para un mismo par (a,b). No tiene sentido la clase por si sola.

Permiten agregar atributos, operaciones a las asociaciones. La clase de asociacion agrega una restriccion extra,en la cual solo puede existir una sola instancia de la clase de asociacion entre dos objetos participando.

Para entender bien la clase de asociacion hay que compararla contra otra forma similar de representarla.

La principal diferencia consiste en que la cantidad de asociaciones posible para un objeto de la clase A y otro objeto de la clase C cuando se usa clase de asociacion es que solo hay una mientras que en la otra podrian tenerse mas de una (para las mismas instancias).

Una forma de crear un historial es usando la segunda forma.

7.2.9 Generalización

Sirve para modelar cuando una clase hereda los atributos y relaciones de la otra. Las subclases pueden incorporar nuevos atributos o relaciones que las superclases no tienen. *Herencia :* es una relación entre una clase general y una versión más específica de dicha clase. Semanticamente se podria decir que es una relacion de inclusion o es un tipo espacial de la clase.

Un principio en cuando al codigo cuando se usa la herencia consiste en la sustituibilidad , en cualquier lugar donde el codigo indique que requiere algo de la superclase, cualquier otra clase que herede de esta podra sustituirse sin problemas.

7.2.10 Clase Abstracta

Una clase abstracta es aquella que no puede ser instanciada directamente. La unica forma de instanciarla es por medio de una subclase.

7.3 Diagramas de objetos

Es el mundo en una instante dado. La relacion entre los objetos se corresponde con la de sus clases.
Instanciacion : Un objeto es una instancia de una clase. Un diagrama de objetos es una instancia de un diagrama de clases.

7.4 OCL

7.4.1 Colecciones

Una navegacion simple de una asociacion resulta en un *Set*, navegaciones combinadas en una *Bag*, y una navegacion entre asociaciones con {*ordered*} resultan e *OrderedSet*. *OCL* distingue tres tipos de colecciones:

- *Set* Es un conjunto matematica, no tiene duplicados
- *Bag* Es un conjunto pero que puede tener repetidos. Se puede tranformar a *Set* usando $-> asSet()$
- *Sequence* Es como una *Bag* pero con orden.

Operaciones :

- *Select* Especifica un subconjunto del conjunto, donde la expresion dado es Verdadera.
- *Reject* Es igual a *Select* con la diferencia que devuelve todo lo que la expresion da Falso
- *Collect* Sirve para especificar una coleccion que se deriva de alguna coleccion pero tiene objetos distintos.Por ejemplo, sirve para traer en una coleccion todas las fechas de cumplea;os de empleados. Importante : cuando el origen es un *Set* la coleccion resultante es una *Bag*. Por ejemplo cuando hay dos empleados con misma fecha, resultaran en elementos distintos de un *Bag*. La notacion se simplifica self.employee-¿collect(BirthDate)-¿asSet() como self.employee.birthdate.
- $forAll$ Se utiliza para poner una restriccion sobre una coleccion.
- *exists* Similar a $forAll$ pero existe.
- *iterate* Es una generalizacion de las anteriores (permite escribir las anteriores).

7.5 LTS

Scope control,concurrencia,dinámica. Poder computacional limitado.

Usos especificación de par de dominios reactivos o autónomos, computacion reactiva, protocolos, arquitecturas de software.

LTS : Labelled Transition System (Sistema de transiciones etiquetadas).

LTS Sea *Estados* el universo de estados, *Act* el universo de acciones observables, y $Act_T = Act \subseteq \{r\}$. Un *LTS* es una tupla $P = (S, L, triangle, s_0)$, donde $S \subseteq Estados$ es un conjunto finito, $L \subseteq Act_T$ es un conjunto de etiquetas, $triangle \subseteq (SxLxS)$ es un conjunto de transiciones etiquetadas, y $s_0 \in S$ es el estado inicial. Definimos el $alfabeto$ de comunicacion de P como $\alpha P = L - \{r\}$.

Concurrencia Procesamiento logicamente simultaneo. No implica multiples unidades de procesamiento.

Paralelismo Procesamiento fisicamente simultaneo. Involucra multiples UPs

Ejecucion Sea $P = (S, L, \triangle, q)$ una *LTS*. Una ejecucion de P es una secuencia $w = q_0 l_0 q_1 \ldots$ de estados q_i y labels $l_i \in L$ talque $q_0 = q$ y $(q_i, l_i, q_{i+1} \in \triangle$ para todo $0 \leq i \leq |w/2|$

Proyeccion Sea w una palabra $w_0w_1w_2w_3\ldots$ y A un alfabeto. La proyeccion de w en A,la cual se denota $W|_A$, es el resultado de eliminar de w todos los elementos w_i tal que $w_i \notin A$

Traza Sea P un LTS. Una palabra w sobre el alfabeto $\alpha(P)$ es una traza de P si hay alguna ejecucion de w' de P tal que $w = w'|_{\alpha(P)}$. Notar que las traza no incluyen las acciones no observables. Tambien se denotan con $tr(P) = \{w|wesunatrazadeP\}$.

7.5.1 Modelado de procesos

Un proceso es la execucion secuencial de un programa. El estado de un proceso en cualquier punto del tiempo consiste en el valor de las variables. Un programa cambia de estado, por medio de transiciones. Cada transicion hace que el programa cambie de un estado a otro. El orden en la cual deben ocurrir las acciones es determinada por un grafo de transiciones, el cual es una representacion abstracta del programa.

7.5.2 Modelado : Concurrencia

El primer problema del modelado de la concurrencia lo trae la velocidad de ejecucion de un proceso en comparacion con otro. Para solucionar este problema se considera que las velocidad son arbitrarias,esto significa que un proceso puede tomar tiempo arbitrario para pasar de una accion a otra. La desventaja es que pierde propiedades real-time, sin embargo tiene la ventaja de independencia de scheduling lo que implica portabilidad de los programas concurrentes.

La forma de modelar concurrencia es usando interleaving. En un proceso las acciones son ejecutadas en orden. sin embargo como los tiempos son arbitrarios, las acciones de diferentes procesos son arbitrariamente intercaladas (interleaved). Lo bueno de tener las acciones interlaveadas (intercaladas,interleaved actions,etc) es que es un buen modelo de abstraccion de la concurrencia porque no dice nada (abstrae) sobre como las cosas (usualmente procesadores) se deben intercalar. Esto resulta en un modelo de ejecucion asincronico e independiente de scheduling. Se dice que una accion a es concurrente con otra accion b si permiten que las acciones se ejecuten en cualquier orden $a \to b$ o $b \to a$.

La composición en paralelo de todos los procesos resultan en una maquina que genera todos las posibles trazas de acciones interleaved.

La forma de modelar interaccion entre procesos es por label comportidos.

7.5.3 Modelando : Eleccion

Ejemplo de la maquina de cafe, donde se puede elegir segun el boton que se presiono si se quiere cafe o te.

En este ejemplo en particular la eleccion de la transicion la realiza el enterno. Pero puede ser realizada internamente en el proceso.

7.5.4 Modelando : Eleccion no deterministica

Cuando dos o mas maquinas tienen mismos labels, se dicen que son compartidas. Estas labels compartidas se utilizan para modelar interacciones entre las maquinas. Una accion compartida implica que al mismo tiempo todos los procesos que la comparten deben ejecutar tal accion. VER CAP3

Sincronizacion de dos tareas, sin importar el orden de ejecucion de las dos tareas

Ejemplo productor consumidor, con buffer tama;o 1.

Ejemplo de handshake.

Ejemplo de dos productores, que no importa el orden en que producan cosas. Lo que importa es que los dos producieron el producto.

Ejemplo de recurso compartido.

Ejemplo cliente servidor

7.5.5 Equivalencias

Equivalencia Sea X un conjunto. Una relacion binaria R es un subconjunto de XxX. R se dice equivalencia si cumple :

- R es reflexiva : esto es que $x\ R\ x$ para cada $x \in R$
- R es simetrica : estos es que $x\ R\ y$ implica $y\ R\ x$ para todo par $(x, y) \in X$
- R es transitiva : esto es que $x\ R\ y$ y $y\ R\ z$ implica $x\ R\ z$ para todo $x, y, z \in R$

Equivalencia llevado a maquinas de estado :

- Reflexividad : cada proceso es una correcta implementacion de si mismo
- Congruencia : Dos procesos equivalentes en comportamiento pueden ser intercambiados sin afectar el comportamiento final.
- Transitividad : Soporta derivacion de implementaciones apartir de especificaciones.

Cuando se busca una equivalencia se espera que cumpla : transitividad, reflexibidad y simetria. Que sea abstracta en cuanto al numero de estados, estrucutra del modelo y trazas.

7.5.6 Equivalencia por Trazas

Propuesta : dos LTS son equivalentes $\leftrightarrow$ si definen el mismo conjunto de trazas Problema : La equivalencia por trazas no puede distinguier al no-determinismo y el determinismo. La equivalencia por trazas es demasiado debil. Se podria decir que la equivalencia por trazas no considera las capacidades de comunicacion de los estados intermedios, esto es algo que importa tener en cuenta. Por ejemplo lo que no ve la equivalencia de trazas es que despues de ingresar una moneda en la maquina expendedora en la no-deterministica puede resultar cafe o te, mientras que en la deterministica solo una de las dos es posible.

Deadlock : La equivalencia de trazas puede exihibir distintos comportamientos de deadlock cuando interactuan en paralelo con otros procesos.

7.5.7 Equivalencia isomorfismo

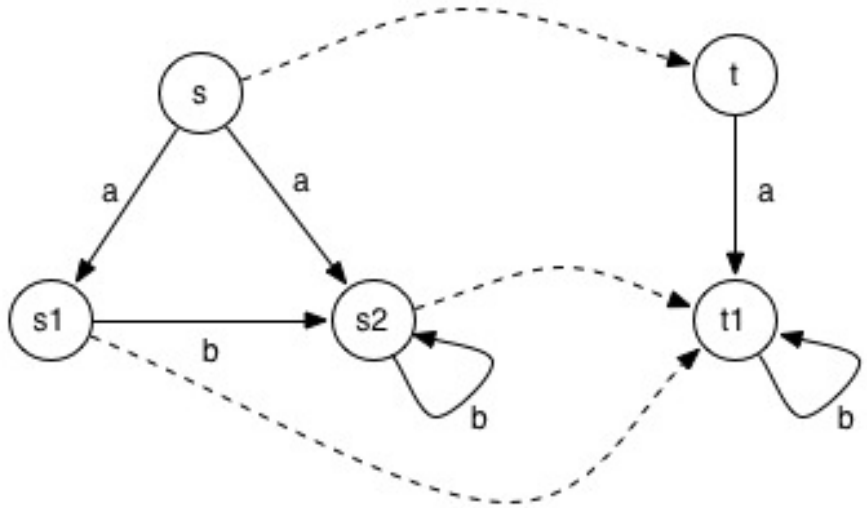
Propuesta : dos LTS son equivalentes $\leftrightarrow$ al quitar sus estados no alcanzables son isomorfos Problema : la cantidad de estados no deberia ser un factor de comparacion, hay LTS que son iguales pero difieren la cantidad de estados.

7.5.8 Bisimulacion

Definicion intuitiva : Dos procesos son equivalente si tienen las mismas trazas y ademas los estados que pueden alcanzar son iguales. No tiene los problemas de trazas, ni la de isomorfismo.

Teorema 7.1 $(P\ Q)$ *si y solo si paa cada accion* $a \in Act$ *:*

Ejemplo de dos maquinas bisimilares.



$$R = \{(s,t),(s_1,t_1),(s_2,t_1)\}$$

en principio no parecen bisimulares, porque parece que hay un no-determismo. Bueno el tema es que la primera maquina no esta minimizada y por esto los estados s_1 y s_2 son equivalentes.

7.5.9 Chequeo bisimulación

Sea R la relacion a chequear
Para cada elemento en R , chequear si todas las posibles transiciones desde los dos estados puede ser matcheadas por las correspondientes transiciones en los otros estados.

7.5.10 Mostrar no-bisimilitud

Existen almenos dos formas de mostrar la no-bisimilitud:

- Enumerar todas las relaciones binarias y mostrar que ninguna contiene $(s_1\ t_2)$ y es una bisimulacion fuere.Esto es muy costoso.

- Realizar algunas observaciones que nos permiten eliminar relaciones candidatas a bisimulacion en un paso. Es decir usar un juego que caracteriza bisimiliridad fuerte.

El juego consiste en que para dos LTS P y Q hay dos jugadores un atacante y un defensor. La idea es que el defensor quiere mostrar bisimiliridad, mientras que el atacante lo contrario. El juego sirve para mostrar la no-bisimiliridad, ya que cuando el defensor gana es cuando el juego es infinito. El juego consiste en que el atacante desde su estado elige una trancision y luego el defensor debe responder haciendo una transicion por la misma etiqueta. Se dice que es fuertemente bisimilar que el defensor gana si y solo si el defensor tiene un estrategia ganadora universal empezando desde la ocnfiguracion (P,Q). Se dice que no es fuertemente bisimilar si el atacante tiene estrategia ganadora universal.

7.5.11 Bisimiliridad Debil

Intuicion : Si P hace a, entonces Q lo puede imitar haciendo algunos tau, un a y mas tau.

Es una relacion de equivalencia. EL juego que la caracteriza es igual al anterior, pero el defensor puede hacer jugadas. Bisimiliridad fuerte implica debil. Es congruencia respecto a ||.

7.5.12 Congruencia

Se dice que una equivalencia es una congruencia si y solo si $P \equiv Q$ implica que $C(P) \equiv C(Q)$

7.5.13 Semanticas LTS

Las LTS tienen dos semanticas lineal y arborea. La linea es la semantica de las trazas, la que sirve para ciertos casos concretos. La arborea esta dada por la bisimulacion, por ejemplo la composicion paralelos

7.5.14 Relación con otros modelos

7.6 State Charts

7.7 SDL flowchart

7.8 Redes Petri

Es un grafo dirigido bipartito de places y transiciones. Los ejes tienen asociados pesos.

preset Son places que alimentan a una transicion

poset Son places que son alimentados por una transicion

marking Es una asignacion de tokens a places

Una transicion puede dispararse cuando la cantidad de tokens en su preset superan los correspondientes valores anotados en los ejes. Cuando se dispara se elimina la cantidad de tokens correspondientes al valor del eje y se agrega en los posets esa cantidad.

marking alcanzable Es una secuencia de transiciones que resulta en ese marking desde el marking inicial.

N-safe Se dice que la red es N-safe si no es alcanzable ningun marking que contenga mas de N tokens en un place.

7.8.1 Semantica

Sistema de transicion u ordenes parciales. Con sistema de transicion se refiere a la misma nocion de maquinas de estado, osea estado inicial y movimientos entre asignaciones de tokens. Las redes petri son simultaneas, es por esto que tiene una semantica de orden parcial ya que si ciertas asignaciones de preset permiten moverse al mismo tiempo podria no existir un orden que diga cual debe realizarse primero.

7.9 MSC

7.10 Diagramas de actividad

Los diagramas de actividad sirven para describir procesos logicos, procesos del negocio y flujo de trabajo. En los diagramas los nodos son llamados acciones, no actividades. Una *actividad* se refiere a una secuencia de acciones,por lo tanto un diagrama muestra una actividad construida por medio de acciones.

Deisición : Tiene una sola entrada de flujo y varias salidas de flujo. Cada salida tiene una guarda. Cada vez que se tiene que realizar una desición solo se puede salir por una guarda.

Merge : Se utiliza para juntar multiples flujos de entrada con una sola salida. Un merge marca el fin de un comportamiento condicional iniciado por una desicion.

7.10.1 Particiones

Las particiones se usan para mostrar quien realiza las acciones del diagrama.

8 Modelo Comportamiento

El modelo intenta capturar en forma general todos los comportamientos que ocurren luego de aplicar una operacion o luego de un evento.

8.1 Maquinas de estado

Los diagramas de maquina de estado son una tecnica para describir el comportamiento de un sistema.

Las maquinas de estado capturan las clases de agentes de comportamiento necesarios en terminos de estados y transiciones de eventos.

En los escenarios los comportamientos suelen estar implicitos, ademas los escenarios no son generales. Las maquinas de estado capturan en forma general y explicita el comportamiento de cualquier instancia de un agente. Cuando se modela con maquinas de estado, se realiza intentado capturar el todos los comportamientos posible de un solo agente. La solucion general del sistema es realizando la composición en paralelo de todas las maquinas.

Lo bueno de las maquinas de estado es que pueden ser validadas atra vez de animaciones y pueden ser verificados por medio de propiedades declarativas. Las maquinas de estado proveen una buena base para la generacion de codigo. El punto malo de las maquinas de estado es que son muy operacionales en las etapas mas tempranas de la etapa de la elaboracion de requerimientos. Su elaboracion puede ser dificultosa.

8.1.1 Definiciones

Estrella $A*$ es el conjunto de todas las trazas finitias, incluyendo $\in$ que puede ser formadas con los simbolos del conjunto A.
Cuando las trazas som restringidas a A, permanecen sin cambios. Esto permite una definicion simple :
$A* = \{s|A = s\}$

after Si $s \in traces(P)$ entonces
P/s $(Pafters)$ es un proceso el cual se comporta igual que P se comporta desde de que P ejecuto todas las acciones grabadas en la traza s.

Ejemplo :
$\alpha VMS = \{coin, choc\}$ maquina $= coin \rightarrow (choc \rightarrow VMS)$ $VMS/coin = (choc \rightarrow VMS)$

En la notacion arborea P/s denota un subarbol.

9 Testing

Testing Es el proceso de ejecutar un producto para :

- Verificar que satisface los requerimientos
- Identificar diferencias entre el comportamiento real y el comportamiento esperado.

9.1 Validacion vs. Verificacion

Validacion La validacion trata sobre la correctitud del requerimiento ante el problema. Estamos construyendo el producto correcto? basada en el uso de modelos.

Verificacion La verificacion trata sobre la correctitud de la solucion contra el requerimiento.El producto esta cumpliendo la especificacion? Estamos construyendo correctamente el producto?

9.2 Aspectos de los errores

Falla Diferencia entre los resultados esperados y los reales

Defecto Esta en el texto del programa, una especificacion, un dise;o, y desde alli se hace visible la falla

Error Un error es una equivacion humana. Un error lleva a uno o mas defectos, que estan presentes en un producto de software.

Un defecto lleva a cero, uno o mas fallas: la manifestacion del defecto.

9.2.1 Verificacion : Estatica vs. Dinamica

Estatica Trata con el análisis de una representacion estática del sistema para descubrir problemas. Tecnicas : Inspecciones,Revisiones,Walktrhrough, Analisis de reglas sintacticas, Analisis Data Flow, Model checking, Theorem Proving.

Dinamica Trata con ejecutar y observar el comportamiento de un producto. Tecnicas : Testing, Run-Time Monitoring, Run-Time Verification.

Testing Verificacion dinamica de la adecuacion del sistema a los requerimientos (de distinto tipo). Es el proceso de ejecurar un producto para verificar que satisface los requerimientos, identificar diferencias entre el comportamiento real y el esperado.

Oraculo Es quien conoce el comportamiento esperado del programa. Puede ser una persona.

Los oraculos tienen problemas : visibilidad y comparacion. Se suele asumir que se conoce el resultado esperado, que puede ejecutarse y que ademas los resultados son comparables.Esto tiene sentido de asumir en muchas situaciones, pero a veces pasa que no se puede asumir.

9.3 Test : Requerimientos no funcionales

- Test de seguridad : validando disponibilidad, integridad y confidencialidad de datos y servicios
- Test de performance : validando los tiempos de acceso y respuesta al sistema
- Test de Stress : Validando el uso del sistema en sus limites de capacidad y verificando sus reacciones mas alla de los mismos
- Test Usabilidad

9.4 Testing Random

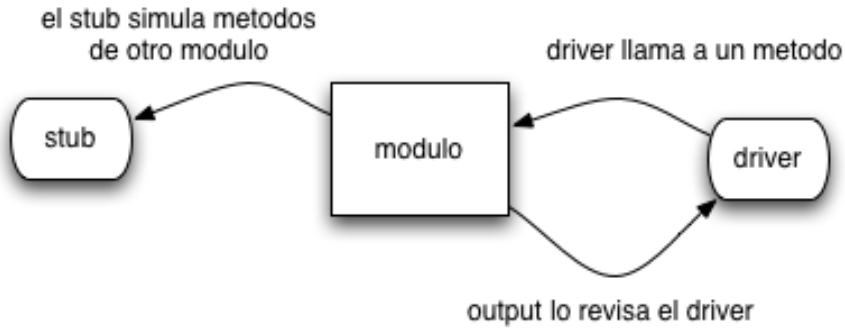
Saber si el testing random es adecuado o no, requiere un previo analisis de la especificacion. Se podria decir que si los parametros tienen relaciones muy complejas, podria complicarse hacer el generador de instancias random para que cubran la especificacion.
Igual el verdadero problema es que los errores podrian no tener una distribucion uniforme.

9.5 Niveles de test

Test de aceptación Es realizado por el cliente.

Test Sistema (o subsistema) Es un testeo sobre todo el sistema.

9.6 Test Integracion



Test Integracion Es un test orientado a verificar que las partes de un sistema que funcionan aisladamente, tambien lo hacen en conjunto. En el test de integracion se testea la interaccion y la comunicacion entre partes (no es lo mismo que test de sistema)

En los test de integracion suelen ser utiles los stub y drivers, ya que para poder testear la integracion a veces ocurre que en ciertas etapas la otra parte no esta terminada.

Stub Simula subprograma que no esta implementado. Se utiliza en tecnicas top-down, ya que lo de abajo no esta implementado es necesario simulargo. El tiempo de desarrollo es mayor, ya que no es trivial desarollar los stubs.

Driver Simula las llamadas y tiene los valores de output. Se utiliza cuando el desarrollo es del tipo bottom-up. Exige un esfuerzo considerable de programacion, al igual que los stubs (creo que los stubs es peor si el driver ya viene armado, p.e. Junit creo que podria decirse que es un driver).

test de modulo o de unidad Es el testing que se realiza sobre un solo modulo

9.6.1 Estrategias de test de integracion

- Sistema organizado jerarquicamente : Top-down,bottom-up o una combinacion de ambos
- Sistemas batch : por partes del flow de corrida
- Sistemas sin jerarquia (objetos) : Es libre, salvo por el orden de desarollo

Test Unidad Es un test que se realiza sobre una unidad de codigo peque;a, claramente definida. Una unidad suele ser : Una clase, un programa, una funcion o procedimientos, un form , un script, un subsistema.

9.6.2 Modelo de ciclo de vida en V

El modelo de ciclo de vida de un software, es una forma de estructurar las etapas del desarrollo de software teniendo en cuenta el tiempo y la abstraccion.

9.7 Testing Funcional

El testing funcional consiste en revisar que un programa implementa correctamente a una funcion f. Tambien, se revisa que se hace enforma razonable testeando condiciones de error y ademas deberia avisar que dada una entrada no valida esta no forma parte del dominio de f.

9.7.1 Particion Categorias

Es una estrategia que divide el dominio del input en subconjuntos (no necesariamente una particion). La particion de categorias tiene como hipotesis que una muestra de cada uno de los subconjuntos alcanza para probar todo el subconjunto (no es verdad, pero parte de esa hipotesis).

$SD_C(P, S)$ $SD_C(P, S)$ = Conjunto no vacio, que es el subdominio definido por un criterio para el programa P y una especificacion S.

Caso de test Especificacion de subdominio y resultado esperado.

Dato de test Valores concretos de los paramatros para ejecutar un caso de test

Test Suite Conjunto de datos de test con los que se testea el programa

Exito Si P es correcto para todo elemento del Test suite, se dice que T es exitoso para P.

9.8 Criterio

Que es un criterio?

9.8.1 Criterios Ideales

Consistente Un criterio C se dice que es consistente para $P \leftrightarrow$ para todo par T_1, T_2 de test sets que satisfacen C, T_1 es exitoso para $P \leftrightarrow T_2$ tambien lo es

Si un criterio es consistente cualquier test set provee la misma informacion.

Completo Un criterio C es completo para $P \leftrightarrow$ si P es incorrecto entonces hay un test set T que no es exitoso para P

Si un criterio es completo se podria decir que se podria probar la correctitud del programa.

*Completo y consistente :*Identifica un test set ideal y permite probar correctitud. Aunque en general esto no parece posible, en testing de sistemas reactivos bajos ciertos supuestos restrictivos es posible tener un criterio completo y consistente.

9.9 Tecnica particion de dominio (Ostrand y M. Balcer)

Heuristica

- 1 - Elegir una funcionalidad que pueda testearse en forma independiente.
- 2 - Determinar sus parámetros u otros objetos del ambiente que pueden afectar su funcionamiento
- 3 - Determinar las caracteristicas relevantes de cada objeto determinado en el punto 2 y de la relacion de los objetos y el output.
- 4 - Determinar elecciones (choices) para cada caracteristica de cada objetos.
- 5 - Clasificacion : errores,unicos,restricciones.
- 6 - Armado de casos.

9.10 Identificacion Categorias

9.10.1 Heuristica 1

Si una condicion de input especificaca un rango de valores (intervalo), identificar una clase valida y dos invalidas: En general se usan las clases : valida, invalidas (fuera de rango para cada lado) y las de borde.

9.10.2 Heuristica 2

Si una condicion de input especifica un número de vaores, identificar una clase válida y dos invalidas.

9.10.3 Heuristica 3

Si una condicion de input especifica un conjunto de valores, y hay razones para pensar que cada uno es manejado por el programa en forma distinta, identificar una clase por cada elemento y una clase invalida

9.10.4 Heuristica 4

Si una condicion'on de input especifica una situación que debe ocurrir, identificar una clase válida y una clase inválida. Por ejemplo, se espera que los puntos formen un triangulo. Una clase valida seria los puntos forman un triangulo. La clase invalida, los puntos no forman un triangulo.

9.10.5 Heuristica 5

Clases invalidas. Una condicion de input acepta un tipo tipo de parametro, probar con otro tipo de parametros. Por ejemplo un input probar ingresar un numero cuando la condicion de input es alfabetico.

9.10.6 Heuristica 6

Tipos de datos de input y output.

9.10.7 Heuristica 7

Cardinalidad del modelo de datos. Cardinalidad minima, cardinalidad maxima.

9.10.8 Heuristica 8

Ciclo de vida de las entidades.

- Condiciones de borde : Los casos de test que exploran los bordes de las clases de equivalencia producen mejor resultado
-

Heuristica

- Si una condicion especifica un rango de valores (intervalo(, identificar una clase valida y dos invalidas
- Si una condicion de input especifica una situacion debe ocurrir, identificar una clase valida y una clase invalida
- Si una condicion de input especifica un conjunto de valores, y hay razones para pensar que cada uno es manejado por el prorama de forma distinta, identificar una clase valida por cada elemento y una clase invalida.
- Probar ingreso de valores de otro tipo que la clase
-
- La cardinalidad de las relaciones define reglas del negocia que deben ser probadas.
- Cardinalidad minima : Cuantas instancias hay como minimo de una entida por cada instancia de una entidad relacionada con ella
- cardinalidad maxima : Cuantas instancias hay como maximo de una entidad por cada instancia de una entidad relacionada con ella
- Tener una vision temporal de las entidad. (?)
-

Tecnicas combinatorias

9.11 Notacion arbórea

9.12 Tecnicas Combinatorias

9.12.1 Grafo Causa-Efecto

Es una heuristica para generar todos los outputs admisibles. La heuristica consiste en buscar las condiciones *or* o *and* entre parametros y combinarlars tal que :
La heuristica tiene complejidad $O(n * k * o)$,donde n es la cantidad de categorias binarias, k la profundidad del DAG y o la cantidad de combinaciones del output validas (las de error se consideran validas).

- Primero Separar las causas y efectos. Las causas se eligen de forma similar a category-partition. Los efectos son los outputs validos.
- Si hay un *or* se consideran todas las opciones con una se;al en true
- Si hay un *and* se consideran todas con solo una en falso

9.12.2 2-wise partition y arreglos ortogonales

Es una tecnica para generacion de casos de test que tiene como supuesto que, los inputs son enumerados, errores son del tipo sngle o double mode. Los pares de parametros que se seleccionan son independientes. Los arreglos ortogonales tienen las caracteristica de que tomando cualquier par de columnas se tienen todas las posibles combinaciones entre dos parametros. Es particularmente util para testing de integracion. Tambien es sumamente util para testing de combinaciones de opciones de configuracion (como una pagina web donde el usuario puede tener distintos sistemas operativos y definiciones).

- Corridas : Es la cantidad de filas
- Factores : Es la cantidad de columnas
- Levels : Es la cantidad de valores de los factores. Usualmente se lo llama choices.
- Strength : Numero de columnas que toma ver $Levels^{Strength}$ posibilidades la misma cantidad de veces.

Notacion $L_{runs}(Levels^{Factores})$

9.13 Testing Estructural

flowgraph Grafo que representa el flujo de control de un programa

Camino Completo Camino que se inicia desde el nodo donde comienza el programa hasta el nodo asociado a la terminacion del programa.

Cada camino del flowgraph corresponde a una ejecucion? No necesariamente, podrian existir un camino pero que alguna parte de este no sea alcanzable Y los caminos completos? Tampoco, podria existir un camino completo (inicio a fin) pero que en el medio tenga un camino no alcanzable nunca. Por ejemplo por alguna condicion, como un if false como parte del camino.

Camino no factible Es un camino para el cual no existen inputs del programa que fuerce su ejecucion.

El problema de los caminos no factibles es que existe codigo no alcanzable nunca, y es un indicio de algo raro.

9.14 Criterios

9.14.1 Cubrimientos de sentencias

9.14.2 Cubrimiento de deciciones (o branches)

9.14.3 Cubrimiento de condiciones

9.14.4 MC/DC

9.14.5 def-use flowgraph

Es un flowgraph que tiene un tipo especial de anotacion :
Sean P un programa y una variable x de P:

- por cada definicion de x, el nodo asociado esta etiquetado con un definicion de x.
- por cada $c-uso$, el nodo asociado esta etiquetado con un uso de x.
- por cada $p-uso$ todos los arcos salientes del nodo asociado estan etiquetados con un uso de x.

DUA (definition-use association) Es una terna (d, u, x) tal que :

- La variable x esta definida en el nodo d
- la variable x se usa en el nodo u
- hay almenos un camino desde d hasta u que no contiene otra definicion de x ademas de la de d (def-clear para x o libre de definiciones para x)

9.14.6 Criterio : Cubrimiento all-uses

Para cada variable en el programa, deben ejecitarse todas las asociaciones entre cada definicion y todo uso de la misma, o cubrir todas las DUAs del programa.

9.14.7 Problema testing estructural

La seleccion de casos no esta basada en el comportamiento funcional, ya que es facil ejecutar todas las instrucciones y sin embargo no invocar ciertas funciones.

Las Tecnicas estructurales se deberian utilizar como criterio de adecuacion, osea como complemento al testing funcional.

9.15 Comparacion de criterios

9.15.1 Subsumicion

Subsumicion Se dice que un criterio C_1 subsume a otro criterio C_2 si para todo test suite T que satisface el criterio C_1, entonces T satisface C_2 (para todo par (P, S)).

Esto quiere decir que si un conjunto de datos satisface *Allpaths* (que subsume *Alldepaths*), entonces el conjuntos de datos satisface B.

9.16 Medidas probabilisticas

La relacion subsumes tiene almenos el problema de que quiza C_2 sea mas preciso que C_1, apesar de que C_1 subsuma a C_2. Para poder explicar el problema se define $M(C, P, S).S$ = Especificacion, P= programa y C = criterio.

$M(C, P, S)$ $M(C, P, S) = 1 - \prod_{i=1}^{n}(1 - \frac{m_i}{d_i}$ donde, d_i = tamano del dominio, m_i = cantidad de inputs que causan falla.

$SD_C(P, S)$ $SD_C(P, S)$ denota los multiconjuntos no vacios de subdominios para los cuales los casos de test satisfacen el criterio C para el programa P y la especificacion S.

9.16.1 Properly Covers

C_1 **properly covers** C_2 Sea $SD_{C1}(P, S) = \{D_1^1, \ldots, D_m^1\}$ y $SD_{C2}(P, S) = \{D_1^2, \ldots, D_n^2\}$.
Se dice que C_1 properly covers C_2 para (P, S) si existe un multiconjunto $M = \{D_{1,1}^1, \ldots, D_{1,k_1}^1, \ldots, D_{n,k1}^1, \ldots, D_{n,k_n}^1\}$ talque M es un sub-multi-conjunto de $SD_{C1}(P, S)$ y :

- $D_1^2 = D_{1,1}^1 \bigcup \ldots \bigcup D_{1,k_1}^1$
- ...
- $D_n^2 = D_{n,1}^1 \bigcup \ldots \bigcup D_{n,k_n}^1$

Cuidado con la definicion, existe una similar que no tiene el properly y tiene problemas. Comunmente se la confunde con esa (es mas intuitiva). Properly covers informalmente dice que si C_1 properly covers C_2 entonces cada subdominio de C_2 puede ser cubierto por subdominios de C_1 (expresados como la union de algunos dominios de C_1). Ademas considera que los subdominios de $SD_{C1}(P, S)$ en el cubrimientos no deben aparecer mas veces que en $SD_{C1}(P, S)$.

10 Testing Sistemas Reactivos

10.1 Testing maquinas de estados

El problema : Dada una maquina de estados que es la especificación, para el cual se tiene su diagrama de transiciones, se quiere testear si una implementación conforma a la espeficación. Esto se suele llamar conformance testing o fault detection. Y la secuencia que resuelve este problema se la llama checking sequence.

Conformance Es una forma de testing para decir si una implementacion conforma a una especificación.

Diferencias con bisimulacion :

- La bisimulacion no tiene encuenta outputs
- La conformance no necesita isoformofismo
- La conformance se hace sobre Mealy machina, las cuales no tienen no-determinismo.

Modelo de fallas : output o transicion equivocada.

10.2 Mealy Machines

Observaciones

- Las $IOLTS$ no admiten no-determinismo. Esto es porque delta y gamma son funciones
- El testing de conformance por sus restricciones hacen que sea un criterio ideal, osea dicen si o no. Esto no pasa casi nunca con testing.

conformance Dada la especificacion de una maquina de estado finita para la cual se tiene el diagrama de transiciones y una implementacion de la cual solo se puede observar IO, queremos saber si la implementacion se ajusta a la especificacion

Mealy Machine Una mealy machine es una tupla $< I, O, S, \Sigma, \lambda >$. Donde I es el conjunto de inputs, O el conjuntos de salida, $\Sigma : SxI \to S$ es la funcion de trancision de estados, $\lambda : SxI \to O$ es la funcion de outputs

checking secuence Dada una maquina de estados A,una checking secuence es un string de inputs que distingue A de todas las otras maquinas de n estados.

La checking secuence es la secuencia de test que garantiza que una implementacion se ajusta a la especificacion.

String Inputs $x = a_1, a_2, \ldots, a_n$ donde $a \in I$, I conjunto de inputs

$\Sigma(estado, inputstring) = estadofinal$ Es una funcion que devuelve el estado final que resulta del string de inputs x

$\lambda(estado, inputstring) = outputstring$ Es una funcion que dado un input string x devuelve el output que produce la maquina.

Para poder dar una equivalencia de Maquinas es necesario primero dar una equivalencia de estados

Equivalencia de estados Dos estados s_i, s_j son equivalentes $\leftrightarrow$ para todo input $x \in I*$ $\lambda(x, s_i) = \lambda(x, s_j)$

Equivalencia de maquina Una maquina A es equivalente a otra maquina B si y solo si para todo estado existe uno equivalente y viceversa.

Observación : Cuando las dos maquinas tienen la misma cantidad de estados, la nocion de equivalencia es la misma que la de isomorfismo. No hay que confundir con el isomorfismo que se propone cuando se ven maquinas de estado, aca hay otros supuestos como mealy machine minimizada.

Maquina minimizada Una maquina de n estados se dice minimizada si y solo no hay dos estados equivalentes en esa misma maquina.

La clase de equivalencia de maquinas, siempre tiene una maquina minimizada y esta es unica. Esta unica maquina minimizada es isomorfa o cualquier otra maquina de la clase minimizada. Se dice minimizada en la cantidad de estados.

Esto de alguna forma propone una familia de maquinas, las cuales se pueden minimizar a una maquina para hacer la comparacion.

Cuando dos FSM son isomorfas =¿ son equivalentes. El contrareciproco no vale, el contra ejemplo es cuando hay estados repetidos (que son equivalentes) en una FSM (o en las dos).

Existe un algoritmo para minimizar maquinas que consiste en particiones los estados segun su output. es $O(pn^2)$.

separating secuence Dados dos estado s_i,s_j una separating secuence x es una secuencia la cual cumple que $\lambda(s_i, x) \neq \lambda(s_j, x)$

Separating Family Una separating family es una coleccion de separating secuences Z_i.

Cada maquina reducida tiene un separating family, que contienen almenos $n-1$ secuencias, cada una de longitud menos o igual a $n - 1$.

Preset Una (preset) secuencia distintiva σ es una tal que para todo par de estados distintos cumple :
s_i, s_k $\lambda(s_i, \sigma) \neq \lambda(s_j, \sigma)$.

10.2.1 Algoritmo TS Chow

Arbol Generador (o en realidad arbol de testing...creo) (Confirmar bien esta definicion) No es el arbol generador tipico de teoria de grafos o no se refiere a eso. Sino al arbol que semantico de desarmar los ciclos de ejecucion. En general se llama camino parcial a un camino que comienza en la raiz y termina en una hoja del arbol de testing

Idea rapida del algoritmo : Armar el arbol de testing, que consiste en un arbol al desdoblar las ejecuciones. Para cada camino parcial, considerar un caso de test (cada camino es x en el algoritmo de abajo). Supuestamente cada x deberia diferenciar a cada estado entre si. Si una piensa en los caminos parciales es parecido a las separating family.

Sea Z_i una familia de conjuntos separadores
Construir un arbol generador(o arbol de testing) de A con raiz en s_1.
Para cada s_i de S_A

PARA CADA X DE Z_i
Resetear B al estado que deberia ser similar a s_1
Moverse usando el camino propuesto pro el arbol de s_1 al supuesto estado s_j
Aplicar x

Para cada transicion no cubierta por el arbol generador hacer algo similar.

11 Testing LTS

LTS Es una tupla $< S, L, \rightarrow, s_0 >$, S conjunto de estados, L labels transiciones, s_0 estado inicial, $\rightarrow$ relacion de transicion

Trazas $Tr(s) = \{\alpha pipe\alpha \in \Sigma ys \rightarrow (por\alpha)ss\}$

Nocion de la definicion de trazas : es el conjuntos de acciones observables que resultan de transitar por el LTS para estar en el estado s.

Alcanzables $Alc(s, \alpha) = \{sspipes \rightarrow ss\}$, en otras palabras dado un string que posibles estados se puede llegar

Determinismo Un LTS es determinista si y solo si para todo estado s y todo $\alpha \in \Sigma$ $count(Alc(s, \alpha) \leq 1$

Un LTS se dice que tiene comportamiento finito si existe un numero natural n tal que para toda traza $tt \in Tr(L)$ se cumple que tt tiene longitud menos a n. Osea todas las trazas tienen una cota en la longitud y es natural.

Preorden trazas $L \leq_{tr} L_{tr}$ sii $Tr(L) \subset Tr(L_{prima})$

Equivalencia de trazas $L =_{tr} L_{prima}$ sii $Tr(L) \subset Tr(L_{prima})$ y $Tr(L_{prima}) \subset Tr(L)$

11.1 Conformance trazas

imp implementa *espec* $\leftrightarrow$ *imp* $\leq_{tr}$ *espec* Se dice que todo el comportamiento de la implementacion es parte del comportamiento especificado

imp implementa *espec* $\leftrightarrow$ *imp* $\geq_{tr}$ *espec* Se dice que la implementacion contempla toda la espeficacion y mas

imp implementa *espec* $\leftrightarrow$ *imp* $=_{tr}$ *espec* Se dice que tanto la implementacion como la especificacion tienen las mismas trazas

12 IOLTS

IOLTS Es una tupla $< S, s_0, \Sigma_I, \Sigma_O, \rightarrow >$, donde S es el conjunto de estadosm s_0 el estado inicial, Σ_I son los labels de input, Σ_O son los labels de output.

Estado quiescente

En otras palabras un estado quiescente es un estado que no tiene ninguna transicion con output. Es comun al principio confundirlo con el grado de salida del nodo, ojo con eso. Hay que ver que sea un label $\mu \in \Sigma$ (normalmente denotado por ! en el label, por ejemplo $te!$.

Sucesor de traza $safter\sigma = \{s' | sRightarrow\sigma s'\}$

son todos los estados que se alcanzan despues de ejecutar la traza, es lo mismo que Alc

Traza quiescente $sigma$ es una traza quiescente de s sii $\exists s' \in (safter\sigma) : \delta(s'))$

Coloquialmente una traza es quiescente sii existe almenos un estado alcanzable que es quisescente.

QTr(s) $QTr(s) = \{\sigma | \sigma esunatrazaquiescentedes\}$

Conjunto de trazas quiescentes de s, osea todas las trazas que tienen almenos un estado quiescente alcanzable.

STr(s) $safter\sigma = \{s' | sRightarrow\sigma s'\} \bigcup \{\delta | \delta(s)\}$

STr se suelen llamar trazas con suspension o trazas de suspension. Se utiliza para ioco y es igual que las trazas normales pero incluyen el δ cuando el estados es quiescente.

out(s) (s minuscula) $out(s) = \{\mu | \mu \in \Sigma_O \wedge s \rightarrow \mu \bigcup \{\delta | \delta(s)\}$

Dado un estado devuelve el conjunto de label de output de ese estado. Si ese estado es quiescente devuelve δ.

out(S) (S mayuscula) $out(S) = \bigcup \{out(s) | s \in S\}$

Conjunto de todos los labels de output de todos los estados.

12.1 Conformance trazas (IOTR)

Se dice que una implementacion es considereda correcta a comparacion de una especificacion si su comportamientpo no contradice el comportamiento especificado.

Comportamiento : en general es expresado en termino de trazas.

Contradiccion : en general una falta de coincidencia del comportamiento observado y el output esperado.

$L \leq_{iotr} L' sii Tr(L) \subset Tr(L') \wedge QTr(L) \subset QTr(L')$ Alternativamente :
$L \leq_{iotr} L' sii \forall \sigma : out(Alc(L, \sigma)) \subset out(Alc(L', \sigma))$ La segunda conformance es lo mismo, pero esta escrito de otra forma. Si uno mira la definicion de out y alcanzabilidad esta haciendo los dos incluye del primero. Si no se entiende esto, hay que volver a revisar las definiciones anteriores y pensar que carajo quieren decir, lo mejor es mirando los ejemplos.

12.2 Limitaciones $\leq$ *iotr*

Una implementacion puede proveer mas operaciones que su especificacion, pero iotr no vale. IOCONF soluciona este problema, si una implementacion hace mas que su especificacion es verdadero. Para solucionar esto se restringe el dominio (no se como llamarlo) de los estados en la defincion. Uno quiere que iotr no de falso cuando se hace una comparacion de una implementacion que hace mas que su especificacion porque hace que las cosas sean mas faciles de modelar. Esto es algo subjetivo, pero es le motivo de que sigan apareciendo mas comparaciones.

12.3 IOCONF

ioconf $Lioconf L' sii \forall \sigma \in Tr(L') : out(Alc(L, \sigma)) \subset out(Alc(L', \sigma))$

ioconf tiene el problema de no ver el no-determinismo. Pero se podria decir que es similar a trazas en LTS o analogo.

12.4 IOCO

IOCO $\forall \sigma \in STr(L') : out(Alc(L, \sigma) \subset out(Alc(L', \sigma))$

Se podria decir que OICO es un refinamiento de IOCONF y considera informacion importante que aportan los estados quiescentes.

13 Preguntas de finales

13.1 Preguntas Sueltas

1 . Como es el testing de arreglos ortogonales?

2 . Que son los requerimietos según jackson?

Segun Jackson los fenomenos se pueden identificar como : mundo,interfaz o maquina. Los fenomenos de ls interfaz son los fenomenos compartidos por el mundo y la maquina, es esto lo que Jackson llama requerimientos.

3. Como relaciona Jackson el problema, los requerimientos y la resolucion?

4. Si tengo que hacer una nueva version del soft que ya tengo, para que me sirve el viejo?

El viejo sirve para conocer los problemas que tiene, ya que la nueva version deberia mejorar al menos uno de estos problemas. Tambien sirve como fuente de conocimiento del dominio.

13.2 20 de nov de 2007

1. Que significa que un criterio de test subsume a otro ? Ejemplifique.

Se dice que C_1 subsume a C_2. Si para todo test set T que satisface el criterio C_1. Si T es exitoso para C_1, tambien lo es para C_2.

Ejemplo : Criterio de testing estructural all-paths subsume a branches. Esto quiere decir que un test set para all-paths que es exitoso tambien lo sera para branches (el mismo caso de test).

2. Suponga un procesimiento foo(int x) que tiene un único defecto que se exhive al ejecutarlo con $x = 0$. Diga si el siguiente criterio de test :

- es completo
- es consistente

Justifique su respuesta.

5. El modelo de Jackson distingue entre aserciones descriptivas y prescriptivas. Explique por qué esta distinción es relevante al momento de hacer verificación y validación del documento de requerimeintos.

Respondido en 2 de marzo de 2010.

6. A qué se refiere la siguiente aserción : El análisis de obstaculos permite desidelaizar objetivos.

Se refiere a que los objetivos a veces suelen ser un grado de idealizacion alto y el analisis de objetivos trata de encontrar situaciones donde estos no se hacen verdaderos. Desidealizar un objetivo permite que este sea mas fuerte. El analisis de obstaculos tambien sirve para la completitud, ya que la negacion de los obstaculos y propiedades del dominio deberia implicar el objetivo de mas alto nivel.

Ejemplo de objetivo muy ideal , Lograr[Ambulancia en lugar de incidente en 15 minutos]. Este objetivo es demasiado ideal, por ejemplo si una ambulancia se rompe en el camino, o si hay trafico hacen que el objetivo no sea verdadero. El que se rompa la ambulancia o el trafico serian los obstaculos. Una posible desidealizacion seria Lograr[Ambulancia en lugar de incidente].

7. Que es una relacion de bisimulacion? Porque adoptarla como criterio de equivalencia en vez de equivalencia por trazas?

Informalmente : una relacion de bisimulacion es una equivalencia y tiene la nocio de que para un observador externo las maquinas son iguales.
Formalmente : Sea Σ el universo de todos los LTS. Sea R una relacion binaria , con $R \subset \Sigma x \Sigma$. R es una bisimulación sii para toda $a \in Act$ vale que :

8. Cual es la diferencia entre un diagrama de flujo de datos (DFD) y un diagrama de actividad?

13.3 27 de nov de 2007

1. Cual es la diferencia entre falla, defecto y error.

- Falla : Es la diferencia entre lo obtenido y lo esperado
- Defecto : Es un error en el texto, especificacion o documento
- Error : Es una equivocacion humana

Un error lleva a una o mas fallas. Un defecto lleva a cero o mas fallas. La falla es la manifestacion del error.

2. Explique que es y para que sirve el def-use graph.

El def-use graph es un flow grafo. En particular el def-use graph es un tipo especial de flow graph que cumple,para cada variable x :

- Cuando hay un def-use ,osea una definicion de x se denota en el nodo u_x.

- Cuando hay un c-use ,osea un uso computacianal de x se denota en el nodo u_x.

- Cuando hay un p-use , osea un uso proposicional de x se denota en el nodo o arista u_x

Ideal poner un ejemplo de while, case, if,etc.

5. Cuales son las diferencias fundamentales entre los sistemas de transicion etiquetados y las redes petri.

Las redes petri tienen asignacion de tokens, mientras que las LTS no tienen esto. Las redes petri funcionan distinto en el sentido de que segun las asginaciones de los places se puede pasar a otra asignacion. Las redes petri pueden pasar tokens si el valor de sus ejes lo permite (y tambien su direccion). Las LTS tienen la idea de $siguen_a$, que intentan modelar como se mueve el sistema (o cambia de estado) segun los eventos o operaciones que se aplican.

7. Segun Jackson, cual es la relacion entre monitorbilidad/controlabilidad y la nocion de requeremiento.

La monitoriabilidad y la controlabilidad tiene que ver con variables relacionadas a cosas (dispositivos segun 4 variables) que se mapean a inputs y outputs del software. La nocion de requerimiento habla sobre la interfaz, que son los fenomenos compartidos por el mundo y la maquina. Entonces lo monitoreable y controlable vive o es parte de la interfaz y juega un papel importante en los requerimientos. El modelo de 4 variables habla sobre sensores o actuadores, pero esto se puede abstraer a otras cosas. El modelo de 4 variables ademas propone una definicion matematica de requerimiento de sistema y requerimientos de software (un requerimiento de software es de sistema, pero no a la inversa). La defincion es la siguiente :
$SysReq \subset CxM$
$SoftReq \subset IxO$

8. Que es el patron de refinamiento por casos. Ejemplificar.

Es una forma de refinar patrones que se basa en propiedades del dominio disjuntas.Estas propiedades disjuntas son los casos. Estos casos deben ser siempre alguno verdadero y completos. Hay que tener cuidado de no confundir con O-refinamiento o con Y-refinamiento. O-refinamiento trata sobre distintas alternativas para refinar un objetivo mas abstracto. El Y-refinamiento introduce casos complementarios. Si uno usa mal el refinamiento por casos y hace un o-refinamiento se estaria con un refinamiento incompleto, ya que en ciertas situaciones el refinamiento no responderia o no diria QUE hacer cuando se cumple alguna propiedad dle dominio.

13.4 25 de febrero de 2008

1. Cual es la diferencia entre verificacion y validacion? Ejemplofique.

Validación : Trata sobre si se esta haciendo el producto correcto. Tiene el objetivo de incrementar la confianza en el producto que se esta desarrollando. Se podria decir que es la comparacion de la especificacion o requerimientos contra el mundo real. En la tarea de validacion deberia existir un experto de dominio para poder dar la aceptacion.

Verificacion : Trata sobre si se esta haciendo el producto correctamente. Usualmente se compara la implementacion contra la especificacion.

Ejemplo de validacion : Ruedas giran ¡=¿ Sensores Envian pulso. Para poder validar es necesario consultar a un experto de dominio, para saber si es correcto. Un buen experto de dominio deberia comentar un problema muy conocido cuando la pista esta mojada.

Ejemplo verificacion : Testing Funcional o model based testing. Testear una funcion y verificar que los resultados obtenidos son los correctos.

2. Explique que modelan las cajas y las flechas en un modelo de contexto.

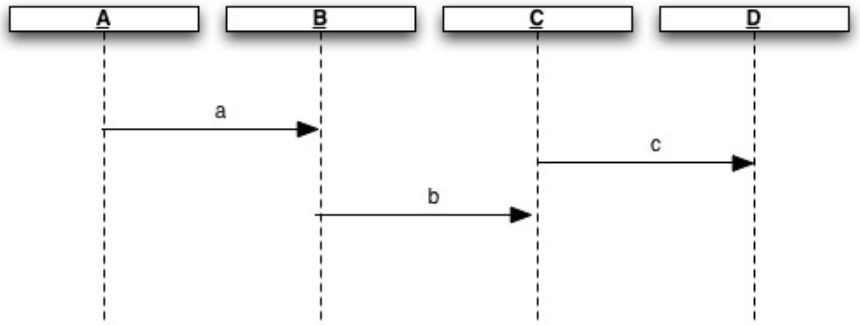
Espero que me pregunten esta en el final :).

3. Cuales son los tres tipos basicos de relaciones entre entidades en un modelo conceptual. Explique sus diferencias.

Aca explicar asociacion, agregacion y composicion.

4. Explique que significa que los Diagramas de secuencia tengan una semantica de ordenes parciales.

En pocas palabras significa que la semantica es de secuencia de eventos. Algunos eventos o mensajes tienen restricciones entre ellos, pero por ejemplo en el grafico siguiente entre a y c no hay orden y permite tener las siguientes secuencias de eventos que no contradicen el diagrama : a, c, b,c, a, b. El diagrama dice que no puede ocurrir b si todavia no ocurrio a.



5. Cuales es la diferencia entre verificacion Dinamica y Estatica? De ejemplos concretos de tecnicas para ambas categorias.

Verificacion Dinamica : Consiste en la ejecucion del producto para verificar su correcto funcionamiento. Tecnicas : Testing,chequeos runtime (tipo valgrind deteccion memory leaks,etc).

Verificaicon Estatica : Consiste en el analisis de la representacion estatica del producto. Tecnicas : Inspecciones, Revisiones, Analisis de reglas sintacticas, Analisis data flow, Prueba de teoremas.

6. que son y cual es el rol de stubs y drivers en testing?

Stubs : Se utilizan en testing de integracion cuando hay una relacion de jerarquia (por ejemplo modulos) de lo que se quiere testeas. Se necesitan los stubs cuando se empieza el testing top-down y los stubs simulan subprogramas los cuales no estan todavia implementados o no se tienen.

Drivers : Se utilizan en testing de integracion ideam anterior pero cuando es bottom-up. Los drivers simlan llamadas y tambien comparan la salida esperada. Esto es necesario porque lo de arriba (usualmente modulos) todavia no estan disponibles.

Drivers y Stubs incrementan el trabajo a realizar y suele incrementar el costo.

13.5 4 de marzo de 2008

1. Explique que es un stakeholder en el contexto de Ingenieria de Requerimientos.

Un stakeholder es un grupo de individuous que se ven afectados por el sistema a construir. Suelen tener un papel importante a la hora de la aceptacion y son una fuente importante de informacion en la ingenieria de requerimientos.

2. A que nos referimos cuando dedimos que los requerimientos, tanto funcionales como no funcionales, deben ser refutables de manera objetiva. Ejemplifique.

Quiere decir que los requerimientos tienen que estar escritos de alguna forma que den lugar a su refutabilidad, ya que si no fuesen refutables no tendrian sentido. En general todos los modelos siempre algun problema van a tener, esto esta relacionado con la refutabilidad de los requerimientos y tampoco quiere decir que son incorrectos, sino que dan a lugar a su refutabilidad por ejemplo buscando un contra ejemplo (se dicen que son refutables porque se puede intentar o dan lugar a buscar contra ejemplo).

Ejemplo , los aviones Las ruedas giran ¡=¿ avion sobre la pista. Un contra ejemplo es el accidente que se dio cuando la pista estaba mojada y la ruedas no giraron.

(ver pagina 166 del libro de Jackson)

3. Explique la diferencia entre lenguajes de especificación de comportamiento basados en interacciones y estados. De ejemplos de notaciones concretas de ambas categorias.

Los de interaccion describen escenarios de intercambios de mensajes, mientras que los basados en estado muestran las acciones que deben ocurrir para cambiar de estado. Los de interaccion tienen semantica de orden parcial y los de estado tienen la lineal y arborea. La mas importante me parece es que los basado en estado modelan todos los comportamientos posibles, mientras que los de interaccion no lo hacen. Esto es el motivo de los implied scenarios.
Ademas los de interaccion podrian no describir ciertas situaciones, osea son lenguajes limitados y los de estados no es asi (creo).

Ejemplos de notaciones :

- Basado en estados : LTS,Maquinas de estados
- Basado en interacciones : Diagramas de secuencia.

4. A que nos referimos cuando hablamos de relaciones derivadas en modelos conceptuales?

Son relaciones que no estan almacenadas y puede obtenerse en base a otras relaciones existentes en el modelo.

5. Que es y para que sirve (en el contexto de testing) un flow graph

Ya preguntado.

6. De tres heuristicas que pueden utilizarse para la particion del dominio en categorias en el contexto de generacion de casos de test.

Heuristica 1 : Cuando un input descrive un intervalo, dar 2 categorias validas y una invalida. Por ejemplo dentro de rango, bardes y la invalida fuera de rango.
Heuristica 2 : Cuando una condicion de input descrive un conjunto de valores y hay alguna razon para pensar que el programa se comportara de forma distinta. Dar una categoria por valor y una invalida. Por ejemplo Camion, Moto y Auto Cada una es categoria y dar otra categoria que no sea ninguno de los tres.
Heuristica 3 : Cuando una condicion de input tiene que cumplir cierta condicion dar una valida y otra invalida. Por ejemplo, tres puntos que deben formar un triangulo. Dar una categoria donde formen un triangulo y otra que no.

13.6 7 de abril de 2008

A que se refiere la siguiente asercion : El analisis de obstaculos permite des-idealizar obstaculos.

Ya preguntado

Indique si la siguiente asercion es verdadera o falsa, justifique su respuesta.
La elaboracion del modelo de objetivos se realiza de manera top-down, es decir refinando los objetivos de alto nivel en objetivos de mas bajo nivel.

Falso, aunque es comun refinar los objetivos y que sea top-down. En realidad se puede refinar hacia arriba haciendo la pregunta porque.

A que nos referimos con que los LTS tienen un semantica arborea. Ejemplofique.

Se refiere a la semantica que tienen los LTS respecto a la bisimulacion. Ejemplo que esta en las diapos de la teorica.

Que significa que los diagramas de secuencia tengan semantica de ordenes parciales. Ejemplofique.

Ya preguntado.

Explique la diferencia entre falle,defecto y error. Explique que rol juega cada uno de estos terminos en testing.

Explique que es un grafo de causa-efecto y para que sirve en el contexto de testing.

13.7 10 de junio de 2008

1 . Que relacion existe entre objetivos blandos y no funcionales? Ejemplifique.

Son de clasificacion ortogonal. Se podria decir que tratan sobre restricciones de comportamientos funcionales. Los no funcionales, responden si o no mientras que los blandos sirven para comparar alternativas.

No-funcional : La respuesta al comando debe ser menor a 50 milisegundos.
Objetivo blando : La respuesta al comando debe ser rapida (Minimizar[tiempo de respuesta al comando])

2. Explique los riesgos asociados a la definicion de alcance de ingenieria de Requerimientos en terminos de la relacion que existe entre sintoma y problema mas en general. Ejemplifique.

3. Explique la nocion de herencia y sus usos en los diagramas de casos de uso.

La nocion de herencia es que si X hereda de Y entonces X es un caso especial de Y. Una semantica posible de herencia es la relacion subconjunto. Si X hereda de Y entonces X incluido en Y. El uso en los casos de uso :

- Abstraccion : Mediante la herencia es posible eliminar algun caso de uso (o juntarlos mejor dicho) y la descripcion del caso de uso es mas abstracta.
- 'Tipo especial de actor : permite hablar de un tipo especial de actor, que puede hacer mas cosas que su

generalizacion.

- Entes Abstractos : Si todo elemento de Y pertenece a alguna de las especializacion $X_1, X_2, \ldots, X_n$ se puede decir que Y es abstracto (no tiene ninguna instancia). Sirve para introducir conceptos relevantes.

4. Indique si la siguiente asercion es verdadera o falsa,justifique su respuestra. La composicion en paralelo de LTS permite modelar la ocurrencia simultanea de eventos.

Se podria decir que es verdadero. Pero la concurrencia de LTS se modela en realidad con el interleave de transiciones entre procesos (o entre cada LTS). Cada proceso individual tiene ejecucion secuencial. El tiempo que puede transcurrir una maquina en un estado es arbitrario. La composicion permite armar una maquina que es la que tiene el interleave de transiciones de las maquinas, pero respeta la secuencialidad. Para sincronizar procesos se utilizan label o transiciones con mismo nombre.

5. En que se diferencia el test de integracion del de sistema y del de unidad. Mencione y ejemplifique alguna estrategia de integracion.

Test de integracion : es aquel que testea las interacciones entre los componentes. Test de sistema : es aquel test que se realiza sobre el sistema cuando esta completamente realizar, si el stakeholder forma parte del test se lo suele llamar test de aceptacion. Test de unidad : es un test que se aplica a una peque;a porcion del software, por ejemplo una funcion, una metodo, un procedimiento, una clase.

Estrategias de integracion :
Estrategia Jerarquica : Cuando hay jerarquias (usualmente con modulos), se utilizan tecnicas top-down,bottom-up o mixta. En estas entran en juego los stub y drivers. Los drivers se utilizan para tecnicar bottom-up, ya que simulan las llamadas a los subprogramas y tambien verifican la salida. Los stubs se utilizan para tecnicas top-down y simulan subprogramas. Tanto el driver como los stubs consumen tiempo de desarrollo importante.

Estrategia Libre : Usualmente se da cuando se utiliza programacion orientada a objetos. Estrategia batch de procesamiento secuencial : Se separan por partes del flow de corrida

6. Explique el uso de arreglos ortogonales para testing. Indique la relacion entre fuerza,niveles y factores.

La tecnica de arreglos ortogonales consiste en elegir 2 parametros (la 2-wise) independientes y con estos dos armar todas las Combinaciones. Esto es valido bajo el supuesto de que los parametros tienen una cantidad finita de choices. El uso en testing es para reducir la cantidad (o la explosion combinatoria) de casos de test. Factor : columnas que son los parametros. nivel : es la cantidad de eleccion que tiene cada columna (choices). fuerza : Cantidad de columnas tales que las $X = nivel^f uerza$ posibilidades aparecen la misma cantidad de veces.

Esta tecnica es ideal cuando hay parametros de configuracion. Como por ejemplo un sistema que corre sobre distintos sistemas operativos, distintas resoluciones y procesadores. Quiza un ejemplo podria ser un web, que corre bajo PCs, celulares,etc.

Tambien se utiliza en programacion orientada a objetos para testear metodos cuando hay una herencia compleja.

13.8 24 de julio de 2008

1. En el contexto de modelado de objetivos, que significa que un objetivo sea de mas alto nivel que otro?
Se dice que un objetivo es de mas alto nivel que otro cuando es mas abstracto. Ademas los subobjetivos con los que se refino el objetivo de mas alto nivel (el mas abstracto) tienen mas detalles (menos abstracto). Osea que al ser de mas alto nivel habla menos sobre el problema y es mas general. Tambien otro paremetro importante es que los objetivos de alto nivel son multiagente, a diferencia de los de mas bajo nivel (las hojas) que son uni-agente (salvo en casos triviales). Una forma de saber si un objetivo es uniagente es cuando el agente o software puede monitorear o controlar algo respecto al objetivo. Tambien cuando puede satisfacerlo.

2. Segun Jackson, cual es la relacion entre monitorbilidad/controlabilidad y la nocion de requerimientos?

Repetida!

3. Asumiendo que en el modelo conceptual no se asignan operaciones a clases conceptuales, como se diferencia la nocion de herencia de un diagrama de clases en el contexto de modelado conceptual y dise;o

Dise;o no se toma mas, calculo que esto no se preguntaria

4. Cual es la diferencia entre un diagrama de flujo de datos (DFD) y un diagrama de actividad.

Los DFD representan el flujo de datos entre componentes de un sistemas. Cada nodo es un componente del sistema y las transicion representa el flujo de datos. Mientras que los diagramas de actividad se utilizan para representar en los nodos actividades y los ejes se utilizan para representar la secuencia o lo que sigue a la siguiente actividad.

5. Que es un oraculo y como se usa en testing?
Un oraculo puede ser un humano o cualquier otra cosa que conozca el resultado correcto de lo que se esta testeando. Se utiliza para comparar la salida de lo que se testea contra el resultado correcto,osea para definir el resultado del test (exitoso o fallido). 6. Que significa que un criterio de test subsume a otro ? Ejemplifique

Sea C_1 y C_2 dos criterios. Se dice que C_1 subsume a C_2 cuando, para un conjunto de datos de test T que fue satisface el criterio C_1 , entonces T tambien satisface el criterio C_2. Ejemplo, en el testing estructural el criterio $all - paths$ subsume a $branch$.

13.9 15 octubre 2008

1. En el contexto de modelado de objetivos, que significa que un objetivo sea de mas alto nivel que otro?
Pregunta repetida 2. Como refina el modelo de objetivos la taxnomia de aserciones de Jackson?
El modelo de objetivos refina las taxonomias de las aserciones en dos ramas principales que son las funcionales y las no-funcionales. Las funcionales son aquellas que definen como se comporta el requerimientos con el entorno. Las no-Funcionales son restricciones sobre el como los requerimientos funcionales satisfacen los requerimientos funcionales o en la forma que deben ser desarollados. 3. En redes de Petri, explique que es un Marking, un marking alcanzable y que una red sea N-safe.
Un marking es una asignacion de token a un place.
Un marking se dice que es alcanzable cuando existe una secuencia de transiciones desde el marking inicial que resulta de un marking (el alcanzable).
Un marking se dice N-safe cuando no es posible alcanzar ningun marking que contengan mas de N tokens.
4. Que es la relacion de bisimulacion? porque adoptarla como criterio de equivalencia en vez de equivalencia por trazas?

Es una relacion de equivalencia que permite comparar LTS. Informalmente se podria decir que intenta capturar que las dos LTS se mueven hacia el mismo estado, dando la sensacion de que las maquinas son equivalente para un observador. Depende, pero la bisimulacion es mejor criterio que la de trazas al momento de detectar no-determinismo por ejemplo, algo que trazas no puede ver.

5. Que significa que un criterio de test subsume a otro? Ejemplofique Repetida 6. Explique que es un grafo de causa-efecto y para que sirve en el contexto de testing.

Un grafo de causa efecto es una tecnica de generacion de tests que se basa en testear todas los outputs posibles de una unidad a testear. Mas en detalle es una heuristica que permite generar estos casos con complejidad O(n*k*o). En el contexto de testing sirve para la generacion automatica de casos de test.

13.10 22 de diciembre de 2008

1. Cual es la diferencia entre un refinamiento por casos y un o-refinamiento?

Un o-refinamiento se utiliza para dar alternativas de refinamiento de un objetivo. Un refinamiento por casos ,es cuando hay propiedades del dominio excluyentes que si alguna se cumple implican una Condicion que esta implica el objetivo refinado. (diagramita y ejemplo).

2. Cuando un objetivo es realizable por un agente? De ejemplos de objetivos realizables y objetivos no realizables.

un objetivo es realizable cuando esta lo suficientemente refinado (de bajo nivel) talque su descripcion haga obvio que puede asignarse a un agente. Ejemplo objetivo realizable : Lograr[Activar alarma si pulso bajo del paciente] Ejemplo objetivo no-realizable : Evitar[Choque de trenes]

3. Explique la relacion entre diagramas de actividad y redes de petri. Ejemplifique.

Los diagramas de actividad son redes petri amigables donde los estados tienen semantica. Presentan un flujo de actividades. Actividades estan conectadas por media de flechas representando el flujo de control. FALTA

4. A que nos referimos cuando decimos que la bisimulacion es una congruencia?

Congruencia : Dado un contexto para P, se quiere poder cambiar P por un proceso equivalente sin alterar el sistema. Una equivalencia es una congruencia si y solo si P≡Q implica C(P)≡C(Q). La semantica informal de bisimulacion es que las dos maquinas se mueven a los mismos estados, y esto para un observador equivale a que son iguales (observa lo mismo para las dos maquinas). Osea que existe una relacion entre los estados de las dos maquinas.

5. Diga si la siguiente asercion es verdadera o falsa. Justifique su respuesta . El testing es un tecnica de verificacion.

Verdadero. La verificacion trata sobre la comparacion de una funcion, metodo, procedimiento contra una especificacion (que puede ser texto, un modelo,etc). El testing no puede garantizar la inexistencia de errores,pero si podria mostrar la existencia de uno o mas fallas.

6. En el contexto de testing estructural , que son y cual es el problema de caminos no factibles?

Un camino no factibles es un camino para el cual no existen valores para las entradas del programa que recorren el camino (no es ejecutable). El problema de un camino no factible es que hay algo raro (y feo) en el programa, ya que es deseable (y mas que deseable) que no existan caminos no factibles. La existencia de un camino no factible es una posible alarma de algo malo en lo que se esta testeando.

13.11 2 de marzo de 2009

13.12 2 de marzo de 2010

1. El modelo jackson distingue entre asersiones descriptivas y prescriptivas. En que consiste esta diferencia? Explique porque esta distinticion es relevante al momento de hacer verificacion y validacion en el contexto del modelo de jackson.

Las aserciones descriptivas son propiedades o leyes que son verdaderas en el mundo. Las prescriptivas son aquellas cosas que se esperamos que sean verdaderas en el mundo.

Son relevantes porque contienen informacion para poder realizar la verificacion y validacion. Las aserciones prescriptivas son relevantes para la verificacion, ya que contienen informacion sobre el resultado esperado. Las descriptivas son utiles para validar, ya que contienen informacion sobre el entorno donde estara el sistema a

construir. Es comun validar propiedades del dominio con un experto de dominio para ver si tienen sentido, por ejemplo avison sobre pista ¡=¿ ruedas en movimiento . Las prescriptivas tambien son utiles para corrobar con el experto (o stakeholders) ya que son estos quienes conocen lo que se espera obtener del sistema. En resumen, tienen informacion relevante (y muy importante) para poder realizar la verificacion y validacion,recordando que la validacion es contra el problema y la verificacion es contra otra especificacion.

2. Explique en general la relacion entre las hojas de un grafo de objetivos y las operaciones en un modelo de operaciones y en particular la multiplicidad de dicha relacion. Ejemplifique

Las hojas de un grafo de objetivos son aquellos objetivos de mas bajo nivel y los cuales estan asignados a agentes (en general a un solo agente). Segun el agente asignado se categorizan en expetativas o requerimiento. Los requerimientos inducen operaciones del software. Las expetativas inducen operacion de agentes. No siempre es una relacion uno a uno.Los casos de uso son o agrupan operaciones.Los casos de uso operacionalizan expectativas.Requerimientos inducen operacionen.

Ejemplo, objetivo si nivel bajo de stock =¿ activar alarma podria refinarse en Mantener[Contabilizacion de stock] (asignado a el agente software) y Sonar alarma si nivel bajo (asignado a la alarma) en un CU podrian estar los dos objetivos a un mismo CU.

3. Explique informalmente en relacion a conjunto de trazas, la semantica de includes y extends en CU.

A — incluye — ¿ B

Semantica : Sea s un escenario denotado por A, entonces una porcion de s contiene un escenario denotado por B. Podrian existir escenarios denotados por B que no aparecen en escenarios denotados por A.

A ¡— extiende — B

Semantica : existe almenos un escenario s denotado por A, que contiene un escenario denotado por B. Pueden existir escenarios denotados por B que no aparecen en escenarios denotados por A.

4. Explique que significa y como puede darse semantica a automatas temporales

Explicando automatas temporales en terminos de otra notacion conocida, como LTS. La semantica de TA hereda una nocion de bisimulacion y la operacion de composicion en paralelo. semantica informal :

- Un estado es un par (s, v) con $s \in S$ y $v \in V$ y $Is[v]= true$.
- El tiempo solo transcurre en los nodos.
- El tiempo de transicion como despreciable.
- El tiempo avanza uniformemente en los relojes
- Se puede pasar a una transicion si su guarda es true.
- Siempre que el valor de los relojes sea verdadero se puede estar en el estado (ojo una transicion podria hacer que salga).

5. Que significa que un criterio de test subsume a otro.
Repetida

6. Si una unidad de software pasa un test suite que garantiza cobertura total de caminos entonces la unidad de soft no tiene fallas. V o F. Justifique

Falso. Aunque un test suite $all - path$ subsume a muchos otros criterios, el testing no garantiza la inexistencia de errores (dijkstra) sino que puede confirmar la existencia de fallas (aunque podria decir que no se encontraron errores, eso no significa que podrian existir). Ademas, la seleccion de casos no esta basada en el comportamiento funcional.

14 Casos de test en testing de sistemas reactivos

Definicion

- Se realiza una composicion en paralelo con la especificacion ‘
- el test pasa , si todos los resultados alcanza un estado OK
- el test falla, si algun test alcanza el estado fallido

Consistente $\forall i : iiocoe \Rightarrow ipasaT$

Exahustiva $\forall ipasaT \Rightarrow iiocoe$

Completa Se dice que T es completa cuando es consistente y exahustiva

En la practica se quiere construir un test consistente.

Para la generacion de casos :

- Problema de grafos del cartero chino, minimo camino que pasa una y solo una vez por cada arista
- Random-walks