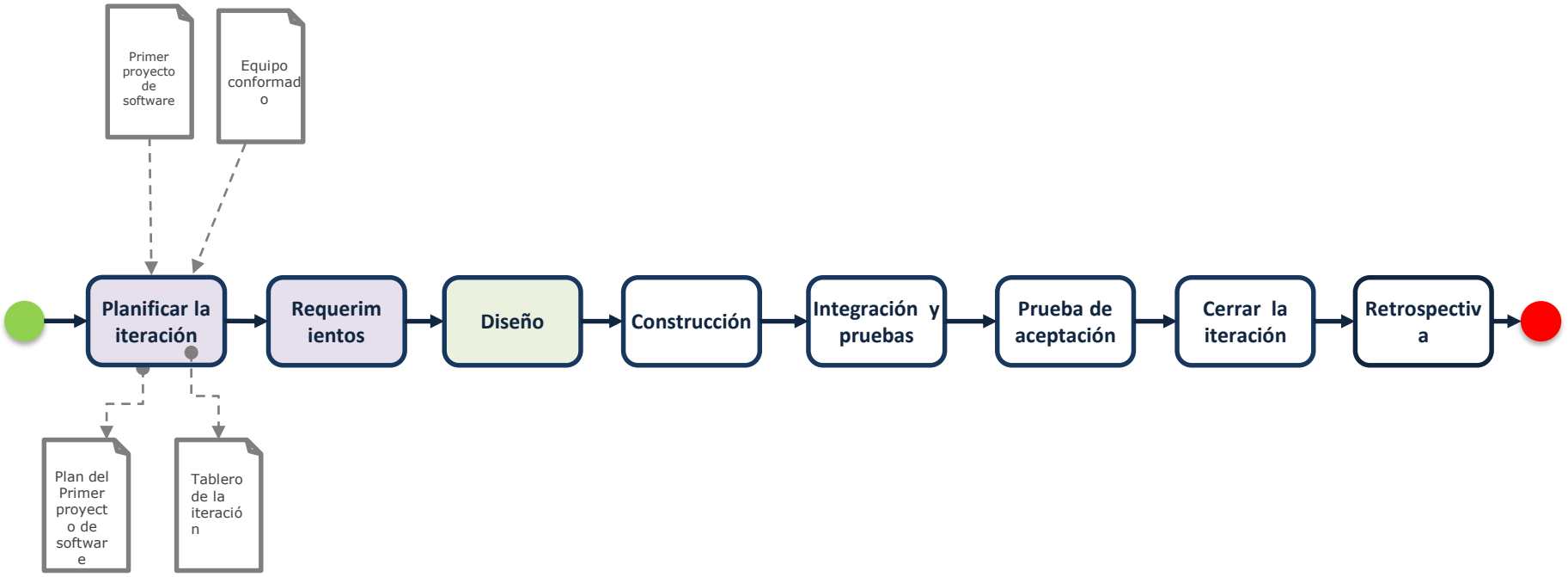
Curso de Ingeniería de Software

Unidad 6
Diseño de software

Guadalupe Ibargüengoitia
Hanna Oktaba

Actividades de la iteración



Entradas a esta unidad

- **Condiciones**
  - Requerimientos de software entendidos
- **Productos de trabajo**
  - *Especificación de Requerimientos de Software*
  - Plantilla de *Especificación de Diseño de Software*
  - *Repositorio* compartido con la carpeta para la iteración
  - *Tablero* de la iteración

Diseño de software

- Diseño de cualquier producto consiste en crear un modelo o representación de lo que se construirá más adelante.

- Diseño de software es el conjunto de actividades creativas mediante las cuales los requerimientos se traducen en una representación del software.

Objetivos del diseño

- Identificar y caracterizar las partes o componentes principales del software

- Definir su interacción e integración en el producto, para llegar al nivel de detalle que permita su mapeo al código en algún lenguaje de programación.

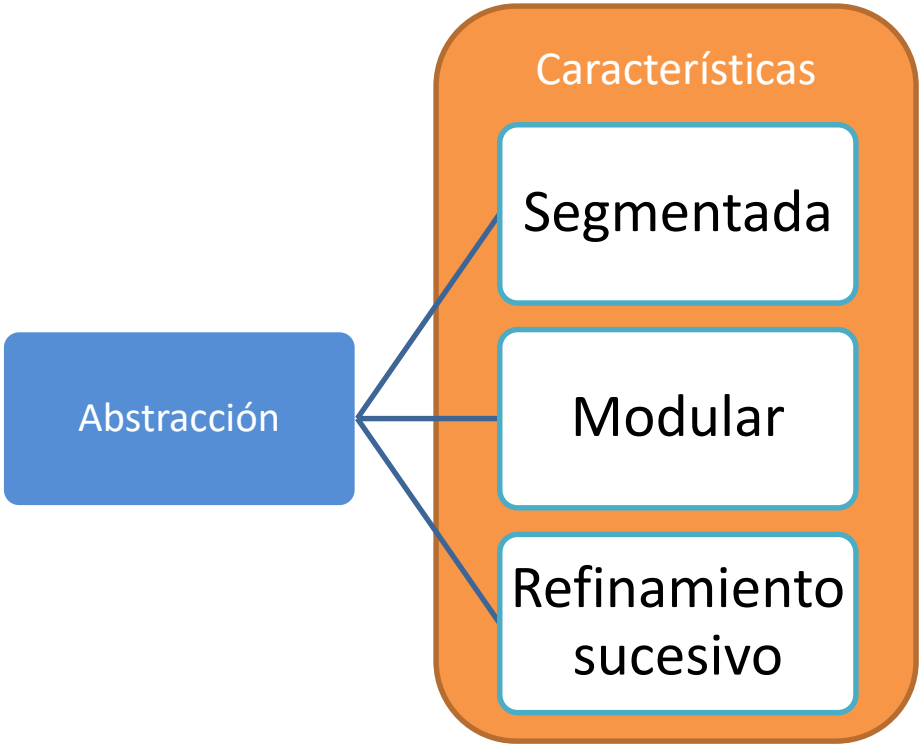
Niveles de abstracción

- ***Diseño arquitectónico* -** describe cómo se descompone y organiza el software en componentes abstractos partiendo de la especificación de requerimientos.
- ***Diseño detallado* -** describe el comportamiento específico de esos componentes tomando en cuenta el ambiente en el cual se codificará.(SWEBOK 2014)

Principios para el Diseño(SWEBOK)

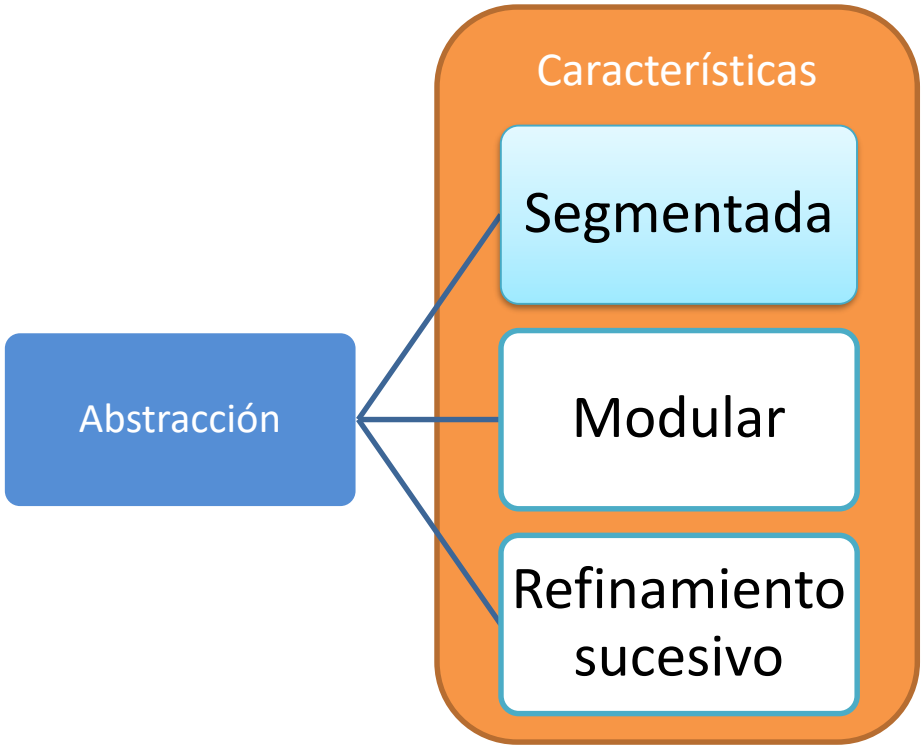
- **Abstracción –** es una vista de un objeto que se centra en la información relevante para un propósito particular e ignora el resto de la información. Para el diseño de software inicialmente se abstraen las propiedades más generales que deberán tener los componentes del software a construir.

...Principios básicos



Se refiere identificar y modelar de forma estructurada propiedades esenciales de un conjunto de objetos omitiendo detalles no esenciales, según sea el caso.

...Principios básicos

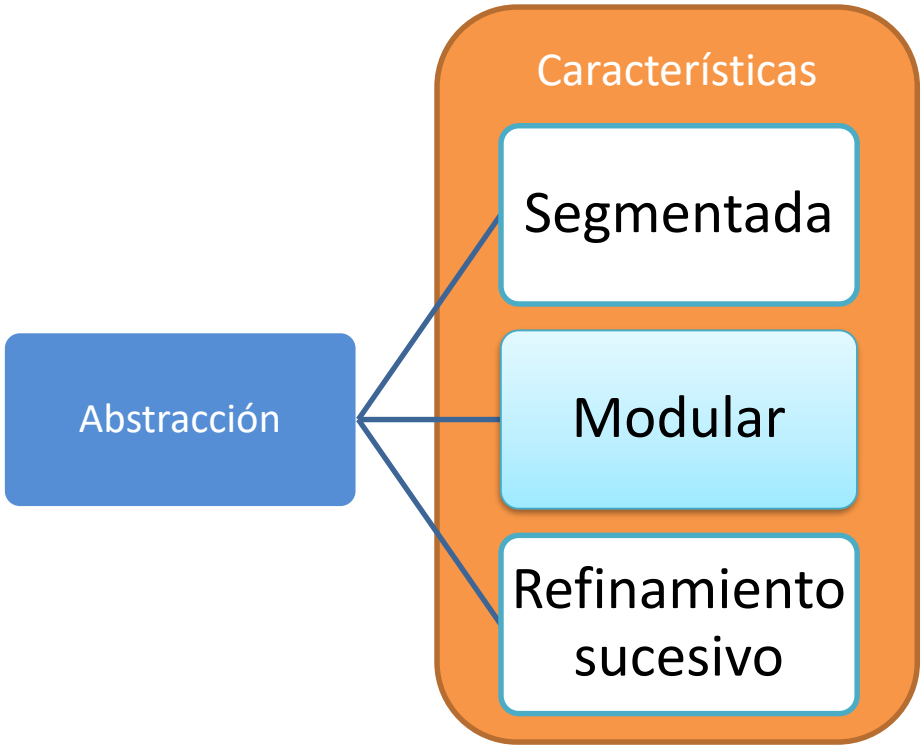


Se refiere a dividir en múltiples elementos que permitan manejar fácilmente un sistema en su conjunto.

Principios para el Diseño

- **Separación de conceptos** - una preocupación de diseño es la separación de elementos relevantes para uno o más de sus involucrados. Cada vista de arquitectura refleja uno o más intereses. Separar los conceptos por vistas permite a los interesados centrarse en cosas a la vez y de esta manera manejar la complejidad.

...Principios básicos



Se refiere a que cada elemento cumpla funcionalidades que lo hagan un componente individual.

Principios para el Diseño

- **Descomposición y modularización -** descomponer y modularizar el software significa dividirlo en una serie de pequeños componentes, que tienen bien definidos interfaces y que describen las interacciones de los componentes. Por lo general, el objetivo es colocar diferentes funcionalidades y responsabilidades en diferentes componentes.

Principios para el Diseño(SWEBOK)

- **Acoplamiento y cohesión** - el acoplamiento está definido como "una medida de la interdependencia entre módulos en un programa de computadora", mientras que cohesión se define como "una medida de la fuerza de asociación de los elementos dentro de un módulo ".
- Un buen diseño busca la cohesión de módulos alta y el acoplamiento débil.

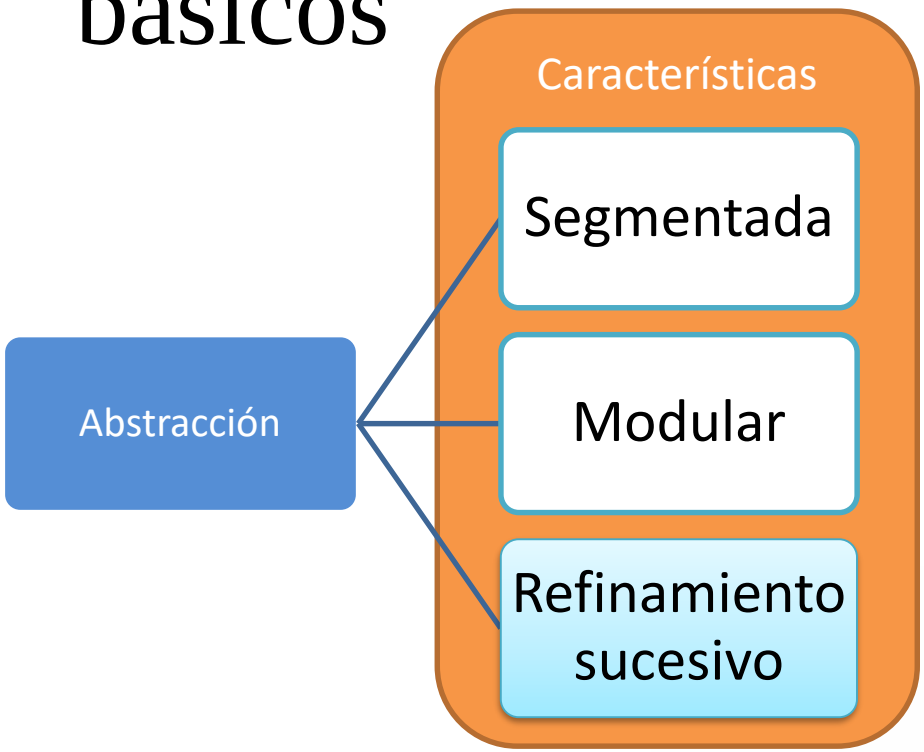
Principios para el Diseño

- **Encapsulación y ocultamiento de la información** - propone agrupar y empaquetar los detalles internos de una abstracción y hacer esos detalles inaccesibles a entidades externas.

Principios para el Diseño

- **Separación de la interfaz y la implementación** - significa definir un componente especificando una interfaz pública la cual está separada de los detalles de cómo el componente se realiza.

...Principios básicos



Se refiere a extender deductivamente el modelo conceptual de requerimientos en una serie de incrementos precisando cada vez, especificaciones que aumentan el nivel de detalle.

Mantenencia

Módulo Consulta

Principios para el Diseño

-
- **Refinamiento sucesivo -**extender deductivamente el modelo conceptual de requerimientos en una serie de incrementos precisando cada vez más, especificaciones que aumentan el nivel de detalle.

Otros principios

- **Diseñar para el cambio** - significa que el diseño debe ser flexible para permitir cambios con relativa facilidad.
- **Diseñar para facilitar el uso del software** - Considerar algunos escenarios del uso del software y su interfaz puede ayudar en el diseño de los componentes apropiados.

- **Diseñar para facilitar la prueba** - Los componentes del sistema deben estar diseñados como unidades que se pueden probar sin depender de la implementación de otros componentes.
- **Diseñar para la reutilización** –Consiste en definir partes genéricas que puedan volver a usarse. Para aplicar este principio se deben identificar los componentes comunes que se podrán reutilizar. El reuso incluye no solo el nivel del diseño, sino de código, casos de prueba, modelos o diagramas.

El diseño es una actividad creativa por lo que no existe "el mejor" diseño.

Cualidades del software

- Al diseñar se deben considerar las cualidades que deberá tener como:
  - De la arquitectura: integridad, corrección, facilidad de construcción y completitud.
  - De uso: seguridad, eficiencia, funcionalidad y usabilidad.
  - De ejecución: facilidad de modificación, de prueba, reusabilidad y portabilidad.

Actividades del diseño

1. Diseñar la arquitectura del software
2. Identificar el ambiente de desarrollo
3. Diseñar los componentes
4. Diseñar la base de datos

Arquitectura de software

Cualidades que debe tener la arquitectura:

- Sencillez. Fácil de comprender y de implementar.
- Extensión. La posibilidad de agregar nuevos componentes.
- Cambio. Los cambios en los requerimientos no afecten mucho a la arquitectura.

Actividades para definir la arquitectura

- Seleccionar el **tipo de arquitectura** del software según el tipo de la aplicación a desarrollar.
- Identificar los **componentes** que conformarán la arquitectura del software y documentarla con un **diagrama de paquetes**.
- En el caso de que sea un sistema distribuido, identificar la lógica de la distribución y documentarla en el **diagrama de distribución** que asigna los componentes a los distintos **nodos de hardware**.
- Identificar el lenguaje de programación y las herramientas para la construcción.

Diseño de componentes

- Una vez identificados los componentes más abstractos de la arquitectura, se continua con la **identificación** de los **elementos a nivel mas detallado del diseño,** que son **clases**.
- Para detallar los componentes seconstruyen diagramas mostrando vistas estáticas y dinámicas de los componentes con diagramas de UML.

Persistencia de datos

- A partir de las **clases** se identifica la información que debe ser **persistente** y se diseña la **base de datos.**
- Uno de los diagramas más utilizados para diseñar las bases de datos, es el de **Entidad-Relación**.
- Los atributos de las clases se convierten en los atributos de las Entidades.
- Las relaciones entre las Entidades se construyen a partir de las relaciones entre clases.

Método Inicial de Desarrollo de Software

- Práctica de Diseño de Software

Práctica de Diseño

Práctica	
Diseño de Software	
Objetivos	
• Seleccionar la arquitectura del software y diseñar los componentes para poder cumplir con la *Especificación de Requerimientos de Software*.	
Entrada	**Resultado**
Condiciones • Requerimientos de software entendidos **Productos de trabajo** •*Especificación de Requerimientos de Software* •*Plantilla Especificación de Diseño de Software* •*Repositorio* compartido con la carpeta para la iteración •*Tablero* de la iteración	**Condiciones** • Diseño del software completo **Productos de trabajo** • *Especificación de Diseño de Software*

Práctica de Diseño

Actividades
1. Establecer la Arquitectura general del software (TD1) (TD2) 2. Especificar el ambiente y las herramientas para la construcción de software. (TD3) 3. Diseñar los componentes principales del software para cada caso de uso dentro del alcance de la iteración. (TD4) (TD5) 4. Definir la navegación entre las pantallas (TD6) 5. Definir la base de datos (TD7) 6. Integrar el documento de Especificación de *Diseño de Software* (TD8)
Herramientas
Herramienta para hacer diagramas de UML Herramienta para hacer modelo Entidad-Relación

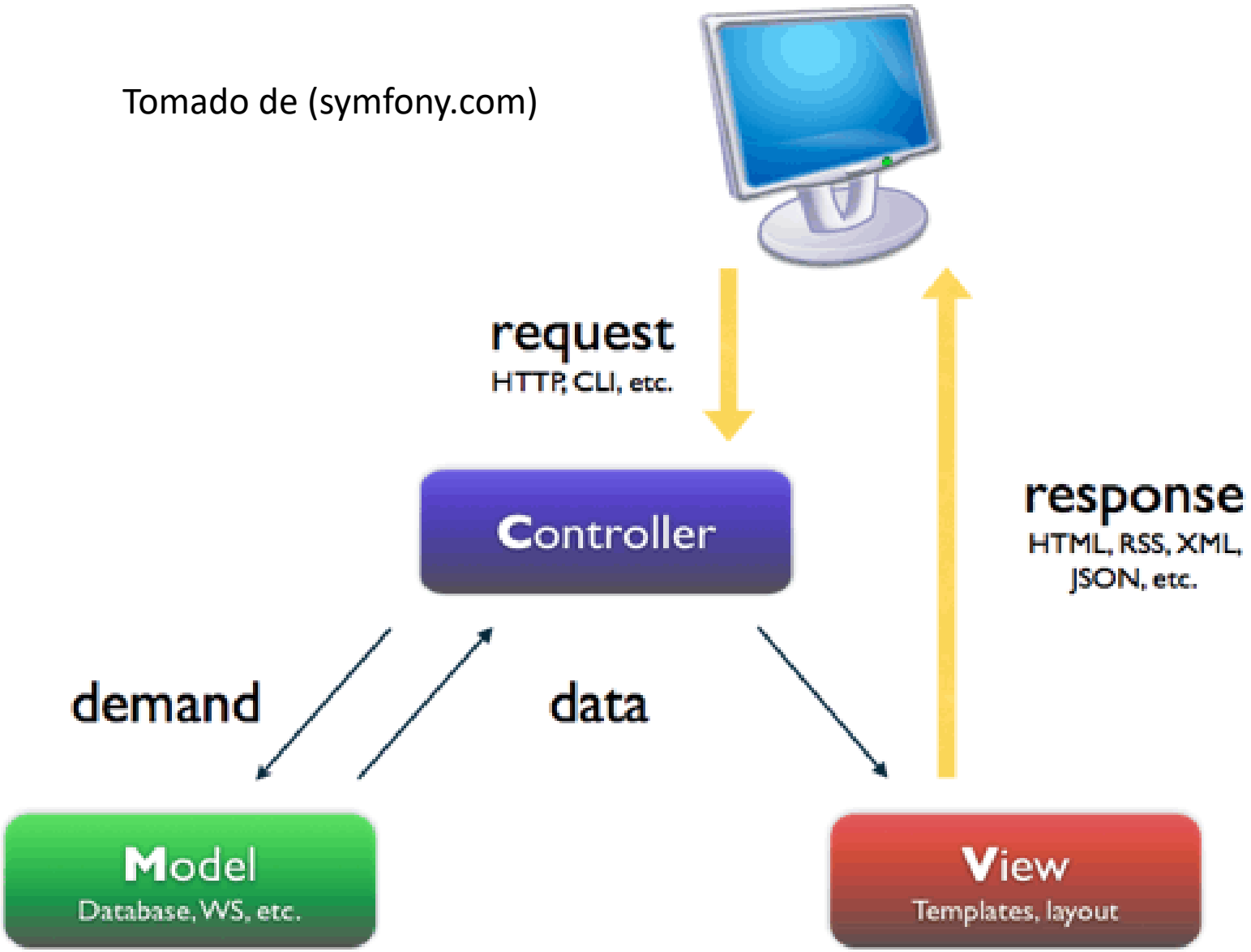
Actividad D1. Establecer la Arquitectura general del software.

- Para aplicaciones web hay un tipo de arquitectura recomendado conocido como patrón arquitectónico *Modelo Vista Controlador (MVC)*.
- MVC tiene varias cualidades como son modularidad, sencillez, facilidad de construcción y adaptabilidad al cambio, separa los datos de una aplicación, la interfaz del usuario y la lógica de control en tres componentes relacionados.
- El patrón MVC es utilizado por los principales marcos de trabajo para construir aplicaciones web como Netbeans, Eclipse, entre otros.

Patrón arquitectónico *Modelo Vista Controlador (MVC)*

- **Modelo** que se encarga representar a los datos de la aplicación y su estructura agrupando todos los componentes que tienen que ver con la persistencia de los datos. Se implementa con apoyo de algún manejador de bases de datos.
- **Vista**, agrupa todos los componentes que permiten la interacción con el usuario. La implementación frecuentemente es a través de páginas HTML.
- **Controlador**, establece la comunicación entre los dos componentes anteriores, agrupando los componentes que manejarán las reglas del dominio de la aplicación. El controlador recibe los eventos que le manda la vista, efectúa las operaciones necesarias solicitando los servicios al modelo y refleja los cambios en la vista.

Tomado de (symfony.com)



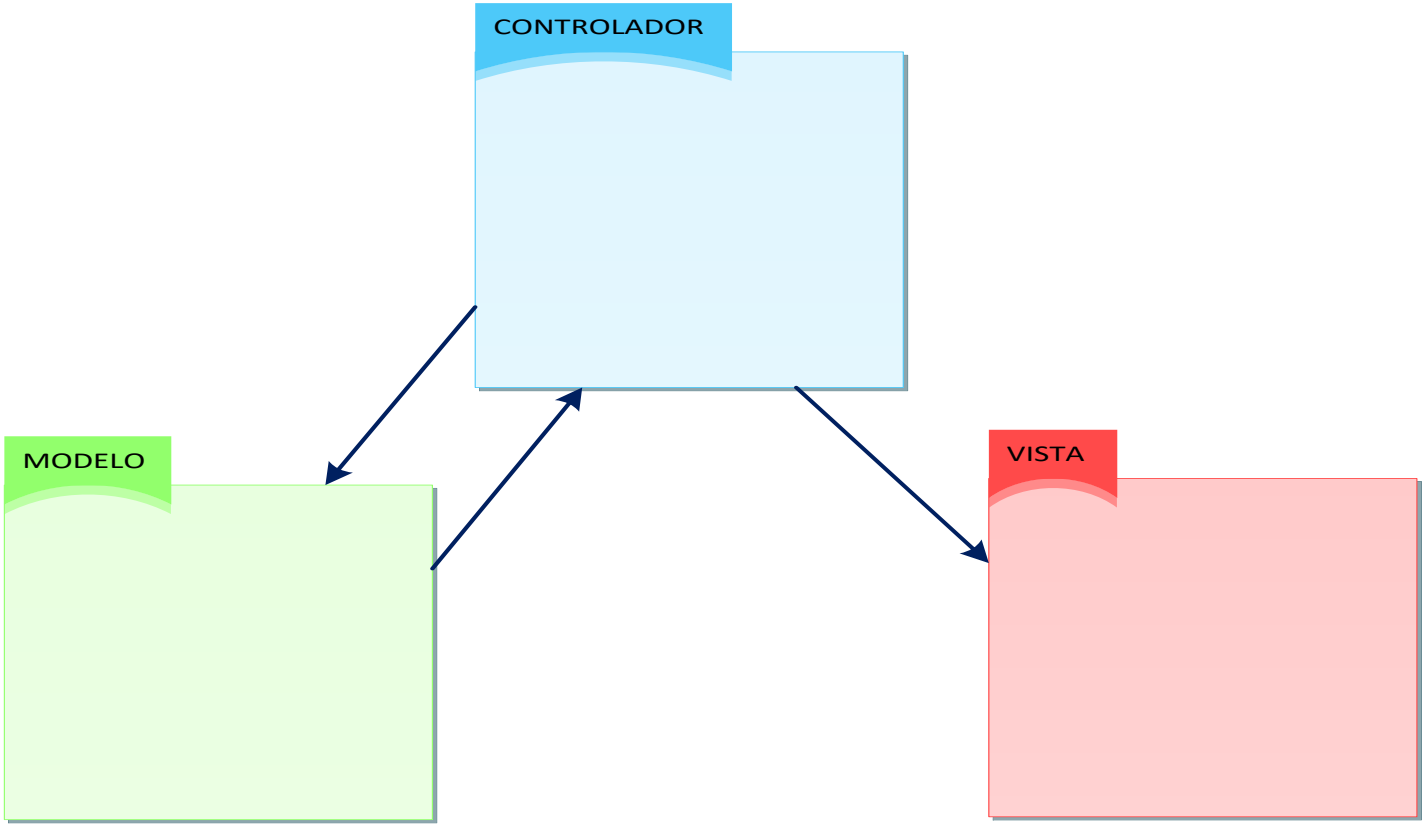
Técnica TD1. Descripción de la arquitectura

- Hacer el diagrama de paquetes que describe los componentes principales del software y sus relaciones basándose en el patrón MVC.
- Hacer un diagrama de distribución del software.
- Especificar el ambiente de implementación

Hacer el Diagrama de paquetes

- Paquete es un mecanismo general de UML para organizar elementos en grupos.
- Un paquete se representa con una carpeta con su nombre.
- Los paquetes pueden contener otros paquetes o clases.
- Los elementos de cada paquete deben tener una alta cohesión y bajo acoplamiento con los elementos de otros paquetes.

Diagramas de paquetes del Modelo Vista Controlador



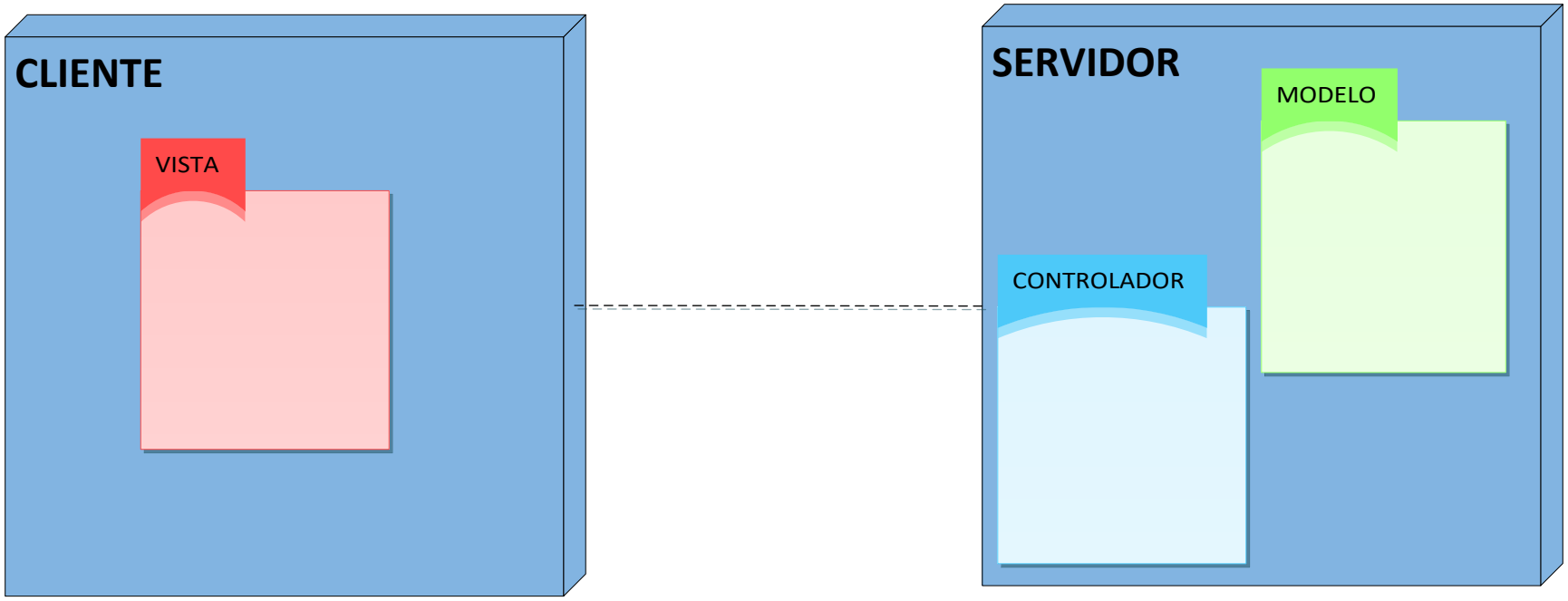
- El **Responsable de calidad** personaliza la plantilla de *Especificación de Diseño de Software*.
- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Hacer el diagrama de paquetes de la arquitectura, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluya el diagrama en la plantilla personalizada de *Especificación de Diseño de Software*.

Técnica TD2. Hacer el diagrama de distribución

- Los diagramas de distribución representan los componentes que se instalarán en cada máquina y las conexiones entre ellos.

- Los elementos de estos diagramas son: los *nodos* y las *conexiones* entre ellos

Diagrama de distribución para un sistema web



- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Hacer el diagrama de distribución, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluya el diagrama en la plantilla personalizada de *Especificación de Diseño de software*.

Actividad D2. Especificar el ambiente y las herramientas para la construcción de software

- Otra decisión importante es definir el ambiente de implementación. Dependiendo del tipo de aplicación a desarrollar, se selecciona el lenguaje de programación, el ambiente y herramientas con que se hará la construcción.

Técnica TD3. Especificar el ambiente de implementación

- El marco de trabajo (framework) para el desarrollo y su versión.
- Las herramientas de diagramación para UML.
- El manejador de bases de datos y su versión.
- El servidor de aplicaciones, si la aplicación es para web.
- El entorno integrado de desarrollo (IDE), si es el caso
- La herramienta que permitirá manejar el control de versiones del código

- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Definir el ambiente de implementación
- El **Responsable Técnico** mueve la tarjeta a la columna de *Haciendo* consigue el software y ayuda a que quede instalado en las máquinas del equipo de trabajo, mueve la tarjeta a la columna *Hecho* cuando ya todo el equipo tiene el ambiente listo en sus máquinas
- El Responsable de calidad se asegura que se incluya el ambiente de implementación en la plantilla personalizada de *Especificación de Diseño de software*.

Actividad D3. Diseñar los componentes principales del software para cada caso de uso dentro del alcance de la iteración.

- Para definir los componentes específicos usaremos una vista estática y otra dinámica.
- Para describir los componentes usaremos diagramas de clases, de secuencia, y de estados de UML.
- Para diseñar la base de datos usaremos diagramas Entidad-Relación.

Técnica TD4: Identificación de clases

- Los diagramas de paquetes y los de clase representan la *vista estática* del sistema.

- De acuerdo a esta vista se codificarán los paquetes y las clases en el lenguaje de programación elegido para la construcción.

Identificación de clases (1)

- Para identificar las clases, se analiza cada caso de uso para imaginar qué clases se necesitan en la Vista, en el Controlador y en el Modelo.

- El responsable técnico coordina la identificación de las clases solicitando a cada miembro del equipo que identifique las clases para el caso de uso del que es responsable.

Identificación de clases (2)

- **Clases para la Vista** (interfaz con el usuario). Son los elementos con los que interactúa directamente el usuario. Estas clases llamaremos con nombres de pantallas y les agregaremos IH (Interfaz Humana) al final del nombre para distinguirlas de las clases de otros componentes.
- **Clases de Controlador** (reglas del negocio o aplicación). Son los elementos que implementan las reglas del negocio y la lógica de la aplicación, es decir el control de todos los caso de uso.
- **Clases para el Modelo** (datos y su persistencia). Son los elementos del que representan los datos de la aplicación y aseguran su persistencia.

Identificar clases y sus relaciones

Ejemplo:

- Para ingresar al sistema, se necesita alguna interfaz de usuario que reciba los datos de ingreso, esto se puede implementar a través de código HTML y se modela como una clase en la Vista.
- Para almacenar las claves de ingreso, se construye una clase correspondiente en el componente del Modelo y se diseña una tabla en la base de datos.
- En el Controlador se construye una clase llamada *Ingreso* con métodos para revisar si los datos dados por la Vista corresponden con los almacenados en el Modelo.

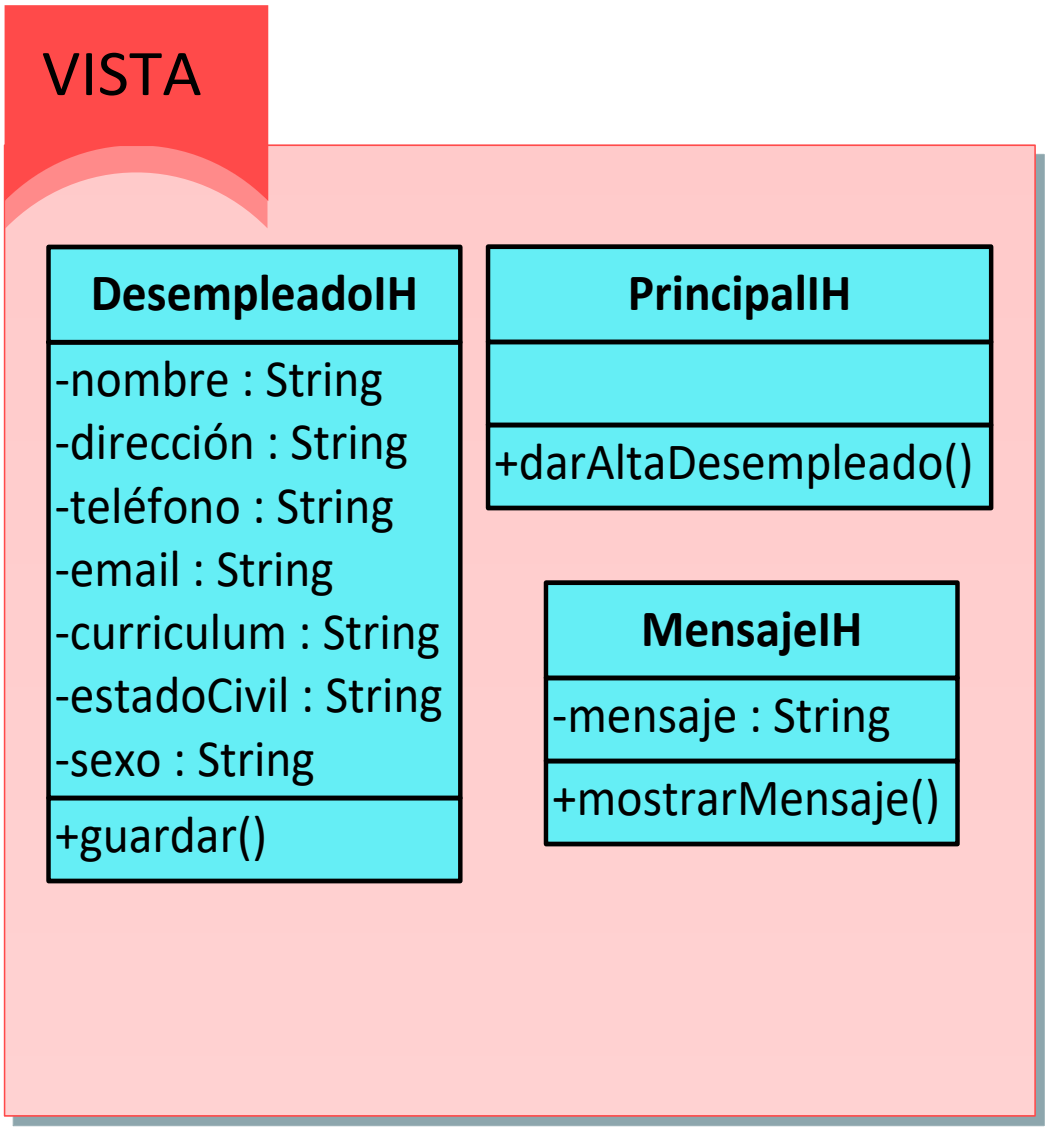
Identificación de clases del Controlador

- Los miembros del equipo revisan el detalle de sus casos de uso y crean una clase (o mas) para el componente del Controlador que se responsabilizará por realizar la funcionalidad básica del caso de uso.
- A esta clase se le da el nombre similar al caso de uso pero usando uno o mas sustantivos.
- Esta clase va a tener por lo menos un método que corresponda a la funcionalidad del caso de uso.

CONTROLADOR

Identificación de clases de Vista

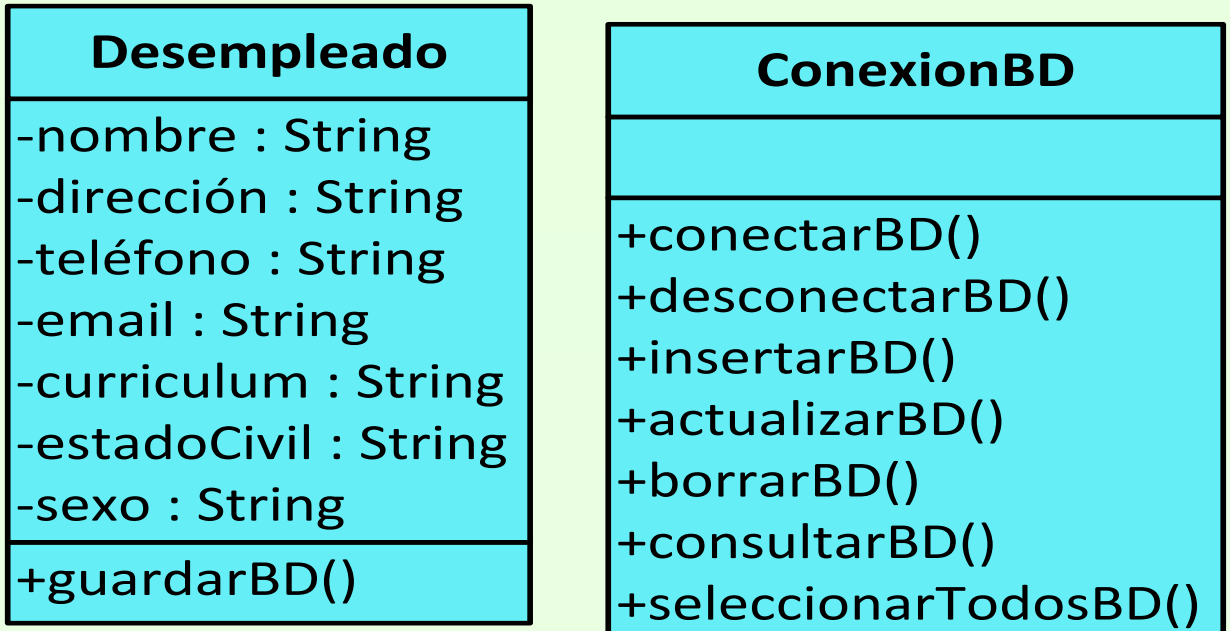
- Para identificar las clases que representarán la interfaz con el usuario se revisa el prototipo de interfaz de cada caso de uso en el documento de requerimientos y se dibuja una clase por cada pantalla.
- A la clase se da el nombre de la pantalla seguido de la abreviación IH (Interfaz Humana).
- Las clases de interfaz son las abstracciones que podrán ser implementadas con diversas tecnologías como por ejemplo código HTML o jsp.


VISTA
DesempleadoIH
-nombre : String
-dirección : String
-teléfono : String
-email : String
-curriculum : String
-estadoCivil : String
-sexo : String
+guardar()
PrincipalIH
+darAltaDesempleado()
MensajeIH
-mensaje : String
+mostrarMensaje()

Identificación de clases de Modelo

- Para identificar las clases del Modelo se analizan los detalles de casos de uso buscando conjuntos de datos que representan cierto tipo de información que manejará el sistema y la cual va requerir de ser resguardada y/o persitente.
- En el caso de requerir resguardo de los datos usando una base de datos se incluye una clase llamada ConexiónBD a través de la cual se hará la conexión real a la base de datos.

MODELO


Desempleado
-nombre : String
-dirección : String
-teléfono : String
-email : String
-curriculum : String
-estadoCivil : String
-sexo : String
+guardarBD()
ConexionBD
+conectarBD()
+desconectarBD()
+insertarBD()
+actualizarBD()
+borrarBD()
+consultarBD()
+seleccionarTodosBD()

Integración de los diagramas de clases

- Cuando todos los miembros del equipo han analizado sus casos de uso y han identificado las clases de cada componente de MVC, el **Responsable técnico** dirige la integración de clases de cada componente MVC en un diagrama de clases.
- En esta integración se toman las decisiones sobre la unificación de los atributos y métodos de las clases comunes y se establecen las relaciones entre clases.

- Cada **Desarrollador** pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad identificar las clases del Modelo, Vista y Controlador para su caso de uso, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Integrar los diagramas de clases del Modelo, Vista y Controlador, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluyan los 3 diagramas en la plantilla personalizada de *Especificación de Diseño de software.*

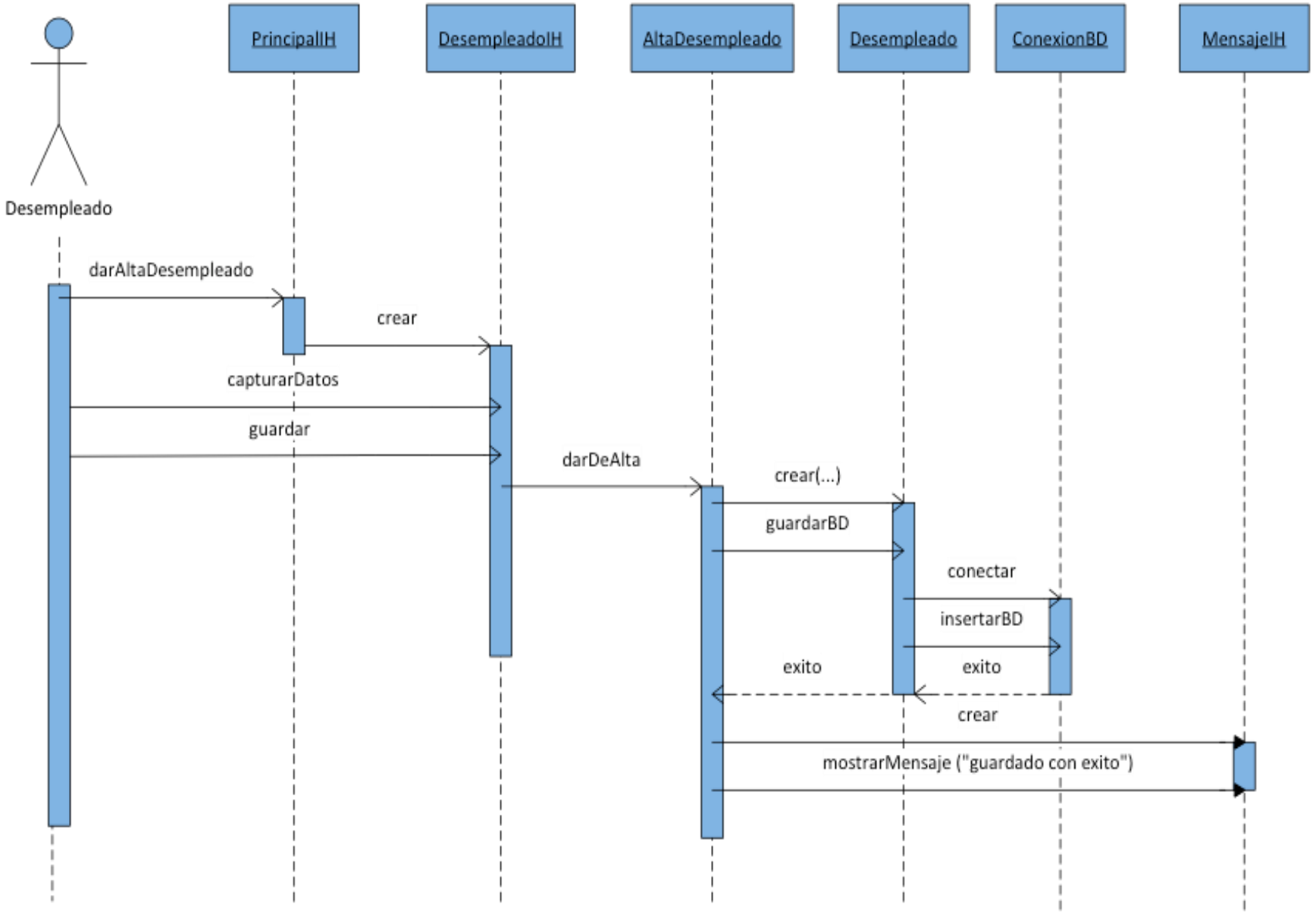
Técnica TD5. Hacer los diagramas de secuencia

- Los **diagramas de secuencia** y los **de estados** representan la ***vista dinámica*** del sistema.

- De acuerdo a esta vista se codificarán los algoritmos de intercambio de mensajes entre los objetos de las clases en el lenguaje de programación elegido para la construcción.

Diagramas de secuencia

- Los diagramas de secuencia muestran la interacción entre los objetos de las clases como una secuencia de envío de mensajes entre ellos, ordenados en el tiempo.
- Para su construcción se parte del detalle de los casos de uso. Para cada flujo normal, alternativo y excepcional de eventos en los casos de uso, se construye un diagrama de secuencia.


Desempleado
PrincipalIH
DesempleadoIH
AltaDesempleado
Desempleado
ConexionBD
MensajeIH
darAltaDesempleado
crear
capturarDatos
guardar
darDeAlta
crear(...)
guardarBD
conectar
insertarBD
exito
exito
crear
mostrarMensaje ("guardado con exito")

Hacer diagrama de secuencia (1)

- Por cada flujo de un caso de uso se crea un diagrama de secuencia de la siguiente manera:
  - Se representa al actor que corresponde al caso de uso poniéndolo arriba en el extremo superior izquierdo del diagrama.
  - El actor inicia las acciones del caso de uso enviando un mensaje a un objeto de una clase de la Vista que corresponde a la primera pantalla de este flujo. Se dibuja este objeto al lado derecho del actor. Los objetos, incluyendo al actor, tienen una linea punteada vertical que representa su "vida" en el tiempo.

Hacer diagrama de secuencia (2)

- Enseguida se analiza cual de las clases del Controlador tiene el método que respondería a la petición hecha por el actor y se dibuja un objeto de esta clase a la derecha del objeto anterior.
- Si es necesario se identifican otras clases del Controlador involucradas en el flujo del caso de uso y se dibujan sus objetos de manera parecida.
- Se continua con el análisis para identificar las clases del Modelo necesarias para completar el flujo dibujando sus objetos.

Hacer diagrama de secuencia (3)

- El flujo de eventos de un caso de uso se representa como envío de mensaje de un objeto al otro.
- Cada mensaje entre objetos aparece como una flecha dirigida del objeto fuente al objeto receptor, etiquetado con el nombre del método correspondiente.
- La duración de la ejecución de un método se representa como una barra más gruesa en la línea de vida del objeto.
- Para visualizar el orden temporal del envío de los mensajes, la flecha de un mensaje posterior se dibuja más abajo que la del mensaje anterior.

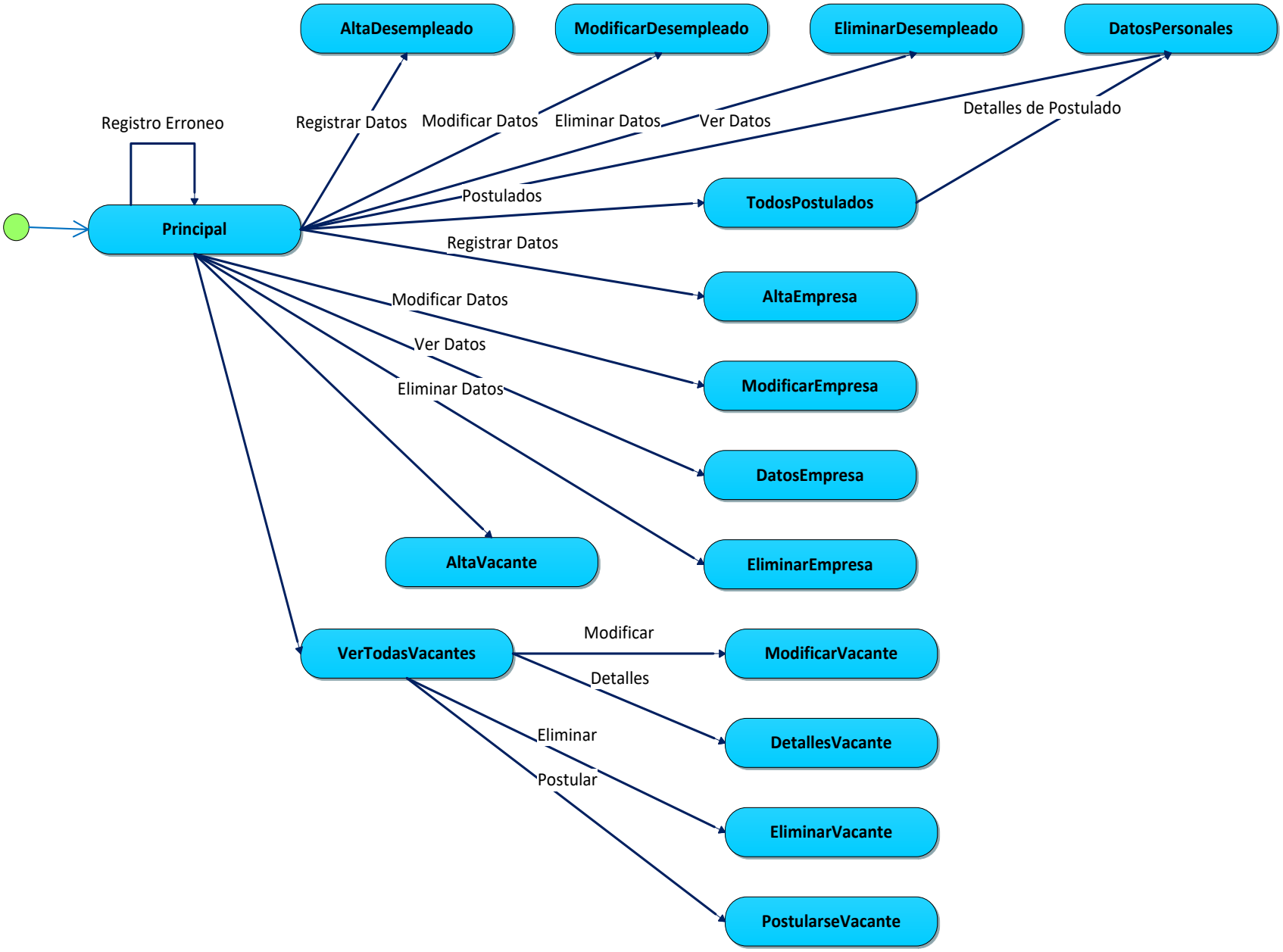
Refinamiento de diagramas de clases

- Una vez terminados los diagramas de secuencia para todos los casos de uso, se revisa la consistencia entre los diagramas de clase y de secuencia.
- Si alguna clase no se usó en ningún diagrama de secuencia se elimina del diagrama de clases.
- Si faltó alguna que no está en el diagrama de clase, se incorpora.
- Se modifican los diagramas de clases según lo encontrado durante la construcción de los diagramas de secuencia.

- Cada **Desarrollador** pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Hacer los diagramas de secuencia para su caso de uso, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad revisar los diagramas de secuencia, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluyan los diagramas en la plantilla personalizada de *Especificación de Diseño de Software*.

Actividad D4. Definir la navegación entre las pantallas

- Un diagrama de estados modela una máquina de estados finitos o autómata, que enfatiza el flujo de control de un estado a otro.
- Es una gráfica con nodos que representan estados y por arcos dirigidos que representan transiciones entre los estados a causa de eventos.


AltaDesempleado
ModificarDesempleado
EliminarDesempleado
DatosPersonales
Registro Erroneo
Registrar Datos
Modificar Datos
Eliminar Datos
Ver Datos
Detalles de Postulado
Postulados
TodosPostulados
Principal
Registrar Datos
AltaEmpresa
Modificar Datos
Ver Datos
ModificarEmpresa
Eliminar Datos
DatosEmpresa
AltaVacante
EliminarEmpresa
VerTodasVacantes
Modificar
ModificarVacante
Detalles
DetallesVacante
Eliminar
Postular
EliminarVacante
PostularseVacante

Diagramas de estados para la navegación

- El modelo de navegación define el orden y las relaciones entre las funcionalidades (casos de uso), se documenta usando diagramas de estados de UML.
- Las pantallas del sistema se definen como estados, se identifica el evento que causa el cambio de pantalla (estado).
- Por ejemplo, si estamos en la pantalla de inicio de un sistema (estado) y seleccionamos *entrar al sistema* (evento) nos llevará a una pantalla que nos permite capturar datos de usuario y contraseña (otro estado).

Técnica TD6. Hacer los diagramas de estado para la navegación

- Para construir el diagrama de estados, que modela la navegación, se parte del prototipo de interfaz de usuario construido para los casos de uso.
- Las pantallas se representan por estados y los eventos, que ocasionan el cambio de una pantalla a otra, por las transiciones entre estados.
- El estado inicial del diagrama, apunta al estado que representa a la pantalla principal del sistema. En esta pantalla, por lo general, se tienen varias opciones para navegar.
- Cada opción representa a un evento que puede llevar a otra pantalla.

- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Revisar e integrar el diagrama de navegación, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluya el diagrama en la plantilla personalizada de *Especificación de Diseño de software*.

Actividad D5. Definir la base de datos

- A partir de las clases de Modelo, que requieren de la persistencia de sus datos , se diseña la base de datos.
- Para modelar la base de datos se usan los conocimientos y técnicas del curso de Bases de Datos.
- Uno de los diagramas más utilizados para este fin, es el de Entidad-Relación, en el cual se hace la correspondencia entre las clases del diagrama de clases del Modelo, que se considere sean persistentes, con una Entidad del diagrama Entidad-Relación.
- Los atributos de las clases se convierten en los atributos de las Entidades. Las relaciones entre las Entidades se construyen a partir de las relaciones entre clases.

- El **Responsable técnico**, pone una tarjeta en el Tablero en la columna de *Por Hacer*, con la actividad Revisar e integrar el diagrama de la base de datos, mueve la tarjeta a la columna de *Haciendo* cuando lo está haciendo y mueve la tarjeta a la columna *Hecho* al terminarlo.
- El **Responsable de calidad** se asegura que se incluyan el diagrama de la base de datos en la plantilla personalizada de *Especificación de Diseño de software*.

Actividad D6. Documentar el Diseño de software

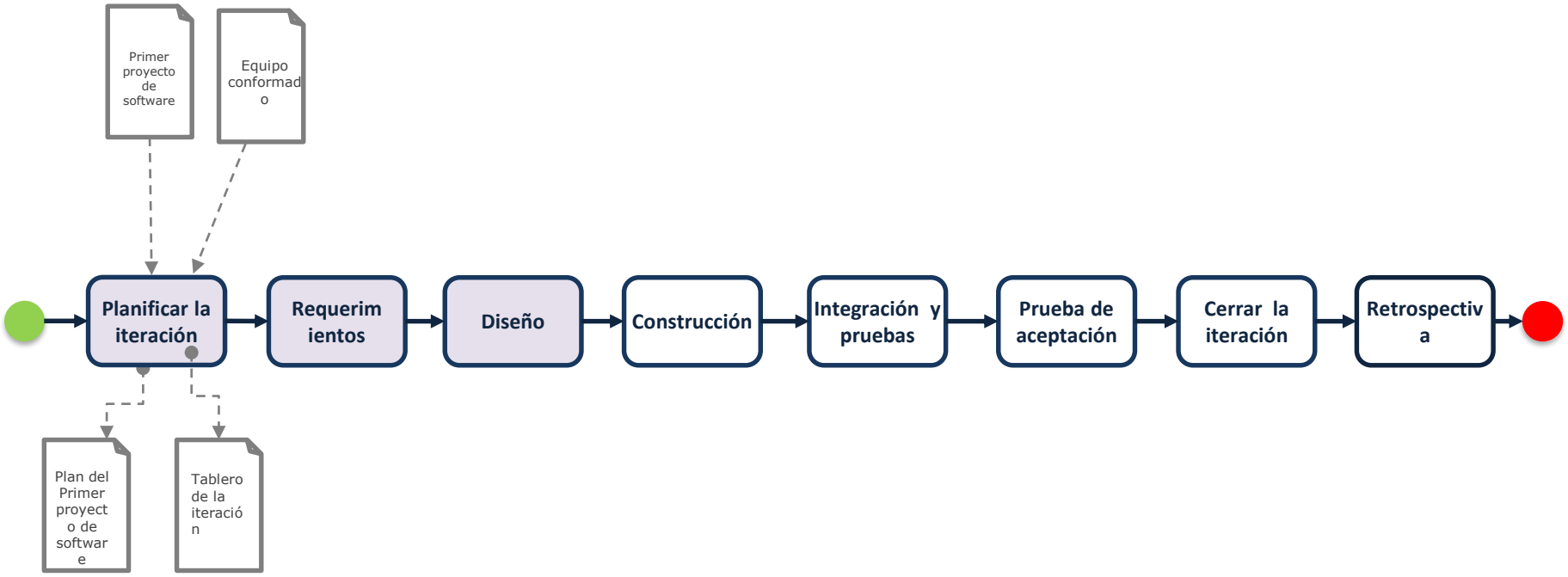
- Una vez aplicadas todas las técnicas anteriores, se han generado una serie de resultados que se reúnen en la *Especificación del Diseño de Software*.

TD7. Documentar la Especificación de Diseño de Software

- La labor del **Responsable de calidad** es asegurar que fueron integrados todos los elementos generados en esta práctica en la plantilla personalizada del documento *Especificación de Diseño de software*

- El **Responsable de Calidad** asegura la integración de todos los elementos del documento de *Especificación de Diseño de Software* siguiendo la plantilla.

Actividades de la iteración



Resultado de esta unidad

- **Condiciones**
- Diseño del software completo

- **Productos de trabajo**
- *Especificación de Diseño de Software*
  - *Descripción de la arquitectura*
    - Diagrama de paquetes o componentes
    - Diagrama de distribución
    - Ambiente de implementación
  - *Detalle de los componentes*
    - Diagrama integrado de clases para la Vista.
    - Diagrama integrado de clases para el Controlador
    - Diagrama integrado de clases para el Modelo
    - Diagramas de secuencia para cada caso de uso, en los flujos normal, alternativo y excepcional
  - Diagrama de navegación representado con diagrama de estados
  - Diagrama de Entidad-Relación
- *Repositorio* compartido con la carpeta para la iteración actualizado
- *Tablero* de la iteración actualizado

Bibliografía

- Ambler S.W. (2005). *The Elements of UML 2.0 Style.* Cambridge University Press.
- Booch G., R. J. (1998). *The Unified Modeling Languajes. Users Guide.* Addison Wesley.
- Buschamann F., R. M. (1996). *A System of Patterns. Patterns-oriented software archuitecture.* John Wiley.
- Humphrey, W. (1999). *Introduction to Team Software Process.* Addison Wesley Professional.
- ISO/IEC25010. (2011). *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.* ISO.
- ISO/IEC29110. (2011). *29110-5-1-2Software Engineering-lifecycle Profiles for Very Small Entities Management and Engineering Guidde. s.1.* Software Engineering.
- Pressman R:S. (n.d.). *Ingeniería de software. Un enfoque práctico.* Mc Graw Hill.
- Rosenberg D., S. K. (2001). *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example.* Addison Wesley.
- Schmuller J. (2000). *Aprendiendo UML en 24 horas.* Prentice Hall.
- Sommerville I. (2011). *Software Engineering.* Pearson.
- SWEBOK. (2014). *Guide to the Software Engineering Body of Knowledge v3.0.* IEEE Computer Society.
- wikipedia. (n.d.). *es.wikipedia.org/wiki/Modelo_Vista_Controlador*. From http://es.wikipedia.org/wiki/Modelo_Vista_Controlador