Curso de Diseño de Software – Impartido por Mario Chacón

Semestre V

Contenido

[1. 3](#_Toc40188318)

[2. Cuestionamientos al Desarrollo de Software 3](#_Toc40188319)

[2.1 Cuestionamientos al desarrollo de Software 3](#_Toc40188320)

[2.2 Eterno problema del diseño 4](#_Toc40188321)

[2.3 Contexto del profesional en Computación 4](#_Toc40188322)

[2.4 Problemas tradicionales del Desarrollo de Software 4](#_Toc40188323)

[2.4.1 Estimación de Tiempos 5](#_Toc40188324)

[2.4.2 Gestión de Riesgos 5](#_Toc40188325)

[2.4.2.1 Riesgos de Líder de Proyecto 6](#_Toc40188326)

[2.4.2.2 Riesgos de Ejecución de Proyecto 6](#_Toc40188327)

[2.4.2.3 Riesgos del Usuario 6](#_Toc40188328)

[2.4.2.4 Riesgos del entorno organizacional y Político 7](#_Toc40188329)

[2.4.3 Control de Calidad 7](#_Toc40188330)

[2.4.4 Diseño Inadecuado 7](#_Toc40188331)

[2.4.5 Experiencia en Tecnologías 8](#_Toc40188332)

[2.4.6 Motivación 8](#_Toc40188333)

[2.4.7 Asignación de personal nuevo 8](#_Toc40188334)

[2.4.8 Fases del Desarrollo de Software 8](#_Toc40188335)

[3. Procesos de Ingeniería de Requerimientos 9](#_Toc40188336)

[3.1 ¿Qué es la Ingeniería de requerimientos? 9](#_Toc40188337)

[3.2 Un proceso de desarrollo (referencia: RUP) 10](#_Toc40188338)

[3.3 Disciplina de Requerimientos 10](#_Toc40188339)

[3.4 Relación con otras disciplinas 11](#_Toc40188340)

[3.5 Entradas y Salidas 12](#_Toc40188341)

[3.6 Procesos de la Ingeniería de Requerimientos 12](#_Toc40188342)

[3.7 Desarrollo y administración 13](#_Toc40188343)

[3.8 Relación entre desarrollo y administración 13](#_Toc40188344)

[3.9 Desarrollo de requerimientos 13](#_Toc40188345)

[3.10 Administración de requerimientos 15](#_Toc40188346)

[3.11 10 trampas a evitar en el proceso 17](#_Toc40188347)

[3.12 Beneficios de un buen proceso de IR 18](#_Toc40188348)

[3.13 Verdades cósmicas sobre los requerimientos 18](#_Toc40188349)

[3.14 Buenas prácticas de IR 19](#_Toc40188350)

[3.14.1 Conocimiento 19](#_Toc40188351)

[3.14.2 Educción (adquisición, indagación) 19](#_Toc40188352)

[3.14.3 Análisis 20](#_Toc40188353)

[3.14.4 Especificación (documentación) 20](#_Toc40188354)

[3.14.5 Validación 20](#_Toc40188355)

[3.14.6 Administración de requerimientos 20](#_Toc40188356)

[3.14.7 Administración de proyectos 21](#_Toc40188357)

[3.15 Implementación de las prácticas 21](#_Toc40188358)

[4. Principios de Diseño de Software 22](#_Toc40188359)

[4.1 Definición 22](#_Toc40188360)

[4.2 Generalidades 23](#_Toc40188361)

[4.3 Análisis vs Diseño 23](#_Toc40188362)

[4.4 Diseño de Software: ¿Proceso o Modelo? 24](#_Toc40188363)

[4.5 Antecedentes: Modelos iniciales 24](#_Toc40188364)

[4.5.1 Problemas 24](#_Toc40188365)

[4.5.2 Consecuencias 25](#_Toc40188366)

[4.6 Modelo: Ciclo de Vida del software 25](#_Toc40188367)

[4.7 Modelo Estructurado 25](#_Toc40188368)

[4.8 Principios del desarrollo de Software 27](#_Toc40188369)

[4.9 Conceptos 28](#_Toc40188370)

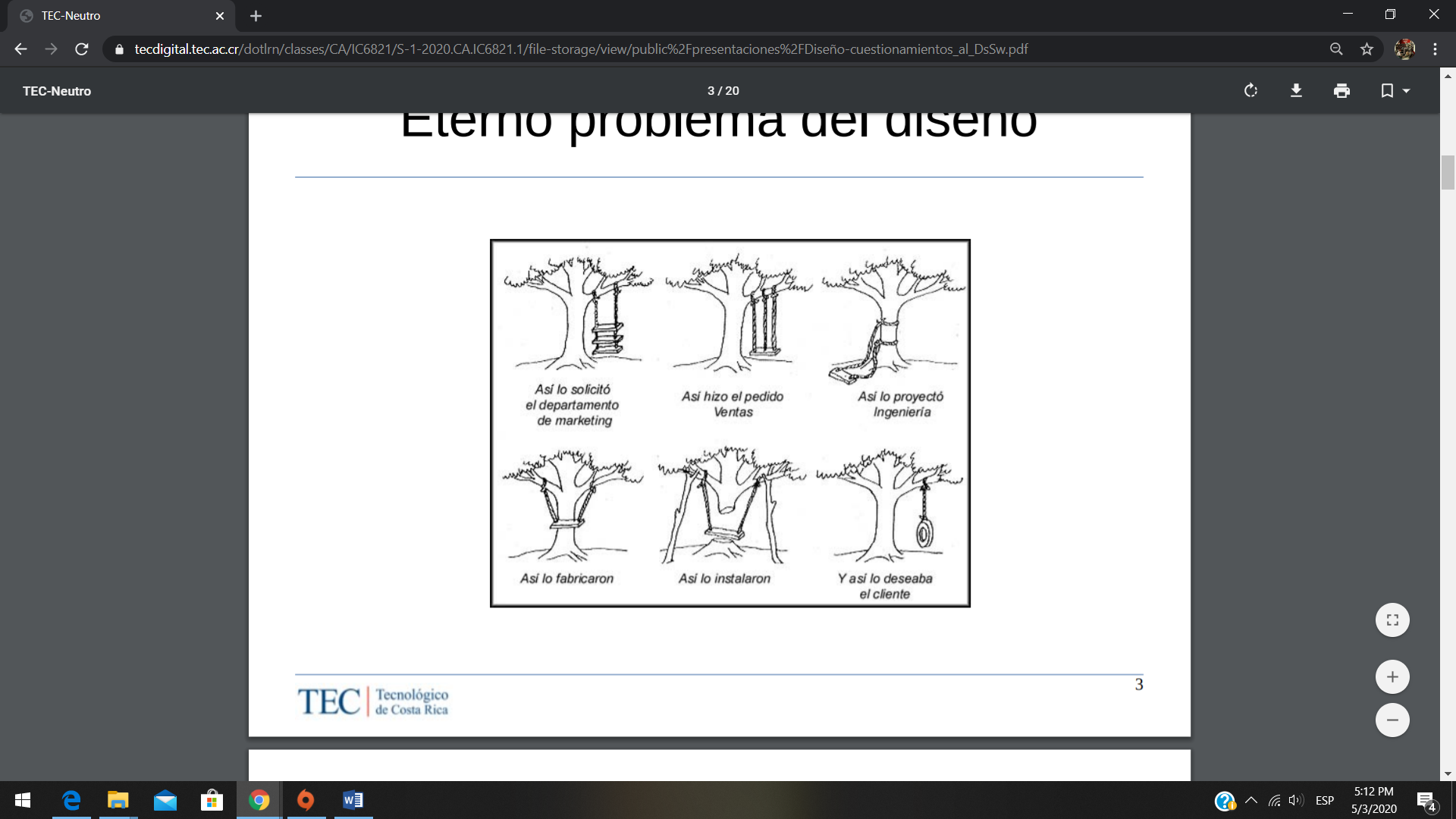
# 

# Cuestionamientos al Desarrollo de Software

## Cuestionamientos al desarrollo de Software

* Los sistemas no responden a las expectativas de los usuarios.
* Los programas fallan con cierta frecuencia.
* Los costos del software son difíciles de prever y normalmente superan las estimaciones.
* La modificación del software es una tarea difícil y costosa.
* El software se suele presentar fuera del plazo establecido y con menos características de las consideradas inicialmente.
* Normalmente, es difícil cambiar el entorno hardware usando el mismo software.
* El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, entre otros.) no suele cumplirse.

## Eterno problema del diseño



## Contexto del profesional en Computación

El contexto abarca las siguientes áreas:

* Ingeniería en Computadores
* Ciencias de la Computación
* Tecnologías de Información
* Ingeniería de Software
* Sistemas de Información

## Problemas tradicionales del Desarrollo de Software

* [Estimación de Tiempos](#_Estimación_de_Tiempos)
* [Gestión de Riesgos](#_Gestión_de_Riesgos)
* [Control de Calidad](#_Control_de_Calidad)
* [Diseño Inadecuado](#_Diseño_Inadecuado)
* [Poca experiencia en tecnologías](#_Experiencia_en_Tecnologías)
* [Poca motivación](#_Motivación)
* [Asignar personal nuevo a proyecto retrasado](#_Asignación_de_personal)
* Cambio vertiginoso de TI
* Comunicación de equipos
* Procesos - metodologías – modelos

### Estimación de Tiempos

* Estimaciones de tiempo poco fiables
* Basado en empirismo
* Con poca documentación
* Es poco probable que al final del proyecto se reflexione y compare la estimación con lo real
  + Escasa documentación de bitácoras y progreso.
  + Escasas estadísticas de las estimaciones y progreso.
* Conocimiento de técnicas
  + Puntos de Fusión
  + Casos de uso
  + Delphi
  + Juicio Experto

### Gestión de Riesgos

**Riesgo**: evento o condición incierta que en caso de que suceda incide en uno o varios de los objetivos del proyecto.

Tipos de Riesgos:

* [Líder del proyecto](#_Riesgos_de_Líder)
* [Ejecución del proyecto](#_Riesgos_de_Ejecución)
* [Usuarios](#_Riesgos_del_Usuario)
* [Entorno organizacional y político](#_Riesgos_del_entorno)

#### Riesgos de Líder de Proyecto

* Experiencia profesional
* Experiencia en el negocio del proyecto
* Competencias de estimación
  + Tiempo
  + Recursos
  + Presupuestos
* Liderazgo
* Identificación de funcionalidades
* Identificación con el proyecto y la organización

#### Riesgos de Ejecución de Proyecto

* Selección y contratación de personal (inadecuado)
* Idoneidad de Metodología de Desarrollo
* Definición de roles y responsabilidades
* Planeación efectiva del proyecto
* Control del proyecto
* Cultura organizacional
* Cumplimiento de objetivos

#### Riesgos del Usuario

* Compromiso de la gerencia
* Identificación de los usuarios claves para el proyecto
* Participación activa y motivada de usuarios
* Generalmente no están al alcance de los líderes del proyecto

#### Riesgos del entorno organizacional y Político

* Cambios en la gerencia organizacional
* Políticas nacionales o internacionales
* Dependerán del margen de acción de la organización
* A nivel nacional:
  + Presupuestos y ejecución presupuestaria
  + Presupuestos anuales 🡪proyectos con presupuesto plurianual
* A nivel internacional:
  + Cultura
  + Industria
  + Conformación de equipos interculturales

### Control de Calidad

* Definición de calidad en la organización
* Posibilidad de inversión
* Medición de costos reales y monitoreo de proyecto
* Control vs Aseguramiento de Calidad
* Apoyos metodológicos
  + ISO
  + CMMI

### Diseño Inadecuado

* Dimensión de etapa de Diseño
* Modelos de Diseños
* Innovación vs aplicaciones tradicionales
* Diseño de Interfaz 🡨🡪 Diseño de Arquitectura 🡨🡪 Diseño de Pruebas

### Experiencia en Tecnologías

* Experiencia del equipo es directamente proporcional en incidencia al éxito del proyecto
* Aprendizaje vs Productividad
* Investigación se paga en tiempo
* Experiencia en Tecnología puede presentar un impacto directo según la Experiencia que se tenga sobre el negocio

### Motivación

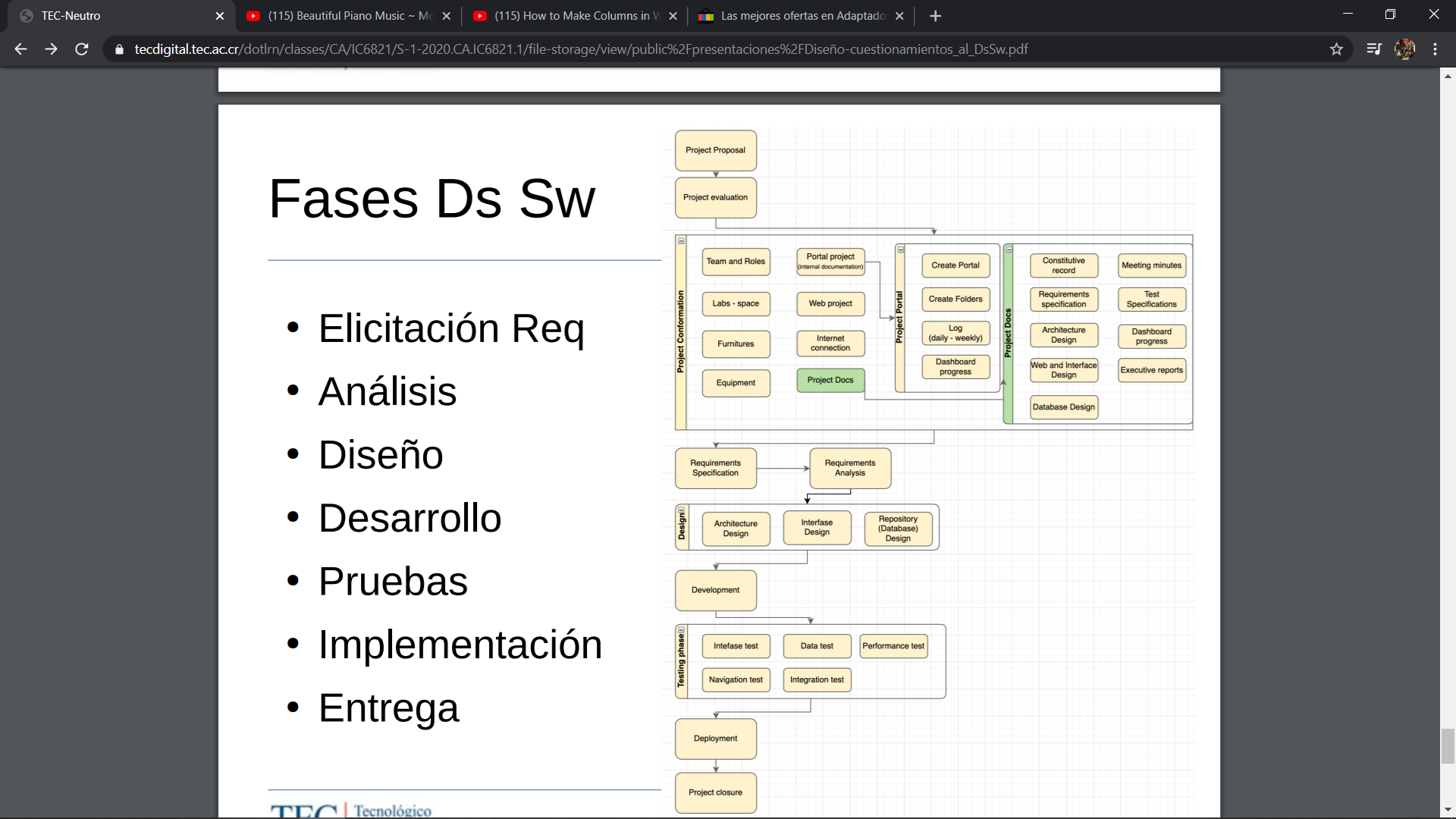
* La motivación del equipo del proyecto incide más que elementos técnicos
* Identificar claramente los roles requeridos del proyecto y las competencias del equipo
* Impacto de los elementos personales

### Asignación de personal nuevo

Este punto va a analizarse por medio del libro 🡪 [“Mítico Mes-Hombre”](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\Libros%20de%20Diseño%20de%20Software\Mítico%20Mes-Hombre.pdf), específicamente en el segundo capítulo el cual tiene el mismo nombre.

El análisis de la lectura se localiza en el “Resumen del Mítico Mes-Hombre”

### Fases del Desarrollo de Software

* Elicitación de Requerimientos
* Análisis
* Diseño
* Desarrollo
* Pruebas
* Implementación
* Entrega

# Procesos de Ingeniería de Requerimientos

## ¿Qué es la Ingeniería de requerimientos?

* Actividades para descubrir, documentar y mantener un conjunto de requerimientos.
* Técnicas sistemáticas y repetibles.
* Documenta las necesidades de negocio y necesidades técnicas.
* Forma parte del proceso de desarrollo del Software (o de otras disciplinas).

## Un proceso de desarrollo (referencia: RUP)



## Disciplina de Requerimientos

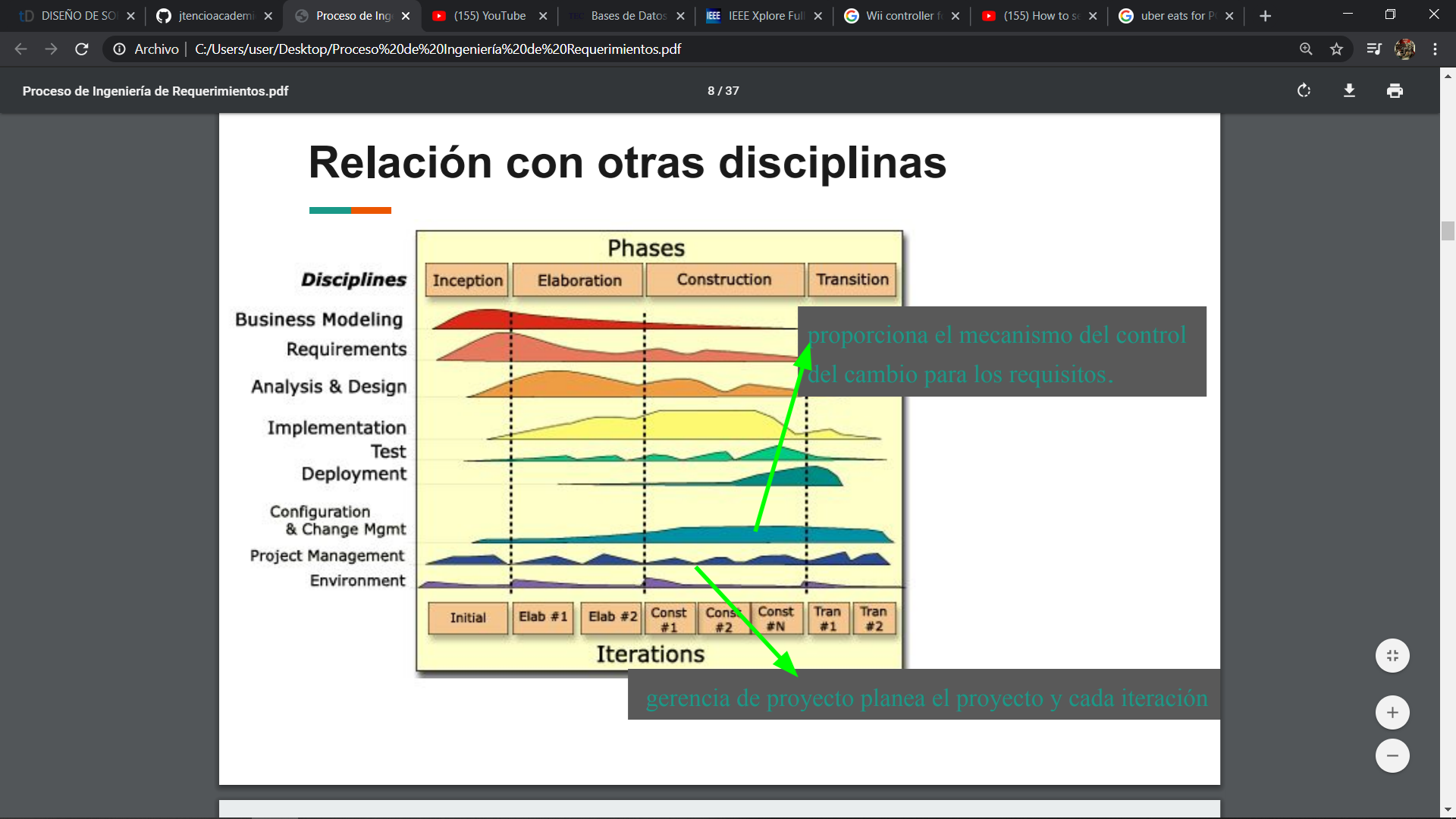
Principales motivos de la importancia de dicha disciplina:

* Para establecer y mantener el acuerdo con los clientes y otros tomadores de decisión de lo que debe hacer el sistema.

Para que los desarrolladores del sistema obtengan una mejor comprensión de los requerimientos del sistema.

* Para definir los límites del sistema.
* Para proporcionar una base para planear el contenido técnico de las iteraciones.
* Para definir una interfaz usuario-sistema, centrándose en las necesidades y las metas del usuario.
* Para alcanzar estas metas, es importante que todos los involucrados entiendan la definición y el alcance del problema que estamos intentando solucionar con este sistema.
* Entradas necesarias:
  + Reglas de negocio
  + Modelos de caso de uso de negocio
  + Modelo del análisis de negocio

## Relación con otras disciplinas

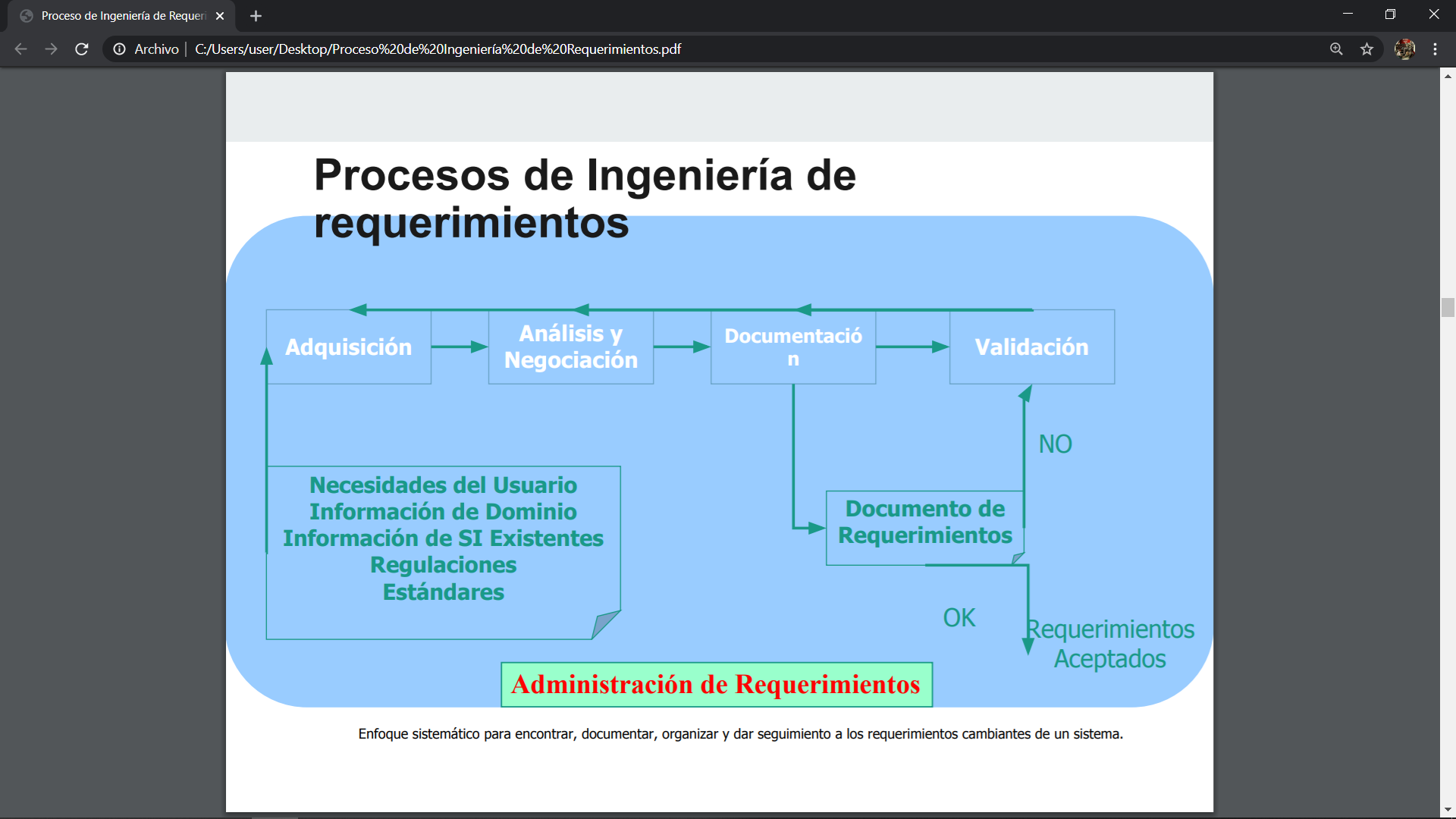


## Entradas y Salidas

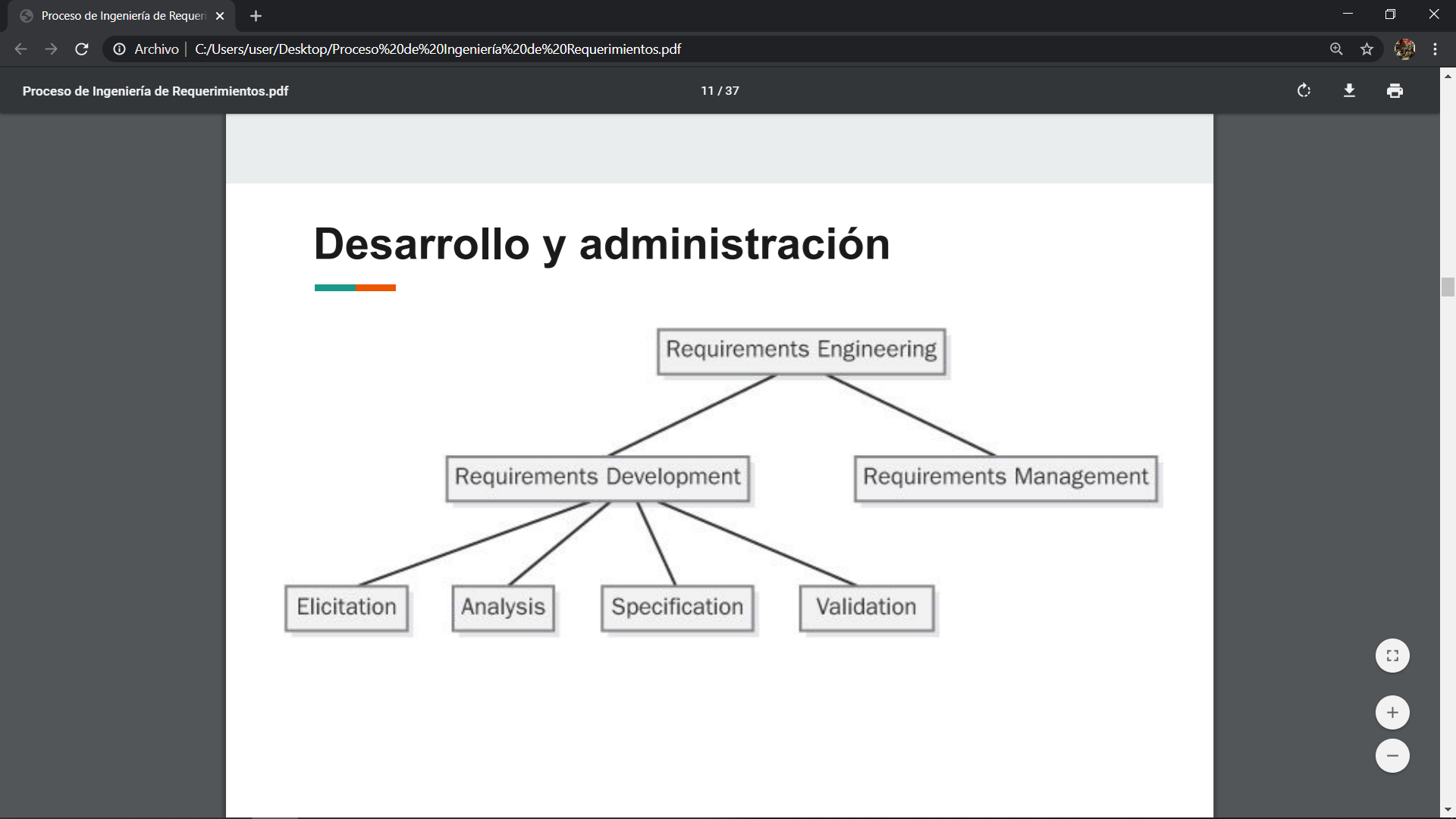


## Procesos de la Ingeniería de Requerimientos

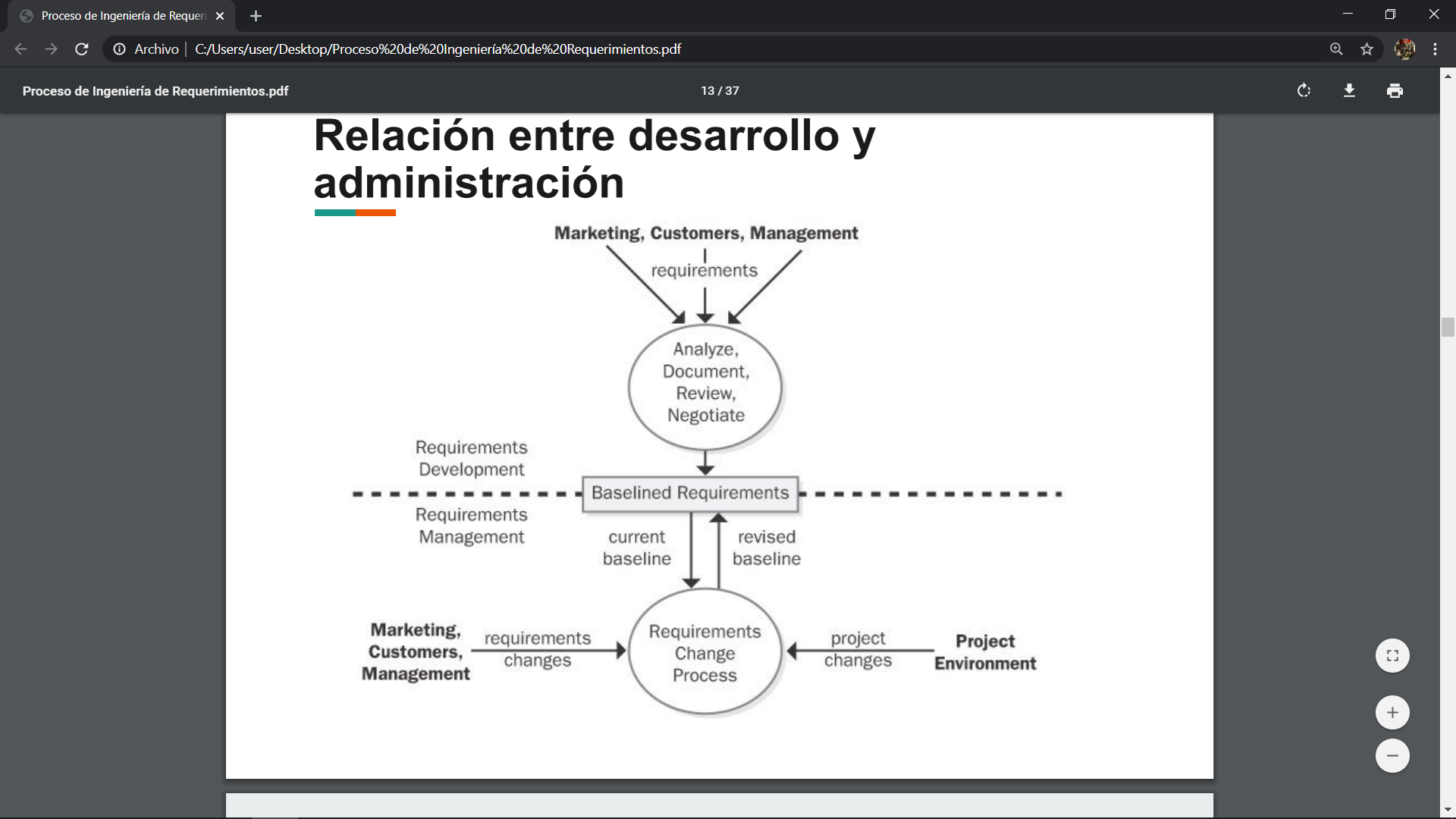
* Conjunto estructurado de actividades las cuales derivan, validan y mantienen un documento de requerimientos de sistema.
* Actividades.
* Roles y responsables.
* Calendarización de Actividades.
* Asignación de Responsables a las Actividades.
* Entregables (artefactos)



## Desarrollo y administración

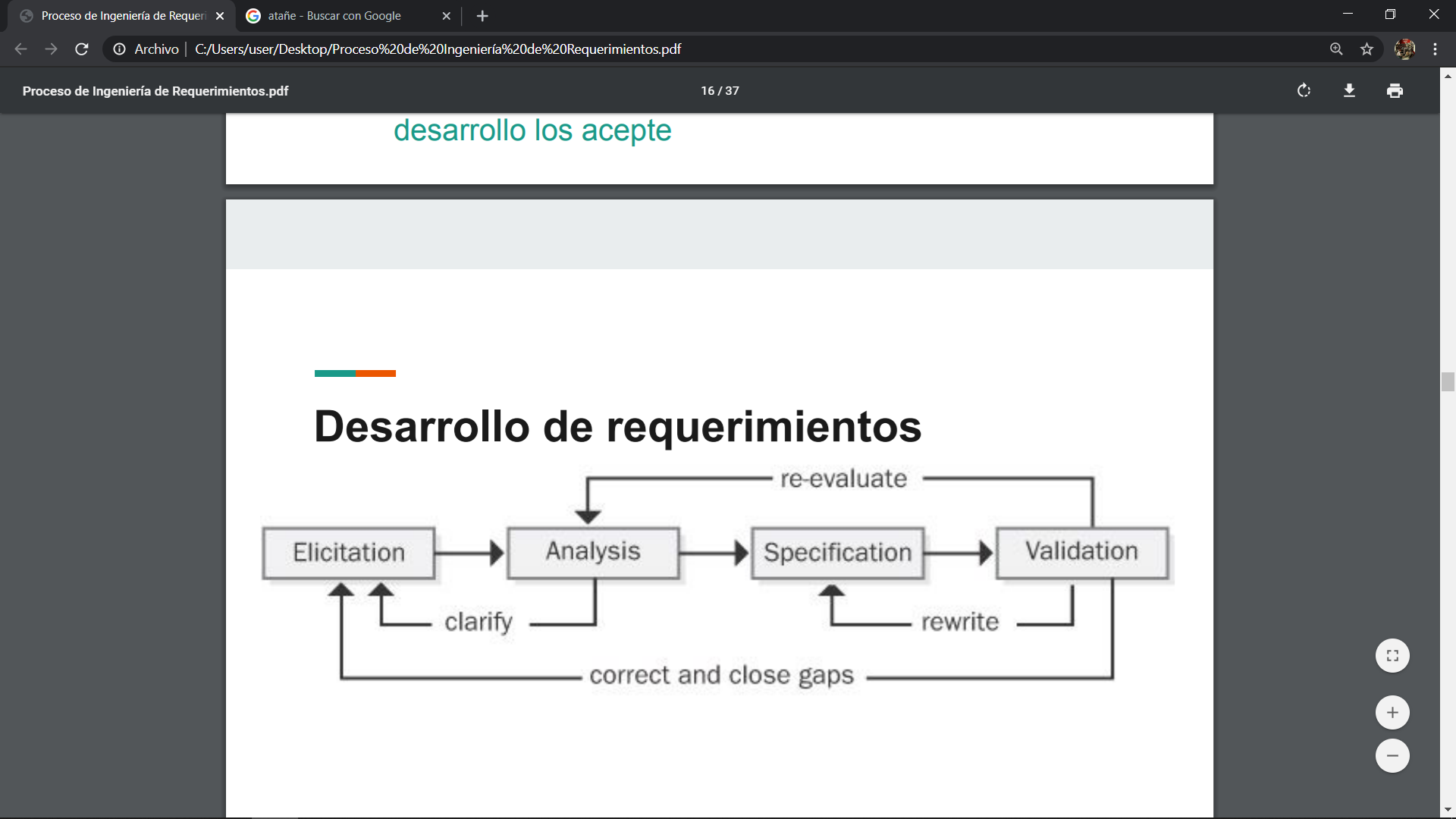


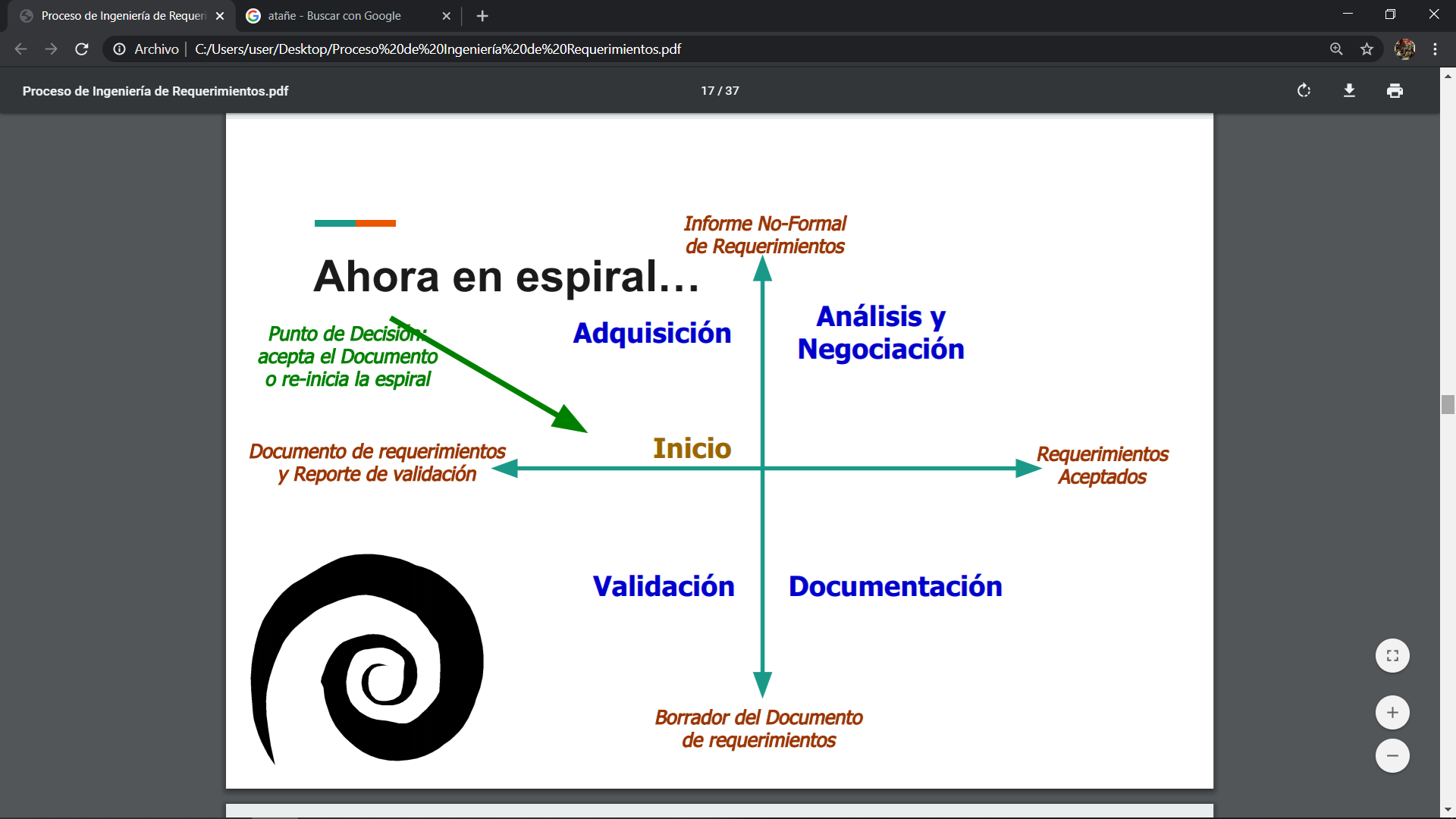
## Relación entre desarrollo y administración

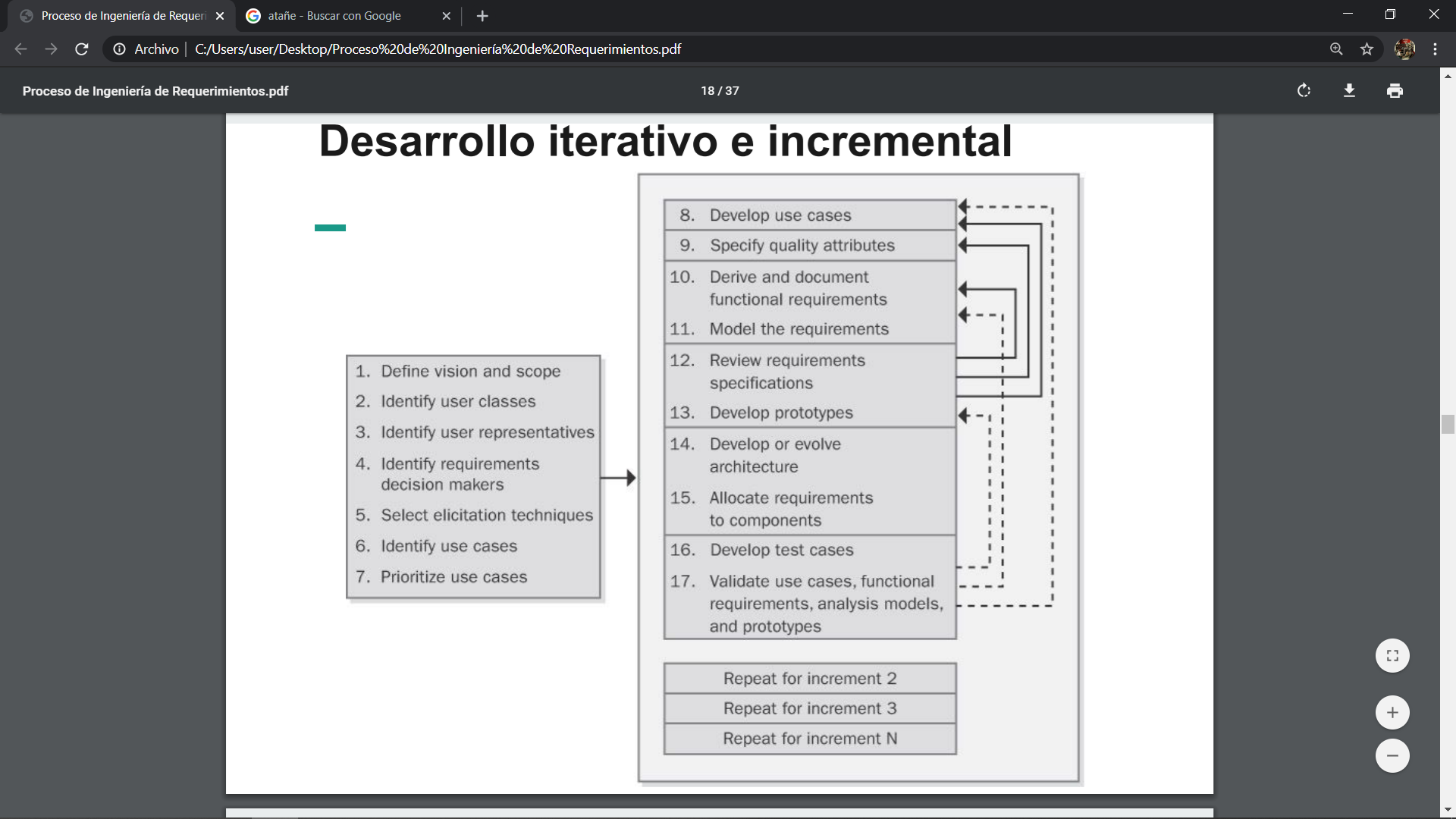


## Desarrollo de requerimientos

* Educción (adquisición), análisis y negociación, documentación (especificación) y validación comprenden:
  + Identificar las clases de usuario esperadas para el producto.
  + Indagar las necesidades de los individuos que representan a cada clase de usuario.
  + Comprender las tareas y metas de los usuarios, así como los objetivos del negocio con los que estas tareas se alinean.
  + Analizar la información obtenida de usuarios para distinguir metas de sus tareas de los requerimientos funcionales, requerimientos no funcionales, reglas del negocio, soluciones sugeridas e información no atinente.
  + Asignar partes de los requerimientos de alto nivel a los componentes del software que se hayan definido en la arquitectura del sistema.
  + Comprender la importancia relativa de los atributos de calidad.
  + Negociar prioridades de implementación.
  + Traducir las necesidades del usuario recolectadas en las especificaciones de requerimientos y modelos.
  + Revisar los requerimientos documentados para asegurar una comprensión común de los requerimientos enunciados por los usuarios y corregir cualesquiera problemas antes de que el grupo de desarrollo los acepte.

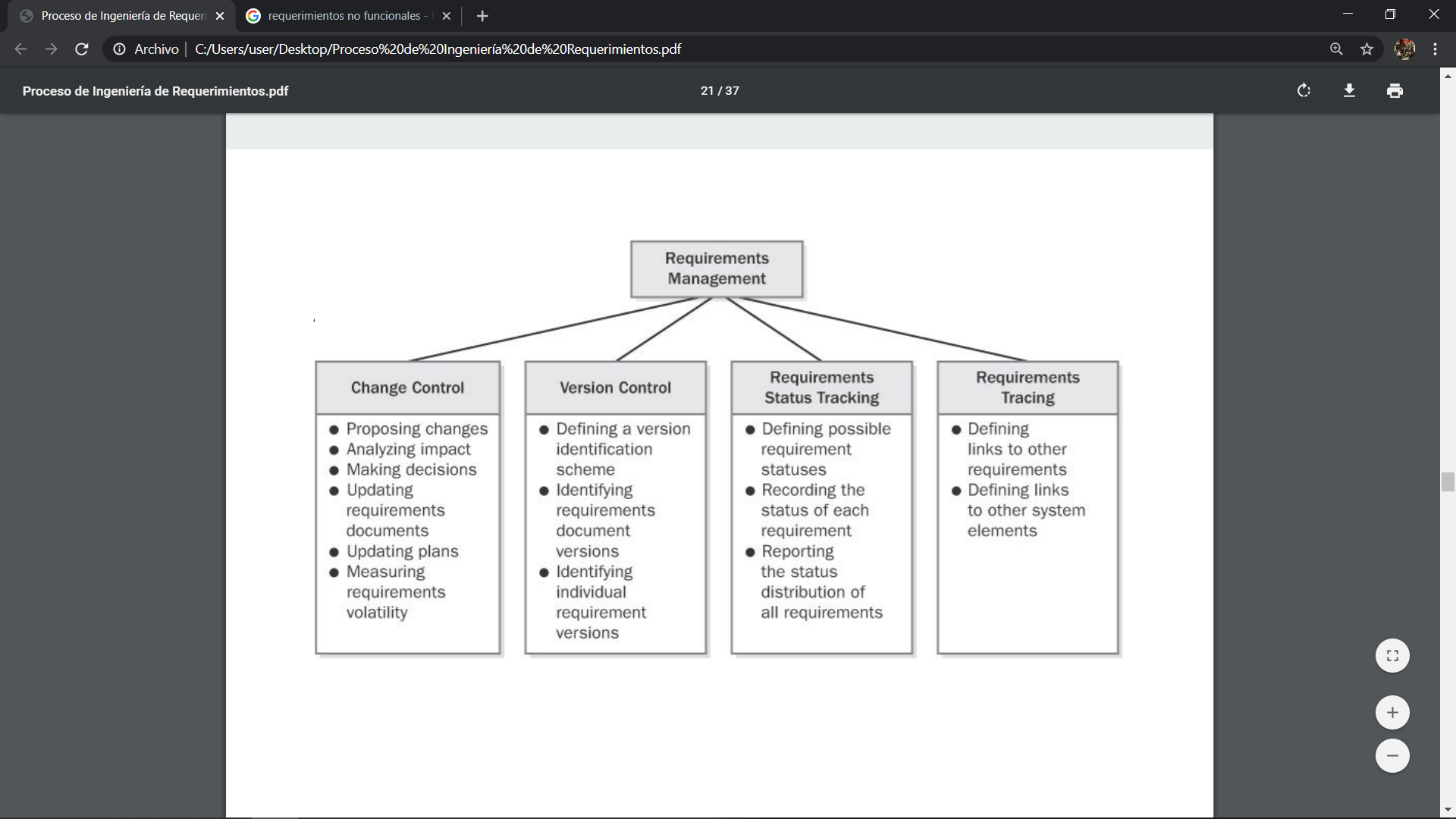


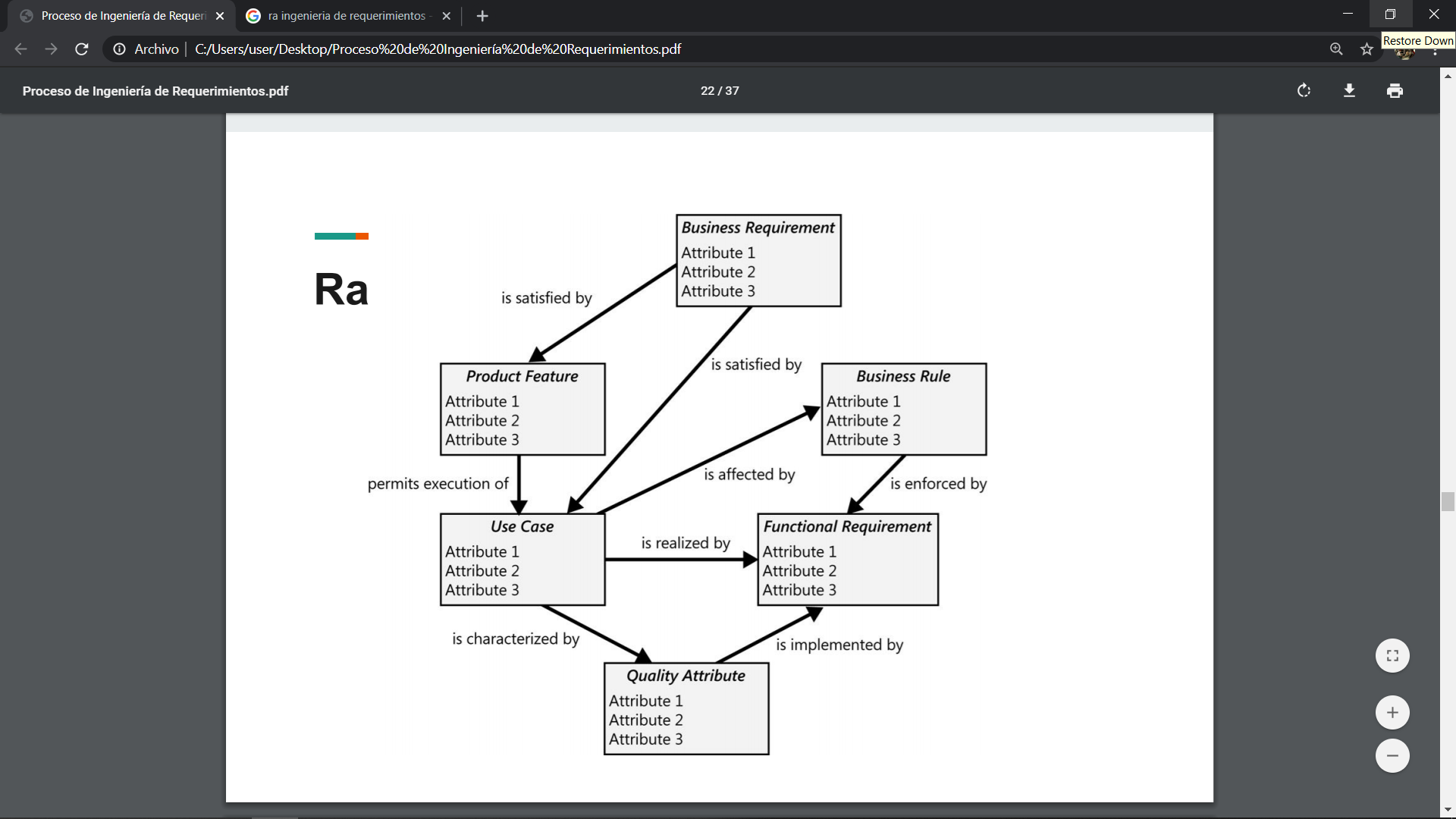




## Administración de requerimientos

* Es el proceso para manejar los cambios a los requerimientos de sistemas, para mantener un acuerdo con el cliente respecto de los requerimientos del proyecto de software.
* Definir una línea base para los requerimientos.
* Revisar los cambios propuestos a los requerimientos y evaluar su posible impacto antes de aprobar cada cambio.
* Documentar cambios aprobados de una manera controlada.
* Mantener los planes del proyecto actualizados con los requerimientos.
* Negociar nuevos compromisos con base en el impacto estimado de los cambios en los requerimientos.
* Rastrear (“trazar”) los requerimientos a las partes correspondientes de casos de uso, diagramas, diseño, código y casos de prueba.
* Dar seguimiento al estado de los requerimientos y su actividad de cambio a lo largo del proyecto.
* Verificar que se implementen correctamente los requerimientos.
* Construir un Comité de Control de Cambios.





## 10 trampas a evitar en el proceso

1. Confusión entre requerimientos.
2. Involucramiento inadecuado del cliente.
3. Requerimientos vagos o ambiguos.
4. Requerimientos sin priorizar.
5. Especificar funcionalidades que el usuario no utilizará.
6. Síndrome: análisis-parálisis.
7. Incrementos descontrolados en el alcance.
8. Proceso inadecuado de control de cambios.
9. Análisis del impacto insuficiente.
10. Control de versiones inadecuado.

## Beneficios de un buen proceso de IR[[1]](#footnote-1)

* Menos defectos de requerimientos.
* Re-trabajo de desarrollo reducido.
* Menos características innecesarias.
* Menores costos de mejoras.
* Desarrollo más rápido.
* Menos malentendidos.
* Alcance “creciente sigiloso” reducido.
* Caos de proyecto reducido.
* Estimaciones de pruebas del sistema más exactas.
* Mayor satisfacción de clientes y miembros del equipo de desarrollo.

## Verdades cósmicas sobre los requerimientos

* Cosmic Truth #1: If you don’t get the requirements right, it doesn’t matter how well you execute the rest of the project.
* Cosmic Truth #2: Requirements development is a discovery and invention process, not just a collection process.
* Cosmic Truth #3: Change happens.
* Cosmic Truth #4: The interests of all project stakeholders intersect in the requirements process.
* Cosmic Truth #5: Customer involvement is the most critical contributor to software quality.
* Cosmic Truth #6: The customer is not always right, but the customer always has a point.
* Cosmic Truth #7: The first question an analyst should ask about a proposed new requirement is, “Is this requirement in scope?”
* Cosmic Truth #8: Even the best requirements document cannot – and should not – replace human dialogue.
* Cosmic Truth #9: The requirements might be vague, but the product will be specific.
* Cosmic Truth #10: You are never going to have perfect requirements.

## Buenas prácticas de IR

Existen distintas áreas en las cuales se requiere realizar buenas prácticas de Ingeniería de Requerimientos y son las siguientes:

* Conocimiento
* Educción (adquisición)
* Análisis
* Especificación (documentación)
* Validación
* Administración de requerimientos
* Administración de proyectos

### Conocimiento

* Capacitar analistas.
* Educar a los representantes de los usuarios y a los administradores acerca de los requerimientos.
* Capacitar a los desarrolladores acerca del dominio de aplicación.
* Crear un glosario.

### Educción (adquisición, indagación)

* Definir proceso de desarrollo de requerimientos.
* Definir visión y alcance.
* Identificar clases de usuarios.
* Seleccionar “campeones” del producto (“paladines”). 🡪Usuarios clave
* Establecer grupos de enfoque.
* Identificar casos de uso.
* Identificar eventos y respuestas del sistema.
* Mantener talleres facilitados de educción.
* Observar usuarios realizar sus trabajos.
* Examinar informes de problemas.
* Reutilizar requerimientos.

### Análisis

* Dibujar un diagrama de contexto.
* Crear prototipos.
* Analizar factibilidad.
* Dar prioridades a los requerimientos.
* Modelar los requerimientos.
* Crear un diccionario de datos.
* Asignar requerimientos a sub-sistemas.
* Aplicar QFD (Quality Function Deployment = Desplazamiento de la Función de Calidad).

### Especificación (documentación)

* Adoptar una plantilla para la ERS.
* Identificar fuentes de los requerimientos.
* Etiquetar cada requerimiento de manera única.
* Registrar reglas del negocio.
* Especificar atributos de calidad.

### Validación

* Inspeccionar los documentos de requerimientos.
* Probar los requerimientos.
* Definir criterios de aceptación.

### Administración de requerimientos

* Definir un proceso de control de cambios.
* Establecer un Comité de Control de Cambios.
* Realizar análisis del impacto de los cambios.
* Hacer líneas base y controlar las versiones de los requerimientos.
* Mantener historia de los cambios.
* Dar seguimiento al estado de los requerimientos.
* Medir la volatilidad de los requerimientos.
* Utilizar una herramienta de administración de requerimientos.
* Crear una matriz de rastreabilidad de requerimientos.

### Administración de proyectos

* Seleccionar un ciclo de vida apropiado.
* Basar los planes en los requerimientos.
* Renegociar compromisos.
* Administrar riesgos de los requerimientos.
* Dar seguimientos al esfuerzo en los requerimientos.
* Revisar lecciones aprendidas del pasado.

## Implementación de las prácticas

|  |  |  |  |
| --- | --- | --- | --- |
| Impact | Difficulty | | |
|  | High | Medium | Low |
| High | - Define requirements development process.  - Base plans on requirements.  - Renegotiate commitments. | - Identify use cases.  - Specify quality attributes.  - Prioritize requirements.  - Adopt SRS template.  - Define change-control process.  - Establish CCB.  - Inspect requirements documents.  - Allocate requirements to subsystems.  - Record business rules. | - Train developers in application domain.  - Define vision and scope.  - Identify user classes.  - Draw context diagram.  - Identify sources of requirements.  - Baseline and control versions of requirements. |
| Medium | - Educate user reps and managers about requirements.  - Model the requirements.  - Manage requirements risks.  - Use a requirements management tool.  - Create requirements traceability matrix.  - Hold facilitated elicitation workshops. | - Train requirements analysts.  - Select product champions.  - Establish focus groups.  - Create prototypes.  - Define acceptance criteria.  - Perform change impact analysis.  - Select appropriate life cycle. | - Analyze feasibility.  - Create a glossary.  - Create a data dictionary.  - Observe users performing their jobs.  - Identify system events and responses.  - Unique label each requirement.  - Test the requirements.  - Track requirements status.  - Review past lessons learned. |
| Low | - Reuse requirements.  - Apply Quality Function Deployment.  - Maintain change history volatility. | - Measure requirements.  - Track requirements effort. | - Examine problem reports. |

# Principios de Diseño de Software

## Definición

* Proceso por el cual un agente crea una especificación de software, dirigido a un objetivo, por medio de un conjunto de componentes primitivas y restricciones.
* Se refiere a “todas las actividades de conceptualización, alcances, implementación, asignación y modificación de sistemas complejos”.
* Actividades entre la especificación de requerimientos y la programación.
* Usualmente involucra la solución de un problema y la planificación de una solución de software.
* Diseño de componentes de bajo nivel y algoritmos.
* Diseño de alto nivel y arquitectura.

## Generalidades

* El diseño de Software es el proceso de visualizar y definir soluciones de software para uno o más conjuntos de problemas.
* Uno de los componentes principales es el software requirement analysis o análisis de requisitos de software (SRA o ARS).
* SRA es una parte del proceso de desarrollo de software que enumera las especificaciones utilizadas en la ingeniería de software.
* Si el software está semiautomatizado o está centrado en el usuario, el diseño del software puede involucrar el diseño de la experiencia del usuario para obtener un guion gráfico para ayudar a determinar esas especificaciones.
* Si el software está completamente automatizado (es decir, sin usuario o interfaz de usuario), un diseño de software puede ser tan simple como un diagrama de flujo o texto que describe una secuencia planificada de eventos.
* Hay métodos de semi-estándar como el lenguaje modelado unificado (UML) y los conceptos de modelado fundamental. En cualquier caso, alguna documentación del plan suele ser el producto del diseño.
* El diseño de Software puede ser independiente de la plataforma o específico de la plataforma, dependiendo de la disponibilidad de la tecnología utilizada para el diseño.

## Análisis vs Diseño

* La principal diferencia entre el análisis y el diseño de software es que la salida de un análisis de software consiste en problemas más pequeños para resolver.
* Además, el análisis no debe diseñarse de manera muy diferente entre los diferentes miembros del equipo.
* Por el contrario, el diseño se centra en las capacidades y, por lo tanto, pueden existir y existirán múltiples diseños para el mismo problema.
* Dependiendo del entorno, el diseño a menudo varía, ya sea creado a partir de marcos confiables o implementado con patrones de diseño adecuados. Los ejemplos de diseño incluyen sistemas operativos, páginas web, dispositivos móviles o incluso el nuevo paradigma de computación en la nube.

## Diseño de Software: ¿Proceso o Modelo?

* Es tanto un proceso como un modelo.
* Proceso: es una secuencia de pasos que permite al diseñador describir todos los aspectos del software para la construcción. La habilidad creativa, la experiencia pasada, el sentido de lo que hace que el software sea bueno y un compromiso general con la calidad son ejemplos de factores críticos de éxito para un diseño competente. Sin embargo, es importante tener en cuenta que el proceso de diseño no siempre es un procedimiento sencillo.
* Modelo: se puede comparar con los planos de un arquitecto para una casa. Comienza representando la totalidad de lo que se va a construir (por ejemplo, una representación tridimensional de la casa); lentamente, la cosa se refina para proporcionar una guía para construir cada detalle (por ejemplo, la plomería). Del mismo modo, el modelo de diseño creado para el software proporciona una variedad de vistas diferentes del software de la computadora.

## Antecedentes: Modelos iniciales

Las principales características de los modelos iniciales son:

* No se “pierde el tiempo” en la planificación, en la calidad, en los documentos que hay que realizar cuando se terminan etapas o en cualquier otra actividad que no sea de codificación.
* Documentación es el código fuente.
* Conforme se implementan cambios, se vuelve no-administrable.
* Dependencia del programador.
* Satisfacción del usuario es equivalente a tener pantallas.

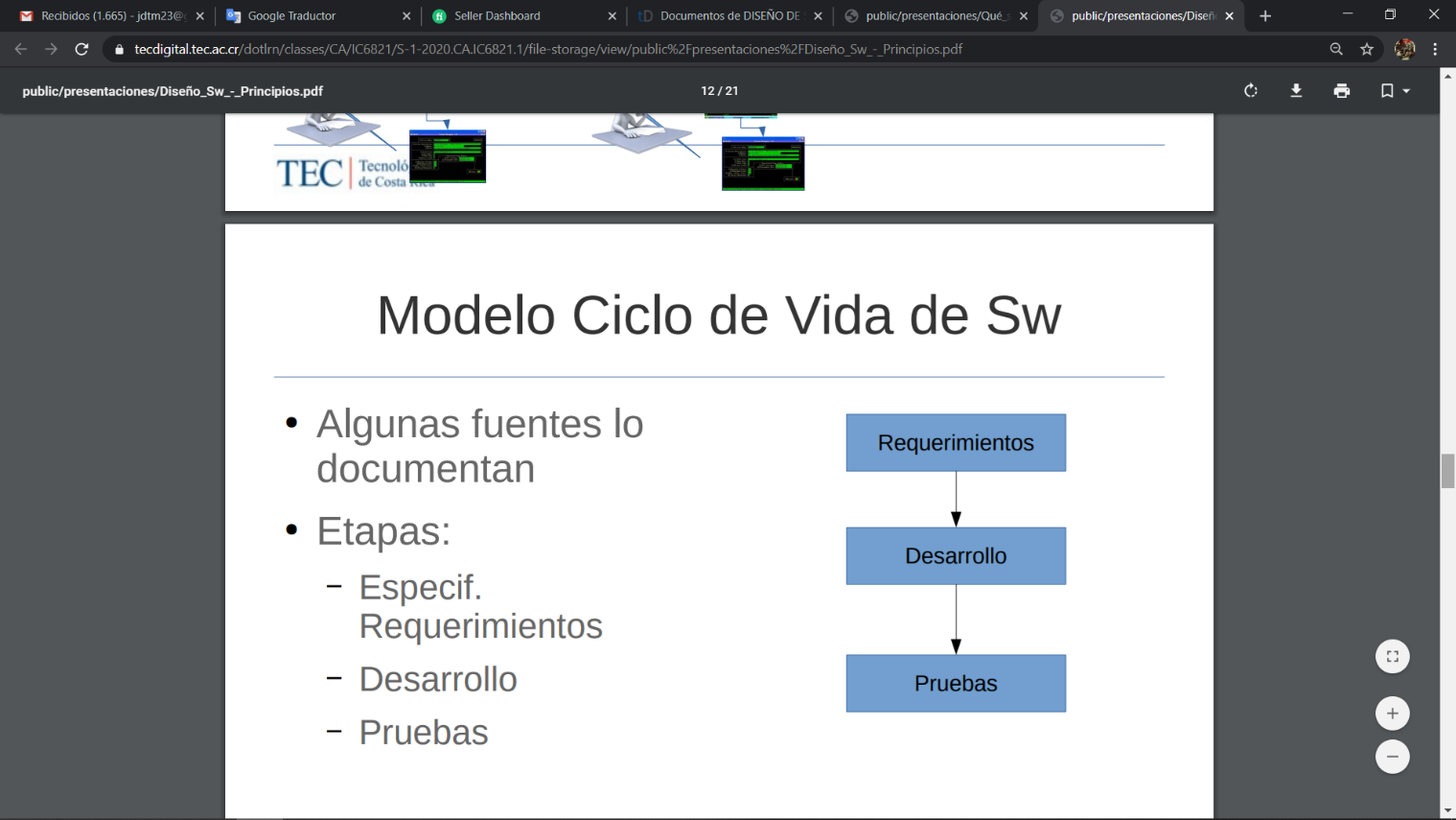
### Problemas

* Gran complejidad del software y su control
* Cambios estructurales del software son casi imposibles.
* Dependencia del programador 🡪 la rotación de personal.
* El software no satisface las necesidades reales del cliente.
* La calidad de software no fue contabilizada durante la creación del modelo.
* El software no es escalable ni administrable.
* El software fuera de plazo y con altos presupuestos.

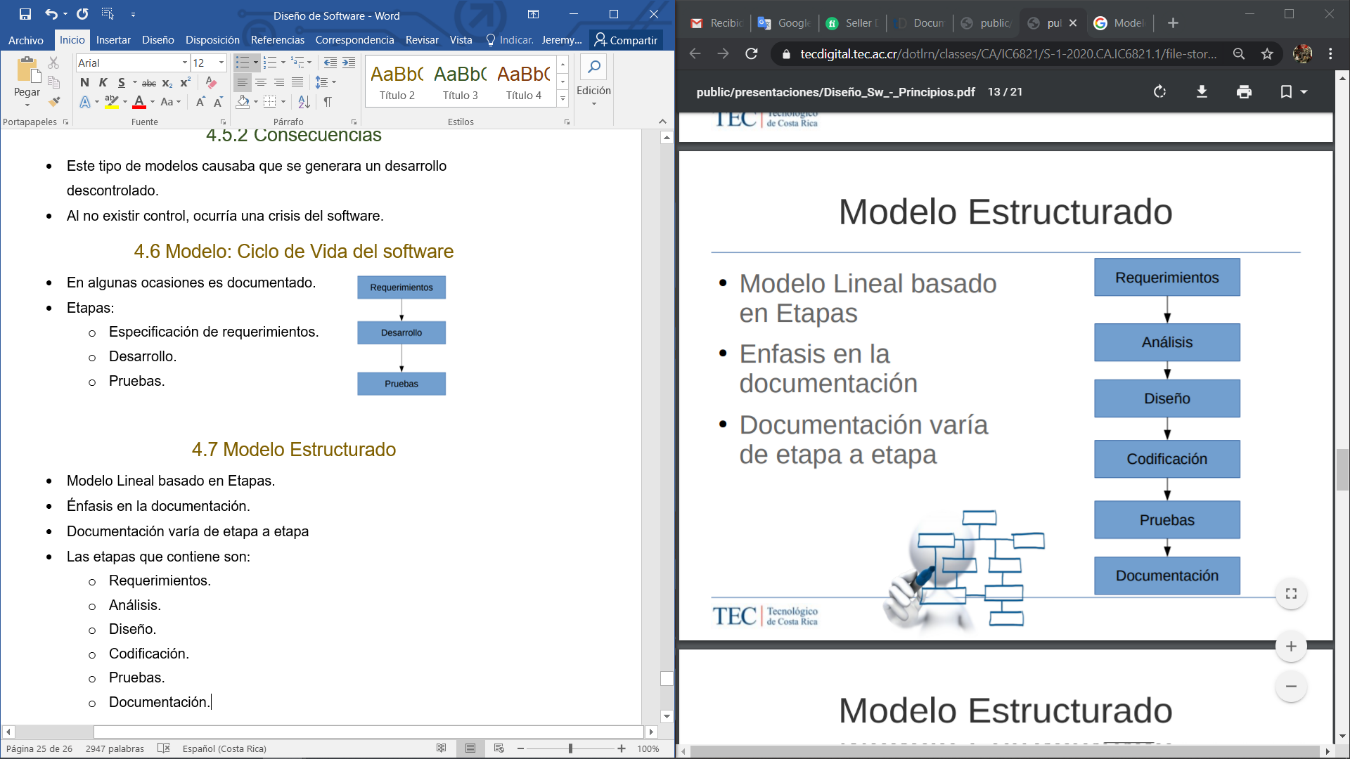
### Consecuencias

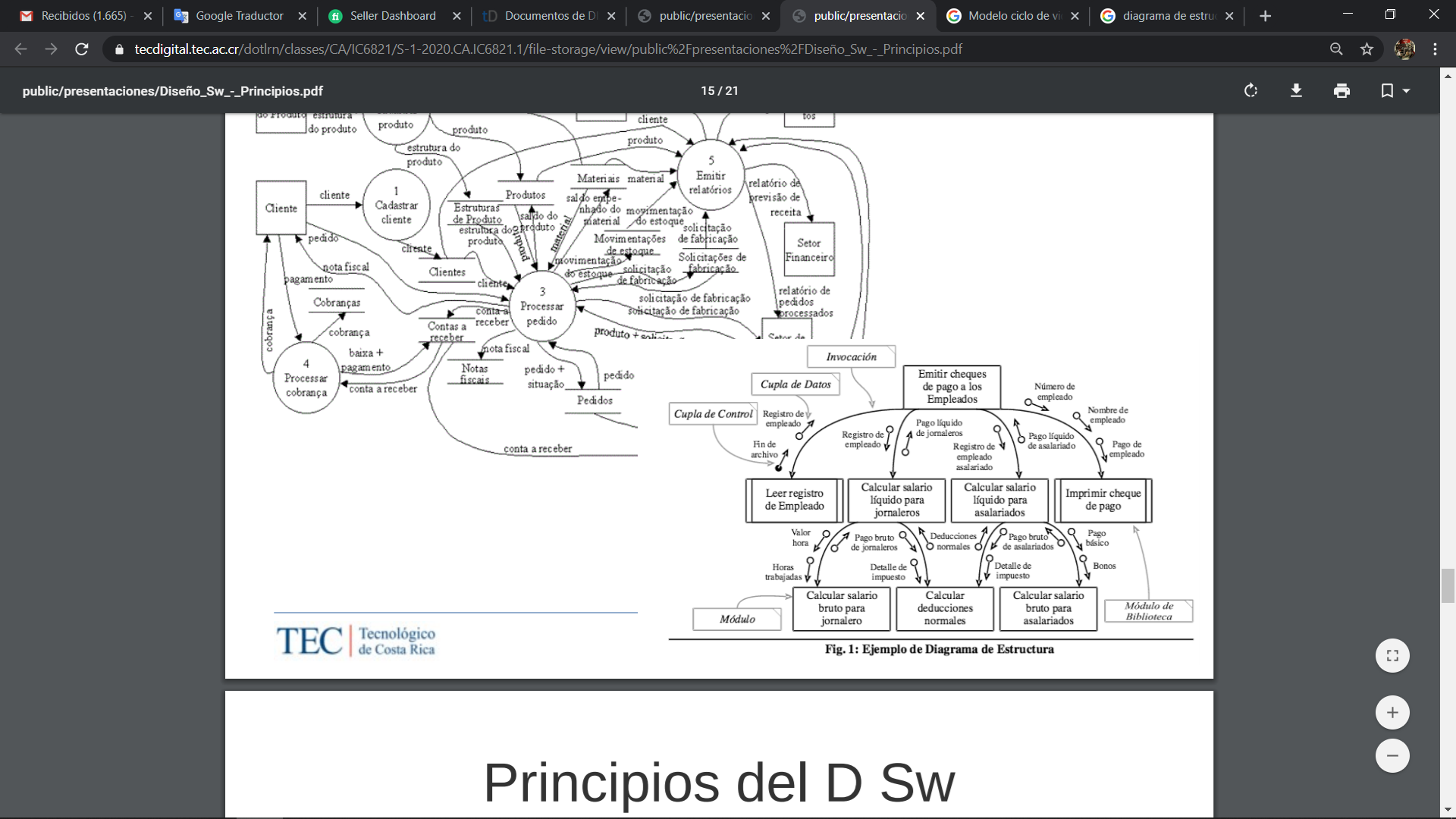
* Este tipo de modelos causaba que se generara un desarrollo descontrolado.
* Al no existir control, ocurría una crisis del software.

## Modelo: Ciclo de Vida del software

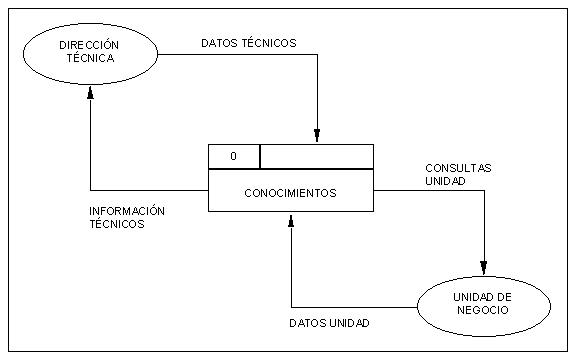
* En algunas ocasiones es documentado.
* Etapas:
  + Especificación de requerimientos.
  + Desarrollo.
  + Pruebas.

## Modelo Estructurado

* Modelo Lineal basado en Etapas.
* Énfasis en la documentación.
* Documentación varía de etapa a etapa
* Las etapas que contiene son:
  + Requerimientos.
  + Análisis.
  + Diseño.
  + Codificación.
  + Pruebas.
  + Documentación.
* Produce:
  + DFD: Data Flow Diagram.
  + Diccionario de Datos.
  + Diseño de BD.



**Diagrama de Estructura**



**Data Flow Diagram**

## Principios del desarrollo de Software

* No debe sufrir de “visión de túnel”: Un buen diseñador debe considerar enfoques alternativos, juzgando cada uno según los requisitos del problema, los recursos disponibles para hacer el trabajo.
* Debe ser trazable al modelo de análisis: Debido a que un solo elemento del modelo de diseño a menudo se remonta a múltiples requisitos, es necesario tener un medio para rastrear cómo los requisitos han sido satisfechos por el modelo de diseño.
* No debe reinventar la rueda: Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales probablemente se han encontrado antes. Estos patrones siempre deben elegirse como una alternativa a la reinvención. El tiempo es corto y los recursos son limitados; Se debe invertir tiempo de diseño en la representación de ideas (verdaderamente nuevas) mediante la integración de patrones que ya existen (cuando corresponda).
* Debe “minimizar la distancia intelectual”: entre el software y el problema tal como existe en el mundo real. Es decir, la estructura del diseño del software debe, siempre que sea posible, imitar la estructura del dominio del problema.
* Debe exhibir uniformidad e integración: Un diseño es uniforme si parece completamente coherente. Para lograr este resultado, se deben definir reglas de estilo y formato para un equipo de diseño antes de que comience el trabajo de diseño. Un diseño se integra si se tiene cuidado al definir las interfaces entre los componentes del diseño.
* Estructurado para adaptarse al cambio: Los conceptos de diseño discutidos en la siguiente sección permiten que un diseño logre este principio.
* Estructurado para degradarse suavemente: Incluso cuando se encuentren datos aberrantes, eventos o condiciones operativas. El software bien diseñado nunca debe “bombardear”; debe estar diseñado para adaptarse a circunstancias inusuales, y si debe finalizar el procesamiento, debe hacerlo de manera elegante.
* No es codificación, la codificación no es diseño: Incluso cuando se crean diseños de procedimientos detallados para los componentes del programa, el nivel de abstracción del modelo de diseño es más alto que el código fuente. Las únicas decisiones de diseño tomadas a nivel de codificación deben abordar los pequeños detalles de implementación que permiten codificar el diseño de procedimiento.
* Calidad del diseño debe evaluarse a medida que se crea, no después del hecho: Una variedad de conceptos de diseño y medidas de diseño están disponibles para ayudar al diseñador a evaluar la calidad durante todo el proceso de desarrollo.
* Revisarse para minimizar los errores conceptuales (semánticos): A veces hay una tendencia a enfocarse en minucias cuando se revisa el diseño, perdiendo el bosque por los árboles. Un equipo de diseño debe asegurarse de que se hayan abordado los principales elementos conceptuales del diseño (omisiones, ambigüedad, inconsistencia) antes de preocuparse por la sintaxis del modelo de diseño.

## Conceptos

* Abstracción: la abstracción es el proceso o el resultado de la generalización al reducir el contenido de información de un concepto o un fenómeno observable, generalmente para retener solo información relevante para un propósito particular. Es un acto de presentar características esenciales sin incluir los detalles de fondo o explicaciones.
* Refinamiento: Es el proceso de elaboración. Una jerarquía se desarrolla descomponiendo una declaración de función macroscópica de manera gradual hasta que se alcanzan las declaraciones del lenguaje de programación. En cada paso, una o varias instrucciones de un programa dado se descomponen en instrucciones más detalladas. Abstracción y refinamiento son conceptos complementarios.
* Modularidad: la arquitectura del software se divide en componentes llamados módulos.
* Arquitectura de software: se refiere a la estructura general del software y las formas en que esa estructura proporciona integridad conceptual para un sistema. Una buena arquitectura producirá un buen retorno de la inversión con respecto al resultado deseado del proyecto, en términos de rendimiento, calidad, cronograma y costo.
* Jerarquía de control: Una estructura de programa que representa la organización de un componente del programa e implica una jerarquía de control.
* Particionamiento estructural: la estructura del programa se puede dividir tanto horizontal como verticalmente. Las particiones horizontales definen ramas separadas de la jerarquía modular para cada función del programa. La partición vertical sugiere que el control y el trabajo deben distribuirse de arriba hacia abajo en la estructura del programa.
* Estructura de datos: es una representación de la relación lógica entre elementos individuales de datos.
* Procedimiento de software: se centra en el procesamiento de cada módulo individualmente.
* Ocultación de información: los módulos deben especificarse y diseñarse de manera que la información contenida en un módulo sea inaccesible para otros módulos que no necesitan dicha información.

En su modelo de objetos, Grady Booch menciona la abstracción, la encapsulación, la modularización y la jerarquía como principios fundamentales de diseño de software. El acrónimo PHAME (Principios de jerarquía, abstracción, modularización y encapsulación) a veces se usa para referirse a estos cuatro principios fundamentales.

## Consideraciones

* Compatibilidad: el software puede funcionar con otros productos diseñados para la interoperabilidad con otro producto. Por ejemplo, un software puede ser compatible con versiones anteriores de sí mismo.
* Extensibilidad: se pueden agregar nuevas capacidades al software sin cambios importantes en la arquitectura subyacente.
* Modularidad: el software resultante comprende componentes independientes bien definidos que conducen a una mejor mantenibilidad. Los componentes podrían implementarse y probarse de forma aislada antes de integrarse para formar un sistema de software deseado. Esto permite la división del trabajo en un proyecto de desarrollo de software.
* Tolerancia a fallas: el software es resistente y capaz de recuperarse de fallas de componentes.
* Mantenimiento: una medida de la facilidad con la que se pueden realizar correcciones de errores o modificaciones funcionales. La alta capacidad de mantenimiento puede ser producto de modularidad y extensibilidad.
* Fiabilidad (durabilidad del software): el software puede realizar una función requerida en las condiciones establecidas durante un período de tiempo específico.
* Reutilización: la capacidad de utilizar algunos o todos los aspectos del software preexistente en otros proyectos con poca o ninguna modificación.
* Robustez: el software puede funcionar bajo tensión o tolerar entradas impredecibles o inválidas. Por ejemplo, se puede diseñar con resistencia a condiciones de poca memoria, pérdida de conexión a internet, entre otros.
* Seguridad: el software puede soportar y resistir actos e influencias hostiles.
* Usabilidad: la interfaz de usuario del software debe ser utilizable para su público objetivo/audiencia. Los valores predeterminados para los parámetros deben elegirse para que sean una buena opción para la mayoría de los usuarios.
* Rendimiento: el software realiza sus tareas en un marco de tiempo aceptable para el usuario y que no requiere demasiada memoria.
* Portabilidad: el software debe poder utilizarse en diferentes condiciones y entornos.
* Escalabilidad: el software se adapta bien para el aumento de datos o al número de usuarios.

1. IR: Ingeniería de Requerimientos [↑](#footnote-ref-1)