Pontificia Universidad Javeriana, Cali
Ingeniería de Software
Profesora: Olga Lucía Roa

1. INGENIERÍA DE SOFTWARE

"La ingeniería de software es una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computadora."[6]

Desde la década del sesenta ha existido un movimiento tendiente a cambiar la programación de computadores de un estado artesanal a una disciplina similar a la ingeniería. El campo de la ingeniería de software surgió con el objetivo de introducir una técnica disciplinada al desarrollo de software.

Fritz Bauer estableció una de las primeras definiciones de ingeniería de software en 1969: "Ingeniería de Software es el establecimiento y uso de principios robustos de ingeniería, orientados a obtener software que sea fiable y funcione de manera eficiente sobre máquinas reales".[7]

Los puntos claves de esta definición pueden observarse en todas las disciplinas de ingeniería : principios robustos y productos económicos y confiables.

La ingeniería de software, al igual que otras ingenierías, debe trabajar con elementos gerenciales y humanos, además de los elementos técnicos propios. Sin embargo, a diferencia de las otras ingenierías, su producto, el software, es inmaterial. El desarrollo de software no puede, por tanto, ser manejado y controlado como otros procesos para productos físicos. El desarrollo de software es una actividad compleja por naturaleza.

La complejidad del desarrollo de software se ve agravada por el constante y acelerado avance tecnológico de la industria electrónica. Cada día los fabricantes son capaces de producir máquinas más económicas y poderosas. Esto posibilita desarrollar actualmente aplicaciones que no podrían realizarse cinco o diez años atrás. Las

---

[6] PRESSMAN, Roger S.. Ingeniería de Software : Un enfoque práctico. 3a edición. McGraw-Hill. España. 1993

[7] Cita de BAUER, Fritz tomada de NAUR, P y RANDELL, B (editores). Software Engineering: A report on a Conference sponsored by the NATO Science Comittee/ NATO. 1969 citada en MARTIN, James y McCLURE, Carma. Structured Techniques for Computing. Prentice-Hall. Englewood Cliffs, NJ, EE.UU. 1985

necesidades del usuario son, cada vez, más complejas. El software, en sí mismo, es cada vez más complejo. El proceso de desarrollo de software es, en definitiva, cada vez más complejo.

1.1 HISTORIA DE LA INGENIERÍA DE SOFTWARE

La ingeniería de software surgió de una serie de investigaciones en la década de los sesenta. Las primeras investigaciones al respecto buscaban hallar mejores mecanismos para escribir programas. Trabajos posteriores, como el análisis y diseño estructurado, comenzaron a presentar un visión más amplia del proceso. La disciplina se ha enriquecido con muchas investigaciones y avances tecnológicos desde esa época.

La ingeniería de software, durante toda su vida, ha estado "marcada" por tres enfoques o paradigmas principales :

- Los métodos orientados a procesos
- Los métodos orientados a la información
- Los métodos orientados a objetos

1.1.1 Los métodos orientados a procesos

Los comienzos de la ingeniería de software se hallan en la década de los sesenta cuando varios matemáticos propusieron nuevos mecanismos para la construcción de programas. En varios artículos de 1965 y 1968 Dijkstra planteó la necesidad de construir aplicaciones basadas en estructuras y no en instrucciones de salto (instrucciones GOTO). Dijkstra culpó a estas instrucciones de los diversos problemas de claridad, errores involuntarios y fallas en el desarrollo de software.

El mayor apoyo a tales teorías y, tal vez, el nacimiento de la Ingeniería de software como disciplina ocurrió en una conferencia organizada por la NATO en 1969, allí se congregaron varios de los principales científicos de computación y se planteó la necesidad de mejores métodos para el desarrollo de aplicaciones, cada vez más grandes y complejas.

A partir de allí varias técnicas surgieron para el mejoramiento del proceso de desarrollo. Una de las más interesantes fue el seudo-código, presentado por David Parnas en 1972. El seudo-código es considerado como el primer modelo de software abstracto, formal y revisable. A pesar de no ofrecer un alto nivel de abstracción fue el comienzo de varios métodos utilizados para la construcción de programas.

En mayo de 1974, Larry Constantine y dos alumnos suyos, Stevens y Myers, presentaron en una edición del IBM Systems Journal el artículo "Structured Design". En este artículo, Constantine presentó, por primera vez en la historia, una notación gráfica que permitía describir el comportamiento de los diferentes módulos de la aplicación.

A diferencia de las técnicas anteriores, Constantine estableció un mecanismo que no sólo permitía describir procedimientos o programas, sino que permitía describir varios módulos y las interrelaciones entre ellos.

Aunque no puede considerarse un método completo, Constantine estableció los primeros pasos en los métodos estructurados (orientados a los procesos). El artículo definió cuatro pasos para el desarrollo de cualquier programa:

1. Establecer un bosquejo del sistema definiendo la función básica que debe cumplir.
2. Identificar los "usuarios externos" del sistema, estableciendo claramente cuales son los datos de entrada y de salida del mismo.
3. Establecer las transformaciones conceptuales, que se efectúan sobre los datos, que sean más importantes para la solución del problema.
4. Definir las estructuras de código que realicen estas transformaciones.

Algunas investigaciones similares sobre mejores mecanismos para la construcción de aplicaciones se generaron por esa misma época: Myers profundizando sobre los diagramas de estructura (1974) y Jackson con su método de diseño (1975).

Sin embargo, el modelo planteado por Constantine tuvo una mayor acogida y fue ampliado y mejorado paralelamente por dos grupos de personas: D.T. Ross y K.E. Schoman con su método SADT (Análisis estructurado y Técnica de desarrollo) y Tom De Marco con SASS (Análisis estructurado y especificación de Sistemas).

Las adiciones al método de Constantine, especialmente las propuestas por DeMarco permitieron aplicar el método de análisis y diseño estructurado en sistemas y organizaciones cada vez más grandes y complejas. DeMarco introdujo el proceso de análisis de requerimientos, el uso del diccionario de datos y de las mini-especificaciones en seudo-código dentro del método de Constantine.

DeMarco y Constantine fueron contratados rápidamente por el investigador Edward Yourdon, quien construyó toda una industria basada en la definición y aplicación de métodos de desarrollo de software. Ed Yourdon es uno de los principales responsables de la evolución de los métodos de desarrollo durante los primeros años y de su difusión por todo el mundo.

Muchos de los desarrollos posteriores de los métodos fueron definidos por consultores de Yourdon Inc.: Ken Orr, Chris Gane, Trish Sarson, Paul Ward, Stephen Mellor. Ellos desarrollaban adiciones a los métodos y los adaptaban de acuerdo a las necesidades de las compañías que los contrataban.

El énfasis de los métodos en el análisis y diseño de los procedimientos de las aplicaciones fue, con el tiempo, perdiendo fuerza y campo de acción entre las empresas del mundo. El advenimiento de las bases de datos y los lenguajes de cuarta generación afectaron el proceso de evolución de los métodos de desarrollo. Algunos métodos estructurados, como el Gane/Sarson comenzaron a incorporar adaptaciones para el diseño de bases de datos. El uso de métodos orientados a los datos comenzarían a ser mucho más populares a medida que avanzaba la década de los ochenta.

Los métodos orientados a los procesos se basan en procesos de descomposición funcional de los sistemas. Estos métodos basan el estudio de la realidad y el modelamiento de los sistemas en los flujos de datos que ocurren en un sistema y en las transformaciones que padecen durante su camino.

1.1.2 Métodos Orientados a los datos

Casi de forma simultánea a como se fueron gestando los métodos estructurados orientados a los procesos, se fueron estableciendo otros métodos para la construcción de aplicaciones basados en las estructuras de los datos que deben manejar.

En 1974 se publicaron algunos estudios del matemático francés J.D. Warnier, quien planteó el diseño de las aplicaciones como una relación directa de las salidas que se esperan de ellas. Estos métodos fueron ampliados por Ken Orr en 1977 y 1981 para ser aplicados en gran variedad de situaciones. El método, en general, es conocido como método de Warnier/Orr.

De forma similar, en 1975, M. Jackson presentó las bases su metodología de desarrollo. Jackson planteó dentro de sus principios, la combinación de modelar el mundo real con modelar las funciones del sistema. Jackson, en lugar de modelar la realidad en función de procesos, como era tradicional, propuso modelar el mundo real en función de entidades, acciones y estructuras.

Las entidades, entendidas como elementos del mundo real que participan en la vida del sistema son la base de todos los análisis de Jackson. Tal información, brinda información importante sobre los datos que constituyen el problema del sistema. Los datos, según esta consideración, son los únicos elementos de las especificaciones que tienen una estructura objetiva y que pueden ser usados como una base lógica y racional para la estructura del programa, evitando utilizar una descomposición funcional que tiene un alto grado de juicios subjetivos.

El alto contenido formal de los métodos de Jackson y Warnier/Orr desestimularon su uso en la construcción de sistemas grandes o donde se requerían grandes esfuerzos de análisis. El estudio de las estructuras de los datos, en lugar de su flujo dentro del sistema, no fue muy utilizado por algún tiempo.

En publicaciones de 1976 Sam Chen presentó uno de los modelos de mayor influencia en el desarrollo de sistemas. Chen presentó un modelo que permitía organizar de forma adecuada los diferentes elementos de los datos y las relaciones existentes entre ellos, presentó por primera vez el modelo entidad-relación.

El modelo, expandido ampliamente por Chen y Flavin (1981), comenzó a ser utilizados para el diseño de bases de datos relacionales. Razón por la cual, el modelo comenzó a ser parte de gran variedad de métodos de desarrollo, incluyendo métodos de desarrollo orientados a procesos como los métodos de Yourdon.

A medida que se popularizaba el uso de los sistemas de bases de datos, las empresas comenzaron a descubrir que los métodos tradicionales no brindaban un soporte adecuado para la construcción de los sistemas. La duplicidad de información y de procesos de actualización comenzó a hacerse evidente y planteó la necesidad de métodos que definieran como uno de sus objetivos fundamentales la construcción de una base de datos a nivel corporativo.

Para 1981 James Martin y Clive Finkelstein presentaron el método de la Ingeniería de Software. El método basa el desarrollo de los sistemas en un proceso de planeación estratégica previo al proceso. Uno de los resultados de este proceso es un modelo de datos a nivel organizacional que debe ser refinado durante la construcción de cada sistema de la empresa.

El método es bastante orientado a los datos y se basa en la estructura de la información de la empresa para realizar la organización de las aplicaciones y la programación de los módulos. El método también manifiesta la necesidad de herramientas automatizadas para la construcción de las aplicaciones. (James Martin es considerado el padre de los lenguajes de cuarta generación y las herramientas CASE).

La ingeniería de la información, durante el análisis y el diseño se concentra en las necesidades de información y las estructuras de datos que soportan las operaciones de la compañía. La construcción de las aplicaciones se realiza, por lo general, utilizando sistemas de generación de código o lenguajes de cuarta generación.

Los métodos orientados a los datos se basan en la estructura de los datos que deben ser procesados en el sistema. Estos métodos establecen una estrecha relación entre la estructura de los datos y los mecanismos que actúan sobre ellos, por lo cual, basan su estudio en la comprensión de tales estructuras.

1.1.3 Métodos orientados al objeto

El enfoque de analizar sólo una parte del problema (procesos o datos), empezó a revaluarse a medida que los sistemas se tornaban más complejos, especialmente con sistemas distribuidos o con alta interacción con otros sistemas autónomos.

La tecnología orientada al objeto basa sus construcciones en objetos, una especie de paquetes que encapsulan datos y procedimientos como una unidad. Los objetos son unos mecanismos muy intuitivos para el modelamiento de la realidad, y fueron utilizados como tales en sistemas de simulación desde 1967. (Simula es un lenguaje de simulación basado en objetos).

En 1972 los objetos comenzaron a ser utilizados para la construcción de sistemas de software más complejos. En esa época el Laboratorio PARC de Xerox diseño y construyo el primer sistema de computador con ratón, interfaces gráficas y un entorno adecuado de desarrollo orientado a objetos: el SmallTalk.

Los primeros métodos de desarrollo orientado a objetos fueron apareciendo basados en experiencias de programación de varios autores, especialmente sobre lenguajes como SmallTalk y Ada. Entre los gestores de los primeros métodos se halla Edwards (1985) y Grady Booch (1986).

Grady Booch presentó inicialmente un método para el diseño de las clases de un programa basado en los conceptos de encapsulamiento y herencia existente en los objetos. Con el tiempo, presentó adiciones al método para brindar cabida a actividades adicionales de análisis y construcción de aplicaciones.

En la actualidad existen una gran variedad de métodos orientados a objetos, muchos de ellos generados por los mismos autores que presentaron métodos orientados a procesos o métodos orientados a datos. Entre la cantidad de métodos orientados a objetos podemos mencionar: Jacobson (Objectory), Rumbaugh (OMT), Coad/Yourdon, Martin/Odell, Shlaer/Mellor, Booch, HP Fusion y MOOSES entre otros.

Los métodos orientados a objetos basan su modelamiento en la identificación de los objetos del problema. Los objetos, entidades del mundo real, son estudiados y descritos en función de sus atributos (datos) y su comportamiento (procesos). Los objetos que son encontrados, se analizan y refinan para ser programados, de forma natural, en lenguajes orientados a objetos. Debido a que los objetos son la unidad durante todo el ciclo de vida del proyecto, estos métodos son bastante uniformes y consistentes.

1.2 PROBLEMÁTICA PRINCIPAL DE LA INGENIERÍA DE SOFTWARE

"Nosotros construimos software como los hermanos Wright volaban aeroplanos. Construimos la cosa por completo, la empujamos por una colina, la dejamos que se estrelle, y comenzamos de nuevo." [8]

La ingeniería de software busca encontrar mejores medios para construir software. Los medios tradicionales de construcción son, por lo general, costosos, y generan productos ineficientes y equivocados. Los nuevos medios de construcción deben ser más confiables, predecibles y capaces de entregar productos económicos y funcionales.

1.2.1 Fracaso de la Ingeniería Clásica

A pesar de los esfuerzos realizados por infinidad de investigadores y teóricos, la ingeniería de software no parece producir los resultados esperados. La ingeniería de software se ha centrado en la construcción de métodos que faciliten y mejoren el proceso de desarrollo de software. Sin embargo, muchos de las organizaciones que aplican tales métodos siguen teniendo problemas con el software que elaboran.

Algunos ejemplos exitosos de aplicación de la ingeniería poseen características casi ideales, muy diferentes a la mayoría de los proyectos de desarrollo de software. Algunos nuevos trabajos en la materia buscan evaluar cuál es el mejor método (metodología) que se puede aplicar en un proyecto específico y cuales son los factores de éxito y fracaso que se presentan en los proyectos reales.

Las herramientas automatizadas de apoyo a métodos de ingeniería de software, o herramientas CASE (*Computer Aided Software Engineering*), surgieron con fuerza a mediados de los ochenta como una posible solución a todos los problemas de las organizaciones con el manejo de los métodos de desarrollo. La verdad, sin embargo, es que muchas de tales herramientas no han funcionado, en muchos casos por un mal uso de los métodos en sí mismos o por limitaciones de las herramientas.

Algunos han definido como una "crisis" y una "aflicción crónica" los diversos problemas que se presentan en el desarrollo de software. De hecho, a pesar de todas las olas tecnológicas y técnicas que han aparecido en todos los tiempos de la informática, la "crisis" patológica que sufre el software no parece haber mermado.

---

[8] Cita de GRAHAM, R.M. tomada de NAUR, P y RANDELL, B (editores). Software Engineering: A report on a Conference sponsored by the NATO Science Comittee/ NATO. 1969 citada en MARTIN, James y McCLURE, Carma. Structured Techniques for Computing. Prentice-Hall. Englewood Cliffs, NJ, EE.UU. 1985

Para muchos, la concepción clásica de ingeniería de software ha fallado y se hace necesario crear nuevos estilos de atacar los problemas para poder solucionarlos.

1.2.2 Evolución de la Ingeniería de Software

A pesar de la "crisis" que parece padecer la ingeniería de software al no entregar los resultados esperados, la verdad es que sólo en los últimos años se han estado dando pasos firmes en pos de una nueva ingeniería, más realista, más concreta y capaz de lograr mejores resultados en el proceso de desarrollo de software.

La historia de la informática ha estado llena de "modas" o "caprichos" en materia de desarrollo de software. Cada vez que se lograba un avance tecnológico importante, se creía que el nuevo paradigma de trabajo podría solucionar los problemas. Cuando surgieron con fuerza las bases de datos, se le atribuyeron a ellas la solución a todos los problemas del software. Lo mismo ocurrió con las tecnologías de lenguajes de cuarta generación (4GL), la programación visual y la programación orientada a objeto.

Aunque los avances tecnológicos contribuyen a solucionar problemas del desarrollo de software, es real que sólo pueden contribuir con una parte de la solución. La ingeniería de software, y los desarrolladores, deben trabajar a un nivel más integral, correlacionando métodos, técnicas y herramientas, y logrando soluciones globales a los problemas del desarrollo.

Los últimos años, los expertos se han dado cuenta que no existen métodos o técnicas infalibles y que cada uno de los métodos sólo pueden ser aplicados con alto grado de éxito en determinados tipos de problemas. Las nuevas tendencias buscan aprovechar la experiencia que se ha obtenido y buscar nuevos tipos de soluciones.

Las nuevas soluciones implican, entre otros, nuevos criterios para la selección de métodos, nuevas teorías de selección de requerimientos, mejores mecanismos de control, mejores técnicas de detección y corrección de errores, mecanismos de métricas y nuevas técnicas para el mejoramiento continuo de los métodos y los departamentos de sistemas de las organizaciones.

Las técnicas requieren, cada vez más, de herramientas de apoyo cada vez más flexibles y versátiles. Una nueva generación de herramientas CASE está gestándose paralela a la nueva visión de ingeniería de software que se esta difundiendo.

La ingeniería de software puede ofrecer ahora un entorno y una visión más global del proceso de desarrollo y los problemas asociados a él. La ingeniería de software parece estar cada vez más cerca de su cometido.

Pontificia Universidad Javeriana, Cali
Ingeniería de Software
Profesora: Olga Lucía Roa

2. PROBLEMÁTICA DE LA INGENIERÍA DE SOFTWARE

2.1 OBJETIVOS DE LA INGENIERÍA DE SOFTWARE

La definición que Fritz Bauer elaboró de ingeniería de software en 1969, incluye una clara definición de los objetivos de la disciplina:

"Ingeniería de Software es el establecimiento y uso de principios robustos de ingeniería, orientados a obtener software que sea fiable y funcione de manera eficiente sobre máquinas reales".[9]

Como elemento aclaratorio, es favorable observar algunos objetivos primordiales, definidos por James Martin, para la programación estructurada. Tales objetivos pueden ser fácilmente atribuibles a la ingeniería de software. Según este criterio, los objetivos primordiales de la ingeniería de software (por lo menos algunos) son :

- Lograr programas de alta calidad de un comportamiento predecible
- Lograr programas que sean fácilmente modificables (y mantenibles)
- Simplificar los programas y el proceso de desarrollo de programas
- Lograr mejores predicciones y controles en el proceso de desarrollo
- Acelerar el desarrollo de sistemas
- Aminorar los costos del desarrollo de sistemas

Como puede observarse, los objetivos se centran en la obtención de mejores productos de software. La única forma en que tal objetivo puede lograrse, es mediante el estudio y mejoramiento de los procesos de desarrollo de software.

La ingeniería de software ha trabajado en pos de sus metas en los últimos 30 años. Sin embargo, muchos de los trabajos han sido investigaciones aisladas, sin cohesión, ni coherencia, con el resto de los adelantos logrados. En los últimos tiempos (desde

[9] Cita de BAUER, Fritz tomada de NAUR, P y RANDELL, B (editores). Software Engineering: A report on a Conference sponsored by the NATO Science Comittee/ NATO. 1969 citada en MARTIN, James y McCLURE, Carma. Structured Techniques for Computing. Prentice-Hall. Englewood Cliffs, NJ, EE.UU. 1985

principios de los noventa), los nuevos teóricos se hallan trabajando en brindar una visión más global y cohesiva a los términos y avances logrados en todo el tiempo de vida de la ingeniería.

2.2 EL PROCESO DE DESARROLLO DE SOFTWARE

Uno de los principales inconvenientes del ingeniero de software es el uso indiscriminado de algunos términos. Diversos expertos, en muchas ocasiones, presentan definiciones distintas para los mismos términos. Muchos de los términos fundamentales como método, modelo, sistema, ciclo de vida son utilizados como comodines, utilizados para nombrar una gran cantidad de conceptos y cosas diferentes. Esta situación crea confusión al comparar diversas metodologías y herramientas CASE, haciendo necesario que tales términos sean definidos claramente antes de realizar cualquier estudio consciente del tema.

El proceso de desarrollo de software es un proceso básico de transformación. En él se busca observar una porción de la realidad, modelarla de acuerdo a ciertos criterios y construir un sistema de software que soporte las actividades de esa realidad.

Cualquier actividad humana que busque transformar una porción de la realidad, tiene tres grande hitos de plasmación:

- El Original.
- El Modelo.
- El Sistema.

2.2.1 El Original

El Original es una porción de la realidad concreta y punto de partida “natural” de la actuación de transformación. El original debe ser captado por el actuador en forma de una o varias representaciones (complementarias o sucesivas) del mismo.

2.2.2 El Modelo

Cada representación abstracta del original, que cubre la misma porción de realidad, es llamada un modelo (modelo simbólico) de ese original. Su inmaterialidad es esencial. Presenta una serie de ventajas para el trabajo del actuador, especialmente economía, predicibilidad, seguridad, controlabilidad y/o diseñabilidad de alternativas.

Los modelos pueden ser de muchos variados tipos. Entre los posibles, tenemos :

- dinámicos o estáticos (si consideran o no sus variaciones con el tiempo)
- determinista o aleatorio (según el grado de incertidumbre asociado)
- descriptivo (descripciones de su composición, características, etc.)
- analítico (comportamiento ante los estímulos, estudios de costo, etc.)

Los modelos simbólicos han sido utilizados sistemáticamente en ingeniería por muchos años. Las razones de su uso extenso se hallan en la semejanza que ofrecen

con el original y la posibilidad de ser más manejables (en tamaño y/o complejidad) que éste (por descomposición/recomposición y generalización /especialización).

2.2.3 El sistema

La generación de posibles modelos sucesivos (cada uno siendo original del siguiente) desemboca en una última representación del original llamada sistema. El sistema es un modelo material, no simbólico, representación del original. El sistema es una nueva porción de la realidad concreta y es el punto de llegada "artificial" de la actuación de transformación.

La transitividad de las sucesivas semejanzas entre los diversos modelos intermedios, aseguran cierta semejanza, lo menos degradada posible, entre el original inicial y el sistema final, situados ambos en el espacio real. El grado de semejanza o diferencia puede ser también una elección del actuador de acuerdo al objetivo final que busca.

2.2.4 Los Métodos

Un método es la secuencia de modelados que ayudan a construir, a partir del original de la realidad, una o varias cadenas de modelos, derivados unos de otros, con el objetivo de lograr un modelo material final o sistema que concrete la nueva porción de la realidad deseada con relación con el original.

En el sentido estricto, metodología es el discurso, ciencia o estudio de los métodos. Sin embargo, dentro de ingeniería de software se denomina metodología también a los métodos. En el sentido estricto, por ejemplo, la expresión "metodologías orientadas a objeto" es incorrecta y debería ser escrita como "métodos orientados a objeto". En el presente texto, los términos se usan indistintamente (incluyendo el mal uso del término metodología) para evitar confusiones con otros textos de temas similares.

2.2.5 Espacio del Problema y Espacio de soluciones

Los teóricos de ingeniería de software utilizan el concepto de espacio para explicar varios conceptos relacionados con sus teorías. Un espacio es considerado en ingeniería, al igual que el concepto matemático del mismo nombre, como un conjunto de valores de diversas variables.

El "espacio del problema" es una porción de alguna parte del mundo real que "encasilla" el original que se desea transformar.

El "espacio de soluciones", por su parte, es un conjunto de combinaciones de hardware y software que representan una solución al problema. Cada combinación constituye un sistema, producto final de la transformación del original.

En muchas ocasiones el "espacio de soluciones" se halla distante del "espacio problema" de la realidad. Para hallar el "espacio de soluciones" se hace necesario

hacer una transformación mental o física a representaciones abstractas del mundo real. La abstracción ayuda a mantener y entender los sistemas, reduciendo los detalles que un ingeniero necesita manejar para conocer el sistema a un nivel determinado.

El espacio del problema también es llamado “espacio - problema” o “dominio del problema”. La expresión “dominio del problema” supone que el espacio de soluciones se genera como una función del espacio problema.

2.3 MÉTODOS DE INGENIERÍA DE SOFTWARE

La ingeniería de software se ocupa del proceso de desarrollo de software: Un proceso de transformación de la realidad. El estudio se basa, entonces, en los diferentes métodos existentes para lograr tal transformación.

Para lograr su cometido, la ingeniería de software estudia los problemas concernientes al desarrollo basado en tres elementos claves :

- Los Métodos
- Las Técnicas
- Las Herramientas

2.3.1 Los Métodos.

Los métodos organizan de manera racional diversas técnicas y herramientas para lograr el proceso de desarrollo del software. Los métodos definen la secuencia en la que se aplican las técnicas, las entregas (documentos, informes, reportes, etc.) que se requieren, los controles que ayudan a asegurar la calidad y las directrices que ayudan a los gerentes a evaluar el progreso de las actividades.

Los métodos son comúnmente llamados por los expertos como metodologías. Otros autores, como Roger S. Pressman, las definen como Procedimientos de desarrollo de software.[10]

Un método involucra por sí mismo la noción de secuencia de técnicas y modelos, y de ninguna forma debe ser correcto enmarcar el método tan sólo como una técnica como sugieren algunos autores.

Para completar su estudio de los métodos, la ingeniería de software debe aplicar un proceso metódico a los métodos, o sea, aplicar un proceso metametódico. Cómo resultado de ello, los métodos se observan como sistemas, con una serie de entidades o elementos relacionados entre sí y una estructura de procedimientos que generan los resultados deseados.

---

[10] PRESSMAN, Roger S.. Ingeniería de Software : Un enfoque práctico. 3a edición. McGraw-Hill. España. 1993

Los métodos (metodologías) usualmente definen y requieren los siguientes elementos:

- Los Modelos
- Las Tareas
- Las Entregas
- Los Roles y las Responsabilidades
- Las Heurísticas

2.3.1.1 Los Modelos

En el proceso de construir el sistema final o solución, los métodos (las metodologías) deben construir una serie de modelos simbólicos abstractos. Tales modelos pueden variar según los criterios utilizados. Cada método debe definir claramente cuales son los modelos que deben generarse y cuál es la notación que deberá utilizarse.

Es importante establecer la diferencia entre los modelos y la notación con que se representan. La notación sólo hace referencia a la forma de "escribir" o especificar gráficamente un modelo en particular.

Es posible que un mismo modelo pueda ser representado con varias notaciones diferentes. Por ejemplo, los modelos de entidad - relación, un modelo que representa las relaciones existentes entre entidades de datos, pueden ser representados por gran variedad de formatos. Las notaciones de Chen, Martin, IDEF1X son notaciones válidas que pueden ser utilizadas para representar un modelo de entidad - relación. Situación similar ocurre con los modelos de flujos de datos y de transición de estado.

Los modelos de ingeniería de software son, a menudo, referenciados en la literatura como "modelos de desarrollo de software" o SDM (*Software Development Models*).

2.3.1.2 Las Tareas

Cada método (metodología) debe especificar claramente las actividades que deben desarrollarse y un orden lógico ideal para ejecutarlas. Por lo general, cada una de estas tareas o actividades involucran técnicas de construcción o revisión de modelos o documentos a entregar.

2.3.1.3 Las Entregas

Las Entregas hacen referencia a los documentos que cada método (metodología) debe producir o alterar como resultado de completar las tareas. En muchas ocasiones las entregas incluyen documentos, manuales y modelos elaborados durante el proceso de desarrollo.

El término "entregas" también puede conocerse como "productos entregables", por su traducción literal del inglés *Deliverables*.

2.3.1.4 Los Roles y las Responsabilidades

Los métodos definen también una serie de roles para el personal involucrado en el desarrollo. La idea es suministrar criterios para determinar que personas deben cumplir cada una de las tareas y que se espera realicen cada una de las personas en el proceso de cumplir cada una de las tareas.

En ocasiones, al interior de la organización, el personal de desarrollo cumple más de un rol en el proceso de desarrollo. Por lo general, la aplicación real de estos criterios se basan en interpretaciones personales u organizacionales de los mismos.

2.3.1.5 Las Heurísticas

Las heurísticas hablan de los mecanismos que se establecen como guías para tomar decisiones sobre las tareas del proceso que se van cumpliendo. Aunque por lo general son reglas empíricas y muy poco documentadas, buscan calificar si las labores cumplidas se han hecho satisfactoriamente o no.

Las heurísticas ayudan a determinar si una tarea o modelo se ha completado de manera correcta y, en ocasiones, permite tomar "atajos" en la culminación de los mismos. Sin embargo, las heurísticas no constituyen reglas definitivas, garantías de éxito.

"Estas ayudas adicionales toman la forma de heurísticas del modelo : métodos que tienen una historia razonables de éxito pero que no lo garantizan."[11]

Aunque no existe una clasificación clara de las heurísticas, estas pueden ser de dos tipos: guías para decisión en la ejecución de las actividades (especialmente criterios de abstracción) y reglas de claridad semántica para los modelos.

2.3.2 Las Técnicas

Cada una de las tareas del método (metodología) se apoyan en procedimientos que prescriben la forma de ejecutarla. Estos procedimientos son el vehículo de comunicación y trabajo entre los conocedores de la realidad original, sus modeladores y los usuarios del sistema final.

Por lo general cada procedimiento obtiene resultados a partir de los resultados de los otros procedimientos previos. Tales productos son, por lo general, entregas o modelos. El conjunto de resultados finales o productos, si son válidos y consistentes, constituyen el sistema deseado.

Las reglas para ejecutar los procedimientos se establecen en las técnicas. Las reglas que definen las técnicas pueden ser textuales o mezclas de reglas diagramáticas y

---

[11] WARD, Paul T. y MELLOR, Stephen J. Structured Development for Real-Time Systems. Yourdon Press. Englewood Cliffs, NJ, EE.UU. 1985

algorítmicas. Por ejemplo, la definición de diagramas de flujos de datos (DFD) es un procedimiento para representar requisitos o funcionalidades de un sistema, con una técnica de diagramación visual, compacta y no tan ambigua como el texto. Los DFD forman parte integral de varios métodos de desarrollo de software (Los diagramas de flujo de datos (DFD) realmente son un modelo, no sólo una notación o diagrama, y deben considerarse como tal).

A menudo las técnicas son confundidas con los métodos (por ejemplo, Roger Pressman define las técnicas de ingeniería de software como métodos). Sin embargo, el significado formal de los términos discrimina las técnicas como puntuales y utilizables en diferentes entornos, y los métodos como una secuencia definida de técnicas y herramientas para el desarrollo de software.

Las técnicas abarcan una gran cantidad de actividades tales como estimación de proyectos, análisis de requisitos, diseño de estructuras de datos o de código, codificación, prueba, mantenimiento, etc.

2.3.3 Las Herramientas

Las herramientas son mecanismos que brindan soporte automático o semiautomático a las técnicas de los métodos. Las herramientas informátizan las técnicas diagramáticas, algorítmicas y textuales: no sólo formalizan documentaciones, sino que deben contener los mecanismos rutinarios formalizados para verificar las consistencias de las técnicas soportadas. Según el alcance de las técnicas que cubra, la herramienta será común a varios métodos o específica de alguno de ellos.

Cuando estas herramientas proveen un soporte real a varias técnicas y actividades de un método de desarrollo de software, conforman un sistema llamado ingeniería de software asistida por computador (CASE - *Computer Aided Software Engineering*). En la actualidad, sin embargo, se denomina comercialmente CASE a casi todas las herramientas de software que apoyan en alguna medida, alguna técnica de ingeniería.

Las herramientas CASE combinan software, hardware y bases de datos sobre el proceso de desarrollo de software, facilitando la labor del ingeniero durante todas las fases del proyecto y permitiéndole completar más fácilmente las diferentes técnicas y etapas de los métodos utilizados.

Pontificia Universidad Javeriana, Cali
Ingeniería de Software
Profesora: Olga Lucía Roa

3. CICLO DE VIDA DEL DESARROLLO

3.1 VIDA Y MUERTE DEL SOFTWARE

En uno de sus libros Alan Davis define la ingeniería de software como: "La ingeniería de software es la aplicación de principios científicos para (1) la transformación ordenada de un problema en una solución operativa de software y (2) el subsecuente mantenimiento de ese software hasta el final de su vida útil." [12]

Esta definición establece claramente que el ingeniero de software debe realizar una serie de actividades adicionales a la programación. Adicionalmente introduce un concepto conocido como "vida útil" del software.

Las actividades requeridas en el proceso de desarrollo, en ocasiones, se repiten reiteradamente durante toda la "vida útil" del software. Estas actividades conforman entonces un ciclo, el ciclo de vida del desarrollo del software.

Aunque podría pensarse que el software no se "estropea" o "envejece", es cierto que los sistemas de software sufren grandes transformaciones durante su tiempo de utilización y en muchas ocasiones es retirado de operación o modificado ampliamente luego de varios años de trabajo. El software tiene entonces, como un ser vivo, un tiempo de vida.

3.2 CICLO DE MUERTE DEL SOFTWARE

El ciclo de muerte propuesto por Rigby y Norris permite ilustrar claramente el concepto de vida útil del software. [13]

Norris y Rigby plantean el énfasis de los modelos tradicionales a preocuparse principalmente por las fases de construcción del software dejando a un lado el proceso de operación y mantenimiento de las aplicaciones. Curiosamente, las fases de

---

[12] DAVIS, Alan M. Software Requirements: Objects, Functions and States. Prentice-Hall. Englewood-Cliffs, NJ, EE.UU. 1993.

[13] NORRIS, Mark y RIGBY, Peter. The Software death cycle. Proc UK IT 90 Conference. Marzo 1990

operación y mantenimiento, por lo general, son mucho más largas y costosas que las etapas iniciales.

El ciclo de muerte del software se centra en los criterios para dar de "baja" un sistema: el costo/beneficio de mantener un sistema comparado con el costo/beneficio de construir uno nuevo.

La premisa básica que sostiene el modelo es que el mantenimiento de un sistema entregado cuesta dinero y tiene que ser compensado con los beneficios que se obtienen del software. En algún punto, el costo de posesión comienza a sobrepasar el beneficio y el neto se torna negativo. En este momento el software debe ser retirado de uso.[14]

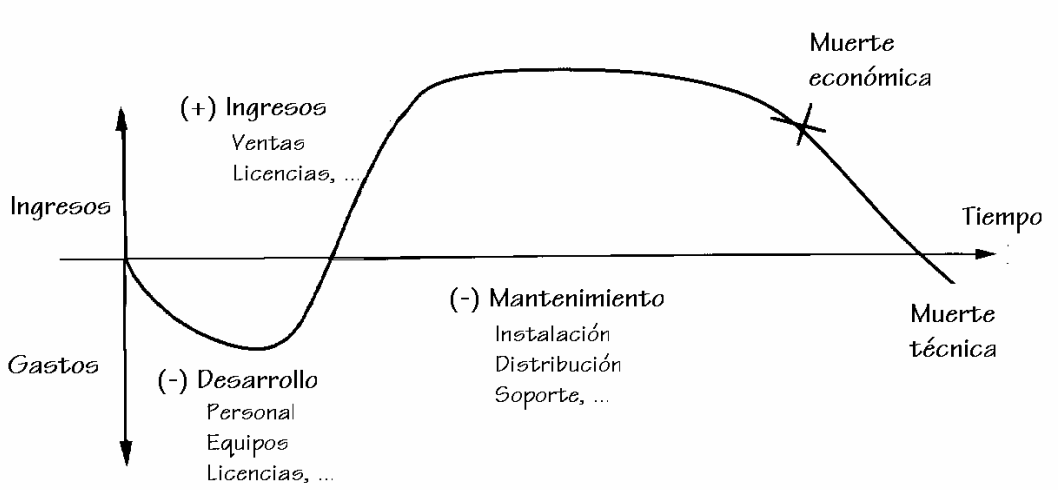

Figura 1. Ciclo de Muerte del Software.

3.3 CICLO DE VIDA DEL DESARROLLO

Tradicionalmente se ha utilizado el término "ciclo de vida del software" para caracterizan las diferentes actividades cíclicas que conforman el desarrollo de software.

Algunos expertos hablan de la mala utilización del término ciclo de vida dentro de la ingeniería de software: Realmente la vida del software no es cíclica, el comportamiento cíclico se presenta en los múltiples desarrollos que se llevan a cabo durante su vida útil. Un término más real y acertado es "ciclo de vida del desarrollo" o "ciclo del desarrollo de software".

---

[14] NORRIS, Mark y RIGBY, Peter. Ingeniería de Software Explicada. MegaByte-Noriega Editores. México. 1994

El ciclo de vida del desarrollo de software, presenta de manera organizada y secuencial, las actividades y procedimientos que deben realizarse en cualquier proyecto de desarrollo. Las fases del ciclo de desarrollo (ciclo de vida) deben completarse todas, sea cual sea el método de ingeniería que se emplee.

El ciclo de desarrollo de software es a menudo referenciado en los libros como ciclo de vida del desarrollo de software o SDLC (*Software Development Life Cycle*).

3.4 MODELOS DE CICLO DE VIDA DEL DESARROLLO

3.4.1 Modelo de Cascada

La noción de ciclo de desarrollo (ciclo de vida) del software y las fases y etapas que lo componen, han variado ampliamente durante los años. Uno de los primeros modelos de ciclo de vida del desarrollo fue establecido por W. Royce en 1970 y es conocido como el "modelo de cascada" (*waterfall model*).

El modelo de cascada es el primero en establecer el proceso de desarrollo como la ejecución de un conjunto de actividades. En su concepción básica, cada una de las actividades generan como salidas productos y modelos que son utilizados como entradas para el proceso subsiguiente. Lo cual supone que una actividad debe terminarse (por lo menos, en algún grado) para empezar la siguiente.

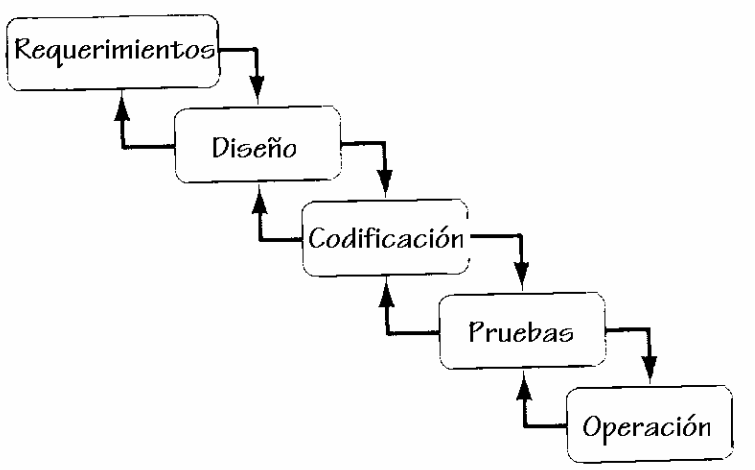

Figura 2. Modelo de cascada del Ciclo de vida del Desarrollo.

El modelo presenta una situación ideal para el proceso de desarrollo: un proceso con progresos ordenados y coherentes. Sin embargo, la idea de obtener tal progreso ordenado no es nada realista. En particular, el enfoque secuencial no permite el tratamiento de fenómenos reales como las relaciones con el personal, las políticas de las organizaciones, las incidencias de la competencia o la economía.

Adicionalmente, los analistas como seres humanos, son tendientes a cometer errores en las diferentes etapas, requiriendo que las etapas deban ser revisadas y repetidas en etapas posteriores para agregar mejoras al trabajo inicialmente imperfecto.

En términos generales el modelo en cascada:

- asume que todo va en una dirección
- no hay sobreposición ni iteraciones entre las fases
- implica una entrega al completar cada una de las fases
- asume que el conocimiento y requerimientos obtenidos en las fases iniciales del proyecto no cambia significativamente durante la vida del mismo.
- asume que no se cometen muchos errores en cada una de las fases

3.4.2 Modelo en espiral

En 1976 Barry Boehm propuso un nuevo modelo de ciclo de vida del desarrollo. El nuevo modelo es conocido como modelo de espiral y busca manejar los riesgos asociados al modelo de cascada. El modelo en espiral es, esencialmente, un desarrollo completo en cascada en cada iteración.

El modelo de Boehm es también conocido como “modelo evolutivo” o “modelo del caracol”.

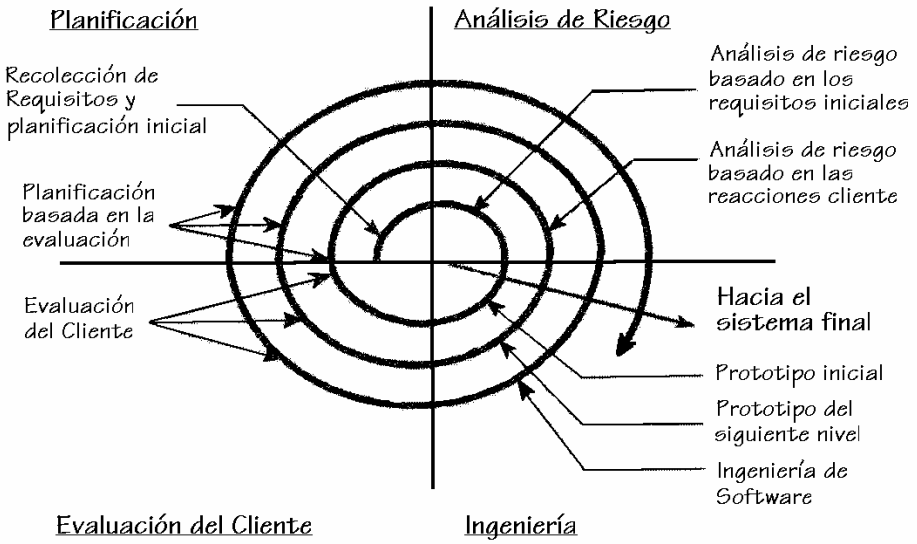

Figura 3. Modelo de Espiral de Ciclo de vida del Desarrollo.

En cada una de la iteraciones, se deben cumplir cuatro actividades principales (una en cada cuadrante) :

- Planeación : determinación de los objetivos, alternativas y restricciones

- Análisis de riesgo : análisis de alternativas e identificación/resolución de riesgos
- Ingeniería : desarrollo del producto hasta “el siguiente nivel”.
- Evaluación : valoración por parte del cliente de los resultados obtenidos.

El movimiento de la espiral, ampliando con cada iteración su amplitud radial, indica que cada vez se van construyendo versiones sucesivas del software, cada vez más completas.

Uno de los puntos más interesantes del modelo, es la introducción al proceso de desarrollo a las actividades de análisis de los riesgos asociados al desarrollo y a la evaluación por parte del cliente de los resultados del software.

El modelo en espiral maneja el concepto de versiones del sistema de software. Cada que se completa una versión del sistema de software, se vuelven a estudiar los requerimientos y el impacto del sistema sobre los mismos para crear una nueva versión del sistema. Este modelo es similar al manejo de productos comerciales que liberan versiones, cada vez más completas y complejas.

3.4.3 Modelo de prototipos

Una variación del modelo en espiral es utilizada por algunos nuevos métodos. Los nuevos métodos se basan únicamente en la construcción incremental de prototipos. En estos modelos no se genera un sistema completo en cada iteración, sino que cada iteración genera tan sólo un nuevo prototipo, cada vez más cercano al sistema final.

Estos nuevos métodos son conocidos actualmente como desarrollo rápido de aplicaciones o RAD (*Rapid Applications Development*). Este estilo fue promovido en 1985 por DuPont Co. (inventores del nylon y la lycra) mediante una metodología propia conocida como producción iterativa de prototipos o RIIP. James Martin realizó un libro posterior definiendo una metodología similar e introduciendo el término RAD.

El proceso iterativo de construcción de prototipos debe estar precedido por una fase de análisis y construcción del modelo conceptual. Posteriormente, cada una de las iteraciones deberá contar con una fase rápida de análisis y con la confrontación permanente de las necesidades del usuario.

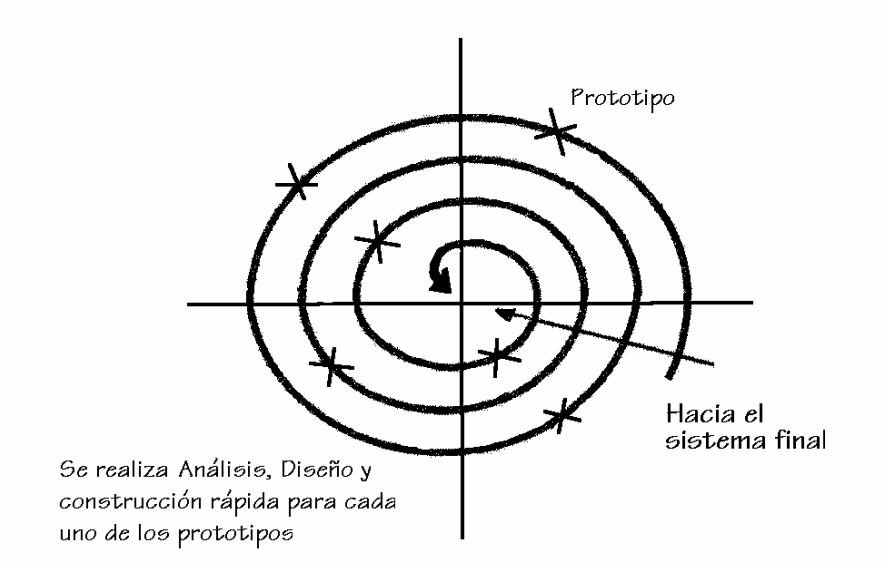

Figura 4. Modelo de Prototipos del Ciclo de vida de Desarrollo.

La construcción iterativamente de prototipos se representa diferente para enfatizar la tendencia hacía el sistema final ideal. (El modelo en espiral construye un sistema cada vez más completo y complejo en cada iteración).

El manejo de prototipos trae consigo, sin embargo, muchos nuevos problemas a ser considerados. De manera especial observaremos los problemas relacionados con las iteraciones mismas :

- ¿Cuantas veces es necesario iterar?
- ¿En que momento es “razonable” dejar de iterar?

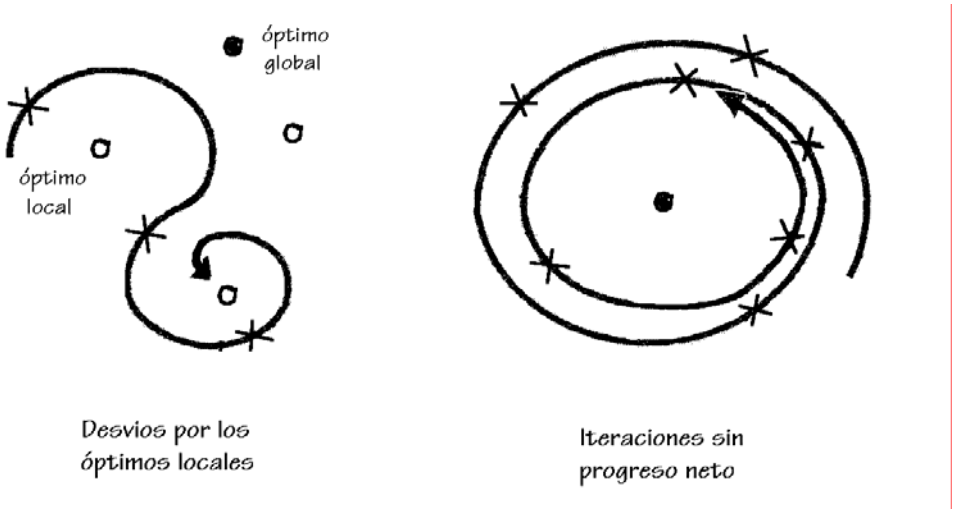

Figura 5. Problemas asociados al uso de prototipos.

Aunque el proceso ideal busca llegar a un sistema óptimo, puede ser que existan varios sistemas "óptimos locales" que desvíen constantemente el proceso de desarrollo y que no permitan observar avances significativos en las iteraciones. La principal causa de tales iteraciones perdidas esta en no poseer un buen modelo conceptual del sistema, lo cual lleva a los diseñadores a explorar en varias direcciones.

Otro problema puede estar en las iteraciones sin mejoras notorias. Establecer objetivos verificables de desarrollo puede ser la mejor forma de eliminar este inconveniente.

3.4.4 Modelo en "V"

Los modelos presentados, sin embargo, desconocen las simetría existente en varias etapas del ciclo de vida, especialmente las concernientes al proceso de revisiones y pruebas de los proyectos.

Algunos expertos han propuesto un nuevo modelo teniendo en cuenta tales aspectos. Uno de los más interesantes y completos es propuesto por Alan Davis como base para el estudio y aplicación de métodos.

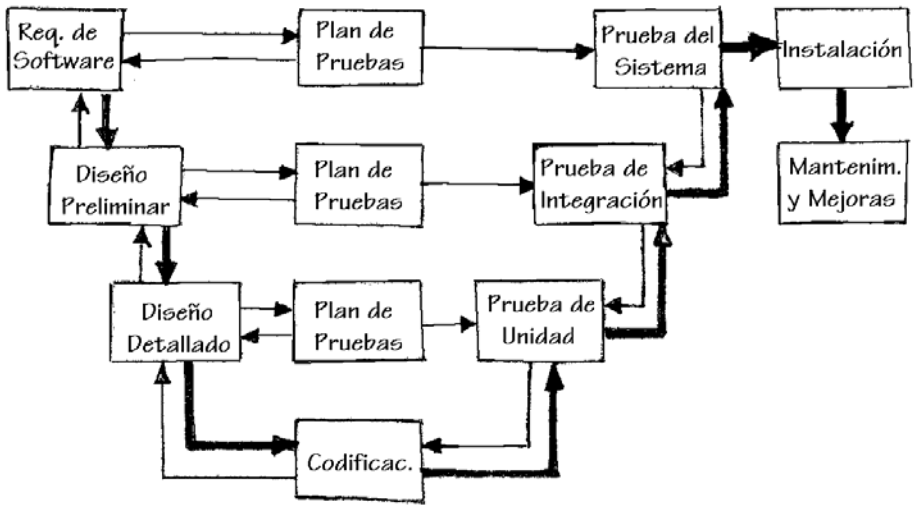

Figura 6. Modelo en "V" del Ciclo de Vida de Desarrollo.

Las principales consideraciones se basan en la inclusión de las actividades de planeación y ejecución de pruebas como parte del proyecto de desarrollo.

3.5 FASES GENÉRICAS DEL PROCESO DE DESARROLLO

Para facilitar el entendimiento de las diversas metodologías, los autores han preferido establecer varios pasos genéricos que caracterizan el desarrollo de software de manera independiente del método o el esquema de ciclo de vida utilizado.

De hecho, sea cual sea el ciclo de vida que se adopte, o la metodología que se use, las fases genéricas deben cumplirse. Este es el criterio con el cual fueron seleccionadas.

3.5.1 Modelo de cinco fases genéricas

Aunque algunos expertos presentan unas variaciones básicas, todo proyecto de desarrollo de software requiere de ejecutar cinco pasos fundamentales :

- Análisis
- Diseño
- Implementación
- Pruebas y corrección de defectos
- Operación y mantenimiento

3.5.1.1 Análisis

Comúnmente se denomina análisis a la etapa centrada en establecer el “qué” del software. Su preocupación básica es el problema a ser resuelto, las necesidades de los usuarios y las funciones que debe desempeñar el software. En definitiva lo qué debe hacer el software.

Algunos expertos, por ejemplo Edward Yourdon y Roger Pressman, diferencian el estudio de los requerimientos del análisis. Para los propósitos de esta división genérica, estos dos aspectos son cubiertos en una única fase.

3.5.1.2 Diseño

Esta etapa se centra en el “cómo”, en la forma cómo debe construirse el sistema de software de acuerdo a la información obtenida de la etapa de análisis.

Edward Yourdon define esta etapa como la construcción del modelo de implementación. Es decir, en esta etapa se define como deberá implementarse el sistema de software. Para Yourdon los modelos creados en la fase de análisis determinan claramente cuál debe ser el comportamiento general del sistema en un entorno ideal. Los modelos a crear en la fase de diseño determinan, ya sobre el entorno propio de la organización, cómo deberá implementarse el sistema.

3.5.1.3 Implementación

La implementación se establece como la “construcción” del sistema. La actividad sólo lleva a la práctica el sistema que se modeló en la fase de diseño. La fase incluye las actividades de codificación e integración de los diferentes módulos constitutivos del sistema.

3.5.1.4 Pruebas y corrección de defectos

El software generado en la fase de implementación no puede ser "entregado" a los clientes, para que funcione, sin practicarle antes una serie de pruebas. Las pruebas son tendientes a encontrar defectos en el sistema final debidos a omisión o mal interpretación de alguna parte del análisis o el diseño.

Los defectos deberán entonces detectarse y corregirse en esta fase del proyecto. En ocasiones los defectos pueden deberse a errores en la implementación de código (errores propios del lenguaje o sistema de implementación), aunque en esta etapa es posible realizar una efectiva detección de los mismos, ellos deben ser detectados y corregidos en la fase de implementación.

3.5.1.5 Operación y Mantenimiento

En esta fase el software es puesto en funcionamiento con los usuarios y el entorno de la organización. Es común que estas etapas requieran de trabajo adicional del equipo de desarrolladores, especialmente en procesos de configuración adecuada, utilidades menores de copias de respaldo, administración de seguridad o reportes adicionales.

Los desarrolladores pueden verse tentados a construir funcionalidades completas adicionales al software como parte del mantenimiento. Sin embargo, los métodos de ingeniería deben inculcar la no ejecución de tales tareas como parte de esta fase. Cualquier nuevo requerimiento para una nueva funcionalidad del software o su adaptación a un entorno diferente, debe ser cuidadosamente analizada y diseñada antes de ser implementada. En definitiva, cualquier adición o modificación sustancial del software deberá generar la ejecución iterativa de todas las fases del proceso nuevamente.[15]

3.5.2 Modelo de tres fases genéricas

Roger Pressman, define tan sólo tres pasos fundamentales. Aunque esta división es un poco extraña y es solamente utilizada varios expertos y teóricos, estas tres fases fundamentales pueden aclarar fácilmente el propósito de cada una de las fases de cualquier metodología y pueden ser útiles al momento de enseñar y comparar diversas técnicas.[16]

Las fases sugeridas por Pressman son :

- Definición
- Desarrollo
- Mantenimiento

---

[15] YOURDON, Edward. Análisis Estructurado Moderno. Prentice-Hall Hispanoamericana / México. 1993

[16] PRESSMAN, Roger S.. Ingeniería de Software : Un enfoque práctico. 3a edición. McGraw-Hill. España. 1993

3.5.2.1 Definición

Según las propias palabras de Pressman, la fase de definición se centra sobre el "qué". Es decir, esta fase busca identificar que información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, que restricciones de diseño existen y que criterios de validación se necesitan para definir un sistema correcto. En definitiva la etapa de definición debe identificar los requisitos clave del sistema y del software.

Aunque la fase de definición puede variar de acuerdo al esquema utilizado, esta fase incluye, básicamente, tres pasos específicos:

- Análisis del sistema.
- Planificación del proyecto de software.
- Análisis de requerimientos (requisitos).

Algunos autores denominan esta fase también como "fase pre-ejecutora". Esta denominación busca señalar que la fase no realiza, o ejecuta, ninguna modificación real al "espacio problema".

3.5.2.2 Desarrollo

La fase de desarrollo se ocupa del "cómo". Esto es, durante esta fase, el desarrollador de software intenta de descubrir cómo han de diseñarse las estructuras de datos y la arquitectura del software, cómo han de implementarse los detalles, cómo ha de traducirse el diseño a un lenguaje de programación y cómo han de realizarse las pruebas.

Básicamente la fase de desarrollo comprende tres pasos específicos:

- Diseño del software
- Codificación
- Prueba del software

Esta fase es denominada en algunos textos como "fase ejecutora", debido a que esta fase ejecuta la transformación del "espacio problema" y realiza la construcción del "espacio solución".

3.5.2.3 Mantenimiento

La fase de mantenimiento se centra en el "cambio" del software. Los cambios se asocian a la corrección de errores, a las adaptaciones requeridas por la evolución del entorno del software y a las modificaciones requeridas por el cambio de los requisitos del cliente dirigidos a reforzar o a ampliar el sistema.

Los cambios, y actividades, realizadas en la fase de mantenimiento son de tres tipos:

- Correctivos, tendientes a corregir defectos y errores.
- Adaptativos, tendiente a adaptar el software a los cambios del entorno.
- de Mejoramiento, tendiente a incorporar nuevas funcionalidades al software.

La fase de mantenimiento puede ser denominada en algunos textos como “fase post-ejecutora”. El término indica que esta fase agrupa las labores posteriores a la “ejecución del espacio solución”. Para ser estrictos, esta fase debe incluir también labores como documentación, formación de usuarios, creación de utilidades, etc.

3.5.2.4 Comentarios al modelo de tres fases genéricas

Aunque el modelo de tres fases genéricas puede dar claridad sobre algunos aspectos del desarrollo de software. En muchos casos no corresponde a las técnicas y métodos de ingeniería utilizados en la vida real.

Aunque es cierto que existe una estrecha relación entre el proceso de diseño y la implementación del sistema final, casi todos los métodos y estudios definen solamente una alta relación entre las fases de análisis y diseño. En muchas ocasiones, la relación entre las fases de diseño y construcción es obviada o no tenida en cuenta por los teóricos.

Otro inconveniente asociado a esta división en fases se halla en situar las pruebas del software y los cambios correctivos como parte de fases diferentes. Es indudable que los cambios correctivos deben asociarse al establecimiento de pruebas de software. Aunque es cierto que, en la actualidad, muchos de los errores son detectados y corregidos durante el mantenimiento del software, los métodos de ingeniería deben buscar cambiar esta situación y hallar la mayor parte de los defectos utilizando técnicas de pruebas.

Sobre la fase de mantenimiento, es necesario definir que los cambios adaptativos y de mejoramiento involucran cambios en los requisitos del sistema. Los requisitos pueden cambiar, ya sea por cambios en el entorno, o por la búsqueda de mejoras en el sistema. La práctica de ingeniería debe buscar crear nuevas fases de definición y desarrollo para tales modificaciones. La mayoría de los autores establecen para esos tipos de mantenimiento una nueva iteración en el ciclo de vida, de manera similar al modelo de ciclo de vida en espiral.

3.6 ANÁLISIS Y DISEÑO

Con el fin de analizar una serie de factores que afectan el conocimiento de las diversas metodologías y técnicas de ingeniería de software, deberemos estudiar con más detalle las relaciones entre el análisis y el diseño del software.

Las fases genéricas de análisis y diseño son los temas que mayor estudio han tenido dentro de la ingeniería de software. Gran cantidad de documentación, teorías, técnicas y trabajos se han ocupado de tales aspectos. Es interesante comprobar, sin embargo, que los diferentes expertos y textos sobre el tema presentan, en muchas

ocasiones, conceptos y definiciones contradictorias de las labores y actividades a realizar en las etapas de análisis y diseño.

La definición más común es bastante vaga: el análisis se ocupa del "qué" y el diseño del "cómo". Para establecer una mayor claridad al respecto es necesario establecer una diferenciación más objetiva del "qué" y del "cómo" en el desarrollo de software.

3.6.1 Dilema del Qué versus el Cómo

A pesar que la definición de análisis y diseño, es muy similar para todos los expertos. Muchos de los autores confunden, omiten o difieren las actividades a realizarse en cada una de esas fases. El problema se debe, según Alan Davis, a la existencia de una paradoja (o dilema) sobre lo que significa "el qué" versus "el cómo".[17]

El dilema del "qué versus cómo" puede resumirse brevemente como "el cómo de una persona equivale al qué de otra", algo vagamente similar a "el piso de una persona es el techo de otra".

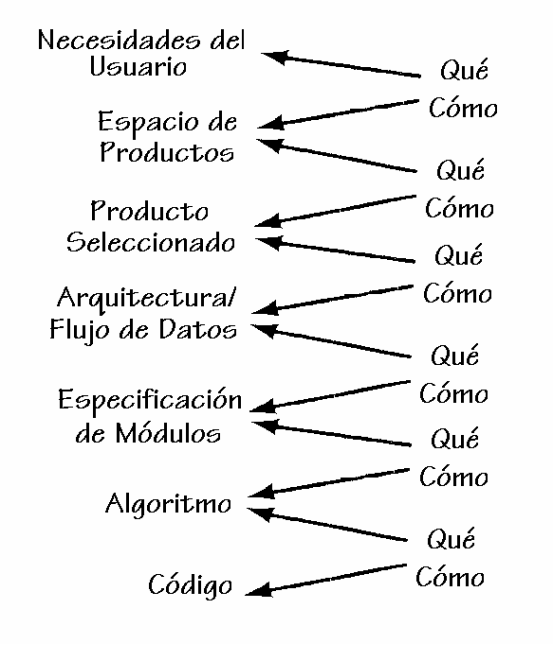

Figura 7. Dilema del Qué vs. el Cómo.

[17] DAVIS, Alan M. Software Requirements: Objects, Functions and States. Prentice-Hall. Englewood-Cliffs, NJ, EE.UU. 1993.

Para establecer claramente el problema a que hace referencia, observaremos unos cuantas situaciones :

- Los analistas podrían establecer claramente cuales son las necesidades del usuario. Este nivel es claramente una definición del qué debe efectuar el sistema y no del cómo debe hacerlo. Viéndolo así el siguiente paso podría ser la definición de todos los posibles sistemas (espacio de soluciones) que satisfacen tales necesidades.
- Por otro lado, podríamos establecer que el conjunto de los posibles sistemas solución sea una definición de lo que queremos que haga el sistema y no cómo debe ser construido o cómo debe comportarse. Visto así, el próximo paso, que define el cómo, podría ser el definir claramente cuál es comportamiento del sistema solución
- Sin embargo, el comportamiento general del sistema a construir podría ser visto como el qué, ya que no define cómo opera internamente. El próximo paso, que define el cómo, podría ser definir los componentes arquitectónicos principales del sistema.
- Podríamos entonces decir que la definición de los componentes arquitectónicos principales definen qué compone el sistema, pero no establece cómo trabaja cada uno de los componentes. El próximo paso, para definir el cómo, podría ser dividir los componentes principales en una serie de elementos menores que describan el comportamiento interno de cada uno.

Las situaciones y niveles podrían repetirse muchas más veces más. En definitiva es claro que, para cada una de las etapas, existe un qué y un cómo que difiere del qué y el cómo de las demás. No es posible definir, usando únicamente “qué” y “cómo”, las labores a realizar en las etapas de análisis y diseño.

3.6.2 Análisis de Requerimientos

El término correcto a emplear para definir la etapa de análisis es “análisis de requerimientos” ó “análisis de requisitos”. El objetivo principal de esta fase es la consecución, comprensión y explicación de los diversos requisitos para la construcción del sistema de software.

Los requisitos son, básicamente, todos los elementos y características que son requeridas, necesitadas o deseadas por el usuario. Existen dos tipos de requerimientos:

- Requerimientos propios del problema
- Requerimientos deseables para el nuevo sistema

Dentro de la etapa de análisis, existen dos tipos diferentes de actividades:

- Análisis del problema
- Descripción de la solución

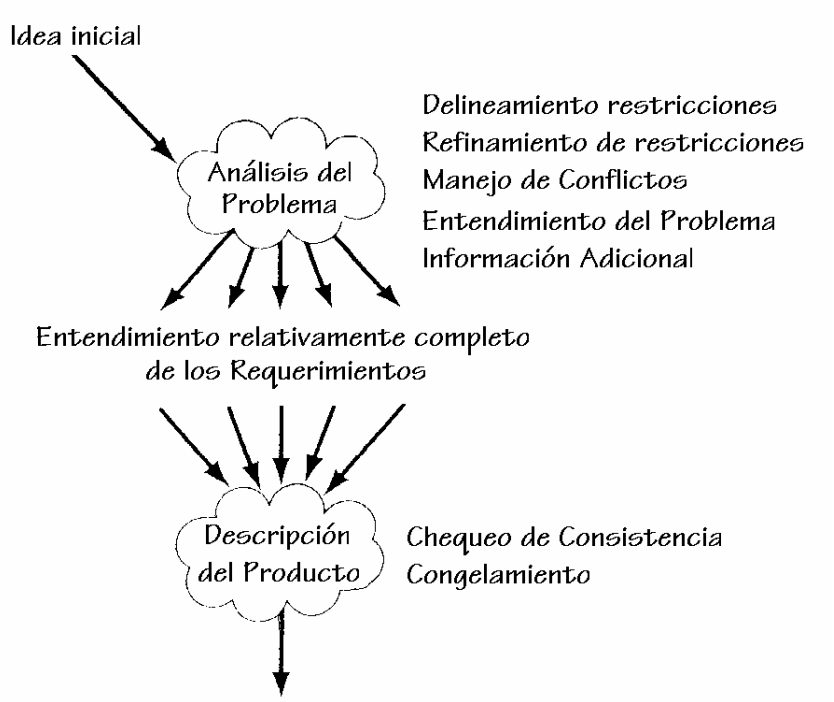

Figura 8. Esquema de funcionamiento de la fase de Análisis.

3.6.2.1 Análisis del problema

Durante esta etapa, los analistas ocupan su tiempo en el entendimiento del espacio problema. En esta actividad, los analistas deben trabajar y entrevistarse con el personal con mayor conocimiento del problema, que les permita identificar todas las posibles necesidades y restricciones sobre la solución del problema. Al culminar esta etapa, los analistas deben poseer un completo entendimiento del problema a solucionar.

Algunos desarrollos de software pueden requerir de muy poco, o ningún, análisis del problema. En particular, el análisis del problema sólo es necesario para problemas nuevos, difíciles o aún no resueltos.

3.6.2.2 Descripción de la solución

En esta etapa, el trabajo de los analistas se centra en describir el comportamiento externo que se desea del sistema a construir. Este comportamiento debe resolver claramente el problema que se conoció adecuadamente en la etapa anterior. En este momento se deben resolver conflictos de visión, eliminar inconsistencias y ambigüedades.

El producto final es una descripción general del comportamiento del sistema ideal (posible de ser realizado) que satisfaga el problema. Edward Yourdon lo define como el “modelo de la tecnología perfecta”.

A pesar que las dos etapas presentadas para el análisis de requerimientos poseen objetivos distintos, es muy poco probable que puedan efectuarse de manera secuencial en el tiempo. En muchas ocasiones las dos etapas se desarrollan en conjunto y se complementan unas a otras.

3.6.2.3 Importancia del Análisis de Requerimientos

Para el concepto de algunos desarrolladores, el análisis de requerimientos (y aún el diseño) son tareas innecesarias, que sólo agregan más trabajo y tiempo al proceso de desarrollo de software.

La importancia, sin embargo, del análisis de requerimientos esta sustentada en varias observaciones e hipótesis de ingeniería:

- Entre más tarde, dentro del proceso de desarrollo, se detecten los errores, mayor será el costo de corregirlo.
- Los errores en el análisis de requerimientos son, típicamente, incorrecciones, omisiones, inconsistencias y ambigüedades.
- Los errores de requerimientos pueden ser detectados más fácilmente.

El impacto de los errores en los requerimientos son substanciales:

- El software resultante puede no satisfacer las necesidades del usuario
- Es imposible probar la validez de un sistema si no se poseen los requerimientos que permitan establecerla.
- Si no se maneja una especificación válida de los requerimientos, se gasta tiempo y dinero en la construcción de un sistema errado.

La realidad es que son los requerimientos, y no otra cosa, lo que define claramente y sin lugar a dudas, cuál debe ser la funcionalidad del sistema a construirse. Los requerimientos constituyen la mejor forma de validar las diferentes etapas del desarrollo. Es posible probar, ya sea con el sistema o los modelos, si la solución que se esta construyendo satisface los requerimientos y si todos ellos están cubiertos en la solución.

Observando detalladamente el espacio de soluciones, el espacio esta determinado por un rango de soluciones que satisfacen las diferentes restricciones y requerimientos hallados en la etapa de análisis. Tales restricciones y requisitos pudieron definirse por los usuarios, los clientes, los desarrolladores, la tecnología disponible, las leyes, los estándares, etc.

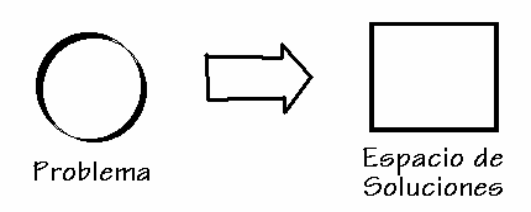

Figura 9. El espacio de soluciones esta determinado por el problema

Cada uno de los requisitos o restricciones delimitan el espacio de soluciones.

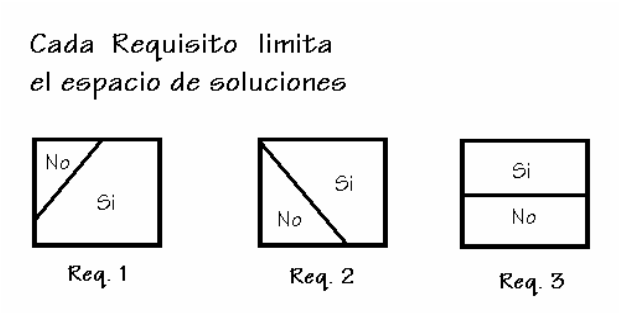

Figura 10. Los requisitos delimitan el espacio de soluciones.

A medida que se van incluyendo todos los requisitos o restricciones del problema (labor de la etapa de análisis), el espacio de soluciones se va reduciendo solamente a las soluciones “aceptables” o válidas para solucionar el problema.

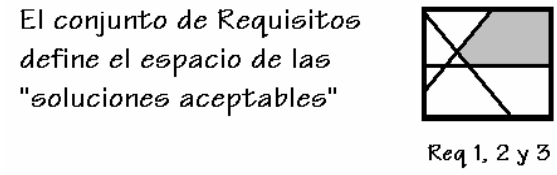

Figura 11. Las soluciones aceptables son definidas por los requisitos

En la etapa de diseño deberá entonces construirse un sistema que se halle dentro del espacio de soluciones aceptables o válidas que definieron los requerimientos.

3.6.3 Diseño y Especificación del Producto

La fase de diseño de un sistema de software se encarga de diseñar y solucionar uno de las soluciones “aceptables” encontradas en la etapa de análisis.

En muchas ocasiones los desarrolladores no son conscientes, ni se preocupan, de la existencia de más soluciones para ese problema. La verdad, se ocupan de crear un sistema que satisface los requerimientos y restricciones definidas con anterioridad.

Esto esta reforzado por una tendencia del mercado conocida como “software suficientemente bueno” (*good enough software*). La tendencia procura no concentrarse en buscar varias alternativas y seleccionar de ellas la mejor de todas, sino crear un software lo suficientemente bueno, que satisfaga todas las necesidades del usuario. En fin, un software que cumpla con todos los requerimientos y restricciones definidas.

La elaboración del diseño continúa siendo un proceso estructurado, de análisis descendente (*top-down*). Básicamente se toma la definición del producto que se obtuvo de análisis y se definen los elementos constitutivos principales de ese producto. Luego se toma cada uno de tales elementos y se descompone en elementos cada vez más pequeños. Como parte final, se definen los algoritmos, estructuras de datos y archivos que son requeridos para que los elementos menores funcionen adecuadamente.

Curiosamente, en la etapa de diseño no se han presentado cambios significativos en la forma de trabajo como si se ha presentado en la etapa de análisis.

Algunos expertos optan por dividir la fase de diseño en dos etapas :

- Diseño Preliminar o Arquitectónico : Donde se descompone el sistema en sus componentes principales, especificando de manera concreta su objetivo, función e interface.
- Diseño Detallado o Algorítmico : Donde se definen y documentan los detalles y algoritmos de cada una de las funciones (por lo menos las principales). También es conocido como diseño de programas.

3.6.3.1 Importancia del Diseño y Especificación del Producto

La fase de diseño ha sido, generalmente, bastante omitida en la aplicación práctica de los métodos de desarrollo. Algunos desarrolladores consideran que la especificación del producto generada como resultado de la fase de análisis es suficiente para comenzar la construcción de las aplicaciones. Aunque el proceso de construcción a partir de la fase de análisis puede ser fácilmente posible en sistemas pequeños, el proceso se hace más difícil a medida que el sistema se torna más grande y complejo.

La ejecución de la fase de diseño garantiza que el producto resultante cumplirá completamente con los requerimientos establecidos en la fase de análisis. Esta fase establece un mecanismo metódico para derivar la arquitectura del software a construir a partir de los requerimientos del usuario. La elaboración consciente de un proceso de diseño permite, adicionalmente, generar productos optimizados para la tecnología y plataformas de la compañía.

La ejecución de la fase de diseño puede traer consigo otras ventajas:

- Incorporación de estándares
- Integración con sistemas existentes (software, tecnología, etc.)

- Reutilización de código
- Consistencia en la implementación de funciones

Analizando las dos etapas de análisis y diseño, el proceso que se lleva a cabo en las dos etapas parece ser muy similar. Incluso parece una tendencia natural realizar algunas consideraciones de diseño mientras se efectúa la tarea de análisis. La razón principal de efectuar una diferenciación entre las dos etapas se halla en la obtención de la optimización: No es posible optimizar una solución si no se conoce claramente el problema que debe resolver.

Desde este punto de vista, las etapas de análisis y diseño tienen dos objetivos diferentes: al análisis busca entendimiento del problema y el diseño una solución optima al problema encontrado.

Fase	**Aspecto**	**Actividad Primaria**	**Meta Primaria**
Análisis de Requerimientos	Análisis del Problema	Descomposición	Entendimento
	Descripción de la solución	Descripción del Comportamiento	
Diseño	Diseño Preliminar	Descomposición	Optimización
	Diseño Detallado	Descripción del Comportamiento	

Tabla 1. Similitudes entre las fases de análisis de diseño.

3.6.4 Comparación de diversos métodos

Los términos Análisis y Diseño son, con mucha frecuencia, utilizados de manera no consistente entre los diferentes métodos (metodologías). A continuación se presenta una tabla que ilustra este punto.[18]

Actividad	**Davis (1993)**	**Berzins (1985)**	**Boehm (1976)**	**Freeman (83)**
Análisis del Problema	Análisis del Problema	Análisis de Requerimientos	Requerimientos	Análisis
Definición del Comportamiento Externo	Escritura del SRS (especificación)	Especificación Funcional		Especificación Funcional
Definición de los componentes constitutivos	Diseño	Diseño	Diseño	

[18] La tabla se basa en algunos trabajos de Alan Davis en la comparación de diversos métodos y en sus trabajos de análisis de requerimientos.

Actividad	**Kerola (1981)**	**Roman (1984)**	**IEEE (1984)**	**IEEE (1984)**
Análisis del Problema	Fase Pragmática	Definición del Problema	N/A	Análisis Requerimientos
Definición del Comportamiento Externo	Fase de Perspectiva de E/S	Diseño del Sistema	Requerimientos	Especificación
Definición de los componentes constitutivos	Fase Constructiva	Diseño del Software	Diseño	Diseño

Actividad	**DoD (1985**	**Ross (1977)**	**Wasserman (1986)**	**Yourdon (1989)**
Análisis del Problema	Requerimientos del Sistema	Análisis de Contexto	Análisis de Requerimientos	Modelamiento Esencial
Definición del Comportamiento Externo	Análisis de Requerimientos del Software	Especificación Funcional	Diseño Externo	Modelamiento de Implement. del Usuario
Definición de los componentes constitutivos	Diseño Preliminar	Diseño	Diseño Interno	Diseño

Tabla 2. Comparación de términos en varios métodos de desarrollo