UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

ESCUELA DE POSGRADO Y EDUCACIÓN CONTINUA

MAESTRÍA EN INGENIERÍA DE GESTIÓN EMPRESARIA

“Propuesta de Gestión para la Adaptación de una metodología de grandes empresas para las pruebas de software en PyMEs”

Ing. Melisa Argüello

Directora: Mg. Ing. Karina Cedaro

Co-director: Dr. Ing. Carlos Casanova

2020

RESUMEN

Frente a un contexto de crisis mundial, hoy más que nunca los productos de software forman parte de todos los aspectos de nuestra vida cotidiana, haciendo que las industrias que los producen ocupen un sector cada vez más importante tanto en la economía global como regional.

Ante esta demanda masiva, se ha comenzado a cuestionar cada vez más la calidad de estos productos y de los procesos que se llevan a cabo para producirlos. Distintas instituciones han desarrollado diferentes modelos y estándares para la mejora continua, focalizados principalmente en la etapa de pruebas del proceso de desarrollo. Es en esta proliferación de modelos de mejora, donde se detecta que no existe ninguno que sea aplicable a las PyMEs desarrolladoras de nuestra región, industrias donde serían de gran ayuda dado que se enfrentan desafíos como la falta de recursos, habilidades y experiencia en su búsqueda por crear software de calidad y sobrevivir en el mercado.

En este trabajo se presenta una propuesta de gestión que permite, a partir de un conjunto de métodos exitosos para las pruebas de software en grandes empresas, mejorar el proceso de pruebas para obtener productos de software de mayor calidad en las PyMEs de la región.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1: INTRODUCCIÓN

1.1 Descripción del problema

En los últimos años la industria del software, al igual que todo el sector de servicios digitales, creció de manera excepcional y se posicionó como uno de los más pujantes de la economía local. Es un “superávit” ininterrumpido, ya que aún frente a un panorama complejo en materia laboral, opera con pleno empleo y expansión de la demanda de personal.

Un estímulo importante que ha incentivado este crecimiento es la denominada Ley de Promoción de Economía del Conocimiento 27.506. La misma, recopila posiciones y necesidades del sector productivo local vinculado a la industria tecnológica en desarrollo en la Argentina, lo que incluye 5 “unicornios”, *clusters* tecnológicos en distintas provincias y un importante capital humano con formación técnica y profesional, entre otros. Además, toma como fuente las experiencias positivas que arrojaron las llamadas leyes de promoción de la industria del software 25.922 y 26.692, que habían establecido principalmente beneficios fiscales e incentivos en forma encapsulada y exclusiva a dicha industria (Politi, 2020).

Aunque su versión original fue suspendida a principios del año corriente, su versión modificada ya se encuentra en el Senado con media sanción de la Cámara de Diputados y, al contrario de otras propuestas, la situación de pandemia global ha impulsado esta Ley. Esto sucede, ya que en esta época de crisis global que afecta tantos aspectos de nuestra vida social y económica, la industria del conocimiento ha demostrado poder adaptarse mejor que otras, tomando un papel fundamental para enfrentar la recesión.

En este marco legal auspicioso que potencia el desarrollo de actividades tecnológicas, sumado a la nueva perspectiva sobre el valor de estas industrias, se ha comenzado a observar un fenómeno que amenaza el sano desarrollo de la industria del software y la consecución de las metas de crecimiento de esta. Este fenómeno, que se ve acentuado particularmente en las Pequeñas y Medianas Empresas (PyMEs) desarrolladoras de software, se trata de una tendencia a suprimir las instancias de verificación y validación de los productos de software. Este se puede observar tanto en la reducción de los perfiles destinados a las pruebas de aplicaciones como la eliminación de etapas de pruebas formales

(*testing*) en los procesos de desarrollo que se llevan a cabo, realizando en consecuencia grandes inversiones en soporte y mantenimiento posterior a las implementaciones (*customer service*).

Por un lado, dadas las características particulares del software como producto, existe una complejidad intrínseca en la detección de errores y realización de pruebas, independientemente del tipo de industria. Contar tiempo suficiente para probar todas las entradas posibles de una aplicación, evaluar sus salidas para validar instrucciones desarrolladas y detectar los ajustes necesarios, es algo que generalmente no sucede. A pesar de que hoy existen diversas herramientas que nos permiten automatizar las pruebas (test de *automation*), si las mismas no son diseñadas correctamente, sólo permiten evaluar parcialmente un desarrollo, ignorando algunas cuestiones claves del negocio como la integración con otros sistemas o un conjunto de entradas no esperadas que puede ingresar el usuario final. Planificar, ejecutar y evaluar los resultados de estas pruebas para que sean representativas requiere tanto de perfiles capacitados en la herramienta específica que se utilice como de perfiles con el conocimiento funcional del negocio para el cual está destinado el producto de software.

Por otro lado, las PyMEs de nuestra región en particular deben enfrentan muchos desafíos adicionales en su búsqueda por crear software de calidad y sobrevivir en el mercado. Primeramente, este tipo de industrias no creen que los mismos procesos y métodos utilizados por las grandes compañías de software sean aplicables en su contexto. Las razones para no adoptar estos métodos son el costo, la falta de recursos, el tiempo y la complejidad. En segundo lugar, los procesos y estructuras organizativas en este tipo de industrias son en su mayoría informales y conducen a un entorno caótico. Una posible razón para esto es que deben enfocarse en el tiempo de comercialización para sobrevivir y, a menudo, descuidan los procesos formales más intensivos en recursos. El tercer desafío está asociado a la falta de recursos, habilidades y experiencia, dado que las PyMEs a menudo no pueden darse el lujo de emplear desarrolladores de software experimentados.

Finalmente, a pesar de que existen muchos programas de Mejora de Procesos de Software (SPI) como CMM, CMMI e ISO / IEC15504, los beneficios de estos son desconocidos por las PyMEs, o bien, las prácticas de ingeniería de software propuestas no se centran en las necesidades y problemas a los que se enfrentan

este tipo de empresas. En un estudio realizado por Broy & Rombach[1] sobre la industria de software alemana se indicó que solo el 5% de todas las casas de software alemanas tienen un nivel de CMM de 2 o más. El otro 95% tiene una madurez de nivel 1 de CMM.

Dicho lo anterior, se evidencia no solo que alcanzar los niveles de calidad propuestos en modelos probados es prácticamente inalcanzables para las PyMEs, sino que también, no existen modelos que sean aplicables para ellas. Es por eso, que resulta necesario adaptar los modelos probados globalmente a la realidad de este tipo de empresas con el fin de proporcionar una metodología de pruebas aplicable en el desarrollo de sus productos, y así brindar mayor calidad en sus proyectos de una forma alcanzable.

En este contexto, la razón de ser de este trabajo de investigación es partir de las herramientas y características que proponen los modelos aplicados por las grandes empresas para la realización de pruebas de software, para luego realizar una adaptación que pueda ser implementada a la realidad de una PyME. De esta manera, se busca añadir beneficios sustanciales en la calidad de las soluciones de software que desarrollan, cumpliendo con las expectativas de sus usuarios, evitando parches y ciclos de retrabajo. Además, pretende disminuir los costos asociados con los desvíos en las planificaciones y, como fin último, obtener una primera aproximación de un marco metodológico para llevar a cabo este tipo de actividad**.**

[1] Broy M, Rombach D. Software Engineering: Wurzeln, Stand und Perspektiven. Informatik Spektrum, December 16; pp. 438-451; 2006.

1.2 Hipótesis

Considerando las complejidades de la actividad de producción de software y la situación actual de la industria en la región, este trabajo propone como hipótesis que partiendo de los modelos de mejora de procesos de prueba reconocidos globalmente, se puede generar una adaptación alcanzable para las PyMEs.

1.3 Objetivo general:

El objetivo general de este trabajo es diseñar una propuesta de gestión que permita, a partir de un conjunto de métodos exitosos para las pruebas de aplicaciones en grandes empresas, mejorar la calidad de los productos de software de las PyMEs de la región.

1.4 Objetivos específicos:

- Desarrollar un marco teórico/conceptual relativo a las pruebas de software, haciendo hincapié en las diferentes metodologías y estándares que son reconocidos mundialmente,
- Identificar las complejidades en la realización de pruebas y en la detección de errores en las Pequeñas y Medianas Empresas de la región.
- Realizar un estudio comparativo de los modelos y estándares actuales relativos al *Testing* de Software.
- Diseñar un modelo base para el proceso de *testing* a partir de las características más efectivas de cada uno de los modelos estudiados, aplicable en el contexto de las PyMEs.
- Validar mediante expertos el modelo propuesto con el fin de refutar o validar la hipótesis.

1.5 Organización de la Tesis

La tesis se organiza de la siguiente manera. En el Capítulo 2 se presentan los conceptos generales del software, la forma en que se produce y sus complejidades respecto a otros productos.

En el Capítulo 3, luego de una breve introducción a la calidad de software, se detallan las actividades, técnicas y niveles pruebas, y cómo cada uno de ellos impacta en la calidad del producto final.

El Capítulo 4 contiene el marco teórico/conceptual relativo a los modelos de mejora, junto a un breve análisis sobre los modelos probados más utilizados en la industria.

En el Capítulo 5 se realiza una caracterización de la situación actual que atraviesan las Pymes desarrolladores de software en la región. Luego, a partir de esas características se elaboraron los criterios para la comparación de los modelos de mejora analizados anteriormente.

Finalmente, en el Capítulo 6, en base a todo lo analizado a lo largo de este trabajo se realiza una propuesta de modelo de mejora. Por último, se presentan las conclusiones, principales aportes de esta tesis y trabajo futuro.

CAPÍTULO 2: CONCEPTOS BÁSICOS DEL DESARROLLO DE SOFTWARE

El presente trabajo se centra en una metodología para la mejora de las pruebas o *Testing* de Software. Dado que generalmente esta es una de las etapas del ciclo de vida del desarrollo de este producto, será necesario establecer algunos conceptos básicos para poder abordar la problemática en el contexto donde se desarrolla. En el siguiente capítulo, se detallan aquellas definiciones y convenciones enunciadas por algunos de los autores formalmente aceptados como exponentes de la Ingeniería del Software tales como son Roger Pressman e Ian Sommerville.

2.1 La Naturaleza del Software

En la actualidad, el software tiene un papel dual. Es un producto y al mismo tiempo es el vehículo para entregar un producto. En su forma de producto, brinda el potencial de cómputo incorporado en el hardware de cómputo o, con más amplitud, en una red de computadoras a las que se accede por medio de un hardware local. Ya sea que resida en un teléfono móvil u opere en el interior de una computadora central, el software es un transformador de información —produce, administra, adquiere, modifica, despliega o transmite información que puede ser tan simple como un solo bit o tan compleja como una presentación con multimedios generada a partir de datos obtenidos de decenas de fuentes independientes—. Como vehículo utilizado para distribuir el producto, el software actúa como la base para el control de la computadora (sistemas operativos), para la comunicación de información (redes) y para la creación y control de otros programas (herramientas y ambientes de software) (Pressman, 2010).

El software distribuye el producto más importante de nuestro tiempo: *información*. Transforma los datos personales (por ejemplo, las transacciones financieras de un individuo) de modo que puedan ser más útiles en un contexto local, administra la información de negocios para mejorar la competitividad, provee una vía para las redes mundiales de información (la internet) y brinda los medios para obtener información en todas sus formas (Pressman, 2010).

En este contexto, Pressman define el **Software** como: 1) instrucciones (programas de cómputo) que cuando se ejecutan proporcionan las características, función y

desempeño buscados; 2) estructuras de datos que permiten que los programas manipulen en forma adecuada la información, y 3) información descriptiva tanto en papel como en formas virtuales que describen la operación y uso de los programas.

2.2 Diferencias con el Hardware

Aunque existen definiciones más complejas, esta es una que deja en claro cuál es la percepción que se tiene de este particular producto. Es por esta particularidad que es importante examinar las características propias del software que lo hacen diferente de otros objetos que construyen los seres humanos. Al tratarse de un elemento de un sistema lógico y no de uno físico, posee características que difieren considerablemente de las del hardware (Pressman, 2010):

1. **El software se desarrolla o modifica con intelecto: no se manufactura en el sentido clásico.** Aunque hay algunas similitudes entre el desarrollo de software y la fabricación de hardware, las dos actividades son diferentes en lo fundamental. En ambas, la alta calidad se logra a través de un buen diseño, pero la fase de manufactura del hardware introduce problemas de calidad que no existen (o que se corrigen con facilidad) en el software.

   Ambas actividades dependen de personas, pero la relación entre los individuos dedicados y el trabajo logrado es diferente por completo. Las dos actividades requieren la construcción de un "producto", pero los enfoques son distintos. Los costos del software se concentran en la ingeniería. Esto significa que los proyectos de software no pueden administrarse como si fueran proyectos de manufactura.
2. **El software no se "*desgasta*".** El software no es susceptible a los problemas ambientales que hacen que el hardware se desgaste. Los defectos ocultos ocasionarán tasas elevadas de fallas al comienzo de la vida de un programa. Sin embargo, éstas se corrigen y la curva se aplana. Aun así, la implicación está clara: el software no se desgasta, ¡pero sí se deteriora!

   Durante su vida, el software sufrirá cambios. Es probable que cuando éstos se realicen, se introduzcan errores que ocasionen que la curva de tasa de fallas tenga aumentos súbitos. Antes de que la curva vuelva a su tasa de fallas original de estado estable, surge la solicitud de otro cambio que hace que la curva se

dispare otra vez. Poco a poco, el nivel mínimo de la tasa de fallas comienza a aumentar: el software se está deteriorando como consecuencia del cambio.

Cuando un componente del hardware se desgasta es sustituido por una refacción. En cambio, no hay refacciones para el software. Cada falla de éste indica un error en el diseño o en el proceso que tradujo el diseño a código ejecutable por la máquina. Entonces, las tareas de mantenimiento del software, que incluyen la satisfacción de peticiones de cambios, involucran una complejidad considerablemente mayor que el mantenimiento del hardware.

3. **Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del software se construye para un uso individualizado.** A medida que evoluciona una disciplina de ingeniería, se crea un conjunto de componentes estandarizados para el diseño, como por ejemplo los tornillos estándar o los circuitos integrados preconstruidos. Los componentes reutilizables han sido creados para que el ingeniero pueda concentrarse en los elementos verdaderamente innovadores de un diseño; es decir, en las partes de éste que representan algo nuevo. En el mundo del hardware, volver a usar componentes es una parte natural del proceso de ingeniería. En el del software, es algo que apenas ha empezado a hacerse a gran escala. Un componente de software debe diseñarse e implementarse de modo que pueda volverse a usar en muchos programas diferentes. Los modernos componentes reutilizables incorporan tanto los datos como el procesamiento que se les aplica, lo que permite que el ingeniero de software cree nuevas aplicaciones a partir de partes susceptibles de volverse a usar. Por ejemplo, las actuales interfaces interactivas de usuario se construyen con componentes reutilizables que permiten la creación de ventanas gráficas, menús desplegables y una amplia variedad de mecanismos de interacción. Las estructuras de datos y el detalle de procesamiento que se requieren para construir la interfaz están contenidos en una librería de componentes reusables para tal fin.

2.3 Una primera aproximación a la calidad del Software

Cuando hablamos de la calidad del software profesional, debemos tener en cuenta que el software es usado y modificado por personas distintas de sus desarrolladores. Por lo tanto, la calidad no solo concierne a lo que hace el software,

sino que además tiene que incluir el comportamiento del software mientras se está ejecutando, la estructura y organización de los programas del sistema y la documentación asociada. Esto se refleja en los llamados atributos de calidad del software o requerimientos no funcionales. Ejemplos de estos atributos son el tiempo de respuesta del software ante una consulta del usuario y la comprensibilidad del código del programa. El conjunto específico de atributos que puede esperar de un sistema de software, obviamente depende de su aplicación. Por lo tanto, un sistema bancario debe ser seguro, un juego interactivo debe ser receptivo, un sistema de conmutación telefónica debe ser confiable, entre otros.

De forma general, se dice que las características que debe poseer un buen software son las siguientes (Sommerville, 2011):

- **Mantenibilidad:** el software debe escribirse de tal manera que pueda evolucionar para satisfacer las necesidades cambiantes de los clientes. Este es un atributo crítico porque el cambio de software es un requisito inevitable de un entorno empresarial cambiante.
- **Confiabilidad**: la confiabilidad del software incluye una gama de características como la exactitud, confianza y seguridad. El software confiable no debe causar daños físicos o económicos en caso de falla del sistema. Los usuarios malintencionados no deben poder acceder o dañar el sistema.
- **Eficiencia**: el software no debe hacer un uso inútil de los recursos del sistema, como la memoria o los ciclos de procesamiento. Por lo tanto, la eficiencia incluye la capacidad de respuesta, el tiempo de procesamiento, la utilización de la memoria, entre otros.
- **Aceptabilidad**: el software debe ser aceptable para el tipo de usuarios para el que está diseñado. Esto significa que debe ser comprensible, utilizable y compatible con otros sistemas que utiliza.

2.4 El Proceso del Software

Un **Proceso** es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo**.** Una **Actividad** busca lograr un objetivo amplio (por ejemplo, comunicación con los participantes) y se desarrolla sin importar el dominio de la aplicación, tamaño del proyecto, complejidad del

esfuerzo o grado de rigor con el que se usará la ingeniería de software. Una **Acción** (diseño de la arquitectura) es un conjunto de tareas que producen un producto importante del trabajo (por ejemplo, un modelo del diseño de la arquitectura). Una **Tarea** se centra en un objetivo pequeño pero bien definido (por ejemplo, realizar una prueba unitaria) que produce un resultado tangible (Pressman, 2010).

En el contexto de la ingeniería de software, un proceso de software es un conjunto de actividades relacionadas que conducen a la producción de un producto de software (Sommerville, 2011). Sin embargo, no es una prescripción rígida de cómo elaborarlo. Por el contrario, debe ser ágil y adaptable (al problema, al proyecto, al equipo y a la cultura organizacional). Por tanto, un proceso adoptado para un proyecto puede ser significativamente distinto de otro adoptado para otro proyecto (Pressman, 2010).

Aunque existen muchos procesos de software diferentes, todos deben incluir cuatro actividades que son consideradas fundamentales para la ingeniería de software (Sommerville, 2011):

- **Especificación** del software: Se debe definir la funcionalidad del software y las restricciones de su funcionamiento.
- **Diseño** e **implementación** del software: Se debe producir el software que cumpla con las especificaciones.
- **Validación** del software**:** El software debe validarse para garantizar que hace lo que el cliente quiere.
- **Evolución** del software: El software debe evolucionar para satisfacer las necesidades cambiantes de los clientes.

De alguna forma, estas actividades son parte de todos los procesos de software. En la práctica, por supuesto, son actividades complejas en sí mismas e incluyen subactividades como la validación de requisitos, el diseño de la arquitectura, las pruebas unitarias, etc. También hay actividades de proceso de apoyo como la documentación y la gestión de la configuración del software.

Cuando se describen y discuten los procesos, generalmente se detallan las sobre actividades incluidas en esos procesos, como especificar un modelo de datos, diseñar una interfaz de usuario, etc., y el orden en que se realizarán las mismas. Sin embargo, además de las actividades, las descripciones de los procesos también pueden incluir (Sommerville, 2011):

- **Productos,** que son los resultados de una actividad de proceso. Por ejemplo, el resultado de la actividad del diseño de arquitectura puede ser el modelo de arquitectura.
- **Roles**, que reflejan las responsabilidades de las personas involucradas en el proceso. Ejemplos de roles son: administrador de proyectos, administrador de configuración, programador, etc.
- **Condiciones previas y posteriores**, que son declaraciones que son verdaderas antes y después de que se haya iniciado una actividad del proceso o que se haya producido un producto. Por ejemplo, antes de que comience el diseño de la arquitectura, una condición previa puede ser que todos los requisitos hayan sido aprobados por el cliente; Una vez finalizada esta actividad, una condición posterior podría ser que se hayan revisado los modelos UML que describen la arquitectura.

Finalmente, a la hora de desarrollar un proceso software se debe de tener en cuenta una serie de aspectos esenciales (Canfora & Ruiz González, 2004):

- Tecnologías de desarrollo y de mantenimiento del software, que aportan las herramientas e infraestructuras necesarias para hacer posible crear y mantener productos software complejos que satisfagan las necesidades actuales y futuras.
- Métodos y técnicas para el desarrollo y el mantenimiento software, que suponen el soporte metodológico esencial para aprovechar de manera eficiente las tecnologías y realizar con éxito las actividades de desarrollo y mantenimiento del software.
- Comportamiento organizacional, es decir, la ciencia de las organizaciones y de las personas es útil, en general, los proyectos software se llevan a cabo por equipos de personas que tienen que ser coordinados y dirigidos dentro de una estructura organizacional eficiente.
- Economía, ya que los proyectos de desarrollo y mantenimiento de software no son esfuerzos autónomos, sino que, como pasa con cualquier otro producto, el software debe estar dirigido a satisfacer las necesidades de clientes/usuarios reales.

El proceso de software es complejo y, como todos los procesos intelectuales y creativos, confía en que las personas tomen decisiones con criterio. No existe un proceso ideal y la mayoría de las organizaciones han desarrollado sus propios procesos de desarrollo. Los procesos han ido evolucionando para aprovechar las capacidades de las personas de una organización y las características específicas de los sistemas que se están desarrollando. Para algunos sistemas, como los sistemas críticos, se requiere un proceso de desarrollo muy estructurado. Para los sistemas empresariales, donde los requisitos cambian rápidamente, es probable que un proceso menos formal y flexible sea más efectivo (Sommerville, 2011).

2.5 Ingeniería del software

Con objeto de elaborar software de calidad listo para enfrentar los retos del siglo XXI, debemos asumir algunas realidades sencillas (Pressman, 2010):

- El software se ha incrustado profundamente en casi todos los aspectos de nuestras vidas y, como consecuencia, el número de personas que tienen interés en las características y funciones que brinda una aplicación específica ha crecido en forma notable. Cuando ha de construirse una aplicación nueva o sistema incrustado, deben escucharse muchas opiniones. Y en ocasiones parece que cada una de ellas tiene una idea un poco distinta de cuáles características y funciones debiera tener el software. **Se concluye que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software.**
- Los requerimientos de la tecnología de la información que demandan los individuos, negocios y gobiernos se hacen más complejos con cada año que pasa. En la actualidad, grandes equipos de personas crean programas de cómputo que antes eran elaborados por un solo individuo. El software sofisticado, que alguna vez se implementó en un ambiente de cómputo predecible y autocontenido, hoy en día se halla incrustado en el interior de todo, desde la electrónica de consumo hasta dispositivos médicos o sistemas de armamento. La complejidad de estos nuevos sistemas y productos basados en computadora demanda atención cuidadosa a las interacciones de todos los elementos del sistema. **Se concluye que el diseño se ha vuelto una actividad crucial.**

- Los individuos, negocios y gobiernos dependen cada vez más del software para tomar decisiones estratégicas y tácticas, así como para sus operaciones y control cotidianos. Si el software falla, las personas y empresas grandes pueden experimentar desde un inconveniente menor hasta fallas catastróficas. **Se concluye que el software debe tener alta calidad.**
- A medida que aumenta el valor percibido de una aplicación específica se incrementa la probabilidad de que su base de usuarios y longevidad también crezcan. Conforme se extienda su base de usuarios y el tiempo de uso, las demandas para adaptarla y mejorarla también crecerán. **Se concluye que el software debe tener facilidad para recibir mantenimiento.**

Estas realidades simples llevan a una conclusión: debe hacerse ingeniería con el software en todas sus formas y a través de todos sus dominios de aplicación.

En base a las realidades expuestas, la IEEE (*Institute of Electrical and Electronics Engineers*) ha definido en su Guía del Cuerpo de Conocimientos de Ingeniería de Software (SWEBOK, *Guide Software Engineering Body of Knowledge*) a la **Ingeniería de Software** como:

"*1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software. 2) El estudio de enfoques según el punto 1*" (Bourque & Fairley, 2014).

2.6 Tipos de Software

Los ingenieros de software se preocupan por desarrollar productos de software (es decir, software que pueda venderse a un cliente). Los productos de software pueden clasificarse en dos tipos (Sommerville, 2011):

- **Productos genéricos:** Estos son sistemas independientes que son producidos por una empresa de software y se venden en el mercado abierto a cualquier cliente que pueda comprarlos. Los ejemplos de este tipo de producto incluye software para PC, como bases de datos, procesadores de texto, paquetes de dibujo y herramientas de gestión de proyectos. También incluye las llamadas aplicaciones verticales diseñadas para algún propósito

específico, como los sistemas de información de bibliotecas, sistemas de contabilidad o sistemas para mantener registros dentales.

- **Productos a medida (o customizados):** Estos son sistemas que son encargados por un cliente en particular. Un contratista de software desarrolla el software especialmente para ese cliente. Los ejemplos de este tipo de software incluyen sistemas de control para dispositivos electrónicos, sistemas para soportar un proceso de negocio en particular, y sistemas de control de tráfico aéreo.

Una diferencia importante entre estos tipos de software es que, en los productos genéricos, la organización que desarrolla el software controla la especificación. Para productos a medida, la especificación generalmente es desarrollada y controlada por la organización que está comprando el software. Los desarrolladores de software deben trabajar con esa especificación. Sin embargo, la distinción entre estos tipos de productos del sistema se está volviendo cada vez más borrosa. Ahora se están construyendo cada vez más sistemas con un producto genérico como base, que luego se adapta para satisfacer los requisitos de un cliente. Los sistemas de planificación de recursos empresariales (ERP), como el sistema SAP, son los mejores ejemplos de este enfoque. En este caso, un sistema grande y complejo se adapta a una empresa al incorporar información sobre las reglas y procesos de negocios, los informes requeridos, etc.

2.7 Modelos de proceso

Ante la diversidad para llevar a cabo un proceso de desarrollo de software, se han reconocido en la industria una serie de modelos de proceso de referencia. Un **modelo de procesos de software** es una descripción simplificada de un proceso del software que presenta una visión de ese proceso. Estos modelos pueden incluir actividades que son parte de los procesos y productos de software y el papel de las personas involucradas en la ingeniería del software (Sommerville, 2005).

En la siguiente sección, se presentan modelos de proceso muy generales (a veces llamados "paradigmas de proceso") desde una perspectiva arquitectónica. Es decir, el marco del proceso que no incluye el detalle de las actividades específicas.

Estos modelos genéricos no son descripciones definitivas de procesos de software. Más bien, son abstracciones del proceso que pueden usarse para explicar

diferentes enfoques para el desarrollo de software. Pueden considerarse como marcos de procesos que pueden ampliarse y adaptarse para crear procesos de ingeniería de software más específicos.

2.7.1 *Modelo en Cascada*

El modelo en cascada o modelo de la cascada, a veces llamado ciclo de vida clásico, sugiere un enfoque sistemático y secuencial para el desarrollo del software, comienza con la especificación de los requerimientos por parte del cliente y avanza a través de la planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado (Pressman, 2010).

El modelo de cascada es un ejemplo de un proceso dirigido por el plan (o *plan-driven)*: se deben planificar y programar todas las actividades del proceso antes de comenzar a trabajar en ellas.

Las etapas principales del modelo de cascada (Figura 2-1 El Modelo en Cascada (Sommerville, 2011)) reflejan directamente las actividades fundamentales del proceso de desarrollo:

1. **Análisis y definición de requisitos**: Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema.
2. **Diseño del sistema:** El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.
3. **Implementación y pruebas unitarias:** Durante esta etapa, se realiza el diseño del software como un conjunto de programas o unidades de programa. Las pruebas unitarias consisten en verificar que cada unidad cumple con sus especificaciones.
4. **Integración y pruebas de sistema:** Las unidades o programas individuales son integrados y probados como un sistema completo para garantizar que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.

5. **Operación y mantenimiento:** Por lo general (aunque no necesariamente), esta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

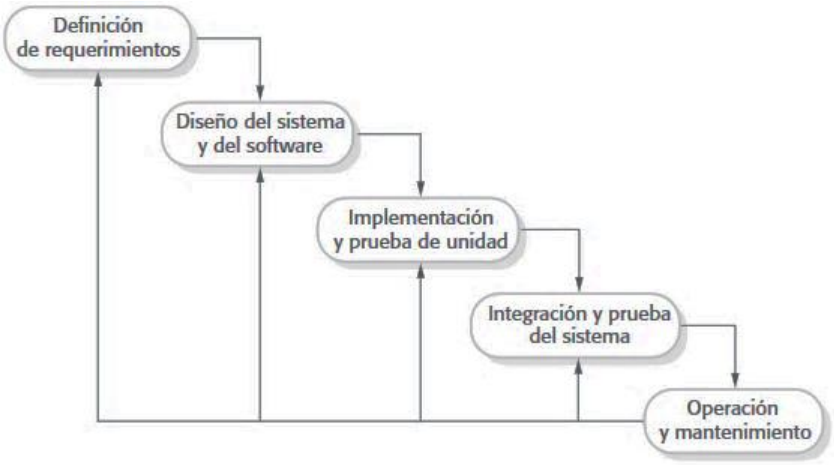

Figura 2-1 El Modelo en Cascada (Sommerville, 2011)

En principio, el resultado de cada fase consiste en uno o más documentos que están aprobados ("firmados"). La siguiente fase no debe comenzar hasta que la fase anterior haya finalizado. En la práctica, estas etapas se superponen y se nutren mutuamente de información. Durante el diseño, se identifican problemas con los requerimientos. Durante la codificación, se descubren problemas de diseño y así sucesivamente. El proceso del software no es un simple modelo lineal, sino que implica la retroalimentación de una fase a otra. Entonces, es posible que los documentos generados en cada fase deban modificarse para reflejar los cambios que se realizan (Sommerville, 2011).

Debido a los costos de producción y aprobación de documentos, las iteraciones pueden ser costosas e implicar un retrabajo significativo. Por lo tanto, después de un pequeño número de iteraciones, es normal congelar partes del desarrollo, como la especificación, y continuar con las etapas posteriores. Los problemas se dejan para una resolución posterior, se ignoran o se programan. Esta congelación prematura de los requerimientos puede significar que el sistema no haga lo que el usuario quiere (Sommerville, 2011).

El modelo de la cascada es el paradigma más antiguo de la ingeniería de software. Sin embargo, en las últimas tres décadas, las críticas hechas al modelo han ocasionado que incluso sus defensores más obstinados cuestionen su eficacia (Hanna, 1995). Hoy en día, el trabajo de software es acelerado y está sujeto a una corriente sin fin de cambios (en las características, funciones y contenido de información). El modelo de la cascada suele ser inapropiado para ese tipo de labor. No obstante, sirve como un modelo de proceso útil en situaciones en las que los requerimientos son fijos y el trabajo avanza en forma lineal hacia el final (Pressman, 2010). Además, el modelo de cascada es consistente con otros modelos de procesos de ingeniería y en cada fase se produce documentación. Esto hace que el proceso sea visible, de modo que los administradores monitoricen el progreso contra el plan de desarrollo (Sommerville, 2011).

2.7.2 *Modelo Incremental*

El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado. Las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades (Sommerville, 2011).

Figura 2-2 El Modelo Incremental (Pressman, 2010)

Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea el producto fundamental. Es decir, se abordan los requerimientos básicos, pero no se proporcionan muchas características suplementarias (algunas conocidas y otras no). El cliente usa el producto fundamental (o lo somete a una evaluación detallada). Como resultado del uso y/o evaluación, se desarrolla un plan para el

incremento que sigue. El plan incluye la modificación del producto fundamental para cumplir mejor las necesidades del cliente, así como la entrega de características adicionales y más funcionalidad. Este proceso se repite después de entregar cada incremento, hasta terminar el producto final (Pressman, 2010).

El desarrollo de software incremental, que es una parte fundamental de los enfoques Ágiles que se enunciarán más adelante, es mejor que un enfoque en cascada para la mayoría de los sistemas empresariales, de comercio electrónico y personales. El desarrollo incremental refleja la forma en que se resuelven problemas. Rara vez se trabaja por adelantado una solución completa del problema, más bien se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores. Al desarrollar el software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme éste se diseña (Sommerville, 2011). Por ejemplo, un software para procesar textos que se elabore con el paradigma incremental quizá entregue en el primer incremento las funciones básicas de administración de archivos, edición y producción del documento; en el segundo dará herramientas más sofisticadas de edición y producción de documentos; en el tercero habrá separación de palabras y revisión de la ortografía; y en el cuarto se proporcionará la capacidad para dar formato avanzado a las páginas. Debe observarse que el flujo de proceso para cualquier incremento puede incorporar el paradigma del prototipo (Pressman, 2010).

Actualmente, el desarrollo incremental es el enfoque más común para el desarrollo de sistemas de aplicaciones. Este enfoque puede ser dirigido por el plan ágil o, como sucede generalmente, una mezcla de ambos. En un enfoque basado en el plan, los incrementos del sistema se identifican por adelantado; Si se adopta un enfoque ágil, se identifican los primeros incrementos, pero el desarrollo de los incrementos posteriores depende del progreso y las prioridades del cliente (Sommerville, 2011).

Desde una perspectiva de gestión, el enfoque incremental tiene dos problemas importantes. Uno es que el proceso no es visible para la gerencia ya que, si se desarrolla rápidamente, no es rentable producir documentos que reflejen cada versión del sistema. El otro es que la estructura del sistema tiende a degradarse a medida que se agregan nuevos incrementos. El cambio regular tiende a corromper

su estructura e incorporar más cambios de software se vuelve cada vez más difícil y costoso, a menos que se gaste tiempo y dinero en la refactorización.

Los problemas del desarrollo incremental se vuelven particularmente agudos para sistemas grandes, complejos y de larga duración, en los que diferentes equipos desarrollan diferentes partes del sistema. Los grandes sistemas necesitan un marco o arquitectura estable y las responsabilidades de los diferentes equipos que trabajan en partes del sistema deben definirse claramente con respecto a esa arquitectura. Esto tiene que ser planeado por adelantado en lugar de ser desarrollado incrementalmente (Sommerville, 2011).

2.7.3 *Prototipado*

No es necesario desarrollar un sistema completo de forma incremental para exponerlo a los clientes para que realicen comentarios. La entrega e implementación incrementales significan que el software se usa en procesos reales y operativos que no siempre es posible, ya que la experimentación puede interrumpir los procesos comerciales normales. Para eso, se utilizan los prototipos. Un prototipo es una versión inicial de un sistema de software que se utiliza para demostrar conceptos, probar opciones de diseño y obtener más información sobre el problema y sus posibles soluciones. El desarrollo rápido e iterativo del prototipo es esencial para que los costos se controlen y las partes interesadas del sistema puedan experimentar con el prototipo al principio del proceso del software (Sommerville, 2011).

Los prototipos del sistema permiten a los usuarios ver qué tan bien el sistema soporta su trabajo. Pueden obtener nuevas ideas para los requerimientos y encontrar áreas de fortaleza y debilidad en el software, así como también proponer nuevos requisitos del sistema. Además, a medida que se desarrolla el prototipo, se pueden revelar errores y omisiones en los requisitos que se han propuesto. Una función descripta en una especificación puede parecer útil y bien definida. Sin embargo, cuando esa función se combina con otras funciones, los usuarios a menudo encuentran que su vista inicial fue incorrecta o incompleta. La especificación del sistema se puede modificar para reflejar su comprensión modificada de los requerimientos (Sommerville, 2011).

La creación de prototipos también es una parte esencial del proceso de diseño de la interfaz de usuario. Debido a la naturaleza dinámica de las interfaces de usuario, las descripciones textuales y los diagramas no son suficientes para expresar los requerimientos de la interfaz de usuario. Por lo tanto, la creación rápida de prototipos con la participación del usuario final es la única forma sensata de desarrollar interfaces gráficas de usuario para sistemas de software. Cabe aclarar que los prototipos no tienen que ser ejecutables para ser útiles. El maquetado de la interfaz de usuario del sistema (Rettig, 1994) puede ser efectivo para ayudar a los usuarios a refinar un diseño de interfaz y trabajar a través de escenarios de uso. Son muy baratos de desarrollar y se pueden construir en pocos días (Sommerville, 2011).

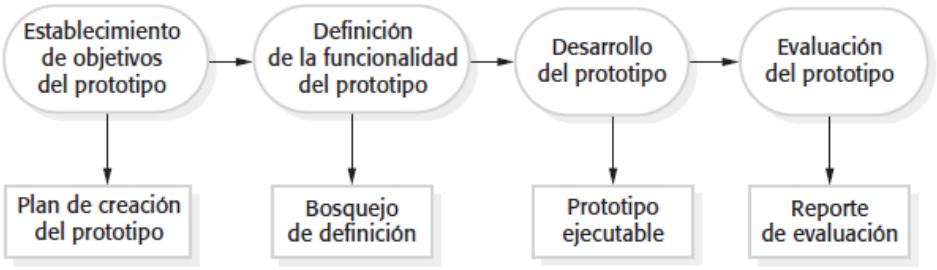

Figura 2-3 Proceso de desarrollo del Prototipo (Sommerville, 2011)

En la Figura 2-3, se detalla brevemente el proceso de desarrollo de un prototipo. Los objetivos de su creación se deben explicitar desde el inicio del proceso. La siguiente etapa consiste en decidir qué colocar y, lo que es más importante, qué dejar fuera del sistema prototipo. Luego sigue la construcción y, por último, la evaluación del prototipo. En esta etapa final se deben tomar medidas para capacitar a los usuarios y se deben utilizar los objetivos del prototipo para derivar un plan de evaluación.

No obstante, el desarrollo de prototipos puede generar algunos problemas. A veces, como los participantes ven lo que parece ser una versión funcional del software, sin darse cuenta de que el prototipo se obtuvo de manera forzosa; no perciben que, en la prisa de hacerlo funcionar, no se considera la calidad general del software o la facilidad de darle mantenimiento a largo plazo. Así como también, es frecuente que se generen compromisos respecto de la implementación a fin de hacer que el prototipo funcione rápido y se utilicen soluciones inapropiadas. Después de un tiempo, quizá esas elecciones que debían ser temporales perduren y se olviden

todas las razones por las que eran inadecuadas. Entonces, la elección de algo menos que lo ideal pasan a formar parte del producto final.

Aunque puede haber problemas, hacer prototipos es un paradigma eficaz para la ingeniería de software. La clave es definir desde el principio las reglas del juego; es decir, todos los participantes deben estar de acuerdo en que el prototipo sirva como el mecanismo para definir los requerimientos. Después se descartará (al menos en parte) y se hará la ingeniería del software real con la mirada puesta en la calidad (Pressman, 2010).

2.7.4 *El Modelo Espiral*

Propuesto en primer lugar por Barry Boehm (Boehm, 1988), el modelo espiral es un modelo evolutivo del proceso del software y se acopla con la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistémicos del modelo de cascada. Tiene el potencial para hacer un desarrollo rápido de versiones cada vez más completas. Boehm describe el modelo de la siguiente manera:

"El modelo de desarrollo espiral es un generador de modelo de proceso impulsado por el riesgo, que se usa para guiar la ingeniería concurrente con participantes múltiples de sistemas intensivos en software. Tiene dos características distintivas principales. La primera es el enfoque cíclico para el crecimiento incremental del grado de definición de un sistema y su implementación, mientras que disminuye su grado de riesgo. La otra es un conjunto de puntos de referencia de anclaje puntual para asegurar el compromiso del participante con soluciones factibles y mutuamente satisfactorias" (Boehm & Hansen, 2001).

En este modelo, el proceso del software se representa como una espiral, en lugar de una secuencia de actividades con un retroceso de una actividad a otra. Cada bucle en la espiral representa una fase del proceso de software. Por lo tanto, el bucle más interno podría estar relacionado con la viabilidad del sistema, el siguiente bucle con la definición de requisitos, el siguiente bucle con el diseño del sistema, etc. (Sommerville, 2011).

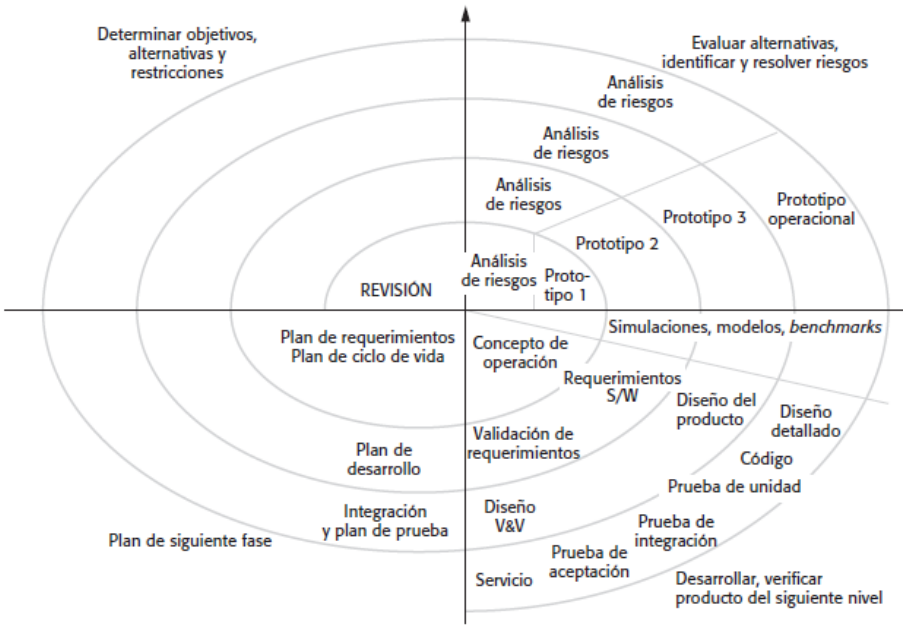

Figura 2-4 Modelo en espiral de Boehm del proceso de software (Boehm, 1988)

A su vez, cada bucle o ciclo de espiral se divide en cuatro sectores (Sommerville, 2011):

1. **Establecimiento de objetivos:** Se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones en el proceso y el producto, y se traza un plan de gestión detallado. Se identifican los riesgos del proyecto. Pueden plantearse estrategias alternativas, según sean los riesgos.
2. **Valoración y reducción del riesgo:** En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso y se realizan acciones para reducir el riesgo. Por ejemplo, si existe un riesgo de que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.
3. **Desarrollo y validación:** Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema. Por ejemplo, la creación de prototipos desechables sería el mejor enfoque de desarrollo, si predominan los riesgos en la interfaz del usuario. Si el principal riesgo identificado es la integración de subsistemas, el modelo en cascada sería el mejor modelo de desarrollo a utilizar, entre otros.
4. **Planeación:** El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral. Si se opta por continuar, se trazan los planes para la siguiente fase del proyecto.

La principal diferencia entre el modelo en espiral y otros modelos de proceso de software es su reconocimiento explícito del riesgo. De manera informal, el riesgo significa simplemente algo que podría salir mal. Por ejemplo, si la intención es usar un nuevo lenguaje de programación, existe el riesgo de que los compiladores disponibles no sean confiables o no produzcan un código de objeto lo suficientemente eficiente. Los riesgos conducen a propuestas de cambios de software y a problemas de proyecto como exceso en las fechas y el costo, de manera que la minimización del riesgo es una actividad muy importante de administración del proyecto (Sommerville, 2011).

Otra de sus características es que, a diferencia de otros modelos del proceso que finalizan cuando se entrega el software, el modelo espiral puede adaptarse para aplicarse a lo largo de toda la vida del software de cómputo. Entonces, el primer circuito alrededor de la espiral quizá represente un “proyecto de desarrollo del concepto” que comienza en el centro de la espiral y continúa por iteraciones múltiples hasta que queda terminado el desarrollo del concepto. Si el concepto va a desarrollarse en un producto real, el proceso sigue hacia fuera de la espiral y comienza un “proyecto de desarrollo de producto nuevo”. El nuevo producto evolucionará a través de cierto número de iteraciones alrededor de la espiral. Más adelante puede usarse un circuito alrededor de la espiral para que represente un “proyecto de mejora del producto”. En esencia, la espiral, cuando se caracteriza de este modo, sigue operativa hasta que el software se retira (Pressman, 2010).

El modelo espiral es un enfoque realista para el desarrollo de sistemas y de software a gran escala. Como el software evoluciona a medida que el proceso avanza, el desarrollador y cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución. El modelo espiral usa los prototipos como mecanismo de reducción de riesgos, pero, más importante, permite aplicar el enfoque de hacer prototipos en cualquier etapa de la evolución del producto. Mantiene el enfoque de escalón sistemático sugerido por el ciclo de vida clásico, pero lo incorpora en una estructura iterativa que refleja al mundo real en una forma más realista (Pressman, 2010).

Sin embargo, como otros paradigmas, el modelo espiral no es una panacea (Pressman, 2010). Algunas veces, es difícil convencer a los clientes (en particular en situaciones bajo contrato) de que el enfoque evolutivo es del todo controlable.

Además, demanda mucha experiencia en la evaluación del riesgo y se basa en ella para llegar al éxito. Sin duda, existirán problemas si algún riesgo importante no se descubre y administra adecuadamente.

2.7.5 *El Proceso Unificado*

Al principio de la década de 1990, James Rumbaugh, Grady Booch e Ivar Jacobson comenzaron a trabajar en un "método unificado" que combinaría lo mejor de cada uno de sus métodos individuales de análisis y diseño orientado a objetos. El resultado fue un lenguaje de modelado unificado, más conocido por sus siglas en inglés UML (*Unified Modeling Language*) que contiene una notación robusta para el modelado y desarrollo de los sistemas orientados a objetos (Pressman, 2010). En este contexto, el UML brinda las herramientas necesarias para apoyar la práctica de la ingeniería de software orientada a objetos, pero no proporciona la estructura del proceso necesaria para guiar a los equipos del proyecto cuando aplican esta tecnología. Es por eso que en los siguientes años, Jacobson, Rumbaugh y Booch desarrollaron una estructura para la ingeniería de software orientado a objetos que utiliza UML conocido como el *Proceso Unificado* (En inglés RUP, *The Rational Unified Process*).

En cierto modo, el Proceso Unificado o PU es un intento por obtener los mejores rasgos y características de los modelos tradicionales del proceso del software, pero en forma que implemente muchos de los mejores principios del desarrollo ágil de software. El proceso unificado reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista respecto de un sistema (Pressman, 2010). Hace énfasis en la importancia de la arquitectura del software y "*ayuda a que el arquitecto se centre en las metas correctas, tales como que sea comprensible, permite cambios futuros y la reutilización"*.

Las características enunciadas en el modelo son las siguientes (Jacobson et al., 1999)**:**

- **Iterativo e Incremental:** El PU es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio o Concepción, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio puede incluir varias iteraciones en proyectos grandes). Estas iteraciones, tal como se muestra

en la Figura 2-5 Esfuerzo en cada una de las disciplinas dependiendo de la Iteración (Jacobson et al., 1999), ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

- **Dirigido por los casos de uso:** En el PU los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.
- **Centrado en la arquitectura:** El PU asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios de este: electricidad, fontanería, etc.

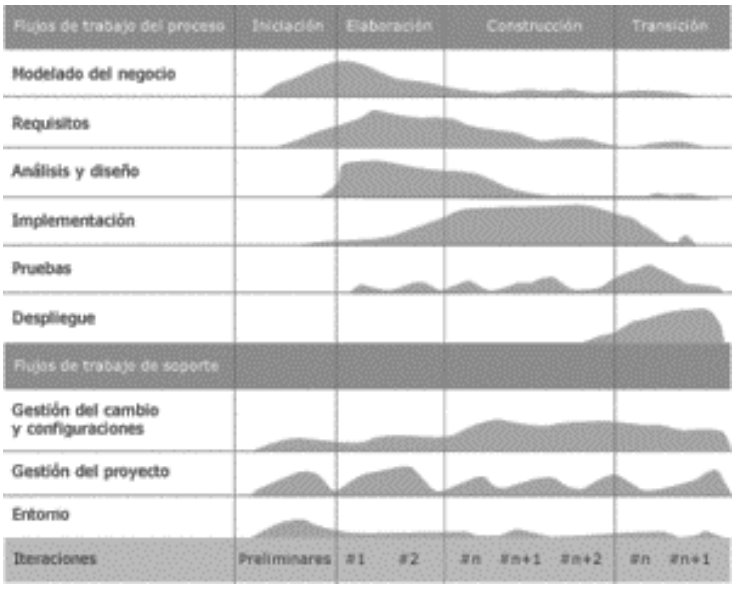

Figura 2-5 Esfuerzo en cada una de las disciplinas dependiendo de la Iteración (Jacobson et al., 1999)

- **Enfocado en los riesgos:** El PU requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

En contraste con los modelos de proceso convencionales, donde se presenta una vista única del proceso, el PU se describe desde tres perspectivas (Sommerville, 2011):

1. Una perspectiva dinámica que muestra las fases del modelo a través del tiempo.
2. Una perspectiva estática que presenta las actividades del proceso que se establecen.
3. Una perspectiva práctica que sugiere buenas prácticas a usar durante el proceso.

Desde la perspectiva dinámica, el PU identifica cuatro fases discretas en el proceso del software. Sin embargo, a diferencia del modelo de cascada en el que las fases se equiparan a las actividades del proceso, las fases del PU están más estrechamente relacionadas con el negocio que con las cuestiones técnicas. Dichas fases son (Sommerville, 2011):

1. **Inicio o Concepción:** La meta de esta fase es establecer un caso empresarial para el sistema. Se deben identificar todas las entidades externas (personas y sistemas) que interactuarán con el sistema y definir dichas interacciones. Luego se usa esta información para valorar la aportación del sistema hacia la empresa. Si esta aportación es menor, entonces el proyecto puede cancelarse después de esta fase.

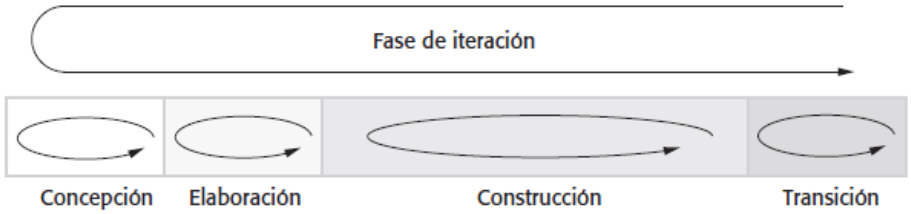

Figura 2-6 Fases dinámicas del Proceso Unificado (Sommerville, 2011)

2. **Elaboración:** Los objetivos de la fase de elaboración consisten en desarrollar una comprensión del dominio del problema, establecer un marco

arquitectónico para el sistema, diseñar el plan del proyecto e identificar los riesgos clave del proyecto. Al finalizar esta fase, se debe tener un modelo de requerimientos para el sistema, que puede ser un conjunto de casos de uso UML, una descripción arquitectónica y un plan de desarrollo para el software.

3. **Construcción:** La fase de construcción involucra el diseño del sistema, la programación y las pruebas. Las partes del sistema se desarrollan en paralelo y se integran durante esta fase. Al finalizar la misma, se debe tener un sistema de software funcionando y la documentación asociada lista para ser entregada a los usuarios.
4. **Transición:** La fase final del PU tiene que ver con mover el sistema del ambiente de desarrollo al ambiente productivo o de usuarios y hacer que funcione en un entorno real. Esto es algo que se ignora en la mayoría de los modelos de proceso de software, aunque, en efecto, es una actividad costosa y en ocasiones problemática. Al finalizar esta fase, debe tener un sistema de software documentado que funcione correctamente en su entorno operativo.

En este modelo, cada iteración se apoya en dos formas. Cada fase puede presentarse en una forma iterativa, con los resultados desarrollados incrementalmente. Además, todo el conjunto de fases puede expresarse de manera incremental, como se muestra en la flecha en curva desde *Transición* hasta *Concepción* en la Figura 2-6 Fases dinámicas del Proceso Unificado (Sommerville, 2011).

La vista estática del PU se centra en las actividades que tienen lugar durante el proceso de desarrollo, denominadas *Flujos de Trabajo*. Hay seis flujos de trabajo centrales identificados en el proceso y tres flujos de trabajo de soporte básicos. El PU se ha diseñado junto con el UML, por lo que la descripción del flujo de trabajo se orienta en torno a los modelos UML asociados, como los modelos de secuencia, los modelos de objetos, etc. Los flujos de trabajo de ingeniería y soporte se describen en la Tabla 2-1 - Flujos de Trabajo (Sommerville, 2011).

Flujo De Trabajo	Descripción
Modelado de negocio	Los procesos de negocios se modelan utilizando casos de uso de negocio.
Requisitos	Se identifican los actores que interactúan con el sistema y se desarrollan casos de uso para modelar los requisitos del sistema.
Análisis y diseño	Se crea un modelo de diseño y se documenta utilizando modelos de arquitectura, modelos de componentes, modelos de objetos y modelos de secuencia.
Implementación	Los componentes del sistema son implementados y estructurados en subsistemas de implementación. La generación automática de código a partir de modelos de diseño ayuda a acelerar este proceso.
Testing	El Testing o prueba de software es un proceso iterativo que se lleva a cabo en conjunto con la implementación. Las pruebas del sistema siguen a la finalización de la implementación.
Despliegue	Se crea una versión del producto, se distribuye a los usuarios y se instala en su lugar de trabajo.
Configuración y gestión de cambios	Este flujo de trabajo de soporte administra los cambios al sistema ya implementado.
Gestión de proyectos	Este flujo de trabajo de soporte gestiona el desarrollo del sistema.
Entorno	Este flujo de trabajo se ocupa de poner a disposición del equipo de desarrollo de software las herramientas de software adecuadas.

Tabla 2-1 - Flujos de Trabajo (Sommerville, 2011)

La ventaja de presentar una vista dinámica y una estática es que las fases del proceso de desarrollo no están asociadas con flujos de trabajo específicos. En principio, al menos, todos los flujos de trabajo PU pueden estar activos en todas las etapas del proceso. En las primeras fases del proceso, la mayor parte del esfuerzo probablemente se dedicará a flujos de trabajo como el modelado y los requisitos del negocio y, en las fases posteriores, a las pruebas y la implementación (Sommerville, 2011).

Por último, la perspectiva de la práctica en el PU describe un conjunto de buenas prácticas de ingeniería de software que se recomiendan para su uso en el desarrollo de sistemas. Se recomiendan seis buenas prácticas fundamentales:

- **Desarrollo de software de forma iterativa:** Planificar los incrementos del sistema en función de las prioridades del cliente y desarrollar oportunamente las características del sistema de mayor prioridad en el proceso de desarrollo.
- **Gestión de requerimientos:** Documentar explícitamente los requerimientos del cliente y realizar un seguimiento de los cambios de dichos requerimientos. Analizar el impacto de los cambios en el sistema antes de aceptarlos.
- **Utilizar arquitecturas basadas en componentes:** Estructurar la arquitectura del sistema en componentes.

- **Modelar visualmente el software**. Usar modelos UML gráficos para elaborar representaciones de software estáticas y dinámicas.
- **Verificar la calidad del software:** Garantizar que el software cumpla con los estándares de calidad de la organización.
- **Controlar los cambios en el software:** Gestionar los cambios al software con un sistema de administración del cambio, así como con procedimientos y herramientas de administración de la configuración.

Las innovaciones más importantes en el PU son la separación de fases y flujos de trabajo, y el reconocimiento de que la implementación de software en el entorno de un usuario es parte del proceso. Las fases son dinámicas y tienen objetivos. Los flujos de trabajo son estáticos y son actividades técnicas que no están asociadas con una sola fase, pero se pueden utilizar durante todo el desarrollo para alcanzar los objetivos de cada fase (Sommerville, 2011).

Actualmente, el PU y el UML se utilizan ampliamente en proyectos de toda clase orientados a objetos. El modelo iterativo e incremental propuesto por el PU puede y debe adaptarse para que satisfaga necesidades específicas del proyecto (Pressman, 2010). Sin embargo, el PU no es un proceso adecuado para todos los tipos de desarrollo, por ejemplo, el desarrollo de software integrado.

2.7.6 *Desarrollo ágil*

Hoy, las empresas operan en un entorno global que cambia rápidamente. En ese sentido, deben tener tanto la capacidad para responder ante nuevas oportunidades y mercados, como a los cambios en las condiciones económicas, así como también al surgimiento de productos y servicios competitivos. Como ya se mencionó anteriormente, el software es parte de casi todas las operaciones comerciales, por lo que el nuevo software se debe desarrollar rápidamente para aprovechar las nuevas oportunidades y hacer frente a la presión de la competencia. Por lo tanto, el desarrollo rápido y la entrega oportuna se están transformando en el requisito más crítico para estos sistemas. De hecho, muchas empresas están dispuestas a negociar la calidad del software y el compromiso con los requerimientos, para lograr con mayor celeridad la implementación.

Debido a que dichos negocios funcionan en un entorno cambiante, a menudo es prácticamente imposible derivar un conjunto completo de requerimientos de

software estable. Los requerimientos iniciales cambian de modo inevitable, porque los clientes encuentran imposible predecir cómo un sistema afectará sus prácticas operacionales, cómo interactuará con otros sistemas y cuáles operaciones de usuarios que se automatizarán. Es posible que sea sólo hasta después de entregar un sistema, y que los usuarios adquieran experiencia con éste, cuando se aclaren los requerimientos reales. Incluso, es probable que, debido a factores externos, los requerimientos cambien rápida e impredeciblemente. En tal caso, el software podría ser obsoleto al momento de entregarse (Sommerville, 2011).

Los procesos de desarrollo de software que planean especificar por completo los requerimientos y luego diseñar, construir y probar el sistema, no están orientados al desarrollo rápido de software. A medida que los requerimientos cambian o cuando se descubren problemas en los mismos, el diseño o la implementación del sistema tienen que reelaborarse y probarse de nuevo. En consecuencia, un proceso convencional en cascada o uno basado en especificación se prolongan con frecuencia, en tanto que el software final se entrega al cliente mucho después de lo que se especificó originalmente (Sommerville, 2011).

Durante algún tiempo, se reconoció la necesidad de desarrollo y de procesos de sistema rápidos que gestionaran los requerimientos cambiantes. IBM introdujo el desarrollo incremental en la década de 1980 (Mills et al., 1980). La entrada de los llamados lenguajes de cuarta generación, también en la misma década, apoyó la idea del software de desarrollo y entrega rápidos (Martin, 1981). Sin embargo, la noción prosperó realmente a finales de la década de 1990, con el desarrollo de la noción de enfoques ágiles como el DSDM, Scrum (Schwaber & Beedle, 2001) y la programación extrema (Beck, 1999, 2000) la cual probablemente sea el método ágil más conocido. El mismo integra una serie de buenas prácticas de programación, como las versiones frecuentes del software, la mejora continua del software y la participación del cliente en el equipo de desarrollo. Una ventaja particular de la programación extrema es el desarrollo de pruebas automatizadas antes de que se cree una característica del programa. Todas las pruebas deben ejecutarse con éxito cuando se integra un incremento en un sistema.

Aunque todos estos métodos ágiles se basan en la noción de desarrollo y entrega incrementales, proponen diferentes procesos para lograrlo. Sin embargo,

comparten un conjunto de características, basados en el manifiesto ágil (Sommerville, 2011):

1. Los procesos de especificación, diseño e implementación están intercalados No hay una especificación detallada del sistema, y la documentación de diseño es minimizada o generada automáticamente por el entorno de programación utilizado para implementar el sistema. El documento de requisitos del usuario solo define las características más importantes del sistema.
2. El sistema está desarrollado en una serie de versiones. Los usuarios finales y otras partes interesadas del sistema participan en la especificación y evaluación de cada versión. Pueden proponer cambios en el software y nuevos requisitos que deben implementarse en una versión posterior del sistema.
3. Las interfaces de usuario del sistema a menudo se desarrollan utilizando un sistema de desarrollo interactivo que permite que el diseño de la interfaz se cree rápidamente dibujando y colocando iconos en la interfaz. El sistema puede entonces generar una interfaz basada en web para un navegador o una interfaz para una plataforma específica como Microsoft Windows.

Sin embargo, la agilidad es algo más que una respuesta efectiva al cambio. También incluye la filosofía expuesta en el llamado “Manifiesto por el desarrollo ágil de software”. En 2001, Kent Beck y otros 16 notables desarrolladores de software, escritores y consultores (grupo conocido como la “Alianza Ágil”) firmaron el manifiesto donde se establecía lo siguiente (Beck et al., 2001):

“Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas

Software funcionando sobre documentación extensiva

Colaboración con el cliente sobre negociación contractual

Respuesta ante el cambio sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda”.

Un manifiesto normalmente se asocia con un movimiento político emergente: ataca a la vieja guardia y sugiere un cambio revolucionario (se espera que para mejorar). En cierta forma, de eso es de lo que trata el desarrollo ágil (Sommerville, 2011).

Los métodos ágiles son métodos de desarrollo incremental que se enfocan en el diseño rápido, liberaciones frecuentes del software, reducción de gastos en el proceso y producción de código de alta calidad. Hacen que el cliente intervenga directamente en el proceso de desarrollo (Sommerville, 2011).

La agilidad puede aplicarse a cualquier proceso del software. Sin embargo, para lograrlo es esencial que éste se diseñe en forma que permita al equipo del proyecto adaptar las tareas y hacerlas directas, ejecutar la planeación de manera que entienda la fluidez de un enfoque ágil del desarrollo, eliminar todos los productos del trabajo excepto los más esenciales y mantenerlos esbeltos, y poner el énfasis en una estrategia de entrega incremental que haga trabajar al software tan rápido como sea posible para el cliente, según el tipo de producto y el ambiente de operación (Sommerville, 2011).

La decisión sobre si utilizar un enfoque de desarrollo ágil o basado en el plan depende del tipo de software que se está desarrollando, las capacidades del equipo de desarrollo y La cultura de la empresa que desarrolla el sistema.

2.8 Producto y Proceso

Habiendo descrito las características principales del proceso de desarrollo de software y el contexto en el que está inmerso, se plantea el debate sobre cuál es la influencia del proceso que se elija sobre el producto que se obtiene como resultado. Si el proceso es deficiente, no cabe duda de que el producto final sufrirá. Pero también es peligrosa la dependencia excesiva del proceso. En un ensayo corto escrito hace muchos años, Margaret Davis hace comentarios atemporales sobre la dualidad del producto y del proceso (Davis, 1995): "*Cada diez años, más o menos, la comunidad del software redefine "el problema" por medio de cambiar su atención de aspectos del producto a aspectos del proceso. Así, hemos adoptado lenguajes de programación estructurada (producto) seguidos de métodos de análisis estructurados (proceso) que van seguidos por el encapsulamiento de datos (producto) a los que siguieron el énfasis actual en el modelo de madurez de la capacidad, del Instituto de Ingeniería de Software para el Desarrollo de Software*

(proceso) (seguido por métodos orientados a objetos, a los que sigue el desarrollo ágil de software). En tanto que la tendencia natural de un péndulo es alcanzar el estado de reposo en el punto medio entre dos extremos, la atención de la comunidad del software cambia constantemente porque se aplica una nueva fuerza al fallar la última oscilación. Estos vaivenes son dañinos en sí mismos porque confunden al profesional promedio del software al cambiar en forma radical lo que significa hacer el trabajo bien. Los cambios periódicos no resuelven "el problema" porque están predestinados a fallar toda vez que el producto y el proceso son tratados como si fueran una dicotomía en lugar de una dualidad. (..) Privilegiar un punto de vista sobre el otro reduce mucho las oportunidades para la reutilización y, por tanto, se pierde la oportunidad de aumentar la satisfacción por el trabajo. La gente obtiene tanta (o más) satisfacción del proceso creativo como del producto final. Un artista disfruta las pinceladas tanto como el resultado que enmarca".

Es una realidad que diferentes tipos de sistemas necesitan diferentes procesos de desarrollo. Por ejemplo, el software en tiempo real en un avión debe especificarse completamente antes de que comience el desarrollo. En los sistemas de comercio electrónico, la especificación y el programa generalmente se desarrollan juntos. En consecuencia, estas actividades genéricas pueden organizarse de diferentes maneras y describirse con diferentes niveles de detalle según el tipo de software que se esté desarrollando. Un buen proceso sin duda contribuye a la creación de un buen producto, pero también podemos seguir un modelo de procesos ampliamente reconocido, y que no sea el más adecuado para la organización y que por ello el producto no acabe siendo el mejor. Es por eso, que es una mala interpretación asumir que, cumpliendo cierto modelo, nivel de madurez, estándar, norma, metodología, ciclo de vida, o cualquier otra cosa relacionada con el proceso se asegura de forma directa un producto de calidad. Sin embargo, si se tiene en cuenta el tipo de producto a desarrollar, las capacidades con las que debe contar el equipo de desarrollo y el contexto de la compañía, se pueden establecer un marco metodológico que combine las virtudes de los modelos reconocidos y aumente sustancialmente la calidad del producto que se entregue.

CAPÍTULO 3: LA ADMINISTRACIÓN DE LA CALIDAD Y EL TESTING DE SOFTWARE

Tal como se detalló en el capítulo anterior, una de las actividades principales del proceso de desarrollo de software, independientemente del modelo que se utilice, es el Testing o pruebas del software.

A lo largo de la evolución de las pruebas, su aplicación en el desarrollo de software resulta esencial a la hora de garantizar la calidad del mismo, ya que estas técnicas representan una revisión final de las especificaciones, del diseño y de la codificación. A medida que los clientes se han vuelto cada vez más selectivos y han comenzado a rechazar los productos poco fiables o que realmente no dan respuesta a sus necesidades, la calidad del software ha ido ganando peso en la ingeniería del software y con ella las pruebas. Sin embargo, sin una definición clara, concisa y medible de lo que es la calidad del software, no es posible tomar buenas decisiones de negocio respecto al uso de los recursos, ni en qué áreas mejorar la calidad, ni qué herramientas y técnicas utilizar para la mejora de la misma.

3.1 Cambio en el enfoque de calidad

En la década de 1990, las principales corporaciones reconocieron que cada año se desperdiciaban miles de millones de dólares en software que no tenía las características ni la funcionalidad que se habían prometido (Pressman, 2010).

Lo que era peor, tanto el gobierno como la industria se preocupaban por la posibilidad de que alguna falla de software pudiera afectar infraestructura importante y provocara pérdidas de decenas de miles de millones de dólares. Al comenzar el nuevo siglo, CIO Magazine (Levinson, 2001) dio la alerta: "*Dejemos de desperdiciar $78 mil millones de dólares al año*", y lamentaba el hecho de que "*las empresas estadounidenses gastan miles de millones de dólares en software que no hace lo que se supone que debe hacer*". InformationWeek se hizo eco de la misma preocupación con su artículo "El Estado de la Calidad del Software" (Ricadel, 2001).

En 2005, ComputerWorld (Hildreth, 2005) se quejaba de que "*el mal software es una plaga en casi todas las organizaciones que emplean computadoras, lo que ocasiona horas de trabajo perdidas por el tiempo que están fuera de uso las*

máquinas, por datos perdidos o corrompidos, oportunidades de venta perdidas, costos elevados de apoyo y mantenimiento, y poca satisfacción del cliente". Un año después, InfoWorld (Foster, 2006) escribió acerca del "*lamentable estado de la calidad del software*" e informaba que el problema de la calidad no había mejorado. Todas estas manifestaciones sobre la producción de software a nivel global, obligaron a las organizaciones a replantearse algunas cuestiones sobre el desarrollo del mismo como producto masivo. Tradicionalmente, los esfuerzos para mejorar la calidad se habían centrado en el final del ciclo de desarrollo del producto, enfatizando la detección y corrección de defectos. Por el contrario, el nuevo enfoque para mejorar la calidad debe abarcar todas las fases en el proceso de desarrollo del producto, desde un análisis de requerimientos hasta la entrega final del producto al cliente. Cada paso en el proceso de desarrollo debe realizarse con el más alto estándar posible. Un proceso de calidad efectivo debe centrarse en (Gebelein et al., 1998):

- Prestar mucha atención a los requerimientos del cliente.
- Hacer esfuerzos para mejorar continuamente la calidad.
- Integrar los procesos de medición con el de diseño y el de desarrollo del producto.
- Llevar el concepto de calidad al nivel más bajo de la organización.
- Desarrollar una perspectiva a nivel de sistema con énfasis en la metodología y el proceso
- Eliminar los excedentes mediante la mejora continua.

3.2 ¿Qué es la Calidad?

Este cambio de enfoque nos lleva a preguntarnos exactamente qué es la calidad. En un nivel algo pragmático, David Garvin (Garvin, 1984), de Harvard Business School, sugiere que "*la calidad es un concepto complejo y de facetas múltiples*" que puede describirse desde cinco diferentes puntos de vista. El *punto de vista trascendental* dice que la calidad es algo que se reconoce de inmediato, pero que no es posible definir explícitamente. El *punto de vista del usuario* concibe la calidad en términos de las metas específicas del usuario final. Si un producto las satisface, tiene calidad. El *punto de vista del fabricante* la define en términos de las especificaciones originales del producto. Si este las cumple, tiene calidad. El *punto*

de vista del producto sugiere que la calidad tiene que ver con las características inherentes (funciones y características) de un producto. Por último, el *punto de vista basado en el valor* la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto. En realidad, la calidad incluye todo esto y más.

Por su parte, la *calidad del diseño* se refiere a las características que los diseñadores especifican para un producto. El tipo de materiales, tolerancias y especificaciones del desempeño, todo contribuye a la calidad del diseño. Si se utilizan mejores materiales, tolerancias más estrictas y se especifican mayores niveles de desempeño, la calidad del diseño de un producto se incrementa si se fabrica de acuerdo con las especificaciones.

En el desarrollo del software, la *calidad del diseño* incluye el grado en el que el diseño cumple las funciones y características especificadas en el modelo de requerimientos. La *calidad de la conformidad* se centra en el grado en el que la implementación se apega al diseño y en el que el sistema resultante cumple sus metas de requerimientos y desempeño.

En este sentido nace la pregunta si la calidad de diseño y la calidad de conformidad son los únicos aspectos a tener en cuenta para el desarrollo de software. Robert Glass afirma que es mejor plantear una relación más intuitiva:

Satisfacción del Usuario	**=**	**Producto que Funciona**	**+**	**Buena Calidad**	**+**	**Entrega dentro del Presupuesto y Plazo**

En última instancia, Glass sostiene que la calidad es importante, pero que, si el usuario no está satisfecho, nada de lo demás importa. DeMarco refuerza esta opinión al decir que "*la calidad de un producto está en función de cuánto cambia al mundo para bien*". Este punto de vista de la calidad afirma que si un producto de software beneficia mucho a los usuarios finales, éstos se mostrarán dispuestos a tolerar problemas ocasionales de confiabilidad o desempeño (Pressman, 2010).

3.3 La calidad del software

A lo largo de los años, tal como se mencionó anteriormente, los autores y las organizaciones la han definido de manera diferente. Desde el punto de vista más general Roger Pressman define la calidad de software como (Pressman, 2010):

"*Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan*".

Por su parte, en SWEBOK, la calidad del software se refiere a "las características deseables de los productos de software, en la medida en que un producto de software en particular posee esas características, y a los procesos, herramientas y técnicas utilizadas para lograr esas características" (Bourque & Fairley, 2014).

Como se mencionó en el capítulo anterior, es importante diferenciar entre la calidad del producto y la calidad del proceso de desarrollo. No obstante, las metas que se establezcan para la calidad del producto van a determinar las metas a establecer para la calidad del proceso, ya que la calidad del producto se va a encontrar íntimamente ligada a la calidad del proceso de desarrollo. Sin un buen proceso de desarrollo resulta prácticamente imposible obtener un buen producto (Wallace, D.R. Fujii, 1989).

Con el fin de establecer un conjunto de propiedades sobre los productos y los procesos del software que permitan establecer su nivel de calidad surgen los modelos de calidad, los cuales se detallarán con mayor profundidad más adelante. Es importante señalar que no basta con tener en cuenta la calidad una vez se ha finalizado el producto. Ésta debe ser considerada en todos sus estados de evolución (especificaciones, diseño, código, etc.). Además, si surge cualquier problema durante su desarrollo que perjudique su calidad, cuanto antes sea atacado el sobrecosto ocasionado será menor. Por lo cual, es necesario establecer una serie de actividades con el fin de evaluar la calidad de los productos y los propios procesos desarrollados que establecen el control de la calidad. El proceso de verificación y validación tiene especial importancia dentro de estas actividades.

3.4 Verificación y Validación

Como se detalló en el capítulo anterior, el proceso de desarrollo adoptado por un proyecto dependerá de los objetivos y metas que tenga el mismo. Para poder alcanzar dichos objetivos se han desarrollado distintos modelos de ciclo de vida, pero en cualquiera de ellos será necesario un proceso que asegure la calidad del producto durante su desarrollo. Para ello, al final de cada fase del ciclo de vida debería comprobarse que el trabajo realizado hasta ese momento cumple con los objetivos previstos. Este es el punto clave, en el que tiene lugar la evaluación del

producto, donde se decide si está o no preparado para pasar a la siguiente fase. De esta forma, si hay errores y son detectados, será más eficiente corregirlos que si se descubriesen en etapas más avanzadas.

La verificación y validación (V&V) es un conjunto de procedimientos, actividades, técnicas y herramientas que se utilizan, paralelamente al desarrollo de software, para asegurar que un producto software resuelve el problema inicialmente planteados (Sommerville, 2005). La verificación propiamente se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica. La validación es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente (Pressman, 2010). Boehm afirma esto de esta forma:

Verificación: "*¿Construimos el producto correctamente?"*

Validación: *"¿Construimos el producto correcto?"*

Las tareas de V&V se aplican tanto a los productos de software como a los resultantes del proceso de desarrollo, como al análisis y a la especificación de requisitos, por ejemplo. De esta forma, se comprueba que el proyecto es viable y que las especificaciones documentadas son completas, correctas, precisas, legibles, evaluables, y que, en general, responden a las expectativas del cliente (Sommerville, 2005). La V&V del diseño debe garantizar que los requisitos no se encuentran incompletos o incorrectamente diseñados. En el caso de la implementación y la codificación, la V&V de software es comúnmente conocida como las **pruebas de software**.

Existen muchas definiciones incorrectas sobre las pruebas del software, las cuáles se discutirán más adelante en este capítulo ya que pueden conducir a una inadecuada aplicación de este proceso. Lo importante en esta instancia es aclarar que se realizan con el fin de detectar errores que, una vez corregidos, mejoran la calidad o la fiabilidad del mismo. Existen distintos tipos de pruebas en función de la unidad de software a la que se aplique y del objetivo que se persiga, por ejemplo, las pruebas de unidad, de integración, de sistema y de aceptación.

Finalmente, las actividades de V&V son también necesarias durante la operación y el mantenimiento del software. Cuando se realiza un cambio en el mismo, se debe examinar el impacto de este sobre el sistema y considerar qué actividades se

necesitan repetir para garantizar, al menos, la misma calidad en el software antes del cambio.

Otro factor a tener en cuenta es la independencia del proceso de V&V, ya que es una característica reconocida como altamente positiva en distintas áreas, reportando dentro del desarrollo software una serie de características importantes (Sommerville, 2005):

- La separación de intereses, por la cual se evitan conflictos de intereses internos en una organización. Una organización independiente garantiza que requisitos importantes no sean ignorados en el proceso de toma de decisiones.
- La diferenciación de puntos de vista, esto supone una segunda opinión, que siempre es interesante para eliminar ambigüedades u omisiones.
- La efectividad y la productividad, las actividades de V&V son llevadas a cabo por personal experto, con competencias específicas en el área de V&V.

A pesar de estas características, es importante tener en cuenta que la independencia no debe ser un muro infranqueable entre el equipo de desarrollo y el equipo de V&V, por ello es importante fomentar un entorno de participación y de aprendizaje hacia un objetivo común.

3.5 Técnicas de verificación y validación

En términos generales, las actividades para la evaluación de la calidad del software se pueden dividir en dos categorías amplias (Friedman & Voas, 1995), el análisis estático y el análisis dinámico.

3.5.1 *Análisis estático*

Las técnicas de análisis estático (Sommerville, 2011) son técnicas de verificación del sistema que no incluyen la ejecución de un programa. En vez de ello, funcionan sobre una representación fuente del software: un modelo de especificación o diseño, o el código fuente del programa. Las técnicas de análisis estático pueden usarse para comprobar la especificación y los modelos de diseño de un sistema con la finalidad de buscar errores antes de que esté disponible una versión ejecutable del sistema, con todas las ventajas que conlleva esta detección temprana.

El análisis estático tradicional incluye la revisión del código, la inspección, walkthroughs[2], el análisis de algoritmos y pruebas de corrección (Naik & Priyadarshi, 2008). Otra técnica consiste en utilizar herramientas de modelado de diseño para comprobar anomalías en el UML, por ejemplo, el hecho de que el mismo nombre sea usado por diferentes objetos.

Sin embargo, en la actualidad, se emplean herramientas que permiten realizar este tipo de análisis de forma automática, en conjunto con técnicas adicionales como (Sommerville, 2011):

1. Verificación formal, en la que se producen argumentos matemáticamente rigurosos de que un programa se conforma a su especificación.
2. Comprobación de modelo (o Model Checking), en la que se usa un verificador de teoremas para revisar una descripción formal del sistema en busca de inconsistencias. Hoy en día hay muchas herramientas disponibles para realizar model checking, muchas de estas herramientas se basan en Binary Decision Diagrams (BDD).
3. Herramientas de simulación de procesos de negocio: el Modelado de Procesos de Negocio (Business Process Modeling, BPM) y la Simulación de Procesos de Negocio (Business Process Simulation, BPS), ha traído consigo la aparición en el mercado de un gran número de herramientas que permitan realizar estas tareas con el beneficio adicional de poder validar los procesos analizados y diseñados en etapas tempranas del proyecto.
4. Análisis automatizado del programa, en el que el código fuente de un programa se revisa por patrones conocidos que son potencialmente erróneos. Actualmente, este tipo de análisis se ha ampliado y se han desarrollado diversas técnicas y herramientas, algunas de ellas aplicables a nivel industrial. Algunas herramientas se utilizan para la verificación de propiedades de seguridad, como por ejemplo detección de posibles buffer overflows o inyección de código, mientras que otras tratan de detectar problemas de uso adecuado de componentes o detección de condiciones de carreras en código concurrente

[2] **Walkthroughs**: es una técnica de análisis estático en la que un diseñador o programador dirige a los miembros del equipo de desarrollo y a otras partes interesadas a través de un segmento de documentación o código, y los participantes hacen preguntas y hacen comentarios sobre posibles errores, violaciones de los estándares de desarrollo y otros problemas (Veenendaal, 2010).

Sin embargo, a pesar de que las técnicas de verificación estáticas son usadas cada vez más y su implementación es poco compleja, el análisis dinámico sigue siendo la técnica predominante en la validación y verificación.

3.5.2 *Análisis dinámico*

El análisis dinámico, por su parte, implica la ejecución real del programa con el objetivo de exponer posibles fallas, así como también de observar su comportamiento y rendimiento. Los programas se ejecutan con valores de entrada típicos y cuidadosamente seleccionados. A menudo, el conjunto de entradas posibles de un programa puede ser extremadamente largo y complejo. Sin embargo, por consideraciones prácticas, se puede seleccionar un subconjunto finito del conjunto de entradas posibles. Por lo tanto, en las pruebas, se observan algunos comportamientos representativos del programa y se llega a una conclusión sobre la calidad del sistema (Naik & Priyadarshi, 2008). Además, son consideradas como la única técnica de validación para los requerimientos no funcionales, debido a que el software debe de ser ejecutado para ver su comportamiento, proporcionando confianza en el software.

Con el fin de obtener pruebas lo más sistemáticas posibles, se busca identificar ese conjunto representativo de comportamientos del programa, determinados por subclases del dominio de las entradas, los escenarios y los estados, diferenciando dos enfoques (Pressman, 2010):

- **Pruebas de Caja Negra:** Las pruebas dinámicas de caja negra (también llamadas pruebas de comportamiento, basada en datos o de entrada/salida) se enfocan en los requerimientos funcionales del software. Para usar esta técnica (Myers et al., 2012), se debe estar completamente despreocupado por el comportamiento interno y la estructura del programa. En su lugar, el objetivo es encontrar circunstancias en las que el programa no se comporte de acuerdo con sus especificaciones. En este enfoque, los datos de prueba se derivan únicamente de las especificaciones (es decir, sin aprovechar el conocimiento de la estructura interna del programa).

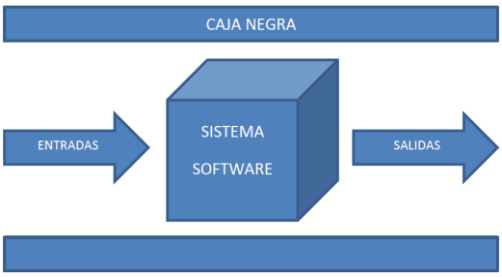

Figura 3-1 Prueba de caja negra

Las pruebas de caja negra intentan encontrar errores en las categorías siguientes: 1) funciones incorrectas o faltantes, 2) errores de interfaz, 3) errores en las estructuras de datos o en el acceso a bases de datos externas, 4) errores de comportamiento o rendimiento y 5) errores de inicialización y terminación. Además, a diferencia de las pruebas de caja blanca que se realizan tempranamente en el proceso de pruebas, la prueba de caja negra tiende a aplicarse durante las últimas etapas.

- **Pruebas de Caja Blanca:** La prueba de caja blanca, en ocasiones llamada prueba de caja de vidrio, permite examinar la estructura interna del programa derivando los datos de prueba a partir de un examen de la lógica del programa. La meta es diseñar pruebas que proporcionen algún nivel de cobertura de programa. Esto es, el conjunto de pruebas debe garantizar que se ejecute toda ruta lógica a través del programa, con la consecuencia de que cada enunciado del programa se efectúe al menos una vez (Sommerville, 2011). En contraposición al enfoque anterior, aplicar Caja Blanca, exige un conocimiento elevado del lenguaje de programación y del código fuente del sistema que se está probando. Este tipo de pruebas dependen directamente de la experiencia de quien realiza las pruebas y en algunas ocasiones es imposible cubrir todos los caminos posibles por lo que puede pasar el sistema, es decir, la cobertura no puede llegar a ser del 100%.

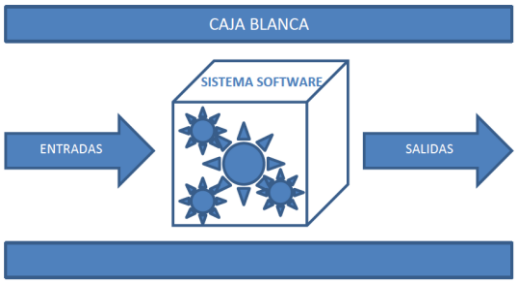

Figura 3-2 Pruebas de Caja Blanca

Las pruebas dinámicas son necesarias, pero no suficientes para proporcionar una seguridad razonable de que el software funcionará según lo previsto. Las actividades de pruebas estáticas adicionales deben realizarse en combinación con actividades de pruebas dinámicas efectivas (*ISO/IEC/IEEE 29119:2013 Software Testing Strandard*, 2013). El análisis estático y el análisis dinámico son complementarios por naturaleza, y para una mejor efectividad, ambos deben realizarse repetidamente y de forma alternada. Los profesionales e investigadores deben eliminar los límites entre el análisis estático y dinámico y crear un análisis híbrido que combine las fortalezas de ambos enfoques (Ernst, 2003).

3.6 El Testing o Prueba de Software

En el campo de la ingeniería del software, podemos encontrar diferentes definiciones para la actividad de Testing o prueba de software. Según la IEEE, el Testing es el proceso de evaluación de un sistema o componente para verificar que satisface los requisitos especificados, o identificar diferencias entre lo esperado y los resultados obtenidos. En el SWEBOK, se define como una actividad realizada para evaluar la calidad del producto y mejorarla, identificando defectos y problemas. Además, aclara que "*se trata de la verificación dinámica del comportamiento de un programa contra el comportamiento esperado, usando un conjunto finito de casos de prueba, seleccionados de manera adecuada desde el dominio infinito de ejecución*" (Bourque & Fairley, 2014). De todas formas, quizás una de las definiciones más acertadas y que más ha perdurado en el tiempo es la propuesta por Myers en 1979 en su libro *The Art Of Software Testing* donde dice que: "*El Testing es el proceso de ejecutar un programa o sistema con la intención de encontrar errores*"(Myers et al., 2012).

Analizando las distintas definiciones se puede extraer un objetivo común a todas ellas. El mismo se trata de proporcionar información sobre la calidad del elemento

que se está probando y cualquier riesgo residual en relación a cuánto se ha probado el elemento; para encontrar defectos en el mismo antes de su lanzamiento para su uso; y para mitigar los riesgos de la calidad deficiente del producto para las partes interesadas (*ISO/IEC/IEEE 29119:2013 Software Testing Strandard*, 2013). Esto asegura que las pruebas de software cubren el ciclo completo de vida del producto desde su desarrollo hasta su mantenimiento. Este concepto da lugar al proceso de verificación y validación del software.

3.7 Actividades del Testing

Con el fin de desarrollar las pruebas de una manera óptima, los casos de prueba cobran especial importancia. Estos son un conjunto de entradas, condiciones de ejecución y resultados esperados que son desarrollados con el fin de cumplir un determinado objetivo, de tal forma que ejecute un camino específico dentro del programa o que verifique su adecuación con los requisitos (IEEE, 1990).

Las pruebas del software se diferencian del resto de actividades del proceso de desarrollo software en que éstas son un proceso destructivo. Lo cual significa que el objetivo del *tester*, que es quien realiza las pruebas, es el de descubrir errores, es decir, encontrar el mayor número de errores. Por ello, un buen caso de pruebas es el que permite localizar un gran número de errores. El trabajo de tester de software, independientemente de la metodología que se utilice y del nivel en que se realicen las pruebas, incluye la siguiente secuencia de actividades (Naik & Priyadarshi, 2008):

1. Identificar un objetivo a ser probado. El objetivo define la intención o el propósito de diseñar uno o más casos de prueba para garantizar que el programa respalda el objetivo. Un propósito claro debe estar asociado con cada caso de prueba.
2. Selección de entradas. Pueden basarse en la especificación de requisitos, el código fuente o nuestras expectativas.
3. Estimar el resultado esperado. En la mayoría de los casos, esto se puede hacer a partir de una comprensión general de alto nivel del objetivo de la prueba y la especificación del programa bajo prueba.
4. Configurar el entorno de ejecución del programa. En este paso se deben cumplir todos los supuestos externos al programa.

5. Ejecutar el programa. En el quinto paso, el ingeniero de pruebas ejecuta el programa con las entradas seleccionadas y observa el resultado real del programa.
6. Analizar el resultado de la prueba. La actividad de prueba final es analizar el resultado de la ejecución anterior. Aquí, la tarea principal es comparar el resultado real de la ejecución del programa con el resultado esperado. La complejidad de la comparación depende de la complejidad de los datos a observar.

   Hay tres tipos principales de resultados de una prueba: aprobado, desaprobado e inconcluyente.

   - Si el programa produce el resultado esperado y se cumple el propósito del caso de prueba, se considera *aprobado*.
   - Si el programa no produce el resultado esperado, se considera *no aprobado*.
   - Una prueba *no concluyente* significa que se deben realizar más exámenes para refinar el veredicto en uno claro de aprobación o rechazo.

   Se debe escribir un informe después de analizar el resultado de la prueba. Debe contener los siguientes elementos: 1) Explicar cómo reproducir el error. 2) Analizar el fracaso para poder describirlo. 3) Un indicador del resultado real y 4) el caso de prueba, completo con la entrada, el resultado esperado y el entorno de ejecución (Naik & Priyadarshi, 2008).

3.8 Niveles de Testing

Las pruebas de software se realizan generalmente en diferentes niveles a lo largo del proceso de desarrollo y mantenimiento. Los niveles se pueden distinguir en función del objeto sujeto a la prueba o del ámbito o destino de las mismas y son aquellos que estructura el proceso de V&V.

El objeto de prueba puede variar: un solo módulo, un grupo de módulos (relacionados por funcionalidad, uso, comportamiento o estructura) o un sistema completo. Se pueden distinguir diferentes ámbitos de prueba y no implican ningún modelo de proceso. Además, se supone que ninguna de ellas es más importante que el resto.

La ventaja de esta estructura es que evita las pruebas redundantes improductivas y además evita que se pasen por alto diferentes clases de errores. La necesidad de pruebas de orden superior tales como las pruebas funcionales, de sistema y de aceptación aumenta junto con el tamaño del programa. La razón es que la proporción de errores de diseño (errores cometidos en los procesos de desarrollo anteriores) y errores de codificación es considerablemente mayor en programas grandes que en programas pequeños (Myers et al., 2012).

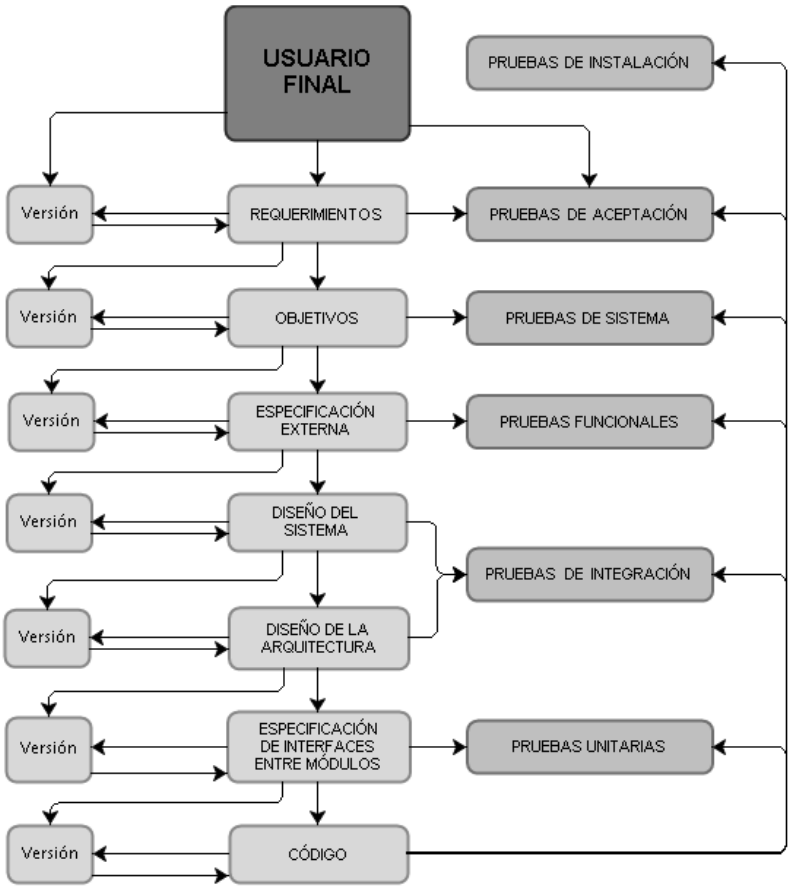

Figura 3-3 La correspondencia entre el proceso de desarrollo y el proceso de pruebas (adaptado de Myers et al., 2012)

Se debe tener en cuenta que la secuencia de los procesos de prueba en la Figura 3-3 no implica necesariamente una secuencia de tiempo. Por ejemplo, dado que la prueba de sistema no se define como "el tipo de prueba que se realiza después de la prueba funcional", sino como un tipo distinto de pruebas centrada en una clase distinta de errores, podrían superponerse parcialmente en el tiempo con otros tipos de pruebas (Myers et al., 2012). Se trata de una situación similar a la de las fases de desarrollo de software descritas en el capítulo anterior.

3.8.1 *Pruebas unitarias*

Las pruebas unitarias verifican los subprogramas individuales, subrutinas, clases o procedimientos en un programa. Más específicamente, en lugar de probar inicialmente el programa en su totalidad, las pruebas se centran primero en los bloques de construcción más pequeños del programa. Estas pruebas suelen llevarse a cabo con acceso al código fuente probado y con ayuda de herramientas de depuración. Frecuentemente, los programadores que escribieron el código conducen estas pruebas, pero no siempre (Bourque & Fairley, 2014).

Las pruebas unitarias son una forma de administrar los elementos combinados a testear, ya que la atención se centra inicialmente en unidades más pequeñas del programa. Además, facilita la tarea de depuración (el proceso de localización y corrección de un error descubierto), ya que, cuando se encuentra un error, se sabe que existe en un módulo en particular. Introduce paralelismo en el proceso de prueba del programa al presentar la oportunidad de probar varios módulos simultáneamente.

El procedimiento de diseño de caso de prueba para una prueba unitaria es el siguiente: Analizar la lógica del módulo utilizando uno o más métodos de caja blanca, y luego complementar esos casos de prueba aplicando métodos de caja negra para las especificaciones del módulo (Myers et al., 2012).

3.8.2 *Pruebas de integración*

Las pruebas de integración verifican las interacciones entre los componentes del software. Las estrategias de integración clásicas, como las de *top-down* (de arriba hacia abajo comenzando con el módulo superior o inicial del programa) y *bottom-up* (desde los módulos terminales hacia a el o los superiores o iniciales), se usan a menudo con software estructurado jerárquicamente.

Las estrategias de integración sistémica modernas están basadas en la arquitectura, lo que implica integrar incrementalmente los componentes o subsistemas de software a partir de la identificación de hilos funcionales. Las pruebas de integración suelen ser una actividad continua en cada etapa del desarrollo durante el cual los ingenieros de software abstraen las perspectivas de bajo nivel y se concentran en la visión del nivel en el que se están integrando (Bourque & Fairley, 2014). Este tipo de pruebas a menudo no se consideran como

un paso de prueba separado; ya que cuando se utilizan las pruebas unitarias de forma incremental, es una parte implícita en dicho proceso (Myers et al., 2012).

3.8.3 *Pruebas Funcionales*

El propósito de las pruebas funcionales es mostrar que un programa no coincide con sus especificaciones externas. Se entiende por especificación externa a una descripción precisa del comportamiento del programa, desde el punto de vista del usuario final.

Los programas no escritos como productos a menudo no poseen requerimientos y objetivos formales. Para tales programas, las pruebas funcionales podrían ser las únicas pruebas de orden superior a ser ejecutadas. Además, la necesidad de pruebas de orden superior aumenta junto con el tamaño del programa. La razón es que la proporción de errores de diseño (errores cometidos en los procesos de desarrollo anteriores) y errores de codificación es considerablemente mayor en programas grandes que en programas pequeños (Myers et al., 2012).

Excepto cuando se utiliza en programas pequeños, la prueba de funcionamiento es normalmente una actividad de caja negra. Es decir, se confía en el proceso anterior de pruebas unitarias para lograr los criterios de cobertura lógicos de caja blanca deseados.

3.8.4 *Pruebas de Sistema*

Las pruebas de sistema se ocupan de probar el comportamiento de un sistema completo. Las pruebas unitarias y de integración habrán identificado muchos de los defectos del software. La prueba del sistema es generalmente se considera apropiado para evaluar los requerimientos no funcionales del sistema, como la seguridad, velocidad, precisión y fiabilidad. Las interfaces externas a otras aplicaciones, utilidades, dispositivos de hardware o entornos operativos también suelen evaluarse en este nivel (Bourque & Fairley, 2014). Las pruebas de sistema no verifican las funciones del sistema o programa completo, ya que esto sería redundante con el proceso de pruebas funcionales. Tiene como propósito particular comparar el sistema o programa con sus objetivos originales tal como se indica en la Figura 3-4.

Este nivel de pruebas es el más incomprendido y difícil de todos. Conociendo el objetivo de este nivel de pruebas, debemos tener en cuenta las siguientes consideraciones (Myers et al., 2012):

- Las pruebas del sistema son el proceso de intentar demostrar cómo el programa, en su totalidad, no cumple con sus objetivos.
- Las pruebas de sistema, por definición, son imposibles si no hay un conjunto de objetivos medibles y escritos para el producto.

En la búsqueda de discrepancias entre el programa y sus objetivos, se debe hacer foco en los errores de traducción cometidos durante el proceso de diseño de la especificación externa. Esto hace que las pruebas de sistema sean un proceso vital ya que, en términos del producto, esta fase del ciclo de desarrollo suele ser más propensa a cometer mayor cantidad de errores y de mayor gravedad. Por lo que implica que, a diferencia de las pruebas funcionales, la especificación externa no se puede utilizar como base para derivar los casos de prueba de sistema, ya que este altera el propósito de las pruebas de sistema. Por otro lado, no se puede utilizar sólo el documento de objetivos para formular casos de prueba ya que no contiene, por definición, descripciones precisas de las interfaces externas de los programas. Este dilema se resuelve utilizando la documentación o publicaciones del usuario final del programa (Myers et al., 2012).

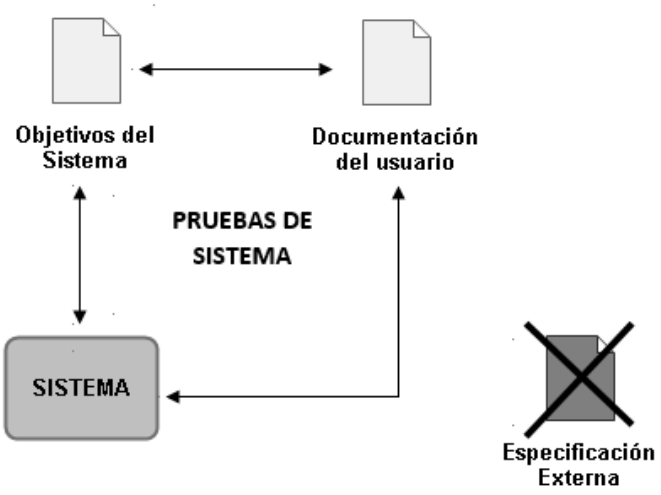

Figura 3-4 Herramientas para la creación de casos de prueba de sistemas (Adaptado de Myers et al., 2012)

3.8.5 *Pruebas de Aceptación*

Las pruebas de aceptación comparan el programa con sus requerimientos iniciales y las necesidades actuales de sus usuarios finales. Es un tipo de prueba inusual ya

que generalmente lo realiza el cliente del programa o el usuario final y normalmente no se considera responsabilidad de la organización de desarrollo.

En el caso de un programa contratado, la organización contratante (usuario) realiza la prueba de aceptación comparando la operación del programa con el contrato original. Como es el caso de otros tipos de pruebas, la mejor manera de hacer esto es idear casos de prueba que intenten demostrar que el programa no cumple con el contrato. Si esos casos de prueba no tienen éxito, el programa es aceptado (Myers et al., 2012). En el caso de un producto de software, como un sistema operativo o un software de base de datos, el cliente sensato primero realiza una prueba de aceptación para determinar si el producto satisface sus necesidades de realizar cualquier tipo de contrato.

Al estar el foco en la satisfacción del cliente, este tipo de pruebas se utiliza como una medida de calidad del producto. Una noción clave (Naik & Priyadarshi, 2008) son las expectativas del cliente ya que, al momento de realizar estas pruebas, el cliente debería haber desarrollado sus criterios de aceptación basados en sus propias expectativas del sistema.

De todas formas, si bien la prueba de aceptación final es responsabilidad del cliente o usuario final, un desarrollador inteligente realizará pruebas de usuario durante el ciclo de desarrollo y antes de entregar el producto terminado (Myers et al., 2012).

3.8.6 *Pruebas de Instalación*

El proceso de prueba restante es la prueba de instalación. Su posición en la Figura 3-3 - La correspondencia entre el proceso de desarrollo y el proceso de pruebas (adaptado de Myers et al., 2012) es un poco inusual, ya que no está relacionada, como todos los demás procesos de prueba a una actividad específica del proceso de diseño. Es un tipo de prueba inusual porque su propósito no es encontrar errores de software sino encontrar errores que ocurren durante el proceso de instalación.

La organización que produjo el sistema debe desarrollar las pruebas de instalación, deben entregarse como parte del sistema y ejecutarse después de que el sistema esté instalado. Entre otras cosas, los casos de prueba pueden verificar que se haya seleccionado un conjunto de opciones compatibles, que todas las partes del sistema existan, que todos los archivos se hayan creado y tengan los contenidos

necesarios, y que la configuración del hardware sea la adecuada (Myers et al., 2012).

3.9 Los principios de las pruebas

A pesar de que las pruebas del software son una tarea técnica, siempre son llevadas a cabo por alguien, por lo que tanto el aspecto humano como el económico resultan especialmente importantes. Myers (Myers et al., 2012) define diez principios fundamentales:

- Una parte necesaria de un caso de prueba es la definición de los resultados esperados.
- Un programador debe evitar probar su propio desarrollo.
- La organización que desarrolla no debería probar sus propios programas.
- La inspección detallada de cada resultado de las pruebas.
- Los casos de prueba deben de ser escritos teniendo en cuenta las condiciones inválidas o inesperadas, de la misma forma que se hacen con condiciones válidas y esperadas.
- La comprobación de que un programa hace lo que debería, y que no hace lo que no.
- Evitar los casos de prueba de un solo uso a no ser que el programa también lo sea.
- No realizar una planificación de pruebas asumiendo que no se encontraran errores.
- La probabilidad de encontrar más errores en una sección es proporcional al número de errores encontrados hasta el momento.
- Las pruebas son tareas que implican con una gran carga creativa e intelectual.

3.10 Estándares de software

Dentro de la administración de la calidad del software los estándares cumplen una función muy importante ya que proporcionan un marco para definir, en un escenario particular, lo que significa el término “calidad”. Como se dijo anteriormente, la calidad del software es subjetiva, y al usar estándares se establece una base para decidir si se logró o no un nivel de calidad requerido. Desde luego, esto depende de cómo se establezcan los mismos para que reflejen las expectativas del usuario

sobre la confiabilidad, la usabilidad y el rendimiento del software. Además, estos aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas. En consecuencia, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo (Sommerville, 2011).

Se pueden identificar dos tipos de estándares de ingeniería de software relacionados que pueden definirse y usarse en la gestión de calidad del software (Sommerville, 2011):

- **Estándares del producto:** Se aplican al producto de software a desarrollar. Incluyen estándares de documentos (como la estructura de los documentos de requerimientos), estándares de documentación (como el encabezado de un comentario estándar para una definición de clase de objeto) y estándares de codificación, los cuales definen cómo debe usarse un lenguaje de programación.
- **Estándares de proceso***:* Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar cómo es una buena práctica de desarrollo. Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Cabe aclarar que, los estándares deben entregar valor en la forma de calidad aumentada del producto. No hay razón para definir estándares que sean costosos en términos de tiempo y esfuerzo, pues aplicarlos sólo conduce a mejoras secundarias en la calidad. Los estándares de producto deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos, y los estándares de proceso deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.

A continuación, se mencionan aquellos estándares que han tenido mayor influencia en la evolución de la concepción de calidad para la producción de software y se detalla dos de los más importantes para el desarrollo de este trabajo: ISO/IEC 29110 y ISO/IEC 29119.

Normas ISO:

- **ISO 9000 (2006):** define el vocabulario común a todos los documentos de la familia ISO 9000, aplicables a cualquier área de trabajo que requiera un sistema de gestión de la calidad
- **ISO 9001 (2015):** toma el vocabulario del ISO 9000 y lo utiliza en la definición del marco conceptual de trabajo y requerimientos de un sistema de gestión de la calidad.

Normas ISO/IEC:

- **ISO/IEC 9003 (2004):** en ella, el esfuerzo es invertido en hallar una correspondencia entre los requerimientos planteados en ISO 9001, las prácticas técnicas y de gerencia de desarrollo de software, esto con el objetivo de hallar una adaptación del modelo de sistema de gestión de la calidad aplicable al ámbito del software.
- **ISO/IEC 9126 (2001):** Este estándar proviene del modelo establecido en 1977 por McCall y sus colegas, los cuales propusieron un modelo para especificar la calidad del software (McCall et al., 1977). El modelo está organizado sobre tres tipos de Características de Calidad:
    - Factores (especificar): Describen la visión externa del software, como es visto por los usuarios.
    - Criterios (construir): Describen la visión interna del software, como es visto por el desarrollador.
    - Métricas (controlar): Se definen y se usan para proveer una escala y método para la medida.

  En este sentido, el estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software.
- **ISO/IEC 2500n-ISO/IEC 2504N (SQuaRE) (2005):** es una colección de 23 estándares y reportes técnicos reunidos bajo el nombre común de Software engineering-Software product Quality Requirements and Evaluation (SQuaRE). Estos documentos pueden clasificarse en cinco divisiones: Gestión de la Calidad, Modelo de Calidad, Medición de la Calidad, Requerimientos de la Calidad y Evaluación de la Calidad. SQuaRE reemplaza a la serie de normas ISO/IEC 9126.

Estándares IEEE:

- **IEEE Std 1028-1997:** provee modelos para los cinco tipos distintos de revisión: revisiones gerenciales, revisiones técnicas, inspecciones, *walkthroughs* y auditorías. En cada modelo, el estándar identifica seis tipos de requerimientos: relativos a responsabilidades, a la entrada, a criterios de entrada, a procedimientos, a criterios de salida y a la salida.
- **IEEE Std 829-1998:** describe, en un grupo de sus cláusulas, ocho distintos documentos asociados con el proceso de pruebas de software, a saber: plan de pruebas, especificación del diseño de pruebas, especificación de casos de prueba, especificación de procedimientos de prueba, reporte de transmisión de ítems de prueba, registro de pruebas, reporte de incidente de pruebas y el reporte del resumen de pruebas. Los documentos son descritos como requerimientos.
- **IEEE Std 730-2002:** provee los requerimientos formales mínimos para la realización de planes de aseguramiento de calidad de software para ser ejecutados en el momento en que el software es desarrollado o mantenido.
- **IEEE Std. 1008-1987:** El objetivo principal es especificar un enfoque estándar para las pruebas unitarias de software. Un segundo objetivo es describir los conceptos de ingeniería de software y los supuestos de prueba en los que se basa el enfoque estándar. Un tercer objetivo es proporcionar orientación e información de recursos para ayudar con la implementación y el uso del enfoque de prueba unitaria estándar, un proyecto para desarrollar un único estándar integral que cubra todos los aspectos de estas pruebas.

3.10.1 *ISO/IEC 29110*

La ISO / IEC 29110 es un conjunto de normas e informes técnicos que se ha desarrollado para entidades muy pequeñas (Por sus siglas en inglés Very Small Entities, VSE). Una VSE se define como una entidad (empresas, organizaciones, departamentos o proyectos) que tiene menos de 25 personas. El conjunto de documentos de la ISO/IEC 29110 ha sido desarrollado para mejorar la calidad de productos y/o calidad de los servicios y el desempeño de los procesos (*ISO/IEC 29110 Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Management and Engineering Guide*, 2011).

El estándar está estructurado como se indica en la tabla 3 -1 según al público objetivo al que va dirigido:

ISO/IEC 29110	Título	Audiencia Objetivo
Parte 1	Visión general	VSEs, evaluadores, productores de normas, proveedores de herramientas, proveedores de metodologías.
Parte 2	Marco de trabajo y taxonomía	Productores de normas, proveedores de herramientas y de metodologías. No destinado a las VSEs
Parte 3	Guía de evaluación	Evaluadores y VSEs
Parte 4	Especificaciones de perfil	Productores de estándares, proveedores de herramientas y de metodologías. No destinado a las VSEs
Parte 5	Guía de administración e Ingeniería	VSEs

Tabla 3-1 Estructura de ISO/IEC 29110

La ISO/IEC 29110-1 define los términos de negocio comunes al conjunto de documentos del perfil de la VSEs. Introduce los conceptos de proceso, ciclo de vida y normalización. Así mismo presenta las características y requisitos de una VSE, aclara los fundamentos para los perfiles o niveles de madurez, documentos, estándares y guías de una VSE específica.

La ISO/IEC 29110-2 introduce los conceptos para el perfil normalizado de ingeniería de Software para las VSEs y define los términos comunes para el conjunto de documentos de dicho perfil. Se especifican los elementos comunes para todos los perfiles normalizados (Estructura, conformidad, evaluación) e introduce la taxonomía (catálogo) de los perfiles.

ISO/IEC 29110-3 define los lineamientos y requisitos de conformidad del proceso de evaluación de los perfiles de las VSE definidos. También contiene información que puede ser útil para desarrolladores de métodos y herramientas de evaluación. Está dirigido a personas que no tienen relación directa con el proceso de evaluación.

Para todas las organizaciones, pero en particular para las microempresas, las certificaciones internacionales pueden aumentar la credibilidad, la competitividad y el acceso a los mercados nacionales e internacionales. Además, para este tipo de empresas, un proceso de certificación debe ser simple, corto y de bajo costo, y tener credibilidad internacional. Es por eso que los perfiles se definen con el propósito de empaquetar referencias a y/o partes de otros documentos de manera

formal, con el fin de adaptarlos a las necesidades y características de las VSEs. Preparar un perfil implica producir dos tipos de documentos:

- **Marco de trabajo y taxonomía:** Específica los elementos comunes a todos los perfiles (estructura, conformidad, evaluación) e introduce la taxonomía (catálogo) de los perfiles.
- **Especificaciones de perfil:** Proporciona la composición definitiva de un perfil, los enlaces normativos al subconjunto normativo de estándares usados en el perfil, y los enlaces informativos (referencias) a documentos de "entrada". Para cada perfil existe un documento de este tipo. Un ejemplo de una especificación de perfil es el documento (Especificación– Perfil Básico). Su objetivo es definir una guía de gestión de proyectos y desarrollo de software, adaptada a las necesidades de las VSE, para un subconjunto de procesos de ISO/IEC 12207 (ISO/IEC 12207:1995).

Con respecto a la verificación y validación, el estándar ISO/IEC 29110 define que estas tareas son necesarias para cada uno de los **Productos de Trabajo**. Se entiende por producto de trabajo a cada uno de los artefactos de valor y relevancia para un esfuerzo de ingeniería de software como por ejemplo un documento o una pieza de software. Las tareas de V&V deben realizarse utilizando los criterios definidos en los procesos de desarrollo para así asegurar la coherencia entre los productos de trabajo de salida y entrada, definidos en cada una de las actividades. Los defectos son identificados, corregidos y los resultados son almacenados en los Registros de Verificación/Validación. El estándar define las siguientes buenas prácticas para las actividades de V&V:

a) una estrategia de verificación es implementada y desarrollada;
b) se identifican los criterios de verificación para todos los productos de trabajo de software requeridos;
c) las actividades de verificación son llevadas a cabo;
d) los defectos son identificados y registrados; y
e) los resultados de las actividades de verificación están a disposición de los clientes y otras partes interesadas.

Para el proceso de Validación de Software:

a) una estrategia de validación es implementada y desarrollada;

b) se identifican los criterios de validación para todos los productos de trabajo requeridos;

c) las actividades de validación requeridas son llevadas a cabo;

d) los defectos son identificados y registrados; y

e) los resultados de las actividades de validación están a disposición de los clientes y otras partes interesadas.

3.10.2 *ISO/IEC 29119*

La ISO/IEC/IEEE 29119 Ingeniería de software y sistemas - Pruebas de software es una serie de cinco estándares internacionales para pruebas de software. Desarrollado por primera vez en 2007 y lanzado en 2013, el estándar define el vocabulario, procesos, documentación, técnicas y un modelo de evaluación de procesos para las pruebas que se puede utilizar dentro de cualquier ciclo de vida de desarrollo de software (*ISO/IEC/IEEE 29119:2013 Software Testing Strandard*, 2013). Se estructura de la siguiente manera:

- ISO/IEC/IEEE 29119-1:2013, Parte 1- Conceptos y vocabulario: facilita el uso de las otras partes de la norma al introducir el vocabulario sobre el cual se construye la norma y proporciona ejemplos de su aplicación en la práctica. Además, proporciona definiciones, una descripción de los conceptos de pruebas de software y formas de aplicar estas definiciones y conceptos a las otras partes de la norma.
- ISO/IEC/IEEE 29119-2:2013, Parte 2- Proceso de Testing: define un modelo de proceso de pruebas genérico para el software que está diseñado para ser utilizado por las organizaciones al realizar pruebas de sus aplicaciones de software. Comprende descripciones de los procesos de prueba a nivel organizacional, la gestión de prueba a nivel de proyecto y a niveles de proceso de prueba dinámica. Los procesos definidos en esta norma se pueden utilizar junto con diferentes modelos de ciclo de vida de desarrollo de software.
- ISO/IEC/IEEE 29119-3:2013, Parte 3 - Documentación de pruebas: incluye plantillas y ejemplos de documentación de pruebas que se producen durante el proceso de testing. Las plantillas son compatibles con los tres niveles principales del proceso de prueba de la Parte 2, y el estándar también incluye la asignación a otros estándares existentes.

- ISO/IEC/IEEE 29119-4:2015, Parte 4 - Técnicas de testing: proporciona definiciones estándar de las técnicas de diseño de pruebas de software (también conocidas como técnicas de diseño de casos de prueba o métodos de prueba) y las medidas de cobertura correspondientes que se pueden usar durante esa etapa. Así como también, los procesos de implementación definidos en la Parte 2. Las técnicas de la Parte 4 están diseñadas para admitir o ser utilizadas por separado de la Parte 2. Las técnicas de diseño de pruebas de la norma se clasifican en tres categorías principales: basadas en la especificación, en la estructura y en la experiencia.
- ISO/IEC/IEEE 29119-5:2016, Parte 5 - Testing dirigido por palabras clave: un enfoque para especificar pruebas de software (normalmente automatizadas) utilizadas en la industria del testing de software. Está destinado a los usuarios "que desean crear especificaciones de prueba dirigidas por palabras clave, crear los entornos de trabajo correspondientes o construir automatización de prueba basada en palabras clave".

Sin embargo, tras la introducción de ISO / IEC / IEEE 29119, que culminó en 2014, algunos evaluadores de software y organizaciones asociadas comenzaron a solicitar que la ISO rescindiera la norma. Dentro de las organizaciones notables que protestaron por el estándar incluyeron la Asociación para Pruebas de Software (*Association for Software Testing*) y la Sociedad Internacional para Pruebas de Software (*International Society for Software Testing*), quienes manifestaron algunas razones como: falta de verdadero consenso de contenido, tal como lo exige ISO / IEC, entre los evaluadores profesionales; y un gran enfoque en la documentación que resta valor al proceso real de las pruebas de software. Según estas entidades, la norma es inconsistente con la forma en que las personas trabajan en tareas cognitivamente exigentes, al tiempo que refuerza la forma en que se aferran a prácticas inútiles mientras hacen organizaciones complejas y estresantes. Además, el estándar no está claro acerca de los principios que deben regir las pruebas, mientras que es demasiado prescriptivo sobre los detalles, lo que lleva a una confusión de medios y fines, donde se excluyen efectivamente un tipo de prueba como las basadas en el contexto.

3.10.3 *El ISTQB*

El ISTQB (International Software Testing Qualifications Board) es una organización de certificación de la calidad del software que opera internacionalmente. Fue fundada en noviembre de 2002 en Edimburgo y está legalmente registrada en Bélgica. Esta organización se encarga de soportar y definir un esquema de certificación internacional. Esta organización, además suministra el plan de estudios y el glosario sobre los que se define como las guías para la acreditación y evaluación de los profesionales del testing a cargo de los comités de cada país.

La certificación ISTQB, junto con TMAP, es la única certificación a nivel personal de cierta importancia internacional a nivel de Calidad del Software. A nivel personal se refiere a que las organizaciones pueden obtener otras certificaciones, pero como tester o ingeniero de calidad, estas son las únicas ampliamente reconocidas (ISTQB, 2020).

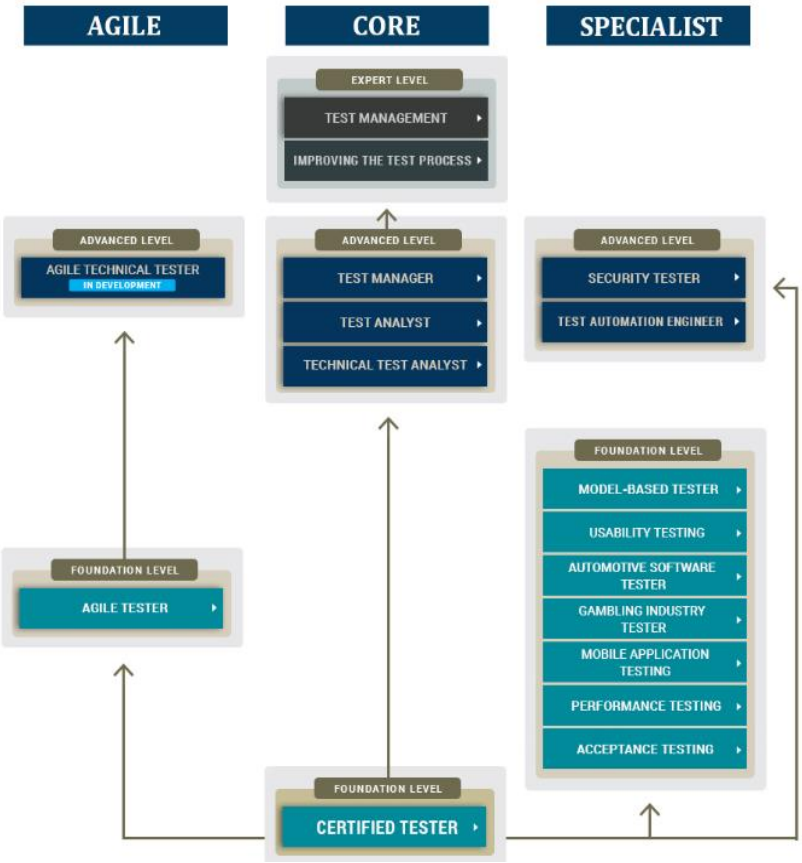

Figura 3-5 Niveles de Certificación ISTQB (ISTQB, 2020)

Actualmente hay 4 niveles de certificación según ISTQB:

- ISTQB Foundation Level (CTFL)
- ISTQB Advanced Level (CTAL)

- Gerente de Pruebas
- Analista de Pruebas
- Analista de Pruebas Técnicas
- Pruebas de seguridad
- Ingeniero de pruebas de Automation

- Expert Level
  - Mejora del Proceso de Pruebas
  - Gestión de Pruebas

CAPÍTULO 4: MODELOS DE MEJORA DE PROCESOS DE SOFTWARE

Tal y como se ha visto a lo largo de este trabajo, existe un estrecho vínculo entre la calidad del proceso de desarrollo de software y la calidad de los productos desarrollados por el mismo. En consecuencia, muchas compañías de ingeniería de software han tomado el camino de la mejora de los procesos para mejorar sus productos. La mejora de procesos significa entender los procesos existentes y cambiarlos para mejorar la calidad del producto, reduciendo costos y tiempo de desarrollo. Para ello, centran sus esfuerzos en la optimización de los procesos para mejorar la calidad del producto y, en particular, en reducir el número de defectos en el software entregado (Sommerville, 2011).

4.1 Introducción a los Modelos de Mejora

A lo largo de la historia, han sido varias las instituciones, como el caso del SEI (*Software Engineering Institute*), que han tenido como motivación la mejora de la calidad a través de la mejora del proceso. Entre los resultados de estas entidades, se encuentran recapitulaciones y catálogos de buenas prácticas que han dado lugar a los modelos de mejora de los procesos software.

Los modelos SPI (*Software Process Improvement*) dotan de un marco de referencia para desarrollar software acorde con la planificación establecida, mientras simultáneamente mejora la capacidad del desarrollador de crear mejores productos. Un modelo de procesos software puede ser usado por una organización para asegurar su madurez e identificar y dar prioridad a las áreas con mayor importancia. En este contexto, el término madurez está referido a la capacidad de una organización de llegar a un estado, definido, continuo y optimizado.

Existen numerosos modelos SPI vigentes en la actualidad. Uno de los más extendidos en la industria del software es CMMI, el cual se detalla más adelante. Estos modelos de mejora suelen guardar cierta estructura común respecto de la realización de actividades de mejora. Siguiendo la estructura desarrollada por el grupo Sogeti[3] en la cual se muestran las distintas actividades de un proceso de mejora, se establece (Dutta et al., 1999):

[3] **Sogeti** Es la división de servicios de Tecnología e Ingeniería de Capgemini. El grupo Sogeti es una de las consultoras de tecnología de la Información más grandes de Europa que ha desarrollado una serie de estándares reconocidos asociados a la calidad del proceso de SW.

- **Concienciación:** La primera actividad de un proceso de mejora consiste en tomar conciencia de la necesidad de mejorar el proceso software. Una de las principales razones para realizar dicha mejora se trata de la existencia de problemas en los procesos que deben de ser solucionados, aunque también destaca la mejora de la eficiencia de un determinado proyecto. La concienciación implica que las diferentes partes implicadas en el proceso acuerden mutuamente las líneas generales y se comprometan al proceso de cambio. Este compromiso no solo debe realizarse al comienzo del proceso, sino que debe permanecer a lo largo de todo el proyecto, realizando un esfuerzo continuo.
- **Determinar el alcance y método:** Una vez que se ha tomado conciencia de la necesidad de mejorar el proceso, es necesario tratar de determinar cuáles son los objetivos y el alcance de la mejora, así es posible establecer qué pruebas se van a realizar, los procesos que son susceptibles de mejora, el tiempo, el esfuerzo y los costos necesarios para hacer frente al proceso de mejora.
- **Efectuar la evaluación actual:** El desarrollo de esta actividad conlleva evaluar la situación actual del proyecto obteniendo, como resultado de la evaluación, los puntos fuertes o débiles de los procesos.
- **Definir las acciones de mejora:** Después de recoger la información obtenida en las actividades anteriores en forma de objetivos de mejora se procede a la elaboración de acciones que posibiliten la mejora paulatina de los mismos.
- **Diseñar el plan:** Una vez establecidas las acciones que se van a realizar en el paso anterior, se diseña un plan de implantación de las acciones de mejora a corto plazo. En este plan se determina en qué momento deben realizarse las acciones de mejora con el fin de alcanzar los objetivos marcados. El plan se dirige tanto a las actividades relacionadas con el contenido de las mejoras del proceso como a las actividades generales necesarias para realizar el proceso de cambio en la dirección adecuada.
- **Implantar acciones de mejora:** Una vez discutido el plan, se pone en marcha, es decir se lleva a cabo la implantación de las acciones de mejora. Debido a que, durante esta actividad, las consecuencias del proceso de

cambio tienen un mayor impacto, hay que prestar mucha atención a la comunicación. Sin duda, puede haber cierta resistencia al cambio, por lo que deberá afrontarse y discutirse abiertamente.

- **Efectuar Evaluación posterior:** Finalmente, después de implantar las acciones de mejora se realiza una evaluación que permite comprobar en qué medida las acciones fueron implantadas con éxito, así como evaluar en qué medida se alcanzaron los objetivos iniciales. A partir de estas observaciones se adoptará la decisión sobre la continuación del proceso de cambio.

A continuación, se detallarán algunos modelos de mejora que se consideran los más relevantes de la industria y que más han influido en el desarrollo de este trabajo.

4.1.1 *Capability Maturity Model (CMM)*

El Modelo de Madurez de Capacidades de Software (CMM, por las siglas de Capability Maturity Model) fue desarrollado por el SEI a principio de los años 90. Este desarrollo fue financiado por el Departamento de Defensa de los EE.UU. como medida ante la llamada “crisis del software”, en busca de obtener un marco para evaluar la madurez de los procesos de software utilizados por dichas organizaciones. Este modelo tiene como objetivo llevar a las organizaciones que desarrollan y mantienen software a un nivel de madurez que potencie de manera eficiente sus capacidades para obtener proyectos exitosos y de calidad (Paulk et al., 1993).

Establecer metas razonables para la mejora de procesos requiere un entendimiento de la diferencia entre organizaciones de software inmaduras y maduras. En una organización de software inmadura, los procesos de software generalmente son improvisados por las personas que intervienen en ellos, al igual que la administración durante el curso de los proyectos. Incluso si se ha especificado un proceso de software, no se sigue ni se aplica rigurosamente. La organización de software inmadura es reaccionaria, y los gerentes generalmente se enfocan en resolver crisis inmediatas. Los horarios y los presupuestos se exceden rutinariamente porque no se basan en estimaciones realistas. Cuando se imponen plazos estrictos, la funcionalidad y la calidad del producto a menudo se comprometen para cumplir con el cronograma. En este tipo de organizaciones, no

existe una base objetiva para juzgar la calidad del producto o para resolver problemas de productos o procesos. Por lo tanto, la calidad del producto es difícil de predecir. Las actividades destinadas a mejorar la calidad, como revisiones y pruebas, a menudo se reducen o eliminan cuando los proyectos se retrasan (Paulk et al., 1993).

Por otro lado, una organización de software madura posee una capacidad para administrar los procesos de desarrollo y mantenimiento de software en toda la organización. El proceso del software se comunica con precisión tanto al personal existente como a los nuevos empleados, y las actividades laborales se llevan a cabo de acuerdo con el proceso planificado. Los procesos obligatorios son aptos para el uso y son consistentes con la forma en que realmente se realiza el trabajo. Estos procesos definidos se actualizan cuando es necesario, y las mejoras se desarrollan a través de pruebas piloto controladas y/o análisis de costo-beneficio. Los roles y responsabilidades dentro del proceso definido son claros a lo largo del proyecto y en toda la organización. Los gerentes monitorean la calidad de los productos de software y la satisfacción del cliente ya que existe una base objetiva y cuantitativa para juzgar la calidad del producto y analizar problemas con el producto y el proceso. Los horarios y los presupuestos se basan en el rendimiento histórico y son realistas; Los resultados esperados de costo, programación, funcionalidad y calidad del producto generalmente se logran. En general, se sigue constantemente un proceso disciplinado porque todos los participantes entienden el valor de hacerlo y existe la infraestructura necesaria para respaldar el proceso (Paulk et al., 1993).

El marco CMM describe un camino evolutivo desde procesos caóticos *ad hoc* a procesos de software maduros y disciplinados. El marco de madurez del proceso de software surge de la integración de los conceptos de proceso de software, capacidad de dichos procesos, rendimiento y madurez.

Tal como se comentó anteriormente, un proceso de software es un conjunto de actividades relacionadas que conducen a la producción de un producto de software (Sommerville, 2011). En ese contexto, el modelo CMM establece los conceptos de Capacidad, Rendimiento y Madurez del proceso (Paulk et al., 1993).

Con base en esos conceptos, la mejora continua del proceso se basa en muchos pasos pequeños y evolutivos en lugar de innovaciones revolucionarias. El CMM

proporciona un marco para organizar esos pasos en cinco niveles de madurez que sientan las bases sucesivas para dicha mejora. Esos niveles definen una escala ordinal para medir la madurez del proceso de software de una organización y para evaluar la capacidad de este, así como también, ayudan a una organización a priorizar sus esfuerzos de mejora.

4.1.2 *Los niveles de Madurez de CMM*

Un nivel de madurez es una meseta evolutiva bien definida hacia el logro de un proceso de software maduro. Cada nivel de madurez proporciona una capa en la base para la mejora continua del proceso. Cada nivel comprende un conjunto de objetivos de proceso que, cuando se satisfacen, estabilizan un componente importante del proceso de software. El logro de cada nivel del marco de madurez establece un componente diferente en el proceso de software, lo que resulta en un aumento en la capacidad de proceso de la organización.

Organizar el CMM en los cinco niveles que se muestran en la Figura 4-1 - Los Cinco Niveles de Maduración del proceso de Software (Adaptado de Paulk et al., 1993) prioriza las acciones de mejora para aumentar la madurez del proceso del software. Las flechas etiquetadas indican el tipo de capacidad de proceso que está siendo institucionalizada por la organización en cada paso del marco de madurez.

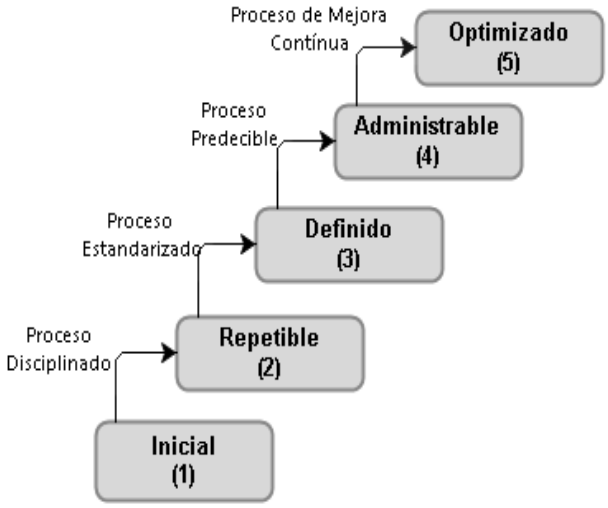

Figura 4-1 Los Cinco Niveles de Maduración del proceso de Software (Adaptado de Paulk et al., 1993)

A su vez, cada nivel de madurez se ha descompuesto en partes constituyentes. Con la excepción del Nivel 1, la descomposición de cada nivel de madurez varía desde resúmenes abstractos de cada nivel hasta su definición operativa en las

prácticas clave. Cada nivel de madurez se compone de varias áreas clave de proceso (Por sus siglas en inglés Key Process Areas, KPAs) que, a su vez, están organizadas en cinco secciones llamadas características comunes. Las características comunes especifican las prácticas clave que, cuando se abordan de forma colectiva, logran los objetivos del área de proceso clave (Paulk et al., 1993). Las características comunes son atributos de prácticas clave que indican si la implementación o institucionalización de un KPA es efectiva, repetible y duradera. Las mismas incluyen: el Compromiso de realización, la Capacidad de desempeño, las Actividades realizadas, la Medición y análisis y, por último, la Verificación de la implementación (Naik & Priyadarshi, 2008).

Nivel 1: Inicial

En este nivel, el software se construye sin seguir un modelo de proceso. No hay mucha planificación involucrada. Incluso si se prepara un plan, no se puede seguir. Los individuos toman decisiones basadas en sus propias capacidades y habilidades (Naik & Priyadarshi, 2008).

El éxito depende completamente de tener un administrador de proyecto (por sus siglas en inglés Project Manager, PM) excepcional y un equipo de software experimentado y efectivo. Ocasionalmente, los PM de software capaces y contundentes pueden soportar las presiones para tomar atajos en el proceso del software; pero cuando salen del proyecto, su influencia estabilizadora se va con ellos. Incluso un proceso de ingeniería sólido no puede superar la inestabilidad creada por la ausencia de prácticas de administración sólidas (Paulk et al., 1993). Como se dijo anteriormente, no hay un KPA asociado con el Nivel 1 por los que una organización alcanza este nivel sin hacer ningún esfuerzo.

Nivel 2: Repetible

En este nivel, existe el concepto de proceso para que el éxito se pueda repetir para proyectos similares. El rendimiento de las actividades comprobadas de proyectos anteriores se utiliza para preparar planes para proyectos futuros. Este nivel puede resumirse como disciplinado porque los procesos se utilizan para la repetibilidad. Todos los procesos están bajo el control efectivo de un sistema de gestión de proyectos (Naik & Priyadarshi, 2008).

Los KPAs del Nivel 2 incluyen (Naik & Priyadarshi, 2008): La Gestión de Requerimientos, para establecer un entendimiento común entre el cliente y los desarrolladores; la Planificación de Proyectos; el Seguimiento y Supervisión del Proyecto; la Gestión de Subcontratos de Software; el Aseguramiento de Calidad del Software, para evaluar procesos y productos para entender su eficiencia y calidad; y por último, la gestión de la configuración del software.

Nivel 3: Definido

En este nivel, la documentación juega un papel clave. Los procesos relacionados con la gestión de proyectos y las actividades de desarrollo de software se documentan, revisan, estandarizan e integran con los procesos de la organización. En otras palabras, hay una aceptación en toda la organización de los procesos estándar. El desarrollo del software se lleva a cabo siguiendo un proceso aprobado. Se realiza un seguimiento de las funcionalidades y las cualidades asociadas. El costo y el horario son monitoreados para mantenerlos bajo control (Naik & Priyadarshi, 2008).

Entre los KPAs del Nivel 3, podemos nombrar (Naik & Priyadarshi, 2008): el Enfoque en el Proceso de la Organización, los Programas de Capacitación; la Administración Integrada, que significa integrar las actividades de ingeniería y administración de software de la organización en un proceso común y definido en función de las necesidades comerciales y tecnológicas; la Ingeniería de Producto, la Coordinación entre grupos y la Revisión por Pares (Peer Review).

Nivel 4: Administrable

En este nivel, las métricas juegan un papel clave. Se recolectan y analizan métricas relativas a procesos y productos. Esas métricas se utilizan para obtener una visión cuantitativa de las cualidades del proceso y del producto. Cuando las métricas muestran que se están excediendo los límites, se activan las acciones correctivas. Por ejemplo, si demasiados casos de prueba fallan durante la prueba del sistema, es útil iniciar un proceso de análisis de causa raíz para comprender por qué fallan tantas pruebas (Naik & Priyadarshi, 2008).

Los KPA en el nivel 4 son los siguientes (Naik & Priyadarshi, 2008): la Gestión Cuantitativa del Proceso, donde los datos del proceso indican qué tan bien está

funcionando el mismo, y la Gestión de la Calidad del Software, donde los atributos de calidad de los productos se miden en forma cuantitativa para tener una mejor visión de los procesos y productos.

Nivel 5: Optimizado

En este nivel, las organizaciones se esfuerzan por mejorar sus procesos de manera continua. Esto se logra en dos pasos: (i) Observando los efectos de los procesos, midiendo algunas métricas clave, en la calidad, el costo y el tiempo de entrega de los productos de software y (ii) realizando cambios en los procesos mediante la introducción de nuevas técnicas y métodos, herramientas y estrategias (Naik & Priyadarshi, 2008).

Dentro de los KPAs en el nivel 5 se encuentran (Naik & Priyadarshi, 2008): la Prevención de Defectos, que tiene por objetivo analizar las causas de diferentes clases de defectos y tomar medidas preventivas para garantizar que no se repitan defectos similares; la Gestión del Cambio Tecnológico, que significa identificar técnicas, herramientas y metodologías útiles e introducirlas gradualmente en los procesos de software; y por último, la Gestión del Cambio de Proceso que trata de mejorar los procesos de una organización para tener un impacto positivo en la calidad, la productividad y el tiempo de desarrollo.

Para que una organización tenga un cierto nivel de madurez, debe cumplir todos los objetivos de todos los KPA en ese nivel y también todos los niveles anteriores. Por ejemplo, para que una organización esté en el nivel 2, se deben cumplir los seis KPA asociados con el nivel 2. Para que una organización esté en el nivel 3, la organización debe cumplir con los seis KPA asociados con el nivel 2 y los siete KPA asociados con el nivel 3.

Además, el SEI proporciona dos metodologías para evaluar las capacidades actuales de las organizaciones: evaluaciones internas y evaluaciones externas. Para evaluaciones internas, desarrolló la evaluación de mejora del proceso interno basada en el modelo de madurez de la capacidad (Por sus siglas en inglés Capability Maturity Model–Based Assessment Internal Process Improvement, CBA-IPI) para ayudar a las organizaciones a la autoevaluación. El CBA-IPI evalúa la capacidad de proceso de las organizaciones al identificar qué KPA se están

cumpliendo y qué se debe mejorar. Por otro lado, desarrolló el marco de evaluación de CMM (Por sus siglas en inglés CMM Appraisal Framework, CAF) para proporcionar un mecanismo para la evaluación formal de las organizaciones. El CAF describe los requisitos y las pautas que deben utilizar los asesores externos para diseñar métodos de evaluación que cumplan con los requisitos de CAF.

4.2 Capability Maturity Model Integration (CMMI)

A pesar de que la aplicación de CMM en el área de software fue exitosa, como se ocupaba de sólo uno de los componentes del producto de una organización, impulsó el desarrollo de CMM en otras áreas asociadas al desarrollo de un producto de software. Al tratarse de un modelo de referencia de prácticas maduras en una disciplina específica que se usa para informar y mejorar la capacidad de un grupo en el desempeño de dicha disciplina, comenzaron a generarse CMM para áreas como la Ingeniería en sistemas, la adquisición de software, entre otras.

Durante ese desarrollo, los CMM se comenzaron a alejar de su modelo original, generando variaciones en su disciplina, estructura y definición de madurez. Esta situación generó dificultades en su aplicación, ya que cuando se implementaba más de un tipo de CMM dentro de la misma organización, se detectaban algunos problemas tales como:

- Los diferentes modelos tienen diferentes estructuras, diferentes formas de medir la madurez y diferentes términos.
- Era difícil integrar los diferentes CMM para lograr el objetivo organizacional común como producir productos de bajo costo y alta calidad dentro de lo programado.
- Era difícil utilizar muchos modelos en la selección de proveedores y la subcontratación.

Estas dificultades, llevaron a la necesidad apremiante de tener una visión unificada de la mejora del proceso completo dentro de la organización. Es así que en 2010 evolucionó la idea del Modelo de Madurez de Integración de Capacidades CMMI (Por sus siglas en inglés Capability Maturity Model Integration, CMMI). El CMMI incluye información de los siguientes modelos:

- Modelo de madurez de capacidad para software (CMM-SW)

- Modelo de madurez de capacidad de desarrollo de producto integrado (IPD-CMM)
- Modelo de madurez de capacidad para ingeniería de sistemas (CMM-SE)
- Modelo de madurez de capacidad para abastecimiento de proveedores (CMM-SS)

4.2.1 *CMMI for Development*

CMMI for Development es un modelo de referencia que cubre el desarrollo y mantenimiento de las actividades aplicadas tanto a los productos como a los servicios. Este modelo en particular surge a partir de la versión 2.0 del CMM-SW, con objetivo de combinar los modelos anteriormente mencionados: CMM-SW, CMM-SE e IPD-CMM. El mismo, define un conjunto de buenas prácticas que cubren un conjunto de áreas de proceso involucradas con el desarrollo y mantenimiento, permitiendo establecer tanto el nivel de madurez como el nivel de capacidad existente en la organización (SEI, 2010).

El modelo CMMI v1.2 (CMMI-DEV) contiene 22 áreas de proceso, de las cuales 2 cubren los procesos de pruebas: Validación (VAL) y Verificación (VER). El propósito de Validación (VAL) es demostrar que un producto o componente de producto se ajusta a su uso previsto cuando se sitúa en su entorno previsto. Mientras que el propósito de la Verificación (VER) es asegurar que los productos de trabajo seleccionados cumplen sus requerimientos especificados.

La utilidad del CMMI se reconoce fácilmente considerando la complejidad del software en la actualidad, la diversidad de sus componentes y las necesidad de plataformas especializadas para ejecutarse (Naik & Priyadarshi, 2008).

El modelo ha conservado los niveles y áreas de proceso de CMM a ser implementadas por las organizaciones para mejorar el desarrollo de productos software, considerando todas las fases que aparecen en el proceso. Además, ha hecho algunos cambios en los títulos de los niveles de madurez y añadido nuevos procesos.

4.2.2 *Aplicación de CMM/CMMI for Developmment*

Como se ha visto, CMMI for Development, así como sus modelos antecesores, es un estándar general para la mejora de los procesos de desarrollo de software, que define un conjunto de niveles y áreas de proceso a ser implementadas por las

organizaciones para mejorar el desarrollo de productos software, considerando todas las fases de ese proceso.

Entre sus fortalezas se pueden destacar: la inclusión de las prácticas de institucionalización, que permiten asegurar que los procesos asociados con cada área sean efectivos, repetibles y duraderos; la guía paso a paso para la mejora, a través de niveles de madurez y capacidad (a diferencia de las ISO citadas en el capítulo anterior); la transición del 'aprendizaje individual' al 'aprendizaje de la organización' por mejora continua, lecciones aprendidas y uso de bibliotecas y bases de datos de proyectos mejorados. Además, se considera que es el más conocido internacionalmente, la documentación es gratuita y ha sido implantado en grandes corporaciones (De La Villa et al., 2004).

Sin embargo, este conjunto de estándares es genérico y no define en detalle procesos ni prácticas a ser implementadas en ninguna de sus áreas, por lo que requiere del uso de otros estándares y de la experiencia del personal para definir los procesos de forma adecuada dentro de la organización (SEI, 2010).

Adicionalmente, dada su generalidad, no cubre el proceso de pruebas en toda su amplitud, simplemente define tres áreas de proceso que, como se ha detallado anteriormente, sólo incluyen algunas de las actividades básicas relacionadas con las pruebas. El CMMI puede llegar a ser excesivamente detallado para algunas organizaciones, es un modelo extenso, puede ser considerado rígido y requiere mayor inversión para ser completamente implementado. Aunque su documentación es gratuita, es un modelo propietario por lo que los servicios de capacitación y consultoría para su implementación son costosos. Por otro lado, son necesarios auditores acreditados por el SEI en CMMI para llevar a cabo las evaluaciones que no tienen un periodo de vigencia. Este tipo de modelos, son de gran complejidad para las organizaciones pequeñas, no está orientado a PyMEs.

4.3 Testing Maturity Model (TMM)

El modelo TMM-Test Maturity Model (Burnstein, 2003) ha sido desarrollado por el Instituto Tecnológico de Illinois como una guía para la mejora del proceso de pruebas. Al igual que CMM, modelo al que complementa, utiliza el concepto de niveles de madurez para la evaluación y mejora del proceso de pruebas; aunque a diferencia de su antecesor no contempla los niveles de capacidad.

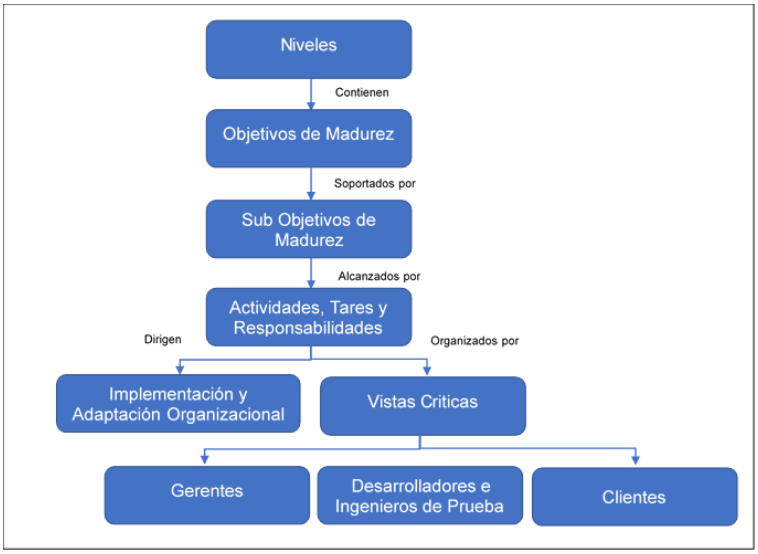

Figura 4-2 Estructura de TMM, Elaboración propia

TMM describe una trayectoria evolutiva de la madurez del proceso de prueba en cinco niveles o etapas, donde Cada etapa se caracteriza por los conceptos de objetivos de madurez, objetivos de soporte de madurez y actividades, tareas y responsabilidades (ATRs), como se explica a continuación (Naik & Priyadarshi, 2008):

- **Objetivos de madurez:** Cada nivel de madurez, excepto el nivel 1, contiene una serie de objetivos que una organización debe cumplir para alcanzar dicho nivel. Los mismos se especifican en términos de objetivos de mejora de las pruebas.
- **Subobjetivos de madurez:** Los objetivos de madurez están respaldados por subobjetivos de madurez ya que, para alcanzar cierto objetivo, puede ser necesario cumplir con varios objetivos secundarios específicos.
- **Actividades, Tareas y Responsabilidades (ATRs):** Los subobjetivos de madurez se logran mediante ATRs que abordan los problemas relacionados con la implementación de actividades y tareas. Los ATRs también abordan cómo una organización puede adaptar sus prácticas para que pueda evolucionar en línea con el modelo TMM, es decir, cómo moverse de un nivel al siguiente. Los ATR se refinan aún más en "vistas", conocidas como vistas críticas, desde las perspectivas de tres grupos diferentes de personas: gerentes, desarrolladores e ingenieros de prueba, y clientes (usuarios / clientes).

Existe una relación no necesariamente recíproca entre los niveles alcanzados en una organización por ambos modelos. Esto es, que un alto nivel TMM implica un alto nivel CMM (o al menos las condiciones de CMM), pues resulta necesario para cumplir con los aspectos de mejoras del testing tener un alto nivel de maduración de las capacidades en la organización. Por otro lado, si se tiene un alto nivel CMM no se puede garantizar un alto nivel TMM, dado que CMM no especifica aspectos orientados a la testabilidad.

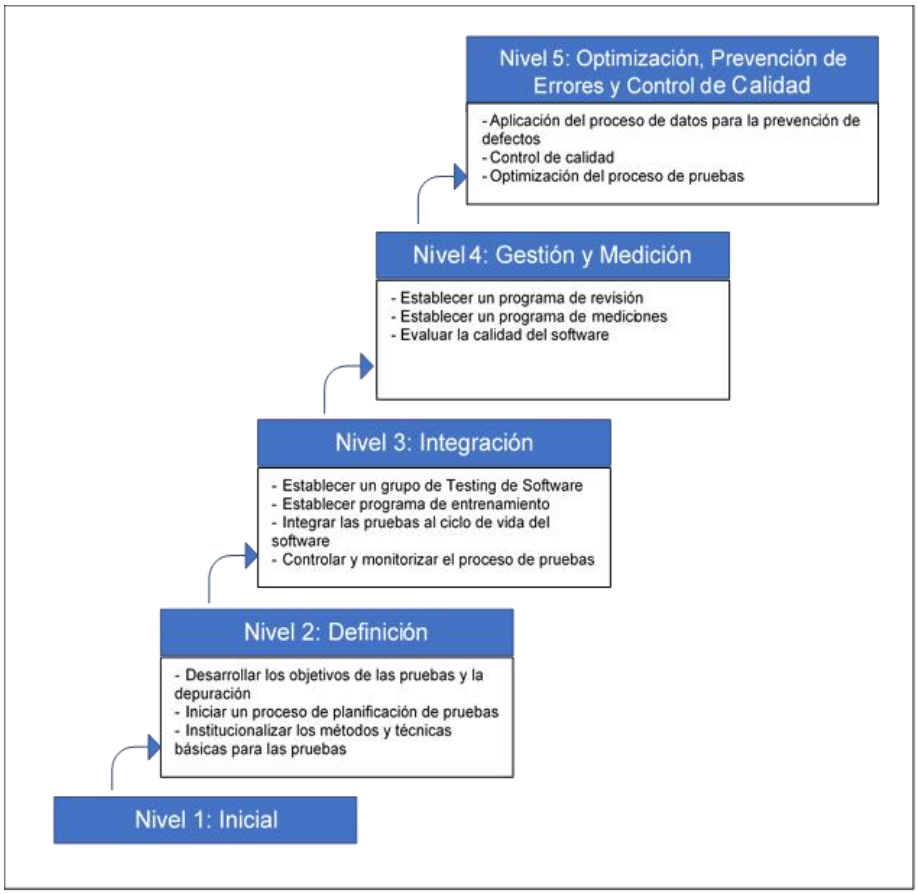

Figura 4-3 Niveles de TMM

4.3.1 *Los Niveles de TMM*

Los niveles de TMM, tal como lo indica la Figura 4-3 - Niveles de TMM, son los siguientes:

Nivel 1 - Inicial

Al igual que en CMM, para que una organización esté en este nivel, no se necesita hacer nada especial. El nivel 1 representa un escenario en el que las pruebas no

se realizan de manera planificada; comienzan después de que se escribe el código y a menudo se realizan solo para demostrar que el sistema funciona. No se hace ningún esfuerzo serio para rastrear el progreso de las pruebas y el nivel de calidad de un producto. Los casos de prueba se diseñan y ejecutan de manera ad hoc. Los recursos de prueba, como probadores capacitados, herramientas de prueba y entornos de prueba, no están disponibles. En resumen, las pruebas no se ven como una fase crítica y distinta en el desarrollo de software.

Nivel 2 - Definición

El objetivo principal de este nivel es verificar que el software satisface los requisitos especificados. Sin embargo, en esta etapa de madurez surgen problemas de calidad debido a que la planificación de las pruebas se realiza tarde. Además, los defectos se propagan desde la especificación de requerimientos y desde las fases de diseño al código ya que no existen revisiones previas a estas etapas. Para el Nivel 2, los objetivos de incluyen: Desarrollar los objetivos de las pruebas y de la depuración de unidades (pruebas unitarias) de forma separada; Iniciar un proceso de planificación de pruebas e Institucionalizar los métodos y técnicas básicas para las pruebas, algunos de los cuales fueron detallados en capítulos anteriores (Naik & Priyadarshi, 2008).

Nivel 3 - Integración

En el nivel 3, las pruebas están totalmente integradas con el proceso de desarrollo desde la planificación del proyecto. A diferencia del nivel 2, la planificación comenzará con la fase de requerimientos y continuará durante todo el ciclo de vida. Existe una organización de pruebas, un grupo de testing independiente, un programa de formación y se reconoce a las pruebas como una actividad profesional. Se realizarán revisiones, aunque no exista un programa formal de revisión y éstas resulten inconsistentes. Los objetivos de madurez en el nivel 3 son los siguientes: Establecer un grupo de Testing de Software independiente; establecer un programa de entrenamiento / capacitación técnica, centrado en el concepto de calidad, planificación de pruebas, métodos y técnicas de testing, estándares y herramientas; Integrar las pruebas por completo al ciclo de vida del

software, es decir, desde el inicio de la planificación del proyecto; controlar y monitorizar el proceso de pruebas (Naik & Priyadarshi, 2008).

Nivel 4 - Gestión y Medición

El nivel 4 se caracteriza porque las pruebas son un proceso medido y cuantificado. Las revisiones de todas las fases del proceso de desarrollo son reconocidas como actividades de pruebas y control de calidad en este nivel. Se prueba el software atendiendo a los atributos de calidad, como fiabilidad, usabilidad y mantenimiento. Las deficiencias en este nivel se producen debido a la carencia en la prevención de defectos. Las actividades de prueba se expanden, incluyen revisiones, inspecciones y walkthroughs a través de las diferentes fases. Los objetivos de maduración de este nivel son: Establecer un programa de revisión para detectar defectos al inicio del ciclo de vida del desarrollo y a un costo menor; Establecer un programa de mediciones para medir la productividad, el progreso y la calidad del trabajo asociado con las pruebas; Evaluar la calidad del software para saber qué nivel de calidad puede resultar de un determinado proceso y tomar medidas para mejorarla (Naik & Priyadarshi, 2008).

Nivel 5 - Optimización, prevención de errores y control de calidad

Finalmente, el nivel 5 se caracteriza porque el proceso de pruebas es repetible, definido, gestionado y medido, por lo que se pueden definir mecanismos que permitan una mejora continua en el proceso de pruebas. Se realiza prevención de defectos y control de calidad. Los objetivos de nivel 5 son: la aplicación del proceso de datos para la prevención de defectos; el Control de calidad que incluye muestreo estadístico, medición de los niveles de confianza, confiabilidad y objetivos de confiabilidad del software; la Optimización del proceso de pruebas (Naik & Priyadarshi, 2008).

4.3.2 *Test Maturity Model Integration (TMMi)*

El modelo TMMi - Test Maturity Model Integration, evolución del modelo TMM (Van Veenendaal, 2018) ha sido desarrollado por la TMMi Foundation como una guía y un marco de referencia para la mejora del proceso de pruebas, siendo un modelo complementario a CMMI. Al igual que este último, TMMi define una representación

por etapas y hace uso del concepto de niveles de madurez para evaluar y mejorar las diferentes áreas de proceso de cada uno de los niveles. Además de las áreas de proceso, identifica el conjunto de objetivos a alcanzar y las prácticas que hay que realizar para ello. El objetivo de TMMi es soportar las actividades de prueba y la mejora del proceso de prueba tanto en la disciplina de ingeniería de sistemas como en la de ingeniería de software.

4.3.3 *Los Niveles de TMMi*

Al igual que TMM, TMMi consta de cinco niveles que van desde la realización de un proceso de pruebas de manera ad-hoc y no gestionada hasta un nivel optimizado, pasando por los niveles gestionado, definido, medido y optimizado. Pasar de un nivel a otro supone haber alcanzado una mejora adecuada y sirve también como punto de partida para el siguiente nivel. La Figura 4-4 - Niveles TMMI muestra la estructura de TMMi que, si bien es muy similar a TMM, logra mejorar algunas limitaciones de este (Van Veenendaal, 2018).

El nivel 1, análogamente a los esquemas anteriores, se caracteriza porque las pruebas se realizan de manera ad-hoc una vez que la codificación se ha terminado, puesto que el objetivo de las pruebas a este nivel es comprobar si el software se ejecuta sin errores importantes. En el nivel 1 de TMMi no hay áreas de proceso definidas. Las organizaciones de nivel de madurez 1 se caracterizan por una tendencia a un compromiso excesivo, el abandono de procesos en tiempos de crisis y la incapacidad de repetir sus éxitos (Van Veenendaal, 2018).

En el nivel 2 de TMMi, el proceso de disciplina reflejado ayuda a garantizar que las prácticas comprobadas se mantengan en momentos de estrés. Sin embargo, muchas partes interesadas aún perciben las pruebas como una fase de proyecto que sigue a la codificación. En el contexto de la mejora del proceso de prueba, se establece una estrategia de prueba para toda la empresa o para todo el programa donde también se desarrollan planes de prueba. Además, en este nivel las pruebas son multinivel y existen objetivos de prueba específicos para cada nivel, definidos en la estrategia. Al Igual que en TMM, el objetivo principal de las pruebas en una organización TMMi nivel 2 es verificar que el producto cumple con los requerimientos especificados. Sin embargo, en esta metodología también se generan muchos problemas de calidad en este nivel, ya que las pruebas se

producen relativamente tarde en el ciclo de vida del desarrollo, por ejemplo, durante el diseño o incluso durante la fase de codificación (Van Veenendaal, 2018).

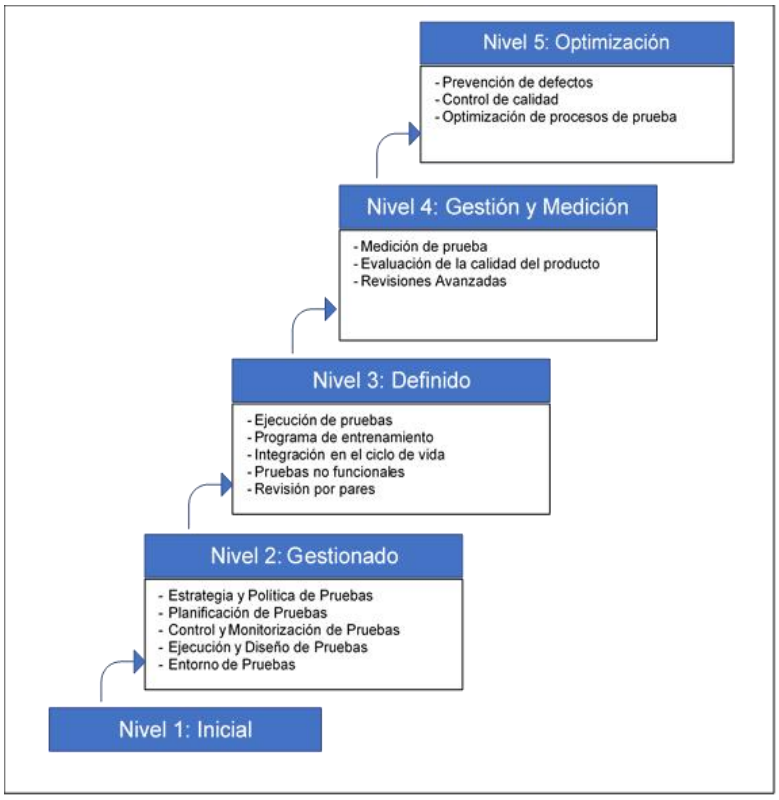

Figura 4-4 Niveles TMMI

En el nivel 3 de TMMi, igual que en TMM, las pruebas ya no se limitan a una fase que sigue a la codificación, están completamente integrados en el ciclo de vida del desarrollo y los hitos asociados. La planificación de las pruebas se realiza en una etapa temprana del proyecto donde se documenta en un plan de prueba maestro. Este proceso se encuentra totalmente institucionalizado como parte de las prácticas aceptadas por la organización. Las organizaciones en el nivel 3 entienden la importancia de las revisiones en el control de calidad; un programa de revisión formal es implementado, aunque todavía no está totalmente vinculado al proceso de prueba dinámica (Van Veenendaal, 2018).

Lograr los objetivos de TMMi nivel 2 y 3 tiene los beneficios de establecer una estructura técnica, de gestión y de personal capaz de realizar pruebas exhaustivas y brindar apoyo para la mejora del proceso de prueba. En este contexto, las pruebas pueden convertirse en un proceso medido para alentar un mayor crecimiento y logro.

En las organizaciones de nivel 4 de TMMi, las pruebas son procesos bien definidos, bien fundamentados y mensurables. Implementa un programa de medición de pruebas en toda la organización que se puede utilizar para evaluar la calidad del proceso de prueba, evaluar la productividad y monitorear las mejoras. Al igual que en TMM, la presencia de un programa de medición permite a una organización implementar un proceso de evaluación de la calidad del producto mediante la definición de las necesidades, atributos y métricas de calidad. Sin embargo, como una mejora a este, las medidas evaluadas se recopilan para respaldar la toma de decisiones basada en hechos, permitiendo, además, realizar predicciones relacionadas con el rendimiento y el costo de la prueba (Van Veenendaal, 2018).
El logro de todos los objetivos de mejora de pruebas en los niveles 1 a 4 de TMMi, crea un marco organizativo para pruebas que admite un proceso completamente definido y medido. En el nivel 5, una organización es capaz de mejorar continuamente sus procesos basándose en una comprensión cuantitativa de los procesos controlados estadísticamente. Análogamente al nivel 5 de TMM, la mejora del rendimiento del proceso de prueba se lleva a cabo a través de procesos incrementales e innovadores y mejoras tecnológicas. Los métodos y técnicas de prueba se optimizan constantemente ya que existe un enfoque a la mejora continua del proceso. Un proceso de prueba optimizado se define en TMMi como:

- Gestionado, definido, medido, eficiente y efectivo.
- Estadísticamente controlado y predecible.
- Enfocado en la prevención de defectos.
- Apoyado por la automatización considerando el uso efectivo de los recursos.
- Capaz de soportar la transferencia de tecnología de la industria a la organización.
- Capaz de soportar la reutilización de activos de prueba.
- Enfocado en el cambio de procesos para lograr la mejora continua.

Para respaldar la mejora continua del proceso de prueba, y para identificar, planificar e implementar mejoras de prueba, a diferencia de TMM, se establece formalmente un grupo permanente de mejora del proceso compuesto por miembros que han recibido capacitación especializada. Su objetivo es identificar y analizar las

causas comunes de defectos en todo el ciclo de vida del desarrollo y definir acciones para evitar que se produzcan defectos similares en el futuro.

Existe un procedimiento establecido para identificar mejoras en los procesos, así como para seleccionar y evaluar nuevas tecnologías de prueba. Las herramientas apoyan el proceso de prueba tanto como es efectivo durante el diseño, la ejecución de las pruebas, las pruebas de regresión, la administración de casos de prueba, la recolección y el análisis de defectos, etc. La reutilización de procesos y software de prueba en toda la organización también es una práctica común y está respaldada. En el nivel 5 de TMMi, la prueba es un proceso con el objetivo de prevenir defectos (Van Veenendaal, 2018).

4.3.4 *Aplicación de TMM/TMMI*

Evaluando las ventajas de los modelos para mejorar el proceso de prueba TMM y TMMi, podemos decir que ambos ofrecen un nivel de pruebas más riguroso que CMM y CMMI, pero siguiendo su misma estructura. Estos modelos, están teniendo un rápido crecimiento por Europa, Asia y USA, sobre todo tMMi que es la versión más actualizada. Estos modelos, proporcionan certificación y procesos de formación/examen, procedimientos y estándares para acreditación formal y pública de Evaluadores y Evaluadores líder. Tanto TMM como TMMi podría ser utilizado en organizaciones que utilizan una estructura organizativa de pruebas basada en equipos independientes o grupos de SQA [4] (De La Villa et al., 2004).

Sin embargo, para el caso donde los desarrolladores son los testers o existen equipos integrados de pruebas, estos modelos de referencia resultan inapropiados, debido a la dimensión reducida de estos equipos la carga de trabajo sería excesiva. Estos modelos, al igual que sus modelos análogos, CMM y CMMI, tienen una gran desventaja cuando son empleados por las organizaciones, puesto que no contienen una definición clara de los procesos, actividades y tareas a ejecutar, elementos de trabajo, productos de trabajo, etc. Por tanto, no existe una descripción formal del proceso; si no que simplemente proporciona información a ser considerada en la definición de los mismos a alto nivel. Este hecho frena su utilización en las

[4] SQA, Software Quality Assurance (aseguramiento de la calidad de SW): es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza adecuada en que el producto logrará satisfacer los requisitos dados de calidad. El aseguramiento de la calidad se enfoca en identificar y evaluar los defectos que pueden afectar al SW.

organizaciones, ya que no proporciona la guía necesaria para la propia definición de los procesos de prueba. Además, para el caso de TMMi, su aplicación está tan fuertemente ligada a CMMI, que requiere que las organizaciones sigan CMMI en su representación por etapas como modelo de referencia para la mejora del resto de procesos no relacionados con las pruebas.

Aunque el modelo es bastante fácil de entender, su aplicación en una organización no siempre es una tarea simple. Está diseñado para organizaciones grandes, no es fácilmente adaptable a estructuras mucho más pequeñas, como las PyMEs (De La Villa et al., 2004).

4.4 Test Process Improvement (TPI)

Tal como se detalló anteriormente, es importante mejorar los procesos de prueba y es ventajoso seguir un modelo definido para su mejora. La idea de mejorar los procesos del modelo TPI fue estudiada por primera vez por Tim Koomen y Martin Pol del grupo Sogeti. Según los autores del libro *Test Process Improvement*, un proceso de prueba debe mejorarse por tres razones (Naik & Priyadarshi, 2008):

- **Calidad:** un mejor proceso de prueba debería dar una mejor perspectiva de las características de calidad del sistema que se está probando.
- **Tiempo de entrega:** un mejor proceso de prueba ahorra tiempo y, por lo tanto, otorga más tiempo a otras actividades del ciclo de vida del software.
- **Costo:** se espera que se lleve a cabo un mejor proceso de pruebas a un costo menor y, por lo tanto, se reduzca el costo general del desarrollo del sistema.

De forma similar a otros modelos detallados en este capítulo, TPI consta de una serie de pasos para implementar mejoras en el proceso de pruebas:

1. Determinar el área de mejora
2. Evaluar el estado actual del proceso de prueba en el área elegida
3. Se identifica el siguiente estado y medio deseado
4. Implementar los cambios necesarios en el proceso de manera planificada.

Aunque estos parecen ser sencillos, su implementación no lo es. Los pasos, especialmente el segundo y el tercero, son demasiado generales y, por lo tanto, es difícil lograr un acuerdo entre los involucrados. En este contexto, el modelo TPI de Koomen y Pol fomenta y apoya la mejora gradual en un proceso de prueba paso a

paso. Demasiados cambios en un proceso pueden no tener éxito por dos razones. Primero, es probable que demasiados cambios requieran demasiados recursos, que pueden no estar disponibles. En segundo lugar, las personas generalmente se resisten a que se realicen demasiados cambios al mismo tiempo (Naik & Priyadarshi, 2008).

Un concepto importante revelado en el modelo intuitivo de mejora del proceso de prueba es la idea de evaluar el estado actual de un proceso de prueba. El estado actual se evalúa desde diferentes puntos de vista, conocidos como “Áreas Clave”, y en este modelo, se han identificado 20. Las áreas clave identificadas en TPI son: Estrategia de prueba, Modelo de ciclo de vida, Momento de participación, Planificación y estimación, Técnicas de especificación de pruebas, Técnicas de pruebas estáticas, Métricas, Herramientas de prueba, Entorno de prueba, Ambiente de oficina, Compromiso y Motivación, Funciones de prueba y capacitación, Ámbito de la metodología, Comunicación, Informes, Gestión de defectos, Gestión de Testware (objetos de prueba), Gestión de procesos de prueba, Evaluación y, por último, Pruebas de bajo nivel.

El estado de un proceso de prueba evaluado con respecto a cada área clave se representa en términos de uno de los cuatro niveles de madurez: A, B, C y D, donde A es el más bajo y D es el más alto. Específicamente, el interés está en saber hasta qué punto ha madurado un área clave determinada. En este contexto, es posible que no todas las áreas clave maduren en la misma medida ya que una organización puede no poner el mismo énfasis en todas. Cada nivel es mejor que su predecesor en una o más formas del grupo de tres parámetros deseados: calidad, tiempo y dinero (costo). Para que un área clave alcance un cierto nivel, debe cumplir con ciertos requisitos, llamados puntos de verificación, asociados con ese nivel y todos los niveles debajo de esa área clave, al igual que el resto de los modelos (Naik & Priyadarshi, 2008).

Una de las características que diferencia este modelo del resto, es que no todas las áreas clave tienen todos los niveles de madurez. Para algunas áreas claves, existen solo los niveles A y B donde B es el nivel más alto de madurez que se puede alcanzar. También, existen dependencias entre niveles de madurez de diferentes áreas clave, es decir, que para que un área clave alcance un cierto nivel de madurez, algunas otras áreas clave deben haber alcanzado ciertos niveles de

madurez. Esta idea de dependencia de madurez exige una mejora en los niveles de madurez de áreas clave de manera priorizada (Naik & Priyadarshi, 2008).

En el año 2013, Sogeti lanzó una evolución del modelo llamada TPI Next, en el cual se mantienen fortalezas del modelo original (áreas clave, niveles de madurez, puntos de control, sugerencias de mejora y mejora gradual) y se incorporan los Clusters y los Facilitadores.

4.4.1 *Aplicación de TPI*

TPI es un medio que puede proporcionar información valiosa sobre el estado actual del proceso de pruebas y sobre cómo se puede mejorar el mismo. Se basa en el concepto de dar pasos pequeños y controlados a partir de prioridades. Debe considerarse como una herramienta para estructurar las acciones de mejora del proceso de pruebas y como un medio para la comunicación. TPI podría ser utilizado en organizaciones que utilizan una estructura organizativa de pruebas basada en equipos independientes o grupos de SQA que utilicen la metodología TMap o Tmap Next, que se detalla más adelante. Cuenta con más de 1.100 profesionales de Software Control & Testing (SCT) repartidos entre Estados Unidos, Europa y la India. Además, podemos destacar que es específico de pruebas (De La Villa et al., 2004).

Sin embargo, una de las limitaciones que surge a la hora de utilizar TPI es que está estrechamente ligado a la metodología TMap, por lo que sería necesario que la organización implemente dicha metodología como mecanismo para la gestión del proceso de pruebas. Es menos difundido que otros modelos y no existe mucha documentación al respecto.

Como se verá en el apartado siguiente, el uso de esta metodología tiene también una serie de desventajas que hacen que sea difícil de implantar en las organizaciones, especialmente cuando esta son pequeñas y medianas empresas, en los casos en que los desarrolladores son los testers o existan equipos integrados de pruebas no muy numerosos, ya que la carga de trabajo para su implementación es excesiva. Está diseñado para organizaciones grandes, y no es fácilmente adaptable a estructuras más pequeñas, como las PyMEs. Además, tal y como se ha mencionado, TPI debe ser considerado como una herramienta para estructurar las acciones de mejora del proceso de pruebas y como un medio para la

comunicación, pero no es un mecanismo que contribuya a la definición de los procesos de prueba y, por tanto, no es directamente implementable en organizaciones que tratan de definir sus procesos de prueba. TPI, será únicamente utilizable de cara a mejorar los mismos en etapas posteriores. Este hecho se resalta aún más cuando se analiza su estructura, puesto que no define actividades y tareas a ser ejecutadas para la realización del proceso de pruebas, sólo define áreas clave y niveles que debería de tener la organización, pero a nivel de evaluación del proceso existente. Por tanto, ni TMM ni TMMi, se corresponden con modelos de procesos para el área de pruebas de software y, únicamente, pueden ser utilizados como una fuente para la definición de los procesos o como un mecanismo para la evaluación de los mismos una vez que ya están siendo utilizados

Para resumir, cuando se requiere un enfoque o método de prueba estructurado, o donde el proceso de prueba necesita mejoras, TPI puede usarse para evaluar esta situación y priorizar las posibles medidas. Cómo se pueden implementar o realizar estas medidas en la práctica se detalla en TMap Next.

4.5 Test Management Approach (TMap) y TMap NEXT

TMAP es una metodología orientada al desarrollo de pruebas software que se genera como complemento de TPI, proporcionando así una herramienta muy centrada en el proceso de pruebas.

En el año 2016, Sogeti evolucionó esta metodología en TMap Next. La diferencia entre los dos es que TMAP es un enfoque sobre cómo realizar un proceso de prueba: cómo configurar una estrategia de prueba, cómo escribir planes de prueba, cómo diseñar casos de prueba, etc. TMap Next, en cambio, proporciona un conjunto completo de herramientas, técnicas, infraestructura, directrices, plantillas, listas de verificación y descripciones detalladas de todas las partes del proceso de prueba. Se trata de un enfoque que puede ser aplicado en todas las situaciones de las pruebas y en combinación con cualquier método de desarrollo (van der Aalst et al., 2010).

4.5.1 *Estructura de TMap/ TMap NEXT*

Tanto Tmap como Tmap Next, se encuentran estructurados en cuatro aspectos fundamentales indicados en la Figura 4-5 - Estructura de TMap/TmapNEXT (Van Driel, 2010):

Figura 4-5 Estructura de TMap/TmapNEXT (Van Driel, 2010)

1. Gestión de pruebas impulsadas por la empresa (por sus siglas en inglés Business Driven Test Management, BDTM).
2. Proceso de pruebas estructurado.
3. Kit de herramientas.
4. Adaptabilidad.

A continuación, se brinda una breve descripción de cada uno de los elementos:

Gestión de pruebas dirigida al negocio (BDTM):

Idealmente, si una organización tuviese infinitos recursos, sería lógico probar todo lo que fuese posible. Como esto no suele suceder, se debe elegir qué y cómo probar. Esta elección depende en gran manera de los riesgos que quiera asumir la organización, a modo de tiempo y dinero, y el resultado que se quiera conseguir. La elección basada en riesgos, resultados, tiempo y costos es la base de la gestión de pruebas impulsadas por la empresa (BDTM) (Koomen et al., 2006).

El BDTM, además, utiliza el concepto del caso de negocio, el cual asume cada vez mayor importancia la perspectiva económica en los proyectos TI. El caso de negocios proporciona la justificación del proyecto y responde las preguntas: ¿por qué hacemos este proyecto? ¿Qué inversiones son necesarias? ¿Qué desea lograr el cliente con el resultado? Tmap Next está dotado de una serie de características para satisfacer este concepto (van der Aalst et al., 2010):

- La aproximación se centra en conseguir un determinado resultado.
- El proyecto para lograr este resultado final se realiza en los plazos marcados.
- El costo del proyecto debe estar equilibrado con los beneficios que la organización espera conseguir.

- Los riesgos han de ser conocidos y lo más pequeños que sea posible.

BDTM, además, parte del principio de que el cliente debe estar implicado en el proceso de pruebas, dotándolo de un carácter económico (Koomen et al., 2006). Para ello, necesita cierta información sobre el esfuerzo total de las pruebas relacionado con los riesgos que tienen de ser probados, el esfuerzo estimado y su planificación para el proceso asociado a la estrategia de pruebas definida (Pol et al., 2002). Hay que comprender que en el enfoque BDTM, es importante no perder de vista el objetivo final que es **proporcionar una evaluación de calidad y una recomendación de riesgo sobre el sistema**. Es por eso, que las actividades de BDTM se estructuran tal como se muestran en la Figura 4-6 - Proceso BDTM (van der Aalst et al., 2010):

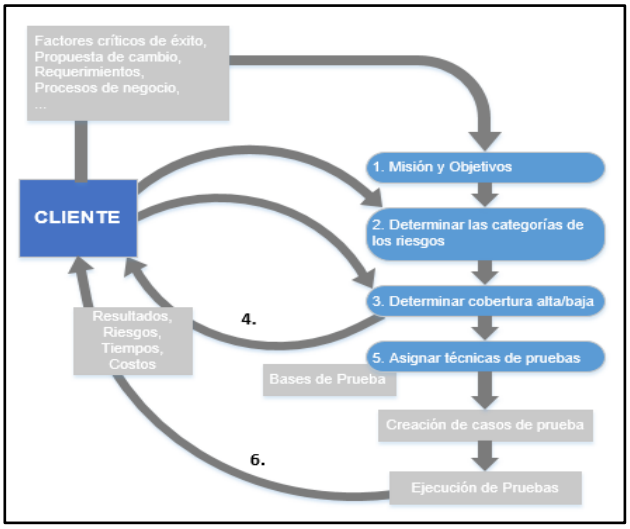

Figura 4-6 Proceso BDTM (van der Aalst et al., 2010)

1. **Formular la misión y recoger objetivos de las pruebas**: Esta actividad se lleva a cabo en contacto con el cliente, y en la misma, el gerente de pruebas designa la misión teniendo en cuenta los cuatro aspectos de BDTM: resultados, riesgos, tiempo y costos.
2. **Determinar las categorías de los riesgos:** Después de formular la misión y los objetivos, se estudia qué conjunto de niveles de prueba deben crearse. A menudo, se determina en base a un análisis de riesgo del producto qué debe probarse (partes del objeto) y qué debe ser investigado (características). El resultado final (ya sea que se llegue inmediatamente o después de uno o más análisis complementarios) es una tabla de riesgo que define una clase de riesgo relacionada con los objetivos de la prueba y la

característica relevante de la parte del objeto asociada ("Tabla de riesgo del plan de prueba maestro").

En este punto, surge un proceso iterativo:

3. **Determinar cobertura alta/baja:** Para determinar la exhaustividad de las pruebas, se utiliza como punto de partida la clase de riesgo por parte de objeto determinada en el paso anterior. Inicialmente, se aplica lo siguiente: cuanto mayor es el riesgo, más exhaustivas son las pruebas requeridas. El resultado se registra en una tabla de estrategia por nivel de prueba ("Tabla de estrategia del plan de prueba").
4. **Estimar globalmente las pruebas y la planificación**: Se realiza una estimación general de las pruebas y se establece una planificación. Esto se comunica al cliente y otras partes interesadas y, según sus puntos de vista, se ajusta según sea necesario. En este caso, los pasos 3 y 4 se ejecutan una vez más. En este punto, se le da al cliente el control del proceso de prueba, lo que le permite administrar, en función del equilibrio entre resultado y riesgo, por un lado, el tiempo y por otro, el costo.

Finalizadas las iteraciones necesarias,

5. **Asignar técnicas de pruebas**: Cuando el cliente y las partes interesadas acuerdan la estimación y la planificación del paso 4, el gerente de prueba asigna quién se ocupa de las pruebas, con qué técnicas se tratan y completa una "Tabla de diseño de prueba". Aquí es donde comienza el proceso de prueba principal.
6. **Proporcionar al cliente información sobre el proceso de pruebas:** Durante todo el proceso, el gerente de pruebas debe informar a las distintas partes interesadas el progreso, los riesgos, los costos y la calidad del proceso.

Como ventaja del enfoque BDTM se puede destacar que el cliente tiene control sobre el proceso en todas las fases. Además, el administrador de pruebas se comunica e informa en la terminología del cliente con información que es útil en el contexto del cliente para facilitar la dinámica. Por otro lado, cabe destacar, que a nivel del plan de prueba maestro, los detalles pueden ser tan intensivos como sea necesario o posible. Esto puede permitir gastar menos esfuerzo en realizar un análisis de riesgo del producto o crear una estrategia de prueba para los niveles de

prueba separados, o incluso omitir estos pasos (explicación del plan maestro de prueba en la sección siguiente).

Proceso de pruebas estructurado

El proceso de pruebas se realiza en todos los niveles de prueba, de manera coordinada y sincronizada, de modo que se transmiten tareas y responsabilidades entre los distintos grupos involucrados en el proceso. Para ello divide el proceso de pruebas en cuatro fases (van der Aalst et al., 2010):

- **Fase 1 - El plan de pruebas maestro:** El plan de pruebas maestro (por sus siglas en inglés Master Test Plan, MTP) es la planificación base de las actividades destinadas a la detección de los defectos más importantes, con el menor costo y tiempo posibles. El MTP se compone de dos fases: la Planificación del proceso de pruebas y el Control general del mismo, con el fin de proporcionar un avance continuo de la calidad del proceso y del objeto de prueba. Además, se adquiere mayor importancia en la realización de informes durante todas las etapas del proceso (Koomen et al., 2006).
- **Fase 2 - Pruebas de aceptación y de sistema:** estas pruebas son consideradas como procesos organizados de forma autónoma que tienen un plan propio de prueba, un presupuesto y, a menudo, su propio entorno. Se tratan de procesos ejecutados en paralelo al proceso de desarrollo, que deben iniciarse al mismo tiempo que se crean las especificaciones funcionales. En ambos niveles de pruebas, con el fin de crear el plan de pruebas y ejecutar distintas actividades dentro del proceso, se utiliza el modelo de ciclo de vida Tmap de la Figura 4-7 como referente. Este modelo es aplicable a todos los niveles de prueba, y permite coordinar y sincronizar estos niveles. Transmite responsabilidades y tareas entre grupos involucrados, además de dividir el proceso de prueba en diferentes fases, actividades y productos.

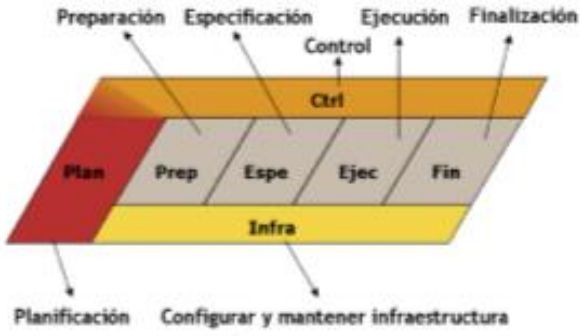

Figura 4-7 Modelo de Ciclo de Vida de Tmap: modelo genérico para pruebas definido en siete fases: Planificación, Control, Organización y Mantenimiento de la infraestructura, Preparación, Especificación, Ejecución y Finalización (van der Aalst et al., 2010)

- **Fase 3 - Pruebas de desarrollo:** Las pruebas de desarrollo (o pruebas unitarias, en este modelo se realizan comenzando desde partes más pequeñas del sistema (rutinas, unidades, programas, módulos y objetos. Una vez que se establece que tienen una calidad aceptable, el sistema es sometido a pruebas integrales. El modelo de ciclo de vida de Tmap puede ayudar a identificar y describir las diferentes actividades para los test de desarrollo.
- **Fase 4 - Procesos de soporte:** Es necesario un entorno para dar soporte a las pruebas, y que en algunos casos cumpla con un conjunto de requisitos genéricos que garanticen la ejecución fiable de las pruebas. Debe ser representativo, manejable y flexible, y debe, además, garantizar la continuidad de las pruebas. Adicionalmente, es necesaria la implantación de procesos para gestionar la configuración y el mantenimiento de estos entornos.

Kit de herramientas:

TMap ofrece un buen soporte para la ejecución del proceso de pruebas estructuradas a través de un completo conjunto de herramientas. Se centra en las técnicas (¿cómo?), la infraestructura (¿dónde? ¿Con qué?) y la organización (¿quién?) (van der Aalst et al., 2010).

Adaptabilidad:

La adaptabilidad de TMap Next, que es más inclusiva que la planteada por TMap, se resume en:

- **Respuesta a los cambios:** Sucede en las primeras actividades del MTP. La obtención de información sobre el entorno en el que se ejecuta el test y el establecimiento de posibles cambios desempeñan un papel importante. Si la estrategia de test no es aceptada por el cliente, el plan se adapta.
- **(Re) utilización de productos y procesos:** TMap Next ofrece la posibilidad de utilizar productos y procesos rápidamente, gracias a sus técnicas de diseño, checklists, plantillas, etc. En la fase final, se definen las actividades para identificar la reutilización y preservación testware.
- **Aprendizaje desde la experiencia:** Al ser un método, TMap Next ofrece el marco necesario para aprender y aplicar lo que ya se ha utilizado con anterioridad. La actividad de evaluación del proceso de test forma parte del Modelo de Ciclo de Vida. Las métricas también son importantes para mejorar el proceso de test de forma continua.
- **Probar antes de usar**: Tmap consta de un instrumento para ensayos de test (intake), creando un espacio para realizar pruebas previas a la puesta en funcionamiento del producto. Las principales herramientas para dichas pruebas son actividades relacionadas con la entrada de datos.

4.5.2 *Roles*

TMap otorga una especial importancia al concepto de rol, si bien hasta ahora se ha venido considerando como un concepto de apoyo para dar a un perfil profesional una mayor especialización dentro de una organización, evitando su equiparación. En TMap, el concepto de rol se centra en las actividades de prueba que son realizadas dentro de un proyecto específico de la organización. Por ello, se forman un conjunto de competencias mucho más especializadas que las asociadas a posiciones o perfiles profesionales. Sin embargo, al encontrarse en el ámbito de una misma organización, se encuentran fuertemente relacionados (Van Driel, 2010). Algunos de los roles son: Tester, Programador de herramientas de pruebas, Experto en métodos de prueba, Coordinador de pruebas, Experto en herramientas de pruebas, Analista de pruebas, Gerentes de pruebas y Consultor de herramientas de pruebas. La particularidad de estos roles es que no concuerdan con ninguna posición usualmente reconocida a modo de guía (Van Driel, 2010).

4.5.3 *Aplicación de TMap*

Dentro de las ventajas de este modelo específico de pruebas, se puede destacar que describe un proceso de Pruebas como un ciclo de gestión de un proyecto. Además, posee un conjunto de herramientas completo para el proceso de Pruebas: análisis de riesgos, estrategia de pruebas y priorización de éstas con una clara orientación al Negocio, técnicas de especificación de pruebas, entre otras, que facilitan que las pruebas sean reutilizables en el futuro.

TMap, al igual que TPI, podría ser utilizado en organizaciones con una estructura organizativa de pruebas basada en equipos independientes o grupos de SQA. En el caso de que los desarrolladores sean los testers o equipos integrados de pruebas, donde los equipos no son muy numerosos, la carga de trabajo sería excesiva. Además, requiere de tener implantada la metodología TPI, donde los procesos no están definidos en detalle y no se describe la interacción entre ambas metodologías. La principal limitación que aparece a la hora de implementar tanto TMap como Tmap Next es la extensión que tiene. La cantidad de procesos que define, roles y actividades a llevar a cabo es más adecuada a grandes organizaciones o departamentos que se dedican únicamente a la realización de actividades de verificación y validación de productos software.

Cabe destacar que poseen una aplicación móvil que organiza las actividades, tareas y roles, colaborando con su implementación. Sin embargo, su adaptación a entornos de pequeñas y medianas empresas es compleja y requiere de un trabajo de consultoría que permita adaptar los procesos, roles y actividades a llevar a cabo a las particularidades de este tipo de organizaciones.

Además, como se dijo anteriormente, para poder determinar acciones de mejoras en el proceso de pruebas, TMap deber ser complementado por TPI, que es el mecanismo definido por Sogeti para este propósito. TMap en sí mismo no proporciona medios para la evaluación y mejora del proceso de pruebas. Cabe destacar además que, si bien es reconocido internacionalmente, no es un estándar y no existe mucha documentación al respecto.

4.6 IT Mark

IT Mark diseñado por el *European Software Institute* (ESI) y sus Socios de la Alianza de los *ESI Centers*, es el primer modelo de calidad internacional diseñado específicamente para las pequeñas y medianas empresas. Es un modelo escalable y muy adecuado para las Pequeñas y Microempresas del sector TIC. Tiene como objetivo brindar un sello de calidad para PyMEs de Tecnologías de la Información, que acredita su madurez y capacidad. También tiene como objetivo mejorar la efectividad organizacional y el éxito en el mercado mediante la mejora de sus procesos. El servicio proporciona conocimiento sobre las capacidades técnicas y gerenciales de la organización a través de la identificación de fortalezas y debilidades, así como del descubrimiento de áreas a mejorar de acuerdo con sus metas de negocio. La comercialización de I.T. Mark se realiza, de momento, en el área de actividad de los ESI Centers hasta ahora constituidos (Brasil, México, Bulgaria, Australia y China), y en los 50 países del mundo en los que tiene cobertura la Red ESI@net (*IT Mark*, 2017).

Este modelo Trabaja desde tres perspectivas de la empresa. El primero es la Gestión General de la empresa, de acuerdo con el modelo 10-squared, que estudia diez categorías de procesos como son estratégica, comercial, financiera, definición de productos y servicios, conocimiento del mercado, marketing, etc., hasta obtener una visión exhaustiva de la empresa. Cada una de estas categorías tiene en cuenta diez elementos, entre los cuales existen algunos elementos críticos, en función del estado de desarrollo de la empresa: Semilla, Start-up, Desarrollo o Expansión.

El segundo se trata de la Seguridad de la Información, basada en la norma ISO/IEC-17799:2005, en la que IT Mark define varios niveles entre los que se destaca el Centrado en la organización de la seguridad, responsabilidades, requisitos legales (Protección de Datos Personales, etc.), y controles de seguridad; la Gestión de la seguridad se haya convertido en un proceso estandarizado en la organización; y la Mejora Continua del Sistema de Gestión de la Seguridad de la Información

Por último, trabaja desde la perspectiva de los Procesos de desarrollo de Software, Sistemas y núcleo del modelo, basado en CMMI.

4.6.1 *Niveles IT Mark*

El esquema I.T. Mark distingue tres niveles posibles, y progresivamente más exigentes, en función de la madurez demostrada en los procesos de cada PYME. Así, el nivel I.T. Mark Standard acredita a una empresa que es consciente de los problemas relacionados con la gestión técnica, de seguridad y del negocio, y que los mantiene habitualmente bajo control. Para ello, se admite que algunas de las áreas de proceso puedan no estar suficientemente elaboradas, trabajándose en el caso particular de los Procesos Técnicos sobre CMMI Nivel 2 y por medio de evaluaciones rápidas orientadas fundamentalmente a la identificación de debilidades (de Clase C).

El I.T. Mark Premium, acredita a una empresa que ha conseguido una Buena Madurez en sus procesos de trabajo técnico, seguridad y del negocio. En este caso los niveles necesarios son considerablemente superiores a los anteriores, exigiendo que todos los procesos evaluados desde los tres puntos de vista estén razonablemente desarrollados. Para Premium, CMMI se trabaja sobre el Nivel 2 de Clase B.

Por último, I.T. Mark Elite acredita a una empresa que ha conseguido un nivel Superior en la Definición e Institucionalización de sus procesos de trabajo técnico, de seguridad y de negocio, por lo que se confía en que la calidad de sus productos sea buena, debido a la madurez de sus procesos y a la mejora continua. En el caso de Elite, los niveles de exigencia son proporcionalmente superiores, y en caso de CMMI se trabaja sobre el Nivel 3.

Nivel	Evaluación de los procesos de negocio	Evaluación de los procesos de seguridad de la información	Evaluación de los proceso de gestión y desarrollo de software	
			Clase de la evaluación	Hallazgos
it mark ELITE	No hay categoría en rojo y > 75%	Nivel 3	Clase B, Nivel de madurez 3	Ningún área de proceso en rojo & Nº áreas de proceso en verde >=11
it mark PREMIUM	No hay categoría en rojo y > 60%	Nivel 2	Clase B, Nivel de madurez 2	Ningún área de proceso en rojo & Nº áreas de proceso en verde >=3
it mark	No más de una categoría en rojo y > 50%	Nivel 1	Clase C, Nivel de madurez 2	Clase C: No más de 2 AP alcanzando menos de 50%. Estas AP no pueden ser ni PP, ni PMC.

Tabla 4-1 Niveles de IT Mark

IT Mark está diseñado de forma que define un Camino de Mejora Continua incluso para Microempresas, que es totalmente compatible y alineado con el modelo CMMI

(como se ve en la Figura 4-8 - Equivalencias de Niveles de IT Mark con CMMI), y que ayuda a las empresas en su permanente búsqueda de competitividad en el mercado.

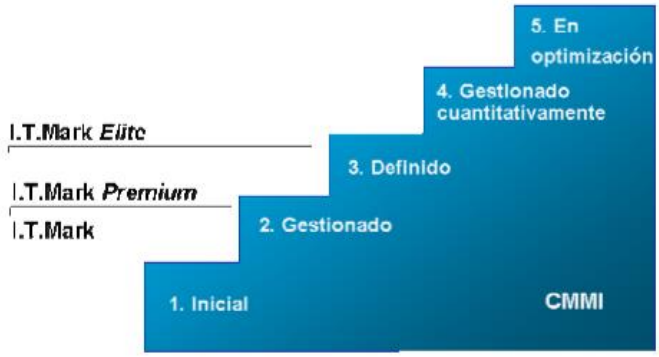

Figura 4-8 Equivalencias de Niveles de IT Mark con CMMI

Cada nivel de certificación IT Mark está compuesto por 5 partes: Sesiones de Conocimiento, Evaluación de la gestión de Negocio, Evaluación de seguridad, Evaluación de procesos de software y Presentación de resultados y oportunidades de mejora.

El servicio que presta IT Mark se realiza por medio de evaluadores experimentados, se basa en modelos mundialmente reconocidos y se completa con la explicación de los resultados preliminares a la organización afectada. Dicha explicación contiene los puntos fuertes y puntos débiles de los Procesos de la organización, de acuerdo con la Buena Práctica y los diferentes modelos descritos. Los puntos débiles constituyen la base para la elaboración del Plan de Mejora que la empresa deberá desarrollar de acuerdo con sus objetivos estratégicos y su proceso de Mejora Continua. Una vez implantadas las mejoras recomendadas, se llevará a cabo una evaluación final que resolverá si la PyME cumple los requisitos necesarios que acrediten la calidad de sus procesos. La PyME que alcance el nivel establecido por el modelo IT Mark Standard será acreedora de la certificación IT Mark. La certificación propiamente dicha no es el objetivo del Modelo, pero sí constituye un hito alcanzable a corto/medio plazo que proporciona a la empresa un reconocimiento del mercado. Los niveles Premium y Elite son ya más ambiciosos, ya que proporcionan un gran nivel de confianza en alcanzar niveles equivalentes del modelo CMMI por un coste inferior.

4.6.2 *Aplicación de IT Mark*

IT Mark es el primer servicio internacional de certificación que evalúa los procesos técnicos y de negocio, diseñado específicamente para PyMEs del sector TI. Cuenta con una serie de beneficios, entre los cuales podemos nombrar que puede utilizarse tanto en PyMEs como en Microempresas (menos de 10 empleados) o en grandes Organizaciones. Este modelo, además, mejora el desempeño del negocio (desde el punto de vista técnico y administrativo), ayudando a ganar reconocimiento en el mercado en cuanto a capacidades de TI. Está basado en modelos internacionalmente reconocidos como CMMI e ISO17799:2005 y funciona como un buen mecanismo para avanzar luego hacia una valoración integral de alguna de estas certificaciones. Se ha aplicado con éxito en Europa, Asia e Iberoamérica.

Por otra parte, el programa incluye valoraciones en Gestión del Negocio y en Seguridad Informática, ignoradas por otros modelos, por encima de los procesos de Software y Sistemas.

Como inconvenientes podríamos destacar que es menos difundido que otros modelos, no apunta exclusivamente a la mejora del producto o del proceso de pruebas, no existe mucha documentación por lo que es necesario contar con consultores para llevar a cabo el proceso y no se puede considerar como un estándar internacional.

4.7 COMPETISOFT

El proyecto COMPETISOFT - Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria del Software de Iberoamérica, fue publicado a fines de 2008 con financiamiento del Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo (CYTED). Este programa, se define como un programa internacional de cooperación científica y tecnológica multilateral, de ámbito iberoamericano, que tiene como objetivo principal contribuir al desarrollo de la Región Iberoamericana, mediante el establecimiento de mecanismos de cooperación entre grupos de investigación de las Universidades, Centros de I+D y Empresas innovadoras de los países iberoamericanos (Competisoft, 2008).

La metodología COMPETISOFT indica que el proceso de desarrollo de software se compone de uno o más ciclos de desarrollo. Cada ciclo está compuesto de las

siguientes fases: Inicio, Requisitos, Análisis, Diseño, Construcción, Integración, Pruebas y Cierre.

En todas las fases existen actividades de V&V que se realizan para asegurar la calidad y corrección de los distintos productos que se generan a lo largo del ciclo de desarrollo. Por ejemplo, en la fase de Inicio se incluye la revisión del Plan del Proyecto; en la fase de Requisitos se incluye un conjunto de actividades para obtener la documentación de Especificación de Requisitos y el Plan de Pruebas de Sistema; en las fases de Análisis y Diseño se obtiene el documento de Especificación del Sistema y el Plan de Pruebas de Integración; entre otras. Las actividades de testing, si bien tienen una fase específica donde ocurren, se realizan en paralelo con el resto de las fases.

Figura 4-9 Fases de COMPETISOFT

En este modelo, al construir el producto en distintas iteraciones, se generan distintas versiones del mismo. Cada versión puede agregar nuevas funcionalidades o mejorar las ya existentes en versiones previas y se deben ejecutar las pruebas planificadas para esa versión. Así el producto va evolucionando hasta obtener una versión lo suficientemente madura para que sea liberada y entregada al cliente.

En el caso de que el tiempo con el que se cuenta para la ejecución de las pruebas para una versión sea demasiado corto y no permita ejecutar todos los casos de prueba diseñados, se debe definir qué pruebas ejecutar para cada versión. Es conveniente utilizar un enfoque basado en los riesgos del producto.

4.7.1 *Estructura de COMPETISOFT*

El modelo COMPETISOFT, está enfocado en procesos y considera los tres niveles básicos de la estructura de una organización para su implementación: la Alta Dirección, Gestión y Operación (Ver Figura 4-10 - Diagrama de paquetes de categorías de procesos (Competisoft, 2008)). El modelo pretende apoyar a las organizaciones en la estandarización de sus prácticas, en la evaluación de su efectividad y en la integración de la mejora continua. A continuación, se detallan brevemente cada una de las categorías:

Categoría de Alta Dirección (DIR)

La categoría de procesos de Alta Dirección (DIR) aborda las prácticas relacionadas con la gestión del negocio. Proporciona también los lineamientos a los procesos de la Categoría de Gerencia y se retroalimenta con la información generada por ellos. Esta categoría contiene el proceso de Gestión de Negocio donde se establece la razón de ser de la organización, sus objetivos y las condiciones para lograrlos.

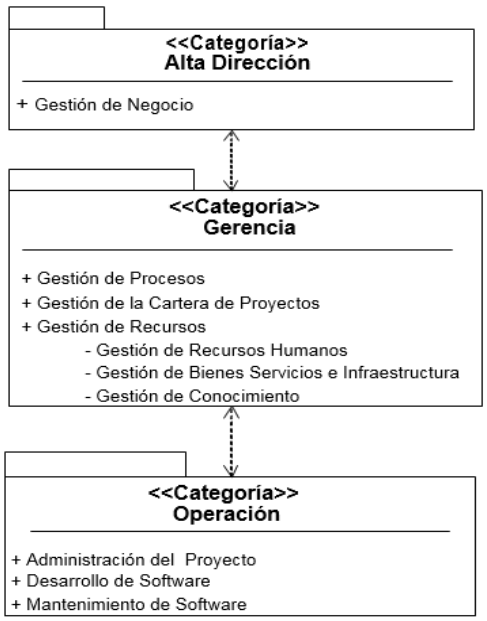

Figura 4-10 Diagrama de paquetes de categorías de procesos (Competisoft, 2008)

Categoría de Gerencia (GER)

La categoría de Gerencia (GER) aborda las prácticas de gestión de procesos, proyectos y recursos en función de los lineamientos establecidos en la Categoría de Alta Dirección. Proporciona además los elementos para el funcionamiento de los procesos de la Categoría de Operación, recibe y evalúa la información generada por éstos y comunica los resultados a la Categoría de Alta Dirección.

Esta categoría está integrada por la de Gestión de Procesos, donde se establecen los procesos de la organización; el proceso de Gestión de Proyectos que asegura que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización; y por la Gestión de Recursos que consigue y dota a la organización de los recursos humanos, infraestructura, ambiente de trabajo y proveedores. Éste último está constituido por los subprocesos de Gestión de Recursos Humanos, Gestión de Bienes, Servicios e Infraestructura y Gestión de Conocimiento.

Categoría de Operación (OPE)

Aborda las prácticas de los proyectos de desarrollo y de mantenimiento de software. Esta categoría realiza las actividades de acuerdo con los elementos proporcionados por la Categoría de Gerencia y entrega a ésta la información y productos generados. Esta categoría está integrada por los procesos de Administración de Proyectos Específicos y Mantenimiento de Software, donde se establece y lleva a cabo sistemáticamente las actividades que permitan cumplir con los objetivos de un proyecto en tiempo y costo esperados. Así como también, por el proceso de Desarrollo de Software que se ocupa de la realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevos cumpliendo con los requisitos especificados y con las normativas de seguridad de información.

4.7.2 *Niveles de Capacidad*

Con respecto al Modelo de Evaluación de Procesos, COMPETISOFT no define uno en particular, sino que deja en manos de quien lo implementa que defina su propio modelo de evaluación de las capacidades. La única condición que se aclara es que esté de acuerdo a las necesidades de su industria de software y conforme a las normas internacionales ISO/IEC 15504-2 Performing an assessment e ISO/IEC 15504-4 Guidance on performing assessment. Es decir, para llevar a cabo una evaluación formal COMPETISOFT se puede usar cualquier modelo de evaluación que sea conforme con esta norma internacional, la cual define cinco niveles de capacidad para los procesos de la organización: Realizado, Gestionado, Establecido, Predecible y Optimizado (Competisoft, 2008).

Nivel	Capacidad de Proceso	Color
1	Realizado	amarillo
2	Gestionado	azul
3	Establecido	verde
4	Predecible	rosa
5	Optimizado	ninguno

Tabla 4-2 Correspondencia entre los niveles de capacidad de proceso y los colores de los productos COMPETISOFT

En la tabla anterior se pueden ver las correspondencias entre los niveles de capacidad de proceso y los colores en que se marcan los productos de entrada,

salida e internos y toda la sección de prácticas del modelo de procesos. Esta correspondencia se puede usar como guía en el orden de implementación de las prácticas, pero no implica que sea suficiente para las evaluaciones.

Se recomienda implementar primero los productos y las prácticas pertenecientes al nivel 1 de capacidades. Una vez implementadas éstas se sugieren agregar los productos y las prácticas de los siguientes niveles. El nivel 5 se logra cuando de manera sistemática a través del tiempo se cumplen los objetivos y se logran mejorar las metas cuantitativas de los procesos.

4.7.3 *Aplicación del COMPETISOFT*

En función de lo expuesto anteriormente y de la revisión de algunos casos testigo de la implementación de este modelo en empresas latinoamericanas, se puede concluir que COMPETISOFT es un modelo que orienta adecuadamente al proceso de mejora continua de una organización, sin los esfuerzos y los costos que demandan modelos más ampliamente aceptados como por ejemplo CMMI. Si bien este modelo entra en la categoría de modelos descriptivos, propone estrategias que facilitan a las organizaciones, especialmente aquellas de tamaño mediano o pequeño, la implantación de un proceso de mejora continua asociado al proceso de desarrollo en general y no sólo enfocado a las pruebas. Esto significa, que puede ser el puntapié inicial para facilitar el proceso de implementación de modelos más complejos en este sector de la industria.

Este modelo, además, considera el proceso de Gestión de Negocio para establecer la razón de ser de la organización, lo que facilita su puesta en marcha en organizaciones que se inician en el camino de la mejora de procesos. La arquitectura que presenta el modelo COMPETISOFT es una guía útil para que una organización analice y reestructure sus procesos considerando aspectos gerenciales, administrativos y técnicos. Las áreas de proceso que proponen acciones que son transversales a diferentes procesos (Verificación, validación y Mediciones) ya se encuentran articuladas en los mismos, y a la vez identifican los productos que hay que verificar y validar facilitando de esta manera su aplicación. Cabe destacar, además, que cuenta con una herramienta que corre sobre el Framework EPF Composer que guía la implementación de COMPETISOFT.

Por otro lado, si bien el modelo no está alineado a un enfoque de desarrollo o marco metodológico en particular, las fases de proceso que propone son más adecuadas para enfoques de tipo tradicional. No se detecta articulación de especializaciones al modelo que facilite su aplicación en diferentes contextos, como, por ejemplo, los de desarrollo ágil. También, a nivel de proceso organizacional, al definir tres categorías dentro de los niveles la organización, no considera aquellas donde los límites no son tan claros como, por ejemplo, las organizaciones horizontales.

Por último, este modelo hace énfasis en la definición de un proceso integrado de Gestión de Conocimiento que permite, desde la estructura misma de los procesos, proponer una estrategia global de gestión y aprovechamiento de los activos de conocimiento producidos y evaluados en los diferentes procesos.

CAPÍTULO 5: CARACTERIZACIÓN DE LAS PYMES DESARROLLADORAS DE SOFTWARE

En la siguiente sección, se realiza una breve caracterización de la industria de las PyMEs desarrolladoras de software en nuestro país. Se detallan cuáles son los desafíos del sector y cómo se encuentran actualmente frente a la calidad de los productos que realizan.

5.1 ¿Qué son las PyMEs?

Según el Ministerio de Producción y Trabajo, una PyME es una micro, pequeña o mediana empresa que realiza alguna de sus actividades en el país, en alguno de estos sectores: servicios, comercial, industrial, agropecuario, construcción o minero. Puede estar integrada por una o varias personas, o por un conjunto de empresas[5].

Asimismo, el Ministerio de Producción establece la categoría de las PyMEs de acuerdo con la actividad declarada, a los montos de las ventas totales anuales de cada empresa o a su cantidad de empleados.

Tomado como referencia la clasificación que realiza el Ministerio de Producción y Trabajo para el año 2019, más las consideraciones internacionales al respecto, para los fines prácticos de este trabajo se establece la siguiente clasificación:

Clasificación	Cantidad de trabajadores
Microempresa	De 1 a 10
Pequeña	De 10 a 50
Mediana	De 50 a 250
Grande	Más de 250

Tabla 5-1 Clasificación de empresas, según su cantidad de trabajadores

Las PyMEs se caracterizan por ser empresas heterogéneas y diversas. Los costos de las inversiones realizadas por estas no suelen ser elevados y pueden convivir y producir en un mismo sector, con diferente cantidad de trabajadores o producción.

[5] Ministerio de Producción y Trabajo de la Nación. (2019). Registrarse como PyME. Buenos Aires, Argentina: argentina.gob.ar. Recuperado de https://www.argentina.gob.ar/produccion/registrarse-como-pyme

5.2 Situación de las PYMEs en Argentina

Las PYMES, tanto en Argentina como en el resto del mundo, son una parte importante para la economía por las contribuciones y la repartición de bienes y servicios que impulsan. Además de generar riqueza, las PyMEs son importantes generadoras de mano de obra y, por lo tanto, de arraigo local; permiten una distribución geográfica más equilibrada de la producción y del uso de recursos y de la riqueza que generan; tienen una flexibilidad que les permite adaptarse a los cambios tecnológicos y económicos y en muchos casos detectar nuevos procesos, productos y mercados. Sobre todo, poseen una capacidad dinámica y una gran potencialidad de crecimiento.

Varios modelos de desarrollo productivo de países que hoy figuran entre las principales economías mundiales (la UE, los Estados Unidos y Japón, por citar algunos ejemplos exitosos) se han basado en políticas exitosas de fortalecimiento y promoción del crecimiento de sus empresas de menor porte relativo. En Argentina los fundamentos de su sector industrial lo construyeron las PyMEs surgidas de la gran corriente inmigratoria del siglo XIX. Hoy el país cuenta con más de 650.000 PyMEs, que representan el 99.6% del total de unidades económicas y aportan casi el 70% del empleo, el 50% de las ventas y más del 30% del valor agregado. Desempeño que fue logrado a pesar de que en más de un período de la historia económica del siglo XX las políticas implementadas atentaron contra la industria nacional en general y contra las empresas de menor dimensión en particular (Roura, 2019)

En este contexto, recientemente se promocionó la Ley Pyme 27.264 que fomenta la actividad de las PyMEs otorgando beneficios como la eliminación del impuesto a la ganancia mínima presunta, la compensación del impuesto a créditos y débitos bancarios y el diferimiento del pago del IVA a 90 días, entre otros. Además, fomenta las inversiones con la Desgravación del Impuesto a las Ganancias, hasta el 10% de las inversiones realizadas, y la Devolución de IVA de dichas inversiones a través de un Bono de crédito fiscal para el pago de impuestos. Otorga más crédito, amplía el cupo prestable de la Línea de Créditos de Inversión Productiva (*PROGRAMA DE RECUPERACIÓN PRODUCTIVA - Ley 27264*, 2016).

5.3 Situación del Sector de Software y Servicios Informáticos de la República Argentina

En este apartado, se detallan algunos de los hitos a tener en cuenta para poder establecer el contexto de la producción de software en Argentina.

5.3.1 *Ley de Software*

En el 2004, fue promulgada en Argentina la “Ley de Software” con el fin de que se reconociera al software como industria. En la misma, se establece que *“la actividad de producción de software debe considerarse como una actividad productiva de transformación asimilable a una actividad industrial, a los efectos de la percepción de beneficios impositivos, crediticios y de cualquier otro tipo”*.
Los beneficios para las empresas del sector han sido considerables: hasta 70% de crédito fiscal, para ser utilizado para pagar impuestos nacionales, hasta 60% de desgravamen sobre el Impuesto a las ganancias, estabilidad fiscal durante el período de vigencia del régimen de promoción (por lo menos hasta el año 2019). Para optar a estos beneficios, es necesario que las empresas cumplan con requisitos asociados a las actividades promovidas: contar con más del 50% de las personas empleadas y con más del 50% de la masa salarial dedicada al software, destinar más del 3% de la facturación a investigación y desarrollo de software, certificarse en estándares de calidad de software internacionales, garantizar estabilidad laboral.

5.3.2 *CESSI*

Un actor fundamental es la Cámara de Empresas de Software y Servicios informáticos (CESSI). Esta cámara es específica para el sector de software (a diferencia de otros países donde el software se subordina al sector más amplio de TIC) y que existe desde hace más de 30 años. CESSI representa a más de 600 empresas de software, que en conjunto, comprenden más del 80% de los ingresos del sector y más del 80% de los empleos (*CESSI Argentina*, 2020).
CESSI opera distintos programas que apoyan el desarrollo de la industria, entre ellos:

- EmplearTEC. Capacitación gratuita para formar personal que se pueda desempeñar en la industria de software.

- Bridge IT. Red de mentores que proporciona asesoría y vinculación a nuevos emprendedores.
- Espacio TI. Apoyo para difundir la oferta de pequeñas empresas de TI.

5.3.3 *Fundación Sadosky*

Este es un organismo, es una fundación sin fines de lucro cuyo objetivo es promover la articulación entre el sistema científico-tecnológico y la estructura productiva, con el fin de hacer llegar los beneficios de las TIC a toda la sociedad (*Fundación Sadosky*, 2020). Estas son algunas de las iniciativas en las que la Fundación actualmente está involucrada:

- Program.ar: Iniciativa para fomentar el aprendizaje de programación en los jóvenes. Destaca que recientemente se consiguió que el Consejo Federal de la Educación declare como estratégica la enseñanza de la programación en todas las escuelas del país por lo que próximamente se enseñará programación de la misma manera que se enseñan materias básicas como matemáticas o historia.
- Estudiar Computación: Es un sitio web con información dedicada a ayudar a que los jóvenes descubran las oportunidades y beneficios que implica estudiar una carrera relacionada con las TI. Además, brinda información precisa y sencilla sobre las diferencias y similitudes existentes entre las distintas disciplinas que conforman el mundo de la computación (ej. ¿Cuál es la diferencia entre Ciencias de la Computación e Ingeniería de Software?, ¿en qué instituciones puedo estudiar estas carreras?).
- Ciencia de Datos: El Ministerio de Ciencias y Tecnología ha reconocido que contar con capacidades locales para el manejo y análisis de grandes datos es clave para la autonomía tecnológica del país, así como para su desarrollo económico y social. El objetivo de este programa es contribuir a que Argentina se convierta en líder regional en el campo de analítica de datos.

5.3.4 *La Industria en números*

Para entender el crecimiento y la potencialidad del sector SSI, a continuación, se detallarán algunos resultados publicados en el Reporte anual sobre el Sector de

Software y Servicios Informáticos de la República Argentina para el año 2018[6] realizado como parte del programa Observatorio Permanente de la Industria de Software y Servicios Informáticos (OPSSI), iniciativa de la CESSI mencionada en el apartado anterior.

Según la CESSI, el empleo en este sector ha aumentado un 47,8% entre 2009 y 2018, a una tasa anual acumulativa del 4,4%. A modo de comparación, el empleo registrado de todo el sector privado entre ambos años creció un 11,4% a una tasa anual acumulativa del 1,2% (Fuente: Observatorio de Empleo y Dinámica Empresarial, OEDE). A pesar de la crisis de 2008-2009 y las recientes devaluaciones de principios de 2014, finales de 2015, 2018 y los sucesos del corriente 2019, las ventas del sector medidas en dólares aumentaron en los últimos diez años un 2,9% acumulativo anual.

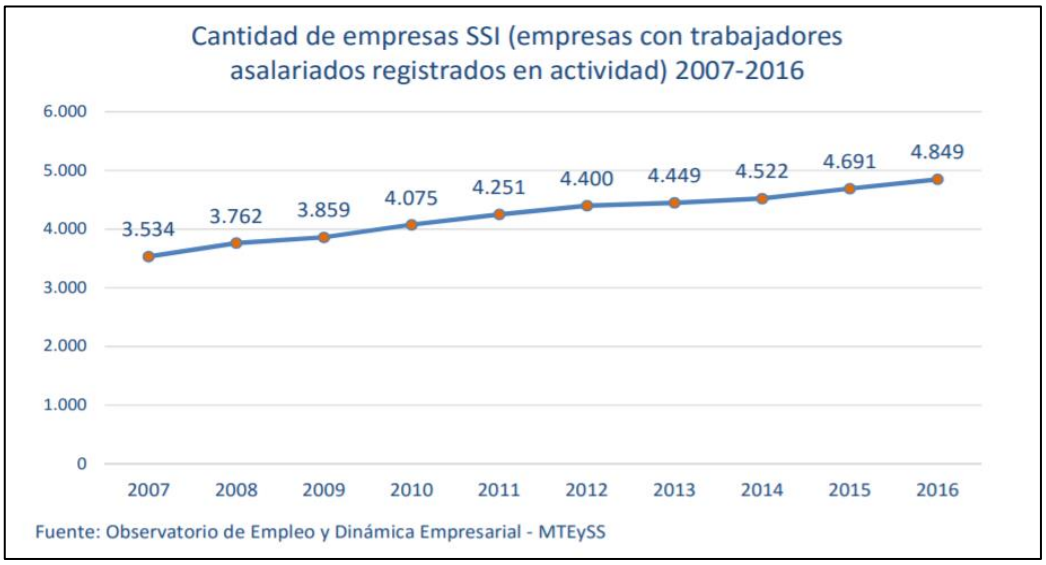

Figura 5-1 Cantidad de empresas SSI (Empresa con trabajadores asalariados registrados en actividad). Fuente: (OPSSI & CESSI, 2019)

En 2013 se tenía conocimiento de 4,288 empresas activas en la industria de software. De estas, el 75% reportaron tener menos de 10 empleados, otro 20% entre 10 y 50 empleados, 4% entre 50 y 200 empleados, y solo el 1% reportó tener más de 200 empleados, lo que denota el peso que tienen las PyMEs particularmente en este sector.

[6] Cámara de la Industria Argentina del Software. (2019). Reporte del Observatorio Permanente de la Industria del Software y Servicios Informáticos de la República. Recuperado de http://cessi.org.ar/opssi-reportes-949/index.html

El desarrollo de software a medida y las ventas de productos y soluciones propias -e implementación e integración asociados a estos productos-, explican el 66% de las ventas del sector durante los últimos tres años. Los servicios financieros (que incluyen bancos, aseguradoras, servicios de pago electrónico, etc.) son el principal cliente del sector, correspondiéndole un 35% de lo facturado por las empresas SSI en los últimos dos años. Bastante por detrás, le siguen el propio sector SSI (con un 12% de la facturación), las telecomunicaciones (con el 11%) y el comercio (con el 8%).

Al analizar las certificaciones de calidad, un 64% de las empresas manifestó tener algún tipo de certificación a diciembre de 2018 (el 60% había certificado al menos ISO9001). El 4% de ese total posee algún nivel CMMI.

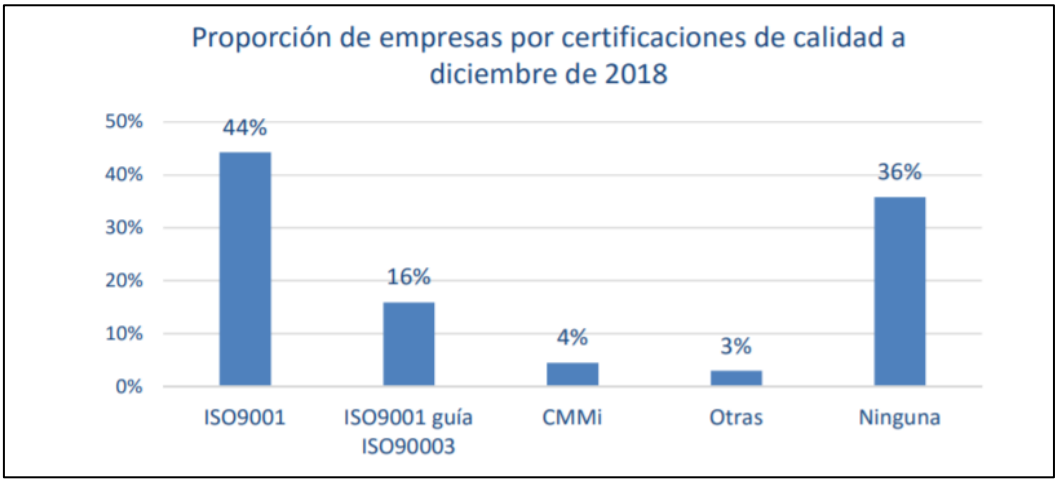

Figura 5-2 Porción de empresas por certificaciones de calidad a diciembre de 2021. Fuente:(OPSSI & CESSI, 2019).

Esta elevada proporción no es de extrañar siendo que las certificaciones de calidad son uno de los requisitos para acceder al régimen de promoción de la Ley de Software (la mayoría de las empresas relevadas percibe beneficios por el régimen o está en proceso de inscripción). Cabe aclarar que el total no suma 100% porque un 4% de las empresas indicó tener dos o más tipos de certificación.

Con respecto a las inversiones realizadas, un 70% de las empresas realizó inversiones en I+D+i durante 2018, con un volumen promedio del 7,4% de la facturación del período. A continuación, se indica en la figura 5-2 la proporción de empresas que destinaron inversión en I+D+i para cada uno de los diversos objetivos durante 2018:

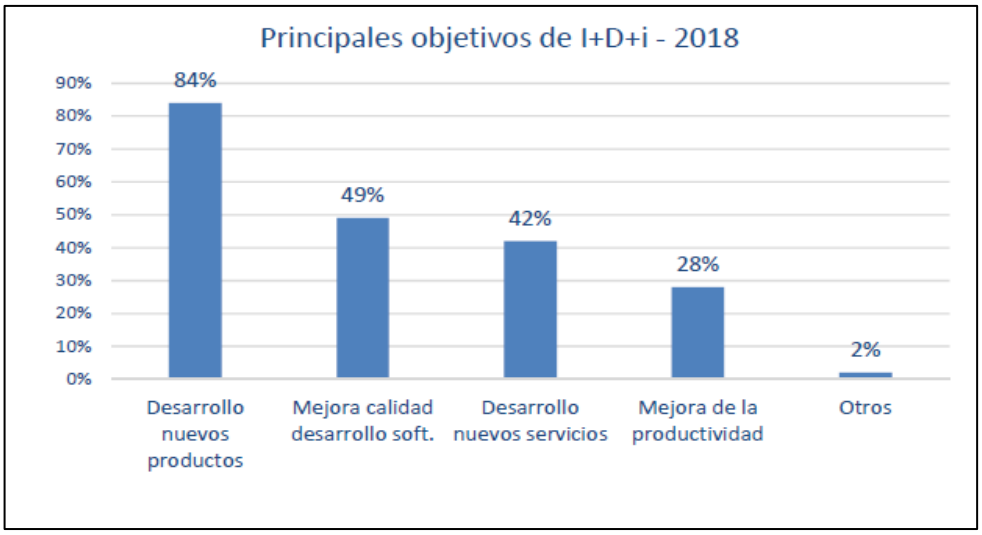

Figura 5-3 Principales objetivos I+D+I - 2018. Fuente:(OPSSI & CESSI, 2019).

Con respecto al financiamiento, en el 2018, más de la mitad de las empresas (un 63%) manifestó haber requerido financiamiento para solventar su capital de trabajo en general. Bastante por detrás aparecen necesidades de financiamiento para actualización tecnológica (32%), para capacitación del personal (22%), para RR.HH. en general (21%) y para desarrollo de nuevos mercados (20%). Lo que evidencia una de las limitaciones más importantes que tiene este sector.

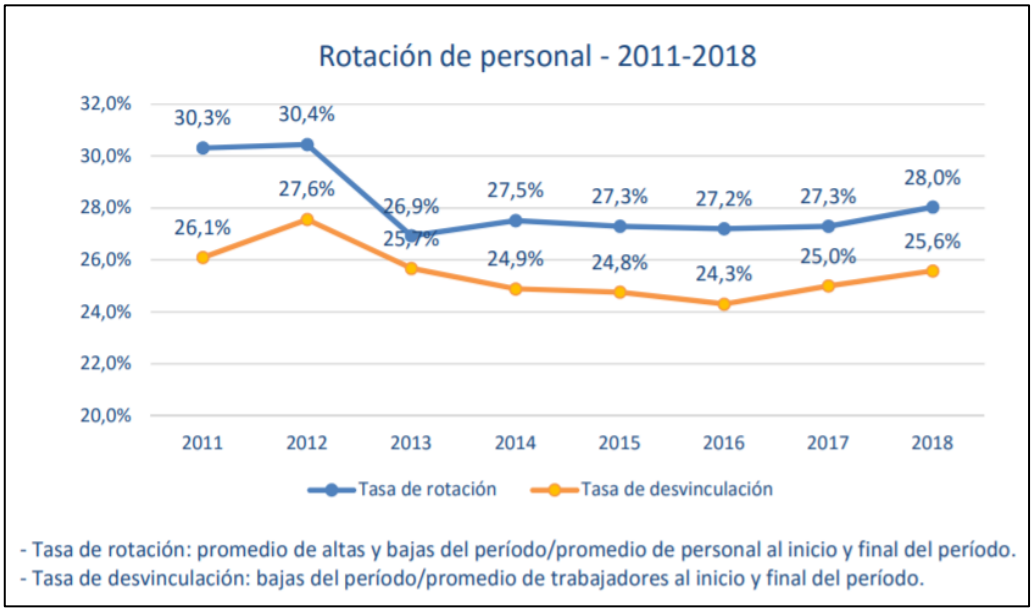

Figura 5-4 Rotación de personal - 2011-2018. Fuente:(OPSSI & CESSI, 2019).

Por último, el sector SSI, como se ha indicado en varias oportunidades en este trabajo, se caracteriza por ser mano de obra intensiva. Su estructura de costos se encuentra compuesta principalmente por los salarios y otros gastos asociados al personal, alcanzando un el 72% de los costos de las empresas, considerando tanto

recursos humanos directos como indirectos. En el gráfico anterior, se puede observar que las empresas SSI rotan en promedio más de un cuarto de sus trabajadores. Esto da cuenta de la alta demanda y baja oferta de recursos calificados en el sector.

5.4 Estado del Testing en las empresas de la región

Para poder exponer la situación actual de las PYMES del sector, con respecto al proceso de prueba de software que llevan a cabo, las habilidades y conocimientos de sus colaboradores con respecto a las pruebas y, como objetivo final, determinar el nivel de madurez de las mismas; se realizó una encuesta de carácter anónimo a personas que ocupan diversos puestos en diferentes empresas desarrolladoras de software de la región, utilizando la herramienta Google Forms. En la misma, se realizaron las siguientes preguntas de investigación:

- *¿Qué modelo de ciclo de vida utilizan las PyMEs desarrolladoras de software de la región?:* Esta pregunta permite determinar qué actividades están llevando a cabo las empresas para producir software y la tendencia en el uso de las mismas. Por ejemplo, las metodologías ágiles tienden a ciclos de desarrollo más cortos y orientados al cliente, por lo que, comparado con modelos más tradicionales, poseen menos volumen de documentación de requerimientos, análisis o de diseño, de modo que es esperable que las empresas que utilizan este tipo de metodologías dediquen una buena cantidad de horas a pruebas.
- *¿Tienen área de Análisis Funcional? ¿Tienen área de testing o testers?:* Estas preguntas nos permiten obtener una aproximación de la estructura que poseen estas empresas. Especialmente, nos permite saber si existe personal dedicado exclusivamente a la ejecución de pruebas. Una característica fundamental para determinar la madurez de un proceso de pruebas.
- *¿Tienen área de QA (Quality Assurance*)? Complementaria a las anteriores, esta pregunta de investigación permite establecer que las actividades de pruebas y aseguramiento de la calidad se están realizando con cierta formalidad que ubica, a quienes contestaron de forma afirmativa, en un nivel de madurez superior en relación a su proceso de prueba.

- *¿Los desarrolladores ejecutan pruebas unitarias? ¿Se ejecutan pruebas funcionales? ¿Trabajan bajo el concepto de "Criterios de Aceptación"?:* Este conjunto de preguntas se realizaron para poder establecer con claridad qué nivel de testing poseen las empresas que se están analizando. Además, en otro de los puntos, se pidió a quienes contestaron que marquen cuáles son las actividades de testing que se realizan dentro de su empresa para poder completar el panorama analizado.
- *¿Qué porcentaje del tiempo por proyecto se invierte en testing? ¿Existe planificación de estas actividades? ¿Qué cantidad de casos de prueba generalmente desarrollan por proyecto?:* Este grupo de cuestiones, permite caracterizar los esfuerzos invertidos en las pruebas a nivel de proyecto. Dado que el tipo de organizaciones que estamos analizando tienden a adaptar sus procesos según los clientes o los proyectos, estas preguntas también nos dan idea de la madurez de las mismas.
- *¿Qué técnicas de testing utilizan?:* Esta pregunta en particular pretende conocer el grado de conocimientos técnicos que se están aplicando para llevar a cabo las actividades de prueba.
- *¿Con qué herramienta registran los Requerimientos (ó Historias de Usuario)? ¿Cómo se registran los errores que se detectan o reportan?:* Estos puntos, además de conocer cuáles son las herramientas que utilizan y si esas herramientas son específicas para ese tipo de tareas, pretende verificar si existe una tendencia al uso de este tipo de herramientas en la muestra que se está estudiando.
- *¿Tienen algún tipo de certificación?:* Esta pregunta pretende exponer si existe una relación entre las buenas prácticas y el hecho de haber alcanzado alguna certificación.
- *¿Cuáles son los conocimientos que consideras que debe tener un buen tester?:* Por último, se pretende con este punto determinar cuáles son los conocimientos más valorados a la hora de realizar las pruebas. Además, nos permite conocer cuáles son las expectativas de aquellos que no realizan pruebas para con los que si las realizan como parte de sus tareas diarias.

Adicionalmente, para poder caracterizar esta muestra, se realizaron preguntas como: ¿Dónde está radicada la empresa?, ¿Qué puesto ocupa quien responde? y

¿Cuántas personas la componen? Se obtuvieron un total de 79 respuestas de personas que se encuentran desarrollando actividades en empresas ubicadas en su mayoría (91.1%) en las ciudades de Rosario, Buenos Aires, Córdoba, Concepción del Uruguay y Gualeguaychú. De la totalidad, el 64.9 % encuadra en la clasificación de PyME. Con respecto a los roles que ocupan las personas que respondieron, se encuentran distribuidos en Desarrolladores (39.2%), Testers (21.5%), Analistas funcionales (16.5%), Líderes de Equipo (12.7%) y otros (3.8%).
A continuación, se detallan los resultados y conclusiones asociadas al grupo de interés de esta investigación.

5.4.1 *Metodologías de Desarrollo de Software utilizadas*

Tal como se puede observar en el gráfico, la mayoría de las empresas están utilizando metodologías ágiles o similares (55.7%). De ese porcentaje, la mayoría son medianas empresas (38.6%), mientras que el grupo completo de Pymes alcanza el 65.9%.

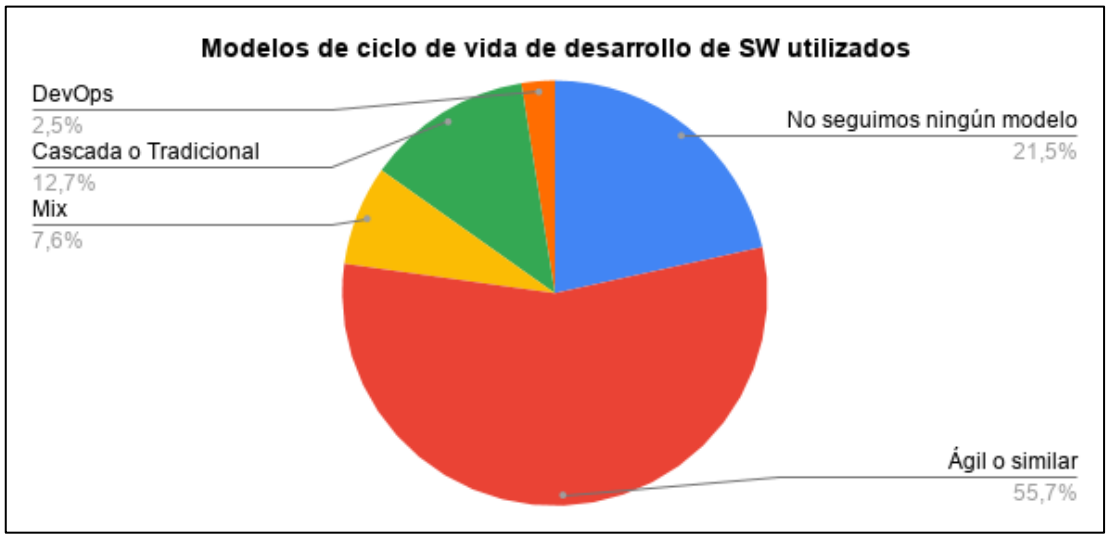

Figura 5-5 Análisis de modelos de ciclo de vida de desarrollo de SW utilizados

Por otro lado, existe un 21.5% que no sigue ningún modelo y algunos que están en un proceso de transición, incorporando alguna metodología a través de un una combinación de los modelos proceso genéricos descriptos en el Capítulo 2.
Esta situación, permite inferir que existe una tendencia a la incorporación de metodologías ágiles y que su implementación se da de manera más rápida en equipos de trabajo más pequeños.

5.4.2 *Personal de testing en las organizaciones*

Con respecto a la estructura de las organizaciones encuestadas, se detectó que en su mayoría poseen área de análisis funcional y área de testing o testers, ya sea dentro o fuera de la empresa. Sin embargo, la mayoría de las PyMEs encuestados respondió que no poseen área de QA (*Quality Assurance*).

Figura 5-6 Análisis de áreas de Testing o testers en las organizaciones encuestadas

Por otro lado, se detectó en algunos casos donde existen tareas de testing informales ya que no existe ni un área dedicada ni el perfil como tal. En muchos casos, tal como se creía, son los desarrolladores quienes realizan el testing o se asigna un responsable por equipo. Además, en función a la comparación de los datos de la encuesta, se pudo concluir que la mitad de este grupo no sigue ningún modelo de ciclo de vida para el desarrollo de sus aplicaciones.

5.4.3 *Testing ad hoc Vs Proceso de testing definido*

En lo que respecta a las pruebas y las actividades asociadas al aseguramiento de la calidad del producto, casi la mitad de los encuestados no realizan pruebas unitarias de forma continua durante el proceso de desarrollo.

Figura 5-7 Análisis de pruebas unitarias durante es desarrollo

Además se detectó que el 32.9% no ejecuta pruebas funcionales. De los cuales, la mayoría no posee área de QA, muy pocas poseen un área de testing o tester y más del 46% no sigue ningún modelo de desarrollo.

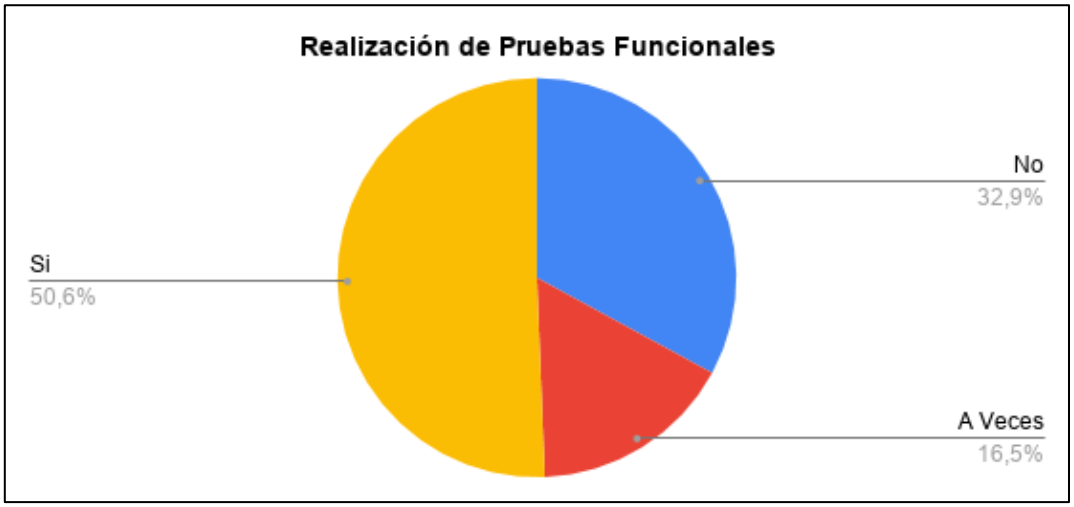

Figura 5-8 Análisis de realización de pruebas funcionales

Adicionalmente, sólo la tercera parte de los encuestados indicó que trabaja con criterio de aceptación, mientras que el 38.9% indicó que no trabaja con este criterio en todos sus proyectos, por lo que no consideran esta actividad dentro de su proceso base.

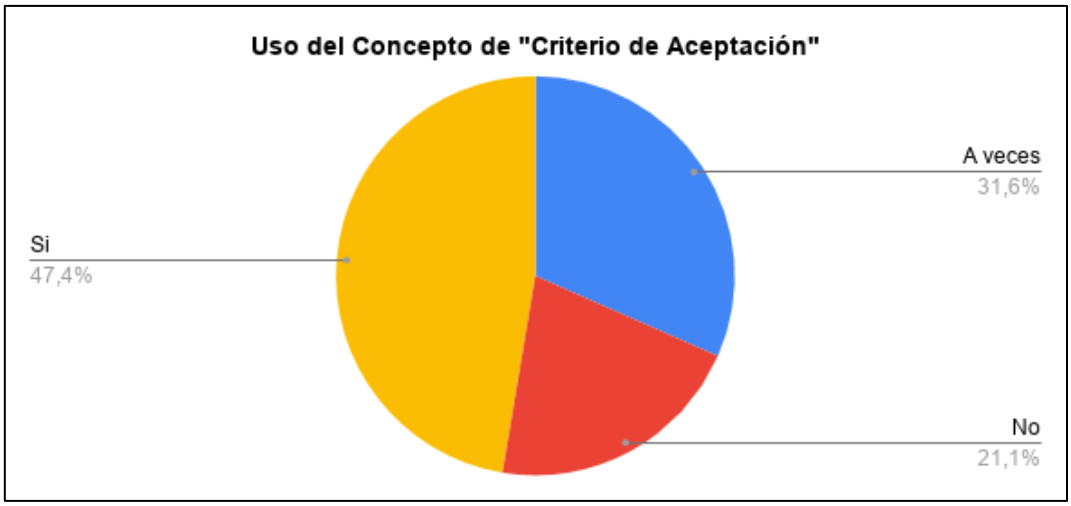

Figura 5-9 Análisis del uso del concepto de "Criterio de Aceptación"

Con base en estas respuestas, se puede concluir que independientemente que exista o no un proceso definido de pruebas, en la mayoría de los casos de las empresas encuestadas, se realiza un testing Ad Hoc. Carece de tareas clave asociadas tanto a los niveles bajos como en los superiores de pruebas.

5.4.4 *El testing en los proyectos*

En cuanto a lo relacionado con el tiempo que se invierte en tareas de pruebas, más de la mitad de los encuestados invierte menos del 30% del tiempo del proyecto y el 12.3 % no llega a invertir el 10% del tiempo. Cabe destacar que estos porcentajes pertenecen a quienes afirmaron contar con un área de testing o testers, independientemente que sea dentro o fuera de la empresa.

Figura 5-10 Análisis del porcentaje de tiempo invertido en testing por proyecto

Por otro lado, podemos interpretar a aquellos que no cuantifican el tiempo como la informalidad en el proceso que impide identificar oportunidades de mejora.

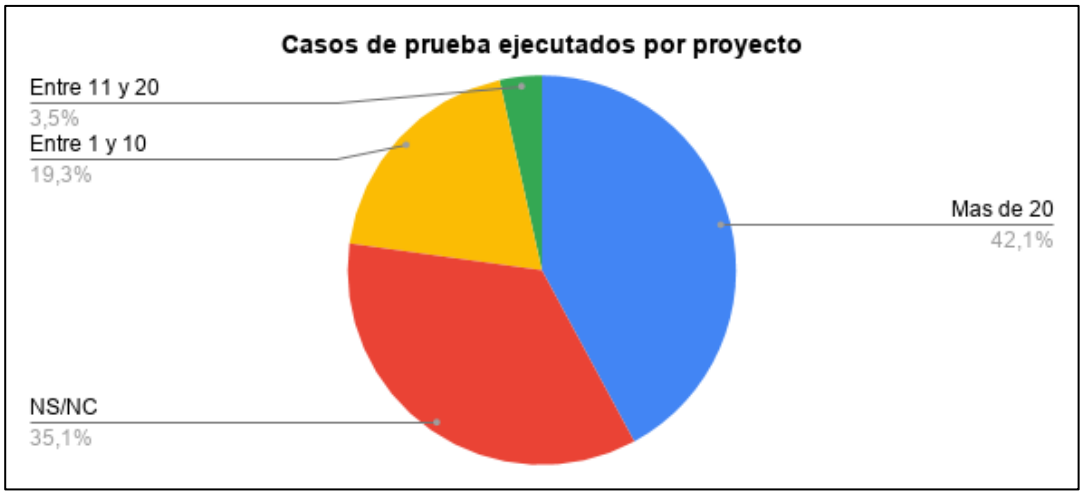

Figura 5-11 Análisis de casos de prueba ejecutados por proyecto

Con respecto a los casos de prueba, se relevó que un 22,8% no llega a ejecutar 20 casos de prueba por proyecto, lo que denota un nivel de pruebas en general muy precario. Además, como se dijo anteriormente, aquellas respuestas NS/NC exponen los niveles de informalidad de los procesos.

En lo referido a la planificación de las actividades de pruebas, el 30.4% contestó que no las planifica en ningún proyecto. De ese porcentaje, más de la mitad no sigue ningún modelo para el desarrollo de sus productos.

Figura 5-12- Análisis de planificación de actividades de testing en los proyectos

El hecho de que exista la planificación de las pruebas a realizar, es la base de la mayoría de las certificaciones ampliamente reconocidas.

En cuanto a cuáles son las tareas que se realizan como parte del proceso de pruebas, para aquellos encuestados que afirmaron que se realizaba testing en sus empresas, se obtuvieron las siguientes respuestas:

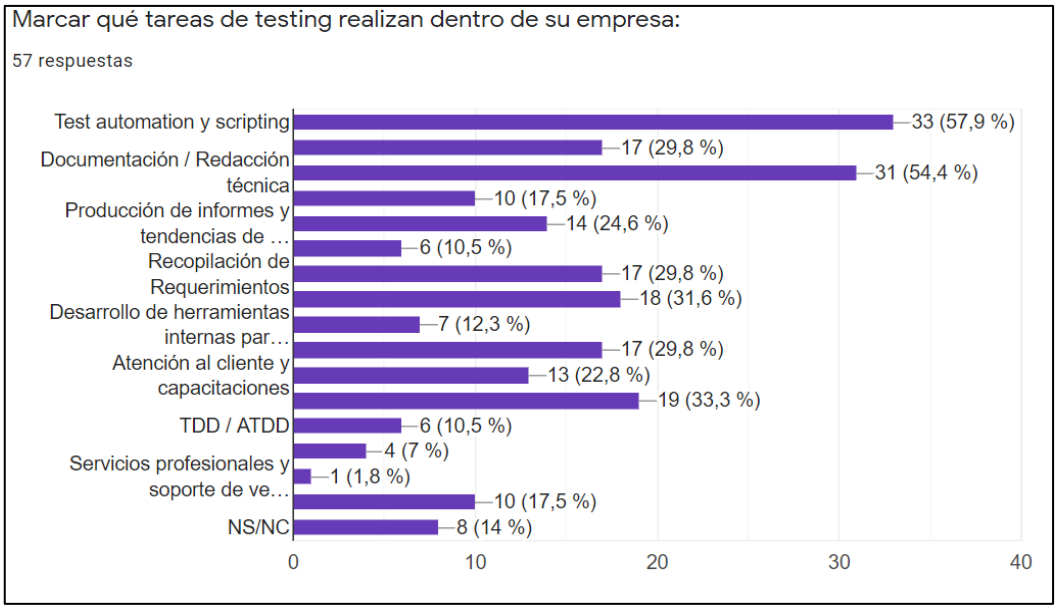

Figura 5-13 Análisis de las tareas de testing que se realizan en las empresas encuestadas

5.4.5 *Uso de herramientas*

En lo relacionado a las herramientas, el 97,5% de los encuestados registra los requerimientos o *Historias de Usuario* en algún tipo de herramienta, ya sea de desarrollo propio, libre o comercial. Sumado a que el 26.1% de los encuestado dijo que varía sus herramientas en función de sus clientes y proyectos

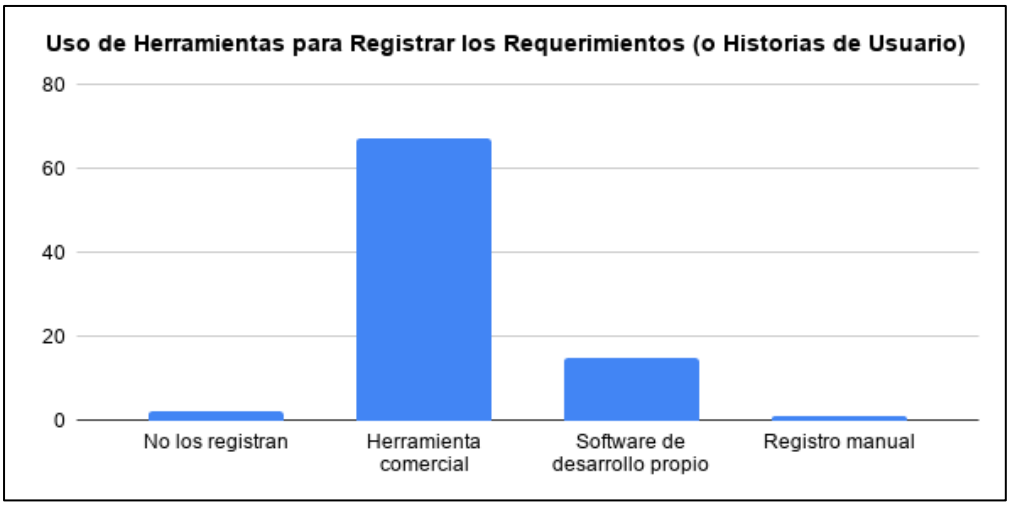

Figura 5-14 Análisis del uso de herramientas para registrar los requerimientos (o historias de usuario)

En base a esto, se puede afirmar que, independientemente si poseen un área o no de análisis funcional y del tamaño de la empresa, estas organizaciones entienden el valor de la documentación y seguimiento de los requerimientos o historias de usuario en su proceso de desarrollo. Dentro de las herramientas comerciales más utilizadas se encuentran *Jira*, *TFS*, *Redmine* y *Trello*

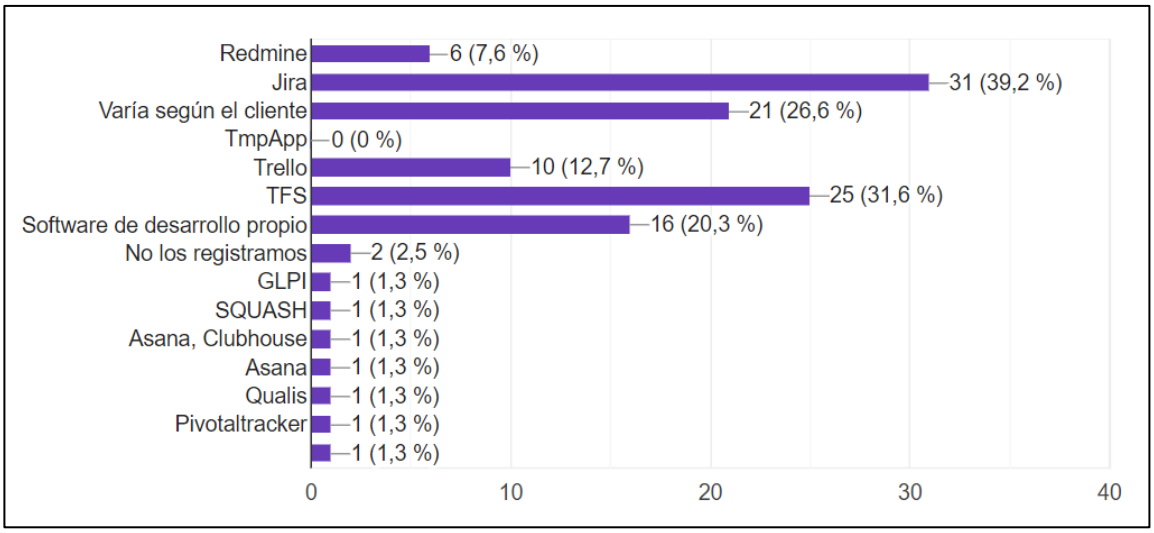

Figura 5-15 Análisis de las herramientas más utilizadas

En cuanto a herramientas para registrar los errores detectados, casi el 40% de los encuestados no usa herramientas específicas para el registro y seguimiento de los *bugs* detectados. Las herramientas no específicas incluyen: mails, Word y Excel.

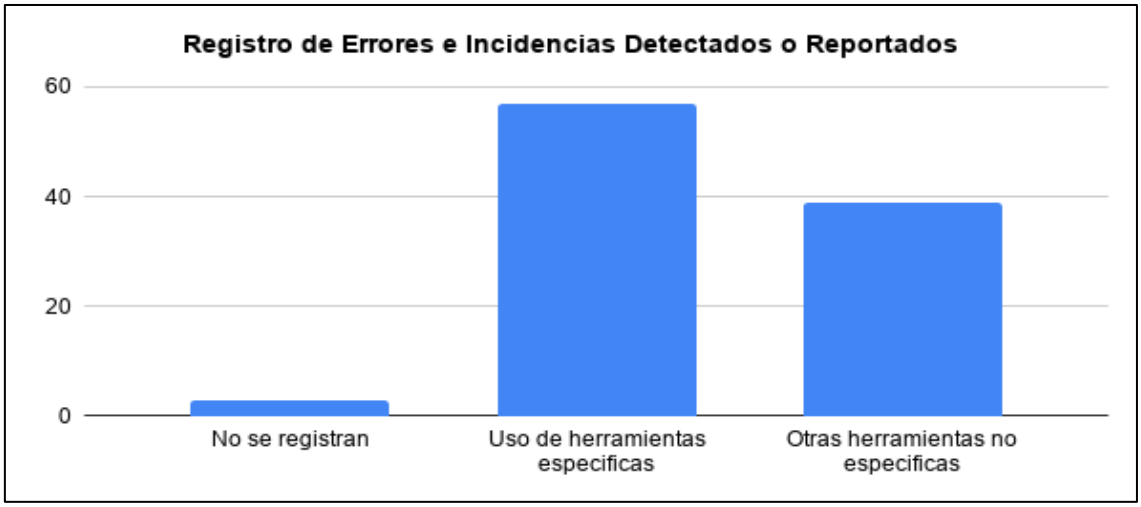

Figura 5-16 Análisis de registro de incidentes y errores

Al igual que para el registro de requerimientos, las aplicaciones más usadas para el registro y seguimiento de los errores detectados son Jira, TFS, Redmine y Trello

5.4.6 *El Rol de las certificaciones*

Por último, con respecto a las certificaciones, se detectó que casi la mitad de los encuestados no posee ninguna y que aquellas específicas de Testing, como por ejemplo la certificación ISTQB, apenas alcanzan al 5% de los encuestados.

Figura 5-17 Análisis de las certificaciones conseguidas

Analizando las empresas que poseen certificaciones conseguidas, se puede destacar que aproximadamente el 75% de ellas: posee área de análisis funcional y un área de testing, casi en su totalidad registran sus requerimientos, los errores detectados, ejecutan pruebas unitarias y planifican sus actividades de pruebas al menos en algunos proyectos. Así como también, ejecutan pruebas funcionales y trabajan con el concepto de "criterio de aceptación", ejecuta más de 20 casos de pruebas por proyecto y utiliza alguna herramienta de test de *automation*.

5.4.7 *Conclusiones*

A partir de la información que se pudo recopilar a través de la encuesta, que concuerda con mi experiencia profesional en mercado, se arribó a las siguientes conclusiones sobre el estado del testing en las PyMEs de la región:

- La mayoría de las empresas están tendiendo al uso de modelos de ciclo de vida ágiles, caracterizados por ciclos de desarrollos cortos y fáciles de adaptar a los cambios de contexto.
- Existe, además, una diversidad de modelos de ciclo de vida que están siendo utilizados para el desarrollo de software. Así como también, un porcentaje considerable de organizaciones que no sigue ningún modelo. Por lo que se puede concluir en este punto, que una buena metodología asociada a las pruebas debe poder adaptarse a cualquier modelo de ciclo de vida utilizado.
- La mayoría de las PyMEs encuestadas realiza sus pruebas de manera informal, caracterizadas por la falta de ejecución de pruebas en los distintos

niveles relevados: Pruebas Unitarias, Pruebas Funcionales y Pruebas de Aceptación. Lo que permite ubicarlas en el nivel más bajo de metodologías como CMMI o TPI.

- Para los proyectos de desarrollo en general, se invierten escasos recursos relacionados a: el tiempo dedicado a las pruebas, la planificación de las mismas y las actividades realizadas.
- Las empresas encuestadas tienen una buena predisposición para el uso de herramientas de registro y seguimiento de requerimientos y errores detectados. Aunque, existe un gran número de ellas que no están usando las herramientas más adecuadas para realizar este tipo de actividades.
- Con respecto a las certificaciones, se puede concluir que el hecho de poseer alguna de ellas, genera un conjunto de actividades y tareas enfocadas a la mejora continua de sus procesos y, en consecuencia, en la calidad del producto final. Sin embargo, en el contexto de las PyMEs, estas certificaciones son muy difíciles de lograr. Los resultados de la encuesta realizada hacen énfasis en la necesidad de un modelo de mejora que sea alcanzable para las PYMEs
- Teniendo en cuenta los puntos anteriores, se puede decir que la mayoría de las organizaciones encuestadas entran en la categoría de organizaciones inmaduras, definida por CMM.

5.5 Evaluación de modelos

Una vez establecida la situación que están atravesando las PYMEs de la industria, nos encontramos en condiciones de poder evaluar los modelos de mejora citados en el capítulo anterior. Para ello, se han definido una serie de criterios en base a lo analizado en este capítulo y lo descrito en capítulos anteriores con respecto a las características propias del proceso de pruebas. Así como también, aportando los conocimientos y la experiencia adquirida al desempeñar tareas en este tipo de industrias. Los criterios de comparación definidos son los siguientes:

- **Licencia:** Un modelo es de licencia Propietario si el usuario tiene limitaciones para usarla, modificarla o redistribuirla y requiere permiso de una organización privada. En cambio, es Abierto si el usuario no tiene limitaciones para usarla o modificarla y se puede redistribuir libremente. En

el contexto de esta investigación, se valoran más aquellas de Licencia Abierta.

- **Tipo**: Puede ser un modelo de mejoras de Procesos cuando se aplica en todo el ciclo de vida del software, puede ser un modelo de mejora de los procesos de prueba exclusivamente o ser un modelo de pruebas de software en sí mismo. Dado el valor que tienen las pruebas y la falta de procesos formales en las PYMEs, se valoran aquellos que sean modelos de mejora de procesos de pruebas.
- **Definición:** Un modelo está Completamente Definido si todas las prácticas descriptas en el mismo están completamente especificadas para llevarse a acabo. Debe incluir todas actividades y tareas. Decimos que está Parcialmente Definido si incluye algunas de las prácticas necesarias y Descriptivo cuando sólo incluye qué es lo que se debe realizar sin más detalles. Dadas las restricciones de recursos que atraviesan las PYMEs, se valoran aquellas que poseen mayor nivel de definición.
- **Acoplado a un ciclo de vida específico:** identifica qué tan dependiente es el modelo estudiado a un determinado ciclo de desarrollo de software. Ya que la encuesta reveló la variedad de modelos de ciclo de vida que se utilizan actualmente, cuanto menos acoplado al ciclo de vida se encuentre el modelo, más accesible es para este sector de la industria.
- **Experiencia:** este aspecto se enfoca a si se necesitan perfiles con cierta formación especial para poder poner en marcha la metodología, tales como consultores o evaluadores externos. Debido a las restricciones presupuestarias detectadas por el relevamiento del CESSI, son aplicables a PYMEs aquellas que prescinden de personal externo para su aplicación.
- **Incorpora técnicas de pruebas:** La propuesta incluye procedimientos técnicos y de gestión que ayudan a desarrollar las actividades de pruebas en los diferentes niveles. Nuevamente, ya que el sector de la industria que estamos estudiando se caracteriza por llevar a cabo estas actividades informalmente, la incorporación de técnicas de prueba en los diferentes niveles puede mejorar sustancialmente los productos desarrollados en estas organizaciones
- **Incorpora herramientas de soporte para su implementación:** con este

criterio, se pretende destacar aquellos modelos de mejora que cuentan con una herramienta que acompañe y soporte el proceso de implementación de la metodología dentro de la organización. Teniendo en cuenta que, en función a la encuesta realizada, las organizaciones que estamos analizando han incorporado este tipo de herramientas de gestión en su operatoria diaria.

En función a los criterios anteriormente detallados, se elaboró una matriz de comparación (Tabla 5-2) con los diferentes modelos que fueron desarrollados en el capítulo anterior. Se incorporaron, además, las normas de la familia ISO que fueron citadas en el capítulo 3, ya que se entienden que estas también son un camino posible para la mejora de los procesos.

Tal como se puede observar, ninguno de los modelos analizados cumple con los criterios deseados. En conclusión, ninguno de los modelos que existe actualmente se adapta de forma eficiente a la realidad de las PYMEs de nuestra región.

Es en este punto en particular, donde se destaca la necesidad de un modelo que permita la mejora de los procesos en este sector de la industria y, como consecuencia, en los productos que desarrollan. En el siguiente capítulo, se realiza una propuesta que abarca tanto las limitaciones de los modelos actuales como las oportunidades que se detectaron en su contexto.

	CMM/CMMi	TMM/TMMi	TPI	Tmap/Tmap Next	IT Mark	COMPETISOFT	ISO/IEC 29110	ISO/IEC 29119
Licencia	Propietario	Propietario	Propietario	Propietario	Propietario	Abierto	Abierto	Abierto
Tipo	Modelo de mejora de procesos	Modelo de mejora del proceso de pruebas	Modelo de mejora del proceso de pruebas	Modelo de mejora del proceso de pruebas	Modelo de mejora de procesos	Modelo de mejora de procesos	Modelo de mejora de procesos	Modelo de mejora del proceso de pruebas
Definición	Parcialmente Definido	Parcialmente Definido	Parcialmente Definido	Completamente Definido	Parcialmente Definido	Descriptivo	Descriptivo	Descriptivo
Acoplado a un ciclo de vida específico	No	No	No	Si	No	Si	No	No
Experiencia	Muy necesaria	Muy necesaria	Muy necesaria	Muy Necesaria	Muy Necesaria por falta de documentación	No	Necesaria la formación en certificaciones ISO	Necesaria la formación en certificaciones ISO
Incorpora técnicas de pruebas	No	No	No	Si	No	No	No	No
Incorpora herramientas de soporte para su implementación	No	No	No	Si	No	Si	No	No

Tabla 5-2 - Comparación de Modelos de Mejora de Procesos, elaboración propia

CAPÍTULO 6: LA PROPUESTA

Tal como se ha de analizado a lo largo de este trabajo, definir un proceso de pruebas dentro de una organización puede ser valioso en la medida en que permita: (i) incrementar la calidad del producto, (ii) facilitar la comunicación y la comprensión entre los miembros del equipo, (iii) apoyar la mejora continua del proceso, y (iv) soportar la ejecución automática de ciertas tareas (Rojas-Montes et al., 2015).

En este sentido y en función a los diferentes conceptos desarrollados, principalmente luego de contextualizar el nivel de madurez de las organizaciones de la región, se propone un proceso liviano, definido para guiar y apoyar la realización de las pruebas en PyMEs desarrolladoras de software. Esta propuesta, además, incorpora diversas técnicas con el objetivo de realizar las pruebas de forma organizada y sistemática, especialmente en lo referido al diseño de las mismas y la forma de llevarlas a cabo, con el fin de subsanar las limitaciones de los modelos evaluados en el capítulo anterior, tomando lo mejor de cada uno de ellos, con el agregado de la experiencia adquirida por desempeñar actividades en organizaciones de este tipo.

6.1 Introducción

Primeramente, tomaremos como punto de partida las características deseadas en base a los criterios definidos para la evaluación de modelos en el capítulo 4:

- Licencia abierta.
- Modelos de mejora de procesos de pruebas.
- Completamente definido.
- No acoplado a un ciclo de vida específico
- Sin experiencia previa necesaria.
- Incorpora técnicas de prueba en los diferentes niveles
- Incorpora herramientas de soporte para su implementación

A continuación, se detallan las implicancias de cada uno de estos puntos en la propuesta. Es aquí donde se justifica la estructura y los objetivos específicos de la misma.

6.1.1 *Licencia Abierta*

Se propone un modelo abierto que sea accesible para las PYMEs, ya que, en base

a los estudios realizados, las limitaciones económicas son un factor predominante en este tipo de industrias.

6.1.2 *Modelos de mejora de procesos de pruebas*

En función a la investigación realizada sobre el impacto de una correcta definición de los procesos de prueba, la propuesta se encuentra clasificada como un Modelo de Mejora de Procesos de Pruebas.

6.1.3 *Completamente definido*

Tal como se explicó en el capítulo anterior, que se encuentre completamente definido implica que exista un gran nivel de detalle en cuanto a las actividades y tareas que se deben realizar. Es por eso que en esta propuesta se define una serie de ejes divididos en niveles, que organizan estas actividades y tareas. Además, se definen los roles que intervienen en cada una.

6.1.4 *No acoplado a un ciclo de vida específico*

Las actividades y tareas son genéricas y no pertenecen a un modelo particular de ciclo de vida de desarrollo.

6.1.5 *Sin experiencia previa necesaria*

En este punto, se define un proceso sencillo representado en un diagrama de flujo que permite orientar al equipo que llevará a cabo el proceso de mejora. Adicionalmente, el hecho de organizarlo en niveles evolutivos y ejes temáticos, permite que no resulte de gran complejidad.

6.1.6 *Incorpora técnicas de prueba en los diferentes niveles*

Considerando todo lo analizado a lo largo del Capítulo 3 sobre el impacto de los diferentes niveles de pruebas en el producto final, se determinó que la metodología debía tener en cuenta: pruebas unitarias y de integración, pruebas funcionales, pruebas de sistema y pruebas de aceptación. Estos tipos de pruebas que se consideran necesarias para dotar de cierta calidad al producto final son condición de cobertura para llevar a cabo el proceso de mejora propuesto.

6.1.7 *Incorpora herramientas de soporte para su implementación*

A su vez, para cada uno de los ejes y niveles, se propone una selección de herramientas que permiten acompañar y dar seguimiento al proceso de pruebas.

6.2 Metodología Propuesta

En esta sección, se realiza una breve introducción sobre la estructura de la metodología y se detallan los elementos fundamentales que componen la propuesta: los ejes, los niveles y las herramientas de apoyo.

Primeramente, para poder conducir el camino de la mejora del proceso de pruebas, tal como se indicó en el punto de especificación, se determinaron una serie de ejes que tienen un doble objetivo: de diagnóstico y de dirección de las mejoras. Tienen un objetivo de diagnóstico ya que, al estar divididos por niveles, podemos evaluar en qué grado de evolución se encuentra cada uno dentro de la organización, en función de las características establecidas. Por otro lado, estos ejes definen la dirección hacia donde se van a realizar las mejoras, dado que una vez que se diagnostica en qué nivel se encuentra, establecen los objetivos que conducen las actividades para posicionarse en el siguiente nivel.

En base al análisis realizado en el capítulo 5, donde se establecieron las condiciones actuales que atraviesan las PyMES sobre las cuales debe poder desarrollarse esta metodología, los ejes se encuentran divididos en tres niveles: Sin implementación, Implementación Parcial e Implementación Completa. El hecho de que sólo sean tres niveles permite realizar un diagnóstico rápido, hacer foco en temas específicos y realizar ciclos cortos de mejora que sean más fáciles de administrar y gestionar.

Figura 6-1 Niveles de evolución de los Ejes

La estructura en niveles es una de las cuestiones que ha permitido que muchos de los modelos de mejora (analizados en el capítulo 4) tengan amplia aceptación a nivel global, dado que permite realizar cambios de forma progresiva y organizada. Cabe destacar que, en el caso de los modelos de mejora mencionados, su estructura suele ser de 5 niveles o más, lo que resulta muy complejo de llevar a cabo cuando se realizan en entornos informales como los de las organizaciones que estamos estudiando. Es por eso, que esta aproximación de 3 niveles resulta superadora para este tipo de empresas, dado que nace a partir de sus necesidades

puntuales y reacondiciona estructuras probadas en grandes empresas para tener éxito. Adicionalmente, al llevarla a cabo, permite sentar las bases necesarias para que si en un futuro se desea obtener alguna certificación reconocida internacionalmente, pueda realizarse de una forma más rápida y con la experiencia previa aportada por esta metodología.

6.2.1 *Ejes y Niveles*

Los ejes que se proponen a continuación se fundamentan en la investigación realizada a lo largo de este trabajo, tanto el impacto que tienen estos aspectos sobre el producto final, como en la definición de cada uno de los niveles para que puedan ser alcanzados por la PyMEs del sector.

Definición del Plan:

Los procesos de prueba son muy visibles porque tanto los miembros del equipo de desarrollo, como la gerencia y los clientes, están interesados en conocer el nivel de calidad actual del sistema y su tendencia. La planificación implica actividades que definen tanto los objetivos de las pruebas como el enfoque para cumplir dichos objetivos dentro de las restricciones impuestas por el contexto.

La planificación depende de la política y la estrategia de prueba de la organización, los ciclos de vida de desarrollo y los métodos utilizados, el alcance de la prueba, los objetivos, los riesgos, las restricciones, la criticidad, la capacidad de ser probado, y la disponibilidad de los recursos. Sin embargo, como una primera aproximación para planificar y establecer un cronograma dentro del presupuesto, se debe preparar un plan de pruebas, que luego se va a ejecutar, se va a rastrear su progreso y sobre el cual se van a tomar medidas correctivas y preventivas en el caso de ser necesarias.

Dicho plan debe incluir información sobre la base de prueba que genera un punto importante en la trazabilidad en el proceso, así como los criterios que se utilizarán durante la monitorización y el control. Las actividades de planificación pueden incluir las siguientes y algunas de ellas se pueden documentar en un plan de prueba:

- Determinar el alcance y los objetivos de las pruebas, es decir, qué probamos, qué no probamos y para qué lo hacemos

- Definir el enfoque general de las pruebas
- Integrar y coordinar las actividades de prueba con las actividades del ciclo de vida de desarrollo
- Las personas y recursos estimados para realizar las diversas actividades
- El calendario para llevar a cabo las pruebas ya sea en fechas particulares (por ejemplo, en los modelos de ciclo de vida en cascada) o en el contexto de cada iteración (por ejemplo, en el desarrollo iterativo)
- Un presupuesto estimado para las actividades de prueba

El nivel de detalle y la forma en que se comunica este plan es el que determinará el nivel de evolución sobre el eje específico. Si no se realizan las actividades de planificación, decimos que no existe un plan. Hablamos de Plan Informal cuando una o más actividades y decisiones de planificación se realizan o se asumen, pero no se encuentran formalizadas en su totalidad, entendiendo por formalizado que se encuentra documentado y distribuido a todos los involucrados en el proyecto. Por último, hablamos de planificación formalizada o Plan Formal cuando se cumplen las condiciones de formalización anteriores, es decir, existe un plan escrito en un papel o alguna herramienta de seguimiento que se distribuyó y es conocido por los involucrados.

En la práctica, se puede documentar en un plan maestro de prueba y en planes de prueba separados para los diferentes niveles de prueba, como las pruebas de sistema y las pruebas de aceptación, o por tipos de prueba separados, como la prueba de usabilidad y la prueba de rendimiento. Cabe aclarar que no es necesario que la documentación del plan sea demasiado compleja, simplemente debe reflejar los resultados de las actividades detalladas arriba. El contenido de los planes varía, y puede extenderse más allá de los temas previamente identificados. En la Figura 6-2, podemos ver un ejemplo de documentación del plan de pruebas, similar al que propone la norma ISO/IEC/IEEE 29119-3.

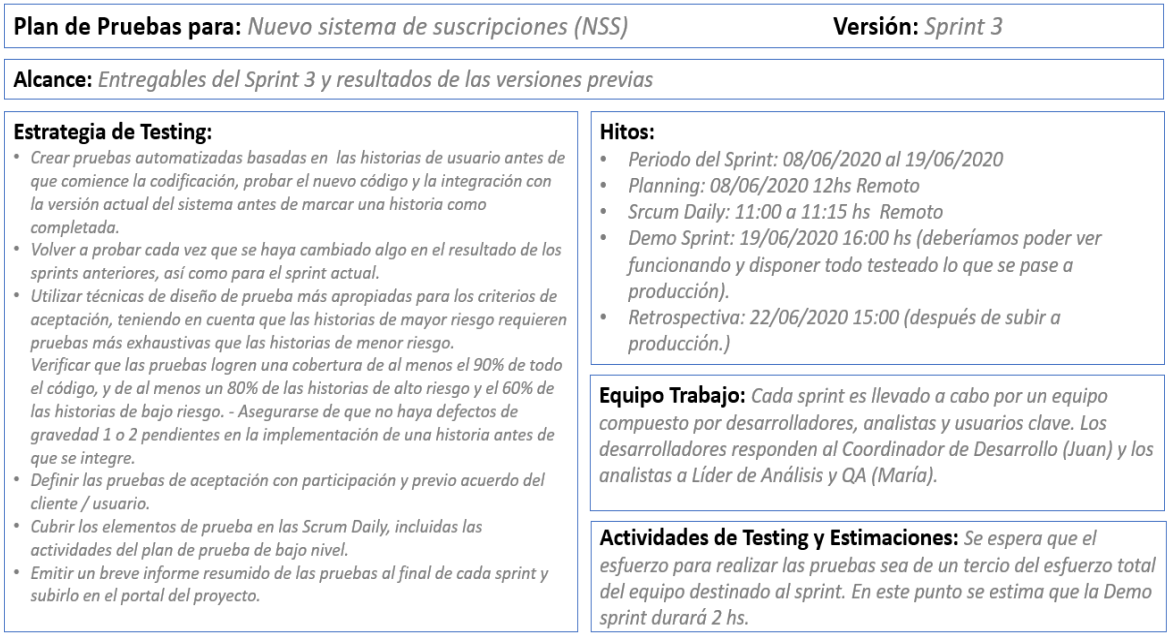

Figura 6-2 Ejemplo de Plan de Pruebas, elaboración propia.

Idealmente, como lo proponen los modelos de prueba ampliamente aceptados, la planificación de las pruebas debe comenzar simultáneamente con la obtención de requerimientos. Las interacciones más rápidas y cercanas entre los desarrolladores y quienes estén encargados de las pruebas, conducen a una mejor comprensión de las necesidades de prueba en términos de recursos y desafíos. A su vez, a medida que el proyecto y la planificación de las pruebas evolucionan, se dispone de más información y se pueden incluir más detalles en el plan. La planificación de la prueba es una actividad que debe realizarse de forma continua a lo largo de todo el ciclo de vida del producto. La retroalimentación de las actividades de prueba que se llevan a cabo se debe utilizar para detectar cambios en los riesgos, de tal forma que la planificación pueda ser ajustada.

Por último, se debe tener en cuenta que a medida de que los planes son funcionales a los objetivos de la organización, institucionalizan ciertas cuestiones que pueden ser reutilizadas y adaptadas a nuevos proyectos.

Niveles de Prueba:

Los procesos de prueba en una organización no pueden centrarse solo en las pruebas de nivel de sistema. Antes de que un sistema completo esté sujeto a los rigores que deben tener las pruebas a este nivel, deben realizarse pruebas a nivel de unidad y nivel de integración. Si no se realizan este tipo de pruebas previamente,

no se pueden realizar pruebas significativas de sistema. La eficacia de las pruebas de bajo nivel da paso a que los defectos se detecten y corrijan mucho más temprano durante el desarrollo. Lo mismo sucede con los niveles superiores, ya que, idealmente, una vez realizadas las pruebas de sistema, se deben realizar pruebas de aceptación y de requerimientos no funcionales (como, por ejemplo, de rendimiento).

Para determinar los niveles afectados, se debe analizar qué pruebas se deben hacer. Para eso, se recomienda realizar las siguientes actividades:

- Analizar la base de prueba disponible con sus respectivos objetos, asociados a los diferentes niveles de pruebas que pueden considerarse. En el *Anexo II: Relaciones entre bases, objetos y niveles de prueba*, se pueden observar las relaciones entre bases, objetos y niveles que propone ISTQB.
- Identificar las prestaciones y conjuntos de prestaciones que se van a probar.
- Definir y priorizar las condiciones de prueba para cada nivel.

Es probable que, en la mayoría de los casos, las PyMEs no posean sus bases de pruebas completas. Lo importante es insumir el tiempo de pruebas de forma inteligente, cubriendo la mayor cantidad de niveles, en el orden en que las pruebas puedan detectar desvíos en forma temprana, teniendo en cuenta la trazabilidad de los requerimientos, las prestaciones y las condiciones de prueba.

Durante este análisis, el hecho de tener que detectar qué es lo que se va a probar, permite llevar a cabo un proceso de revisión que es muy útil si no existen otras actividades de revisión en el proceso. En la Figura 6-3, se propone un ejemplo para ordenar los puntos identificados que pueden formar parte del plan de pruebas.

En este punto, el nivel de evolución del eje que se analiza está en función de los niveles de pruebas que son cubiertos.

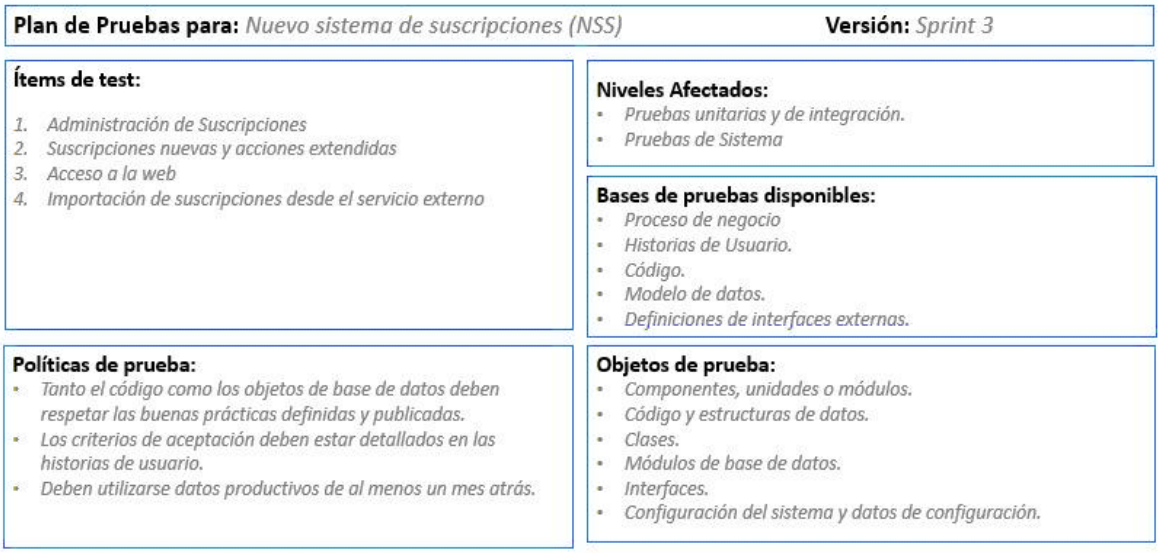

Figura 6-3 Ejemplo de análisis de niveles afectados, elaboración propia.

Definición de Casos de prueba:

Así como en los niveles afectados se debe determinar "qué probar", la definición de casos de prueba responde a la pregunta "cómo probar". Esta actividad involucra:

- Diseñar los casos de prueba, identificando los datos de entradas posibles, las condiciones de ejecución y los resultados esperados.
- Priorizarlos.
- Especificarlos y registrarlos para su seguimiento.

El hecho de aplicar una técnica de especificación de prueba permite realizar de forma adecuada esas actividades. Existen múltiples técnicas de especificación de prueba que permiten derivar casos a partir de un objeto de prueba. Idealmente, se pretende que cada caso de prueba se pueda trazar bidireccionalmente hasta la(s) condición(es) de prueba que cubre.

A menudo, es una buena práctica diseñar casos de prueba de alto nivel, sin valores concretos para los datos de entrada y los resultados esperados (también conocidos como checklists). Estos casos de prueba de alto nivel son reutilizables a lo largo de múltiples ciclos de prueba con diferentes datos concretos, sin dejar de lado el alcance del caso de prueba. En la Anexo I, también se pueden observar una serie de fallos comunes, en los diferentes niveles, que se pueden utilizar como punto de partida para derivar casos de prueba de alto nivel. En la Tabla 6-1 se propone una primera aproximación a la especificación de casos de prueba que, además, incorpora algunas columnas para el seguimiento de estos.

Cabe destacar que la generación de casos de prueba, además, da lugar a la identificación de otras necesidades de manera temprana, como son la infraestructura, los datos y las herramientas necesarias.

Este eje, se encuentra en el nivel más bajo cuando no se lleva a cabo la generación de casos de pruebas. Se encuentra en un nivel de implementación parcial cuando se identifican casos de prueba de alto nivel, pero carecen de una especificación completa. Por último, se encuentra en el nivel más alto cuando los casos de prueba tienen una especificación completa de sus entradas, condiciones de ejecución y resultados esperados.

CP Nº	Caso de Prueba	Detalle	Resultado Esperado	Resultado Obtenido	Obs.	Estado
1	*El usuario de Atención al Cliente puede crear un nuevo tipo de suscripción*	*1 - El usuario debe poder ingresar el nombre, los precios asociados y los comentarios para un nuevo tipo de suscripción* *2 - El usuario debe poder almacenar el nuevo tipo de suscripción*	*Nuevo tipo de Suscripción*			*En proceso*
2	*El usuario de Atención al cliente debe poder consultar y modificar un tipo de suscripción existente*	*1- El usuario debe poder ver un tipo de suscripción existente* *2 – El usuario puede cambiar el nombre y los precios asociados para un tipo de suscripción, siempre y cuando no exista una suscripción de ese tipo* *3 – El usuario puede cancelar el cambio de un tipo de suscripción antes de que se almacene* *4 – El usuario puede almacenar los cambios en un tipo de suscripción.*	*Tipo de Suscripción modificada*	*Suscripción no modificada*		*Error*
3						*Completo*
4						*Pendiente*

Tabla 6-1 Ejemplo de Especificación de Casos de Prueba, elaboración propia.

Tipos de Actividades de prueba:

En un sentido amplio, tal como se ha detallado en el capítulo 3, hay dos tipos de actividades de prueba: estáticas y dinámicas. Las pruebas dinámicas implican la ejecución real de código, mientras que las pruebas estáticas evalúan el código u otro producto de trabajo que se esté probando sin ejecutarlo de forma efectiva. Ambos tipos de pruebas se complementan entre sí al encontrar diferentes tipos de defectos.

Para el caso de las pruebas dinámicas, es recomendable utilizar herramientas de ejecución de pruebas, lo que a menudo se denomina automatización de la prueba. Los beneficios potenciales del uso de las mismas incluyen: la reducción del trabajo manual repetitivo, ahorro de tiempo, mayor consistencia, objetividad y repetibilidad. En el apartado de herramientas, se detallan algunos ejemplos.

Las pruebas estáticas se basan en la evaluación manual de los productos de trabajo (es decir, revisiones) o en la evaluación basada en herramientas del código u otros productos de trabajo (es decir, el análisis estático). Casi cualquier producto de trabajo puede ser examinado mediante una prueba estática, por alguna o ambas opciones, revisiones y análisis estático, por ejemplo: Especificaciones de requerimientos funcionales, de seguridad, arquitectura y diseño; historias de usuarios y criterios de aceptación; Código; Productos de prueba, incluyendo planes de prueba, casos de prueba.; Guías de usuario; Contratos, planes de proyecto, calendarios y presupuestos; Modelos, tales como diagramas de actividad; entre otros.

Las revisiones se pueden aplicar a cualquier producto de trabajo que los participantes sepan cómo leer y comprender. El análisis estático puede ser aplicado eficientemente a cualquier producto de trabajo con una estructura formal (típicamente código o modelos) para el cual exista una herramienta de análisis estático apropiada. El análisis estático, incluso, se puede aplicar con herramientas que evalúan los productos de trabajo escritos en lenguaje natural, como los requisitos (por ejemplo, la corrección ortográfica, la gramática y la legibilidad).

A su vez, la prueba estática, independiente del tipo, y la prueba dinámica pueden tener los mismos objetivos, tales como proporcionar una evaluación de la calidad de los productos de trabajo e identificar los defectos tan temprano como sea posible. Las dos formas de prueba tienen sus propias ventajas y limitaciones. Sin embargo, lo que está claro es que existe la necesidad de realizar ambas formas de prueba para que la prueba general sea efectiva y económica. Es por eso, que en este eje se evalúa el nivel de evolución en función de la capacidad de realizar ambos tipos de prueba.

Trazabilidad:

En cualquier proceso que posea un cierto grado de complejidad para el cual se desarrolló un plan, deben incluirse actividades de monitorización y control. La monitorización (o seguimiento) en el contexto de las pruebas, implica la comparación continua del avance real con respecto al plan de prueba. Por otro lado, el control de la prueba implica tomar las medidas necesarias para cumplir los objetivos del plan de prueba (que puede actualizarse con el tiempo). La

monitorización y el control de la prueba se apoyan en la evaluación de los criterios de salida, por ejemplo, para la ejecución de la prueba como parte de un nivel de prueba dado puede incluir actividades como: Comprobar los resultados de la prueba en relación con los criterios de cobertura especificados, evaluar el nivel de calidad de los componentes o sistemas en base a los resultados obtenidos, y finalmente, determinar si se necesitan más pruebas.

Para implementar una monitorización y control efectivos de la prueba, es importante establecer y mantener la trazabilidad a lo largo del proceso de prueba entre cada elemento de la base de prueba y los diversos productos de trabajo de la prueba asociados con ese elemento. Además de la evaluación de la cobertura de las pruebas, una buena trazabilidad permite:

- Analizar el impacto de los cambios.
- Hacer que la prueba pueda ser auditada.
- Mejorar la comprensión de los informes de avance de la prueba y de los informes resumen de prueba para incluir el estado de los elementos de la base de prueba (por ejemplo, los requisitos que pasaron sus pruebas, los requisitos que fallaron sus pruebas, y los requisitos que tienen pruebas pendientes).
- Relacionar los aspectos técnicos de la prueba con los implicados en términos que éstos puedan entender.
- Aportar información para evaluar la calidad de los productos, la capacidad de los procesos y el avance de los proyectos en relación con los objetivos de negocio.

En modelos como los que propone UML, se realizan matrices de trazabilidad entre los casos de uso, los casos de pruebas y los requerimientos utilizando una codificación específica, que en algunos casos es muy difícil de mantener. Sin embargo, en la actualidad existen muchas herramientas que resuelven estas cuestiones. En este sentido, el nivel de trazabilidad entre las bases de prueba y los productos de trabajo es lo que va a determinar la evolución de este eje.

Entorno de pruebas:

Idealmente, las pruebas a nivel del sistema deben llevarse a cabo en el entorno operativo del sistema. Sin embargo, en la mayoría de los casos esto no es posible, ya que es muy costoso o poco práctico o, si el sistema es de alta disponibilidad y se encuentra operativo, es casi imposible. Por lo tanto, los entornos de prueba deben ser tan cercanos a la realidad como sea posible. Dentro de las actividades que se deben llevar a cabo para la generación de un ambiente adecuado de pruebas, se incluyen:

- Construir el entorno de prueba (incluyendo, posiblemente, arneses de prueba, virtualización de servicios, simuladores y otros elementos de infraestructura) y verificar que se haya configurado correctamente todo lo necesario.
- Preparar los datos de prueba y asegurarse de que estén correctamente cargados en el entorno de prueba.

Hay que tener en cuenta que se necesitan diferentes entornos de prueba para realizar diferentes categorías de pruebas, ya que no es lo mismo realizar una prueba de rendimiento que una de instalación. Sin embargo, aunque se cuente con un único entorno dedicado para las pruebas, lo más importante es que sea tan parecido al entorno operativo como sea posible. Si los entornos de prueba difieren significativamente de los entornos operativos, carece de sentido detectar fallos bajo esas condiciones. Es en este punto donde se evalúa la evolución de este eje, considerando: el nivel más bajo cuando no existe un entorno específico para realizar las pruebas; de nivel intermedio cuando existe un entorno dedicado a las pruebas gestionado y controlado, pero no se encuentra completo (ya sea porque faltan servicios o no se encuentran configurados correctamente, faltan interacciones con otros módulos o sistema, o bien la base de datos no se encuentra actualizada con datos productivos) y el nivel superior cuando es el adecuado, es decir, cuando se asemeja en su totalidad al entorno productivo.

Herramientas de Pruebas:

Las herramientas se pueden utilizar de múltiples formas para apoyar una o más actividades de prueba del proceso:

- Herramientas que se utilizan directamente en la prueba, como herramientas de ejecución de la prueba y herramientas de preparación de datos de prueba.
- Herramientas que ayudan a gestionar requisitos, casos de prueba, procedimientos de prueba, guiones de prueba automatizados, resultados de prueba, datos de prueba y defectos, así como a informar y monitorear la ejecución de la prueba.
- Herramientas que se utilizan para la investigación y la evaluación.
- Cualquier herramienta que asista en la prueba (una hoja de cálculo es también una herramienta de prueba en este sentido).

Dentro de los beneficios del uso de herramientas podemos nombrar: mejorar la eficiencia de las actividades de prueba mediante la automatización de tareas repetitivas o que requieren recursos significativos cuando se realizan manualmente (por ejemplo, ejecución de pruebas, prueba de regresión), mejorar la eficiencia dando apoyo a las actividades de prueba manuales a lo largo de todo el proceso de prueba, mejorar la calidad al permitir pruebas más consistentes y un mayor nivel de reproducibilidad de los defectos, automatizar actividades que no se pueden ejecutar manualmente (por ejemplo, pruebas de rendimiento a gran escala), aumentar la fiabilidad de las pruebas (por ejemplo, automatizando las comparaciones de grandes cantidades de datos o simulando el comportamiento), entre otras. En general, el uso de herramientas hace que la actividad sea menos propensa a errores. Sin embargo, la simple adquisición de una herramienta no garantiza el éxito. Cada nueva herramienta introducida en una organización requiere un esfuerzo para lograr beneficios reales y duraderos.

Aunque más adelante en este trabajo se detalla un apartado para la selección de herramientas de soporte para cada eje, este eje en particular pretende evaluar el nivel en que los diferentes tipos de herramientas se encuentran integrados en el proceso en general. Es por eso que se determina que se encuentra en el nivel más bajo si incorpora herramientas no específicas como hojas de cálculo o mails tanto para la gestión como el seguimiento de sus pruebas. Se considera de nivel medio si utiliza herramientas específicas para la ejecución y seguimiento de las mismas y, se encuentra en el nivel superior si posee una automatización extendida del proceso. Nos referimos a automatización extendida, cuando poseen un conjunto de

herramientas que permiten una gestión de punta a punta, es decir, cuando existen herramientas que dan soporte a todos los ejes del proceso.

Feedback:

Mientras se lleva a cabo el proyecto, existe una serie de hitos que son importantes para la recopilación de datos de las actividades de prueba que se llevaron a cabo en el transcurso del mismo. Algunos de ellos pueden ser: la liberación de un software, un proyecto de prueba es completado (o cancelado), cuando finaliza una iteración de un proyecto Ágil (por ejemplo, como parte de una reunión retrospectiva), cuando se completa un nivel de prueba, o cuando se completa la liberación de un mantenimiento. Esta recopilación permite consolidar la experiencia, los productos de prueba y cualquier otra información relevante. Además, es el primer paso para poder establecer métricas.

Existen diferentes tipos de feedback, reportes o informes involucrados en un proceso de prueba, entre ellos, los informes internos y los informes externos. Informar defectos es una actividad interna. Un ejemplo de este tipo de informes se puede ver en la Tabla 6-2, que se apoya en la tabla de casos de prueba de la Tabla 6-1 Ejemplo de Especificación de Casos de Prueba, elaboración propia. Por otro lado, informar la calidad del sistema bajo prueba a los clientes, se incluyen en informes externos. Al proporcionar reportes precisos y oportunos, tanto de forma interna como externa, es posible detectar desvíos de manera temprana, así como también patrones de fallos que conduzcan al defecto a corregir. Además, proporcionar feedback del proceso y de las actividades contribuye a construir buenas relaciones ya que generan un espacio de diálogo que permite tener visibilidad del progreso y conducen hacia la mejora continua. Finalmente, es esencial llevar a cabo la gestión de acciones de mejora para todo el proceso.

Casos de Pruebas: *Nuevo sistema de suscripciones (NSS)*	
Versión: *Sprint 3*	
Cant. Casos Prueba:	*10*
Cant. Casos Prueba Pendientes:	*5*
Cant. Casos Prueba en Curso:	*1*
Cant. Casos Prueba Finalizados:	*3*
Cant. Casos Prueba con Error:	*1*

Tabla 6-2 Ejemplo de Reporte de ejecución de casos de pruebas.

6.2.2 *Matriz de Diagnóstico y Matrices de Evolución*

A modo de resumen y para facilitar el diagnóstico, se derivó la siguiente matriz en base a lo analizado anteriormente (Tabla 6-3). La misma permite establecer el nivel de evolución para cada uno de los ejes definidos.

De forma complementaria, se elaboró una matriz para cada uno de los niveles donde se detalla cuáles son las actividades de mejora a llevar a cabo para evolucionar al siguiente nivel de cada eje. Se debe tener en cuenta que, si bien los niveles de evolución se corresponden con situaciones que se dan en las PyMEs, los ejes funcionan de manera independiente. La Tabla 6-4 corresponde a la evolución desde el nivel sin implementación, mientras que la Tabla 6-5 corresponde a la evolución desde la implementación parcial. Cabe destacar, que es usual que una organización tenga algunos ejes más desarrollados que otro u otros, por lo que se debe recurrir a ambas matrices para trabajar su plan de mejoras.

	Sin Implementación	**Implementación Parcial**	**Implementación completa**
Definición del Plan	No existe un Plan	Plan Informal	Plan Formalizado
Niveles de Prueba afectados	Sin Pruebas	Pruebas Unitarias y de sistema	Pruebas unitarias y de Integración y Niveles superiores
Definición de casos de pruebas	Sin casos de pruebas	Casos de prueba de Alto Nivel	Casos de prueba especificados de forma completa
Tipos de Actividades de prueba	Sin pruebas	Pruebas Estáticas o bien dinámicas	Pruebas Estáticas y Dinámicas combinadas
Trazabilidad	Sin mantener trazabilidad	Trazabilidad parcial	Trazabilidad completa
Entorno de Pruebas	Sin entorno específico de Pruebas	Entorno gestionado y controlado	Testing en el ambiente adecuado
Herramientas de Pruebas	Herramientas no específicas	Herramientas específicas de seguimiento y ejecución de pruebas	Automatización extensiva del proceso
Feedback	Sin Reportes	Reportes de defectos internos	Reportes de Progreso, Actividades y de defectos prioritarios

Tabla 6-3 Matriz de diagnóstico de ejes, elaboración propia.

Adicionalmente, en la siguiente sección, se recomiendan un conjunto de herramientas que soportan y conducen el proceso de implementación de actividades en estos ejes; permitiendo que las mejoras se realicen de forma más

ágil ya que resuelven muchas cuestiones de antemano, generando una estructura común y más cercana a las requeridas por las certificaciones internacionales.

	Sin Implementación	Actividades sugeridas para mejorar
Definición del Plan	No existe un Plan	● Responder a las preguntas: ¿Qué probamos? ¿Qué no probamos? y ¿para qué lo hacemos? ● Definir el enfoque general de las pruebas. ● Generar un calendario de alto nivel para llevar a cabo las pruebas. Cuándo comienzan, cuando terminan y los hitos más importantes.
Niveles de Prueba afectados	Sin Pruebas	● Analizar la base de prueba disponible con sus respectivos objetos, asociados al nivel de pruebas unitarias y pruebas de sistema. ● Identificar las prestaciones y conjuntos de prestaciones que se van a probar asociados a los objetos analizados previamente. ● Identificar además las condiciones en las que se va a probar
Definición de casos de pruebas	Sin casos de pruebas	● Generar casos de prueba de alto nivel, sin valores de entrada ni condiciones de ejecución. A menudo es recomendable un checklist como una primera aproximación.
Tipos de Actividades de prueba	Sin pruebas	● Realizar pruebas estáticas de revisión, por ejemplo, la revisión del código, la inspección, walkthroughs, el análisis de algoritmos y las revisiones de documentación funcional. ● Realizar pruebas dinámicas exploratorias de forma manual.
Trazabilidad	Sin mantener trazabilidad	● Mantener trazabilidad al menos entre los objetos de prueba y los casos de prueba.
Entorno de Pruebas	Sin entorno específico de Pruebas	● Construir un entorno de pruebas que permita implementar los diferentes desarrollos y probarlos por fuera del entorno de desarrollo y del productivo. ● Preparar los datos de prueba, en lo posible, que sean similares a los productivos.
Herramientas de Pruebas	Herramientas no específicas	● Incorporar herramientas de seguimiento de requerimientos e incidencias. Como primera aproximación, se pueden llevar en una hoja de cálculo, pero es aconsejable utilizar herramientas específicas para este tipo de tareas. ● Incorporar herramientas de ejecución de pruebas que permitan realizar una automatización de estas.
Feedback	Sin Reportes	● Generar reportes de defectos internos. La idea es poder establecer una forma de comunicar cuál es el nivel de calidad del producto sometido a pruebas. Como primera aproximación, se sugiere llevar un conteo de los casos de pruebas ejecutados, el resultado de estos y los defectos reportados una vez que se realizaron los ajustes derivados de los anteriores. Es importante, además, poder hacer una estimación de los recursos/horas insumidos para realizar esta tarea.

Tabla 6-4 Mejoras por eje desde el nivel de Sin implementación, elaboración propia.

	Implementación Parcial	Actividades sugeridas para mejorar
Definición del Plan	Plan Informal.	● Documentar el plan, ya sea en una herramienta específica o en una hoja de cálculo, y distribuirlo a los interesados. ● Integrar y coordinar las actividades de prueba con las actividades del ciclo de vida de desarrollo. ● Estimar personas y recursos para realizar las diversas actividades. Si es posible, realizar una asignación tentativa. ● Generar un calendario con fechas específicas o bien, en el contexto de cada iteración. ● Realizar presupuesto estimado del tiempo requerido para realizar las actividades de prueba.
Niveles de Prueba afectados	Pruebas Unitarias y de sistema	● Analizar la base de prueba disponible con sus respectivos objetos, asociados al nivel de pruebas de integración y pruebas de nivel superior como las pruebas de aceptación. ● Relevar la necesidad de pruebas en niveles especializados como pruebas de Instalación, en caso de que deban considerarse. ● Identificar las prestaciones y conjuntos de prestaciones que se van a probar en relación a los diferentes niveles ● Definir y priorizar las condiciones para cada nivel manteniendo, en lo posible, la trazabilidad con los objetos anteriores.
Definición de casos de pruebas	Casos de prueba de Alto Nivel	● Generar casos de prueba, identificando los datos de entradas posibles, las condiciones de ejecución y los resultados esperados. ● Priorizarlos en base a las prestaciones que se van a probar. ● Especificarlos y registrarlos, usando este registro como base para el seguimiento de los mismos.
Tipos de Actividades de prueba	Pruebas Estáticas o bien dinámicas	● Realizar pruebas de análisis estático guiadas por herramientas como, por ejemplo, las de análisis del código fuente, las de model checking y de simulación de procesos de negocio. ● Automatizar pruebas dinámicas
Trazabilidad	Trazabilidad parcial	● Generar una trazabilidad desde los requerimientos, pasando por los objetos de prueba y finalizando en los casos de prueba, de modo que sea posible determinar el nivel de cobertura.
Entorno de Pruebas	Entorno gestionado y controlado	● Incluir en el entorno de prueba: arneses de prueba, ("test harness"), virtualización de servicios, simuladores y otros elementos de infraestructura. Verificar, además, que se encuentre configurado correctamente. ● Preparar los datos de prueba a partir de datos productivos y mantenerlos lo más actualizados posibles. Se debe tener como resultado, un ambiente lo más similar al ambiente productivo dentro de las posibilidades y restricciones del negocio.

	Implementación Parcial	Actividades sugeridas para mejorar
Herramientas de Pruebas	Herramientas específicas de seguimiento y ejecución de pruebas	● Incorporar herramientas para la Gestión de la Prueba y Productos de Prueba: Se pueden utilizar en cualquier actividad de prueba a lo largo de todo el ciclo de vida de desarrollo de software. ● Herramientas de soporte para la Prueba Estática. ● Incorporar herramientas para el Diseño e Implementación de Pruebas para crear productos de trabajo mantenibles en el diseño e implementación de pruebas, incluyendo casos de prueba, procedimientos de prueba y datos de prueba. ● Incorporar Herramientas para la Ejecución y el Registro de Pruebas. ● Incorporar herramientas para la Medición del Rendimiento y el Análisis Dinámico (esenciales para dar soporte a las actividades de pruebas de rendimiento y de carga, ya que estas actividades no se pueden realizar de forma eficaz de forma manual). ● Incorporar Herramientas para Necesidades de Prueba Especializadas en el caso que las pruebas de mayor prioridad, así lo requieran. Por ejemplo: Pruebas de seguridad, portabilidad, accesibilidad, entre otras.
Feedback	Reportes de defectos internos	● Generar reportes de progreso que permitan evaluar la evolución de las pruebas y realizar un seguimiento de las actividades asociadas a las mismas. ● Elaborar reportes de defectos externos para que el proceso de pruebas y la calidad del producto pueda ser comunicados a los interesados y que los mismo puedan realizar un feedback tanto de calidad conseguida como de las prioridades para futuras versiones.

Tabla 6-5 Mejoras por eje desde el nivel de Implementación Parcial, elaboración propia.

6.2.3 *Las herramientas de prueba*

Las herramientas de prueba pueden clasificarse en función de varios criterios, como el objetivo, el precio, el modelo de concesión de licencias (por ejemplo, comercial o de código abierto) y la tecnología utilizada. Para los propósitos de este trabajo, se utilizará la clasificación en función a la actividad que soportan.

Algunas herramientas claramente dan soporte a una sola o principalmente a una actividad; otras pueden dar soporte a más de una actividad, pero se clasifican según la actividad con la que están más estrechamente asociadas. Las herramientas de un solo proveedor, especialmente aquellas que han sido diseñadas para trabajar juntas, pueden ser suministradas como un paquete integrado (International Software Testing Qualifications Board, 2018).

Hoy en día, y tal como se evidenció en la encuesta realizada a las organizaciones del sector, el hecho de apoyarse en las diferentes herramientas permite que la implementación de cualquier metodología se realice de forma más intuitiva y consistente. Estas herramientas, además, permiten explotar los datos que son recabados en las diferentes partes del proceso, necesarios para la generación de métricas de proceso, requeridas en niveles intermedios de las diferentes certificaciones.

En esta sección, se describen y proponen una serie de herramientas de soporte para el desarrollo en los distintos ejes.

- *Herramientas de soporte para la definición del plan*

  Las herramientas que existen, más que para la definición propia del plan, permiten realizar seguimiento y monitorización, así como también, gestionar los cambios y que los mismos sean visibles para todos los interesados en el proceso.

  Dentro de las herramientas que se pueden utilizar para apoyar este eje, se encuentran las de gestión de la documentación. La gestión documental es muy importante para el buen funcionamiento de la empresa. Estas herramientas abarcan numerosas acciones como: tratamiento de los documentos, el flujo de los mismos, su distribución, consulta, reproducción, almacenamiento, envío, recepción, mejora, y destrucción de los documentos. Algunos ejemplos de las

mismas incluyen Google Drive, Dropbox y SharePoint. Para los fines descriptos en este eje, las herramientas gratuitas que brinda Google son suficientes para los fines propuestos.

Por otro lado, se encuentran las herramientas de gestión de proyecto, donde tanto las genéricas como aquellas diseñadas para ciclos de vida específicos, proporcionan al gestor de proyectos soporte para la planificación y el seguimiento. Adicionalmente, la mayoría de este tipo de herramientas también permite la gestión de requerimientos, donde se utiliza para capturar, organizar, dar prioridad y rastrear los mismos. Jira en su versión gratuita, por ejemplo, permite además la gestión de requisitos, gestión de pruebas, gestión de defectos y paneles con informes. Para ciclos de vida ágiles, entre las más usadas en su versión gratuita se encuentran Trello. Por otro lado, debemos mencionar Redmine que es una opción más enfocada en equipos de QA. Es un software para la gestión de proyectos que incluye un sistema de seguimiento de errores. Redmine destaca por ser software libre y de código abierto, disponible bajo la Licencia Pública General de GNU.

- *Herramientas de soporte para los distintos niveles de pruebas*

  Además de las herramientas de gestión de proyecto y gestión de requerimientos nombradas anteriormente, se le suman las herramientas de gestión de la configuración y de Integración continua. Una herramienta de gestión de la configuración ayuda a realizar el seguimiento de todos los productos de trabajo producidos y sus diferentes versiones (IBM, 2006). Los modelos y el código, concretamente, requieren una configuración gestionada. En este sentido, GitHub brinda un servicio de repositorio basado en la web para alojar y administrar proyectos de software, versiones y código fuente. Proporciona características como edición en línea, seguimiento de errores, administración de tareas, así como funciones de redes sociales que promueven el trabajo en forma colaborativa entre desarrolladores y testers.

  De forma complementaria, las herramientas de integración continua permiten hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes en los diferentes niveles de prueba. El proceso de integración consiste en que cada cierto tiempo (horas), se descargan

los códigos fuentes desde el control de versiones de la herramienta de gestión de la configuración, se compila, se ejecutan las pruebas y se generan informes. Jenkins es la herramienta de integración continua de código abierto más conocida del mercado, está desarrollada en Java, es compatible con múltiples sistemas de control de versiones.

- *Herramientas de soporte para la definición de casos de prueba*
  Tal como se indicó en la definición del eje, es importante la definición de "cómo probar" el producto de software que resultó del desarrollo. Para este punto, existen herramientas de diseño de pruebas que permiten generar entradas de prueba a partir de una especificación almacenada en el repositorio de una herramienta CASE, por ejemplo, o en una herramienta de gestión de requerimientos, a partir de condiciones de prueba especificadas almacenadas en la misma herramienta, o a partir del código. Una vez creados los casos de prueba, pueden almacenarse en una herramienta de gestión de pruebas para su control y seguimiento. Adicionalmente, muchas de las herramientas que se utilizan para la automatización de pruebas brindan interfaces de carga que facilitan la definición de los casos de pruebas. Un ejemplo es Zephyr, un add-on de Jira con las funcionalidades de crear, importar, ejecutar, realizar seguimiento, reportar y documentar los casos de prueba; además de vincularlos con sus respectivas historias de usuario. Esta herramienta permite crear un tipo de ticket denominado "Test", de la misma manera en la que creamos cualquier otro ticket en Jira, con los campos para cargar cada paso ("Test Step"), con los datos necesarios para la ejecución de dichas etapas ("Test Data") y el resultado esperado ("Expected Result").

- *Herramientas de soporte para los distintos tipos de actividades de pruebas*
  Para el caso de las pruebas dinámicas, las herramientas de automatización de pruebas generan ventajas comparativas. Nos referimos a la automatización de pruebas a controlar la ejecución de pruebas y comparar entre los resultados obtenidos y los resultados esperados de forma automática. La automatización de pruebas permite incluir pruebas repetitivas, en forma paralela y en diversos navegadores dentro de un proceso formal de pruebas ya existente o bien como

complemento de pruebas cuya ejecución manual es compleja. Estas herramientas son fundamentales para las llamadas pruebas de regresión.

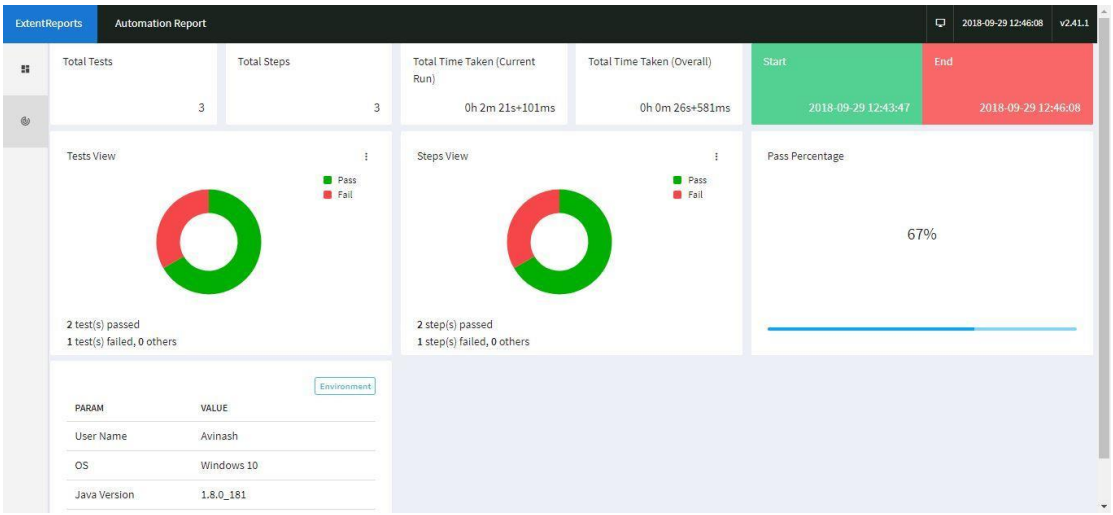

Figura 6-4 Ejemplo de reporte de ejecución de pruebas automatizadas creado con Selenium.

En la Figura 6-4, podemos ver una captura de Selenium uno de los entornos de ejecución de pruebas más utilizados, que provee una herramienta de grabación/reproducción para crear pruebas sin usar un lenguaje de scripting para pruebas. Incluye también un lenguaje específico de dominio para pruebas (Selenese) para escribir pruebas en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python. Las pruebas pueden ejecutarse entonces usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX. Esta herramienta permite además programar pruebas de rendimiento, tanto de carga como de estrés, aunque esto en general lo hacen otras herramientas que obtienen las métricas. Una de esas herramientas es JMeter.

Para las pruebas estáticas, existen herramientas para realizar análisis estático de código. Estas herramientas utilizan un conjunto de reglas para identificar problemas dentro del software. Detecta cosas como código duplicado, código muerto (variables, parámetros o métodos sin usar), complejidad de métodos (if innecesarios, etc.), desvío de los estándares, entre otras cosas. Un ejemplo de este tipo de herramienta es Sonarqube, una herramienta de software libre y gratuito que permite gestionar la calidad del código fuente. Dentro de sus funciones permite recopilar, analizar, y visualizar métricas del código fuente.

También realiza un histórico de todas las métricas del proyecto y genera informes con resúmenes de las mismas.

- *Herramientas de soporte a la trazabilidad*

  Las herramientas de soporte son especialmente importantes en este eje, ya que ahorran mucho tiempo y mejoran notablemente el proceso. Tanto las herramientas de administración de proyecto como las de gestión de requerimientos aportan en gran medida a las tareas relacionadas a la trazabilidad. Las mismas, facilitan el seguimiento y el control de los diferentes productos del proceso de desarrollo, mostrando la trazabilidad de forma visual, y resolviendo cuestiones asociadas con la integridad, permitiendo además la gestión de los defectos detectados y la generación de Informes.

- *Herramientas de soporte para los entornos de prueba*

  En este punto, las herramientas de gestión de la configuración y de integración continua son las más importantes para el mantenimiento y correcto control de los entornos de pruebas. De forma complementaria, existen herramientas más específicas asociadas a las características propias del entorno como aquellas asociadas a tecnologías específicas cuando se trata de aplicaciones Mobile, las que trabajan en con un lenguaje específico de programación o aquellas que se encuentran consumiendo o brindado servicios mediante diferentes protocolos de comunicación

- *Herramientas de soporte para feedback*

  Actualmente, la mayoría de las herramientas, independientemente de su clasificación, permiten en menor o mayor medida recopilar información para consolidar la experiencia y los productos de prueba, generando información necesaria para la elaboración de métricas del proceso. El desafío está en cómo comunicar esta información para que aporte al proceso que se está llevando a cabo.

Factores de Éxito para Herramientas

Al incorporar cualquier herramienta al proceso, es importante tener en cuenta las siguientes recomendaciones:

- Las nuevas herramientas deben desplegarse en la organización de forma incremental.
- Se deben adaptar y mejorar los procesos para que se ajusten al uso de la herramienta.
- Se debe proporcionar formación, entrenamiento y asesoramiento para los usuarios cualquiera sea la herramienta que se incorpore.
- Se deben definir directrices para el uso de la herramienta (por ejemplo, normas internas para la automatización).
- Se debe implementar una forma de recopilar información de uso a partir del uso real de la herramienta.
- Se debe monitorizar el uso y los beneficios de la herramienta.
- Se debe proporcionar apoyo a los usuarios de una herramienta determinada.
- Se deben recopilar las lecciones aprendidas de todos los usuarios.
- También es importante garantizar que la herramienta se integre técnica y desde una perspectiva de la organización en el ciclo de vida del desarrollo de software, lo que puede implicar la participación de organizaciones independientes responsables de las operaciones y/o de terceros proveedores.

6.2.4 *Proceso de aplicación*

El proceso de mejora consiste principalmente en realizar pequeñas mejoras significativas en ciclos cortos, ya que, en base a lo analizado en el capítulo 4, es la forma que se observan mejores resultados en las PyMES. Esta aproximación, permite que tanto la organización como los actores involucrados, hagan foco en actividades específicas, obteniendo resultados de forma más rápida y con menor resistencia a los cambios.

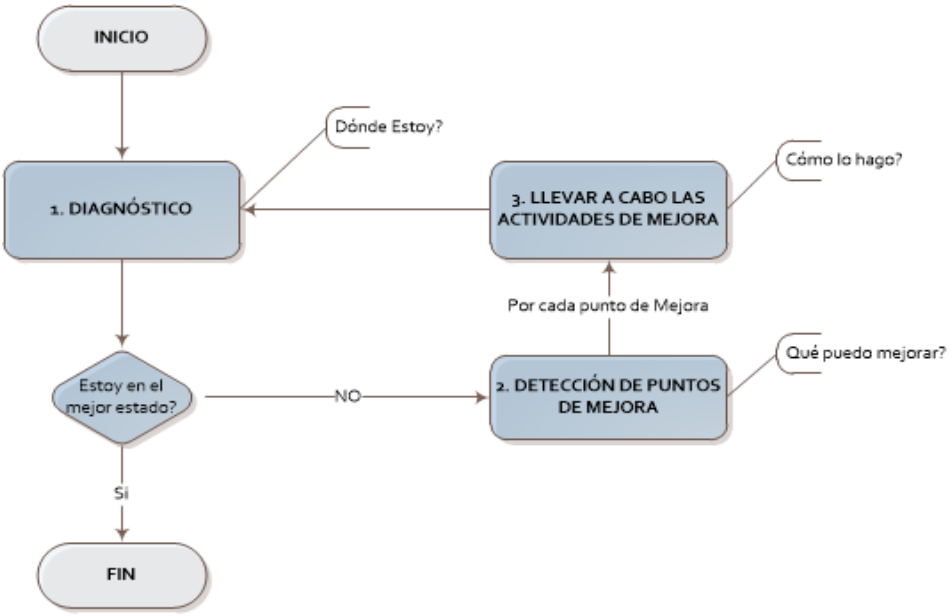

Figura 6-5 Proceso de aplicación de la propuesta, elaboración propia.

El proceso de aplicación, resumido en la Figura 6-5, se implementa de la siguiente forma:

1- **Diagnóstico:** Antes de implementar cualquier mejora, se debe establecer el punto de partida, es decir, el estado actual del proceso de pruebas de la organización. Para eso, se debe establecer un nivel para cada uno de los ejes descriptos en la matriz de diagnóstico. El objetivo en este punto es responder a la pregunta ¿"¿Dónde estoy?" para cada uno de los ejes. Se recomienda que participen de este diagnóstico todos los involucrados tanto en el proceso de desarrollo como aquellos responsables del producto como *Product Owners* o *Key Users.*

	Sin Implementación	Implementación Parcial	Implementación completa
Definición del Plan		X	
Niveles de Prueba afectados			X
Definición de casos de pruebas		X	
Tipos de Actividades de prueba			X
Trazabilidad	X		
Entorno de Pruebas			X
Herramientas de Pruebas		X	
Feedback	X		

Figura 6-6 Ejemplo de diagnóstico realizado, elaboración propia.

2- **Detección de puntos de mejora:** Una vez realizado el diagnóstico, se deben detectar cuáles son los ejes que tienen la capacidad de ser mejorados, es decir aquellos que no se encuentran en el nivel más alto. El propósito de este paso es determinar qué es lo que se puede mejorar del proceso. Aunque los ejes actúan de forma independiente, es decir, que el hecho de que el nivel de uno no depende ni condiciona el grado de evolución del resto, se recomienda comenzar por aquellos que se encuentran menos desarrollados y por el orden en el que se encuentran en la grilla. Este orden no es aleatorio, se encuentran ordenados por la etapa del proceso de desarrollo en la que se involucran estas actividades. Como lo hemos mencionado anteriormente, cuanto más temprano se involucren actividades de pruebas, mejores resultados tienen en la calidad del producto final.

	Sin Implementación	Implementación Parcial	Implementación completa
Definición del Plan		X	
Niveles de Prueba afectados			X
Definición de casos de pruebas		X	
Tipos de Actividades de prueba			X
Trazabilidad	X		
Entorno de Pruebas			X
Herramientas de Pruebas		X	
Feedback	X		

Figura 6-7 Ejemplo de detección de puntos de mejora, elaboración propia.

3- **Llevar a cabo las actividades de mejora:** Una vez detectados cuáles son los puntos de mejora y cuál es el orden en que se van a tratar, comienzan las iteraciones de forma incremental. Se comienza tomando un eje, viendo en qué nivel se encuentra y comenzando a implementar las actividades propuestas de forma conjunta con las herramientas sugeridas para dar soporte a ese eje. Como se dijo anteriormente, la idea es realizar cambios pequeños y significativos, por lo que se recomienda trabajar con un eje a la vez, incorporando de una actividad. Se recomienda que se acompañe esta incorporación con una herramienta que le de soporte, teniendo en cuenta los factores de éxito mencionados para llevarlo a cabo.

	Implementación Parcial	Actividades sugeridas para mejorar
Definición del Plan	Plan Informal.	● Documentar el plan, ya sea en una herramienta específica o en una hoja de cálculo, y distribuirlo a los interesados. ● Integrar y coordinar las actividades de prueba con las actividades del ciclo de vida de desarrollo. ● Estimar personas y recursos para realizar las diversas actividades. Si es posible, realizar una asignación tentativa. ● Generar un calendario con fechas específicas o bien, en el contexto de cada iteración. ● Realizar presupuesto estimado del tiempo requerido para realizar las actividades de prueba.

Figura 6-8 Extracto de la matriz de mejoras correspondiente al nivel de implementación parcial

4- **Verificar resultados:** llevadas a cabo las actividades de mejora, se debe volver a medir el grado de evolución del eje que se trabajó, utilizando la matriz de diagnóstico. Se recomienda verificar los cambios al conseguir un hito asociado al cambio o al finalizar una iteración en el caso de ciclos ágiles. En esta verificación, pueden pasar 3 cosas:

   a. **No se logró mejorar el nivel**. El eje sigue en el mismo nivel de evolución que cuando se diagnosticó primeramente. En este caso, se puede tomar otra de las actividades para lograr implementarlo o una nueva herramienta para profundizar el cambio. No todas las actividades y herramientas de mejora son aplicables en todos los casos y los tiempos no son los mismos para todos los ejes. Muchas veces ciertas herramientas son mejor aceptadas por quienes van a usarlas que otras. Idealmente, esta situación debe detectarse lo antes posible por lo que debe destinarse el tiempo necesario para realizar el control de avance.

   b. **Se mejoró el nivel, pero aún no se encuentra en el nivel más alto.** En este caso, se puede continuar con otro eje que se encuentre en un nivel inferior y, una vez que el resto fueron tratados, si los hubiere, volver a revisarlo con la matriz asociada al siguiente nivel.

   c. **Se encuentra en el nivel más alto.** Si el eje ya se encuentra en el nivel más alto, se debe continuar con otro de los ejes detectados en el punto 3. En el caso de que ya hayan sido todos tratados se debe volver a realizar la misma evaluación integral del punto 1. Si en este diagnóstico, todos los ejes se encuentran en el nivel superior, la implementación de la metodología se encuentra **Completa**.

6.2.5 *Roles de prueba*

Una de las primeras consideraciones que se debe tener en cuenta en la aplicación de esta metodología, es que las tareas de prueba pueden ser realizadas por personas que desempeñan un rol de prueba específico o por personas que desempeñan otro rol (por ejemplo, clientes). Un cierto grado de independencia, como lo han comprobado muchos modelos de mejora, a menudo, hace que quien prueba sea más efectivo para encontrar defectos debido a las diferencias entre los sesgos asociados al conocimiento del autor y del *tester*. Sin embargo, la independencia no es un sustituto de la familiaridad, y los desarrolladores pueden encontrar de forma eficiente muchos defectos en su propio código. Además, tal como se evidenció en el análisis del capítulo 4, las PyMES en su mayoría se encuentran limitadas para destinar perfiles dedicados de forma exclusiva a las tareas de prueba.

De forma general y para la mayoría de los tipos de proyectos, se recomienda que existan diferentes niveles de prueba, con algunos de estos niveles tratados por *testers* independientes. Los desarrolladores deben participar en la prueba, especialmente en los niveles inferiores, de manera que puedan ejercer control sobre la calidad de su propio trabajo.

6.2.6 *El papel de la comunicación*

La comunicación en un proceso de pruebas es fundamental, sobre todo cuando se quiere implantar en una organización que no ha incorporado este tipo de prácticas. Generar un buen ambiente en el equipo de trabajo es fundamental para el éxito de la apropiación de un proceso “nuevo” en una organización. Dicha comunicación involucra detalles tan pequeños como la descripción de un hallazgo o su clasificación, por tanto, se recomienda que para este tipo de procesos se establezcan formas de lenguaje comunes que ayuden al éxito del proyecto y a la calidad del producto desarrollado. Durante todo el proceso de implementación de esta propuesta, se pretende que la mayor cantidad de personas involucradas en el proyecto puedan comunicar sus necesidades sobre la calidad del producto. Tanto desde el momento del diagnóstico como en los sucesivos ciclos de mejora, se pretenden generar conversaciones sobre lo que se espera del producto. Los ciclos

cortos deben dar lugar a reuniones, al menos quincenales, que permitan poner objetivos y revisar avances.

Aquellos involucrados en el proceso de mejora deben poder comunicarse internamente y con los grupos externos. La buena comunicación interna es esencial para trabajar como un grupo cohesionado con la oportunidad de mantenerse al tanto de los acontecimientos en el grupo. La comunicación con los clientes permite que el grupo de prueba conozca las expectativas de los mismos. La comunicación con los desarrolladores es esencial para comprender los detalles de diseño del sistema. El grupo de marketing, al comunicarse con el grupo de prueba, realiza un seguimiento del progreso y el nivel de calidad del sistema bajo prueba.

Adicionalmente, es importante la forma en la que se comunican los fallos y defectos detectados ya que puede ser percibido como una crítica al producto y a su autor. Un elemento de la psicología humana llamado sesgo de confirmación puede dificultar la aceptación de información que esté en desacuerdo con las creencias actuales. Por ejemplo, dado que los desarrolladores esperan que su código sea correcto, tienen un sesgo de confirmación que hace difícil aceptar que el código sea incorrecto. Además del sesgo de confirmación, otros sesgos cognitivos pueden dificultar que las personas entiendan o acepten la información producida por la prueba. Asimismo, es un rasgo humano común culpar al portador de malas noticias, y la información producida por la prueba, a menudo, contiene malas noticias. Como resultado de estos factores psicológicos, algunas personas pueden percibir al proceso de prueba como una actividad destructiva, a pesar de que contribuye en gran medida al avance del proyecto y a la calidad del producto (International Software Testing Qualifications Board, 2018). Para tratar de reducir estas percepciones, la información sobre defectos y fallos debe ser comunicada de manera constructiva. En este punto, las herramientas de gestión de defectos son de gran utilidad ya que estandarizan la forma en la que se comunican esos defectos. De esta manera, se pueden reducir las tensiones entre quienes realizan las pruebas y los analistas, los propietarios de producto, los diseñadores y los desarrolladores. Esto se aplica tanto a las pruebas estáticas como a las dinámicas. Adicionalmente, es importante que quienes realicen las pruebas tengan buenas competencias interpersonales para poder comunicarse eficazmente sobre defectos, fallos, resultados de la prueba, avance de la prueba y riesgos, y para construir relaciones

positivas con las personas que forman parte de su entorno de trabajo. Las formas de comunicarse de manera adecuada incluyen los siguientes ejemplos (International Software Testing Qualifications Board, 2018):

- Comenzar con colaboración en lugar de batallas. Recordar a todos que el objetivo común es lograr sistemas de mejor calidad.
- Enfatizar los beneficios de la prueba. Por ejemplo, para los autores, la información sobre los defectos puede ayudarles a mejorar sus productos de trabajo y sus competencias. Para la organización, los defectos detectados y corregidos durante la prueba ahorrarán tiempo y dinero y reducirán el riesgo general para la calidad del producto.
- Comunicar los resultados de las pruebas y otros hallazgos de manera neutral y centrada en los hechos sin criticar a la persona que creó el elemento defectuoso. Redactar informes de defecto objetivos y fundamentados en hechos y revisar los hallazgos.
- Tratar de entender cómo se siente la otra persona y las razones por las que puede reaccionar negativamente a la información.
- Confirmar que el interlocutor ha entendido lo que se ha dicho y viceversa.

Definir claramente los objetivos de prueba y plasmarlos en el plan, tiene importantes implicaciones psicológicas. La mayoría de las personas tienden a alinear sus planes y comportamientos con los objetivos establecidos por el equipo, la dirección y otros implicados. También es importante que los testers se adhieran a estos objetivos con un mínimo sesgo personal.

6.3 El resultado: La implementación completa

Lo visto anteriormente, nos invita a preguntarnos qué significa que todos los niveles se encuentren en el nivel superior o con una implementación completa. Quiere decir que la organización logró implementar un proceso controlado y formalizado para llevar a cabo sus pruebas. Existe un plan formalizado que abarca los distintos tipos niveles de prueba, donde los casos de prueba se encuentran especificados de forma completa e incluyen actividades de pruebas estáticas y dinámicas de forma combinada. Las pruebas se realizan en entornos controlados y adecuados, con trazabilidad completa y soportado por herramientas. Existe la generación de una base de *testing* que se toma como punto de partida de cada iteración o fase y se

va incrementando mientras el producto evoluciona. El resultado de esta propuesta, además, tiene la capacidad de adaptarse a un proyecto en particular y, a medida que se ejecuta en distintos ámbitos, retroalimenta procesos base gracias a la existencia de feedback permanente.

En la Tabla 6-6, podemos ver como la propuesta es superadora con respecto a los modelos de mejora analizados en el contexto de las PYMES sin alejarse de las características estudiadas y probadas que hacen que estos modelos sean tan populares en el mercado.

En este contexto, debemos tener en cuenta que, al conseguir el nivel de implementación completa, se sentaron las bases para aspirar a una certificación ISO o CMM ya que se reconoce el proceso como tal, lo estandariza y le da forma incorporando actividades y estructuras que son comunes a todos los modelos analizados, por lo que la organización en general se encuentra en una situación más propicia y mejor predispuesta para llevar a cabo estos nuevos desafíos. Es el puntapié en la generación de nuevas tareas, áreas y responsabilidades con conciencia sobre los efectos de las pruebas. Esta mejora en el proceso, indefectiblemente se verá reflejada en la mejora de la calidad del producto final

	CMM/CMMi	TMM/TMMi	TPI	Tmap/Tmap Next	IT Mark	COMPETISOFT	ISO/IEC 29110	ISO/IEC 29119	Propuesta de este trabajo
Licencia	Propietario	Propietario	Propietario	Propietario	Propietario	Abierto	Abierto	Abierto	**Abierto**
Tipo	Modelo de mejora de procesos	Modelo de mejora del proceso de pruebas	Modelo de mejora del proceso de pruebas	Modelo de mejora del proceso de pruebas	Modelo de mejora de procesos	Modelo de mejora de procesos	Modelo de mejora de procesos	Modelo de mejora del proceso de pruebas	**Modelo de mejora del proceso de pruebas**
Definición	Parcialmente Definido	Parcialmente Definido	Parcialmente Definido	Completamente Definido	Parcialmente Definido	Descriptivo	Descriptivo	Descriptivo	**Completamente Definido**
Acoplado a un ciclo de vida específico	No	No	No	Si	No	Si	No	No	**No**
Experiencia	Muy necesaria	Muy necesaria	Muy necesaria	Muy Necesaria	Muy Necesaria por falta de documentación	No	Necesaria la formación en certificaciones ISO	Necesaria la formación en certificaciones ISO	**No**
Incorpora técnicas de pruebas	No	No	No	Si	No	No	No	No	**Si**
Incorpora herramientas de soporte para su implementación	No	No	No	Si	No	Si	No	No	**Si**

Tabla 6-6 Comparativa entre los modelos de mejora y la propuesta de este trabajo.

6.4 Resumen de Evaluación de Expertos

Dada la complejidad de la situación actual que estamos atravesando, la propuesta detallada anteriormente no pudo ser puesta en práctica en un contexto real. En su lugar, la misma fue sometida a juicio de expertos, tanto del área de conocimiento como de la industria, quienes evaluaron su factibilidad y su aporte a la problemática que le da origen.

Desde el punto de vista de los modelos evaluados y los criterios de comparación, los expertos consideraron que los modelos seleccionados son los que actualmente están vigentes en la industria. Así como los criterios considerados eran suficientes y pertinentes.

Con respecto a la propuesta propiamente, consideraron en general que era clara y bien organizada. Destacaron su simplicidad y practicidad, así como también su capacidad de adaptación a los diferentes modelos de desarrollo. Valoraron también su posibilidad de aplicación tanto en empresas desarrolladores como en áreas de sistemas dentro de cualquier tipo de organización que cuenten con escasos recursos destinados a la implementación de procesos de mejora de este tipo, principalmente por su abordaje integral, el aporte de las herramientas de soporte y la guía para su implementación.

Con respecto a los ejes planteados, consideraron que eran oportunos, balanceados y coherentes con respectos a las actividades de mejora que plantean. La división de ejes y niveles se consideró clara dado que permite visualizar el estado real y concreto de cada uno de los puntos que son importantes en los distintos momentos en los que se pueden encontrar estas organizaciones. Se mencionó que el modelo en general permite realizar un análisis exhaustivo sin resultar denso o complejo, y que además su enfoque progresivo aumenta su nivel de aceptación a la hora de llevarlo a cabo. Por otro lado, se recomendó incluir las pruebas de integración en el nivel de implementación parcial de la categoría de "Niveles de prueba afectados", dado que podrían realizarse en conjunto con las unitarias. Así como también, se

recomendó evaluar la posibilidad de incluir métricas en este nivel, como son la de cobertura de código[7], dado que son métricas sencillas de calcular y utilizar.

Por último, con respecto a cuán relevante resulta la propuesta para las PyMEs desarrolladoras de software de la región, todos los expertos reconocieron la problemática como algo que fehacientemente sucede en la industria. En base a su experiencia, los evaluadores consideran que las principales barreras que tienen las PyMEs para adoptar marcos de trabajo o modelos de mejora de este tipo son la complejidad de los modelos tradicionales y la inexistencia de modelos adaptados a sus realidades. Destacaron la necesidad de procesos de calidad para el cumplimiento de los contratos de servicio, el involucramiento del cliente en esos procesos y la necesidad de mejora continua en la industria.

En general, la propuesta se consideró útil tanto para la mejora en la calidad del producto como para la mejora de la calidad del proceso de desarrollo. Se considera como una alternativa válida, acertada, enfocada en las limitaciones típicas de las PyMES y que la misma cuenta con la información suficiente para ponerse en práctica de forma inmediata.

Las devoluciones recibidas se encuentran anexadas al final de este trabajo.

[7] Cobertura de código: método de análisis que determina qué partes del software han sido ejecutadas (cubiertas) por el conjunto de pruebas y qué partes no se han ejecutado, normalmente expresado en porcentaje (Veenendaal, 2010).

Conclusiones

En este punto, y para finalizar, cabe realizar las conclusiones de este trabajo.

Se ha presentado el marco teórico/conceptual relativo a las pruebas de software, los estándares y los modelos de mejora de procesos de prueba más populares en la industria a nivel global. Con el marco teórico desarrollado, se pudo determinar que los modelos, en general, tienen componentes y estructuras similares. Es común la utilización de escalas para diagnosticar el estado de las organizaciones y llevar a cabo sus mejoras. Además, se pudo corroborar que existen relaciones explícitas entre modelos, por ejemplo, cuando un modelo se elaboró basándose en otro.

En cuanto al análisis de la situación actual de las PyMEs de la región, se lograron identificar las complejidades en la realización de pruebas y en la detección de errores en este sector de la industria, tanto a partir de la bibliografía y estudios consultados como a partir de la encuesta realizada y la experiencia profesional en este tipo de industrias. Además, se logró establecer el escenario en el cual se va a llevar a cabo la propuesta a partir de los desafíos que atraviesan estas organizaciones en la actualidad.

Una vez determinadas estas características y necesidades, se realizó un estudio comparativo entre los modelos y estándares, determinando que, aunque las propuestas que entregan son de gran valor, ninguno de los modelos analizados es aplicable en este tipo de empresas. De esta forma, se expone la necesidad y la importancia de llevar a cabo este trabajo.

Cabe destacar que esta propuesta, nace a partir del análisis de las necesidades y limitaciones de las PyMEs y se constituye en base a aquellas actividades y prácticas que han demostrado que aportan valor en los modelos de mejora analizados. En este sentido, se plantea un proceso ágil, sin demasiados requisitos para su implementación, con la mínima documentación posible y que genera productos de trabajo reutilizables para su despliegue en los diferentes proyectos de la organización.

Dado el contexto global que estamos atravesando, no fue viable llevar a cabo esta propuesta en un caso testigo, por lo que se sometió la hipótesis al juicio de expertos.

El hecho de contar con la evaluación de personas que se desarrollan en este tipo de industrias suma un punto de vista adicional al trabajo desarrollado.
Con su devolución positiva, se concluye que se han cumplido los objetivos específicos, y por lo tanto el objetivo general.

Trabajos Futuros

Dado el contexto por el cual está atravesando el mundo y sobre todo la región, el principal desafío que se plantea para el futuro es poner a prueba esta propuesta en un entorno real, en este caso en una PyME desarrolladora de software. Esto último, permitirá plantear nuevos desafíos para perfeccionar el modelo, dado que se podrá ver la capacidad real de la adaptación en las situaciones particulares de cada organización. Esto permitirá la evolución constante en función de las distintas realidades en las que sea aplicado.
Es importante destacar que la propuesta que se expone en este trabajo, pretende ser un primer paso para aquellas organizaciones definidas como inmaduras. Es decir, organizaciones donde los procesos de software generalmente son improvisados por las personas que intervienen en ellos, al igual que la administración durante el curso de los proyectos. Incluso si se ha especificado un proceso de software, no se sigue ni se aplica rigurosamente. Un desafío para el futuro es lograr el paso siguiente, enfocado a aquellas organizaciones que lograron completar este proceso y están en condiciones de avanzar al siguiente nivel.
Finalmente, en función de lo relevado y analizado en la encuesta realizada, se pudo demostrar que este tipo de organizaciones reciben con gran aprecio la incorporación de herramientas libres en sus procesos, por lo que es importante trabajar en investigar y adaptar las herramientas existentes a las formas de trabajo de una organización. Como trabajo futuro se propone la adaptación e integración de algunas de estas herramientas, de tal forma que en su conjunto faciliten al responsable de pruebas obtener indicadores a partir de los cuales se puedan tomar decisiones sobre el proceso y sobre la calidad del producto; esto implica extender la investigación sobre las herramientas de prueba y desarrollo.

ANEXOS

Anexo I: Encuesta para conocer el estado del testing en la región

Encuesta para
Conocer el Estado de

Anexo II: Relaciones entre bases, objetos y niveles de prueba (International Software Testing Qualifications Board, 2018)

	Objetivos específicos.	Bases de prueba, referenciadas para generar casos de prueba.	Objeto de prueba (es decir, lo que se está probando).	Defectos y fallos característicos.
Prueba unitaria	• Reducir el riesgo. • Verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados. • Generar confianza en la calidad del componente. • Encontrar defectos en el componente. • Prevenir la propagación de defectos a niveles de prueba superiores.	• Diseño detallado. • Código. • Modelo de datos. • Especificaciones de los componentes.	• Componentes, unidades o módulos. • Código y estructuras de datos. • Clases. • Módulos de base de datos.	• Funcionamiento incorrecto (por ejemplo, no como se describe en las especificaciones de diseño). • Problemas de flujo de datos. • Código y lógica incorrectos.
Prueba de integración	• Reducir el riesgo. • Verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados. • Generar confianza en la calidad de las interfaces. • Encontrar defectos (que pueden estar en las propias interfaces o dentro de los componentes o sistemas). • Prevenir la propagación de defectos a niveles de prueba superiores.	• Diseño de software y sistemas. • Diagramas de secuencia. • Especificaciones de interfaz y protocolos de comunicación. • Casos de uso. • Arquitectura a nivel de componente o de sistema. • Flujos de trabajo. • Definiciones de interfaces externas.	• Subsistemas. • Bases de datos. • Infraestructura. • Interfaces. • Interfaces de programación de aplicaciones (API por sus siglas en inglés). • Microservicios.	• Datos incorrectos, datos faltantes o codificación incorrecta de datos. • Secuenciación o sincronización incorrecta de las llamadas a la interfaz. • Incompatibilidad de la interfaz. • Fallos en la comunicación entre componentes/sistemas. • Fallos de comunicación entre componentes/ si no tratados o tratados de forma incorrecta. • Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre componentes. Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre sistemas. • Estructuras de mensajes inconsistentes entre sistemas. • Incumplimiento de las normas de seguridad obligatorias.

Prueba de sistema	• Reducir el riesgo. • Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados y especificados. • Validar que el sistema está completo y que funcionará como se espera. • Generar confianza en la calidad del sistema considerado como un todo. • Encontrar defectos. • Prevenir la propagación de defectos a niveles de prueba superiores o a producción.	• Especificaciones de requisitos del sistema y del software (funcionales y no funcionales). • Informes de análisis de riesgo. • Casos de uso. • Épicas e historias de usuario. • Modelos de comportamiento del sistema. • Diagramas de estado. • Manuales del sistema y del usuario	• Aplicaciones. • Sistemas hardware/software. • Sistemas operativos. • Sistema sujeto a prueba (SSP). • Configuración del sistema y datos de configuración.	• Cálculos incorrectos. • Comportamiento funcional o no funcional del sistema incorrecto o inesperado. • Control y/o flujos de datos incorrectos dentro del sistema. • Incapacidad para llevar a cabo, de forma adecuada y completa, las tareas funcionales extremo a extremo. • Fallo del sistema para operar correctamente en el/los entorno/s de producción. • Fallo del sistema para funcionar como se describe en los manuales del sistema y de usuario
Prueba de aceptación	• Establecer confianza en la calidad del sistema en su conjunto. • Validar que el sistema está completo y que funcionará como se espera. • Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados.	• Procesos de negocio. • Requisitos de usuario o de negocio. • Normativas, contratos legales y estándares. • Casos de uso. • Requisitos de sistema. • Documentación del sistema o del usuario. • Procedimientos de instalación. • Informes de análisis de riesgo. Además, como base de prueba para derivar casos de prueba para la prueba de aceptación operativa, se pueden utilizar uno o más de los siguientes productos de trabajo: • Procedimientos de copia de seguridad y restauración. • Procedimientos de recuperación de desastres. • Requisitos no funcionales. • Documentación de operaciones. • Instrucciones de despliegue e instalación. • Objetivos de rendimiento. • Paquetes de base de datos. • Estándares o reglamentos de seguridad.	• Sistema sujeto a prueba. • Configuración del sistema y datos de configuración. • Procesos de negocio para un sistema totalmente integrado. • Sistemas de recuperación y sitios críticos (para pruebas de continuidad del negocio y recuperación de desastres). • Procesos operativos y de mantenimiento. • Formularios. • Informes. • Datos de producción existentes y transformados.	• Los flujos de trabajo del sistema no cumplen con los requisitos de negocio o de usuario. • Las reglas de negocio no se implementan de forma correcta. • El sistema no satisface los requisitos contractuales o reglamentarios. • Fallos no funcionales tales como vulnerabilidades de seguridad, eficiencia de rendimiento inadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada.

Anexo III: Evaluaciones de expertos


Devolucion - A. Tejera.pdf


Devolucion - U. Marsiglia.pdf


Devolucion - S. Scozzesi.pdf


Devolución P. Sadone.pdf


Devolución - G. Gorosito.pdf

BIBLIOGRAFÍA

Beck, K. (1999). Embracing Change with Extreme Programming. *IEEE Computer*, *32*(10), 70–78.

Beck, K. (2000). *Extreme Programming explained* (M. Reading (Ed.)). Addison-Wesley.

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. http://agilemanifesto.org/

Boehm, B. (1988). A Spiral Model for Software Development and Enhancement. *Computer, Vol. 21, N° 5*, 61–72.

Boehm, B., & Hansen, W. (2001). The Spiral Model as a Tool for Evolutionary Software Acquisition. *CrossTalk*, *14*, 4–11. www.stsc.hill.af.mil/crosstalk/2001/05/boehm.html.

Bourque, P., & Fairley, R. E. (2014). SWEBOK V3.0 Guide to the Software Engineering Body ok Knowledge. In P. Bourque, É. de technologie supérieure (ÉTS), R. E. Fairley, & Software and Systems Engineering Associates (S2EA) (Eds.), *IEEE Computer Society* (Version 3.). IEEE Computer Society. https://doi.org/10.1234/12345678

Burnstein, I. (2003). *Practical Software Testing*. Springer.

Canfora, G., & Ruiz González, F. (2004). Tecnología de proceso software. *Novatica: Revista de La Asociación de Técnicos de Informática, ISSN 0211-2124, Nº. 171*, 5–8.

CESSI Argentina. (2020). www.cessi.org.ar

Competisoft. (2008). *COMPETISOFT. Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria del Software de Iberoamérica*. https://alarcos.esi.uclm.es/ipsw/doc/competisoft-modelo_v1.pdf

Davis, M. (1995). Process and Product: Dichotomy or Duality. *ACM SIGSOFT Software Engineering Notes*, *20*(2), 17–18.

De La Villa, M., Ramos, I., & Ruiz, M. (2004). Modelos de Evaluación y Mejora de Procesos: Análisis Comparativo. *Adis 2004*, 1–18. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-120/paper4.pdf

Dutta, S., Lee, M., & Wassenhove, L. Van. (1999). Software Engineering in Europe: A Study of Best Practices. *IEEE Software*, *1*(3), 45 – 53.

Ernst, M. D. (2003). *Static and Dynamic Analysis: Synergy and Duality* (pp. 24–27). ICSE Workshop on Dynamic Analysis.

Foster, E. (2006). Quality Culprits. *InfoWorld Grip Line Weblog*.

Friedman, M. A., & Voas, J. M. (1995). *Software Assessment: Reliability, Safety, Testability*. Wiley.

Fundación Sadosky. (2020). http://www.fundacionsadosky.org.ar/

Garvin, D. (1984). What Does 'Product Quality' Really Mean? *Sloan Management Review*, 25–45.

Gebelein, S. H., Nelson-Neuhaus, K. J., Skube, C. J., Lee, D. G., Stevens, L. A., Hellervik, L. W., & Davis, B. L. (1998). *Successful Manager's Handbook* (L. Marasco (Ed.)). Personnel Decisions International.

Hanna, M. (1995). *Farewell to Waterfalls*. 38–46.

Hildreth, S. (2005). Buggy Software: Up from a Low Quality Quagmire. *Computerworld*, *39*(30), 23–25. http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=17720183&site=ehost-live

IBM. (2006). *Concepto: Herramientas de soporte*. https://cgrw01.cgr.go.cr/rup/RUP.es/LargeProjects/core.base_rup/guidances/concepts/supporting_tools_ABC700D1.html#:~:text=Una herramienta de documentación para,documentos que presentan los modelos.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology* (p. 84). www.mit.jyu.fi/ope/kurssit/.../IEEE_SoftwareEngGlossary.pdf

International Software Testing Qualifications Board. (2018). *Probador Certificado del ISTQB ® Programa de Estudio de Nivel Básico International Software Testing Qualifications Board*.

ISO/IEC/IEEE 29119:2013 Software Testing Strandard (Vol. 2013). (2013). https://doi.org/10.1109/IEEESTD.2013.6588543

ISO/IEC 29110 Software Engineering – Lifecycle profiles for Very Small Entities (VSEs) – Management and engineering guide. (2011).

ISTQB. (2020). *ISTQB*. www.istqb.org

IT Mark. (2017). http://it-mark.eu/?&lang=es

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.

Koomen, T., Van Der Aalst, L., Broekman, B., & Vroon, M. (2006). *TMap Next for result-driven testing*. UTN Publishers: Netherlands.

Levinson, M. (2001). Let's Stop Wasting $78 billion a Year. *CIO Magazine*. www.cio.com/archive/101501/wasting.html

Martin, J. (1981). *Application Development Without Programmers* (Englewood). Prentice-Hall.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in Software Quality, Technical Report RADC-TR-77-369*.

Mills, H. D., O'Neill, D., Linger, R. C., Dyer, M., & Quinnan, R. E. (1980). The Management of Software Engineering. *IBM Systems Journal*, *18*(4), 414–477.

Myers, G. J., Badgett, T., & Sandler, C. (2012). *The Art of Software Testing* (Third Edit). John Wiley & Sons, Inc.

Naik, K., & Priyadarshi, T. (2008). *Software Testing and Quality Assurance Theory and Practice*. John Wiley & Sons, Inc.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). *Capability Maturity Model for Software* (Issue February).

Pol, M., Teunissen, R., & Van Veenendaal, E. (2002). *Software Testing: A Guide to the TMap Approach*. Addison-Wesley.

Politi, P. I. (2020). *Economía del Conocimiento: ¿nueva ley?* https://abogados.com.ar/economia-del-conocimiento-nueva-ley/25979

Pressman, R. S. (2010). Ingeniería Del Software Un Enfoque Práctico. In *McGraw-Hill* (Séptima Ed). McGraw-Hill. https://doi.org/http://zeus.inf.ucv.cl/~bcrawford/Modelado%20UML/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf

PROGRAMA DE RECUPERACIÓN PRODUCTIVA - Ley 27264. (2016). http://servicios.infoleg.gob.ar/infolegInternet/anexos/260000-264999/263953/norma.htm

Rettig, M. (1994). Practical Programmer: Prototyping for Tiny Fingers. *Communications of the ACM, Vol. 37, N°4*, 21–27.

Ricadel, A. (2001). The State of Software Quality. *InformationWeek*. www.informationweek.com/838/quality.htm

Rojas-Montes, M. L., Pino-Correa, F. J., & Martínez, J. M. (2015). Proceso de pruebas para pequeñas organizaciones desarrolladoras de software. *Revista Facultad De Ingeniería*, *24*(39), 55. https://doi.org/10.19053/01211129.3551

Roura, H. (2019). *Las PyMES en el desarrollo de la economía argentina*. http://www.informeindustrial.com.ar/verNota.aspx?nota=Las PyMES en el desarrollo de la economía argentina___169

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum* (Cliffs Eng). Prentice Hall.

SEI. (2010). *CMMI for Development, Version 1.3*. *November*.

Sommerville, I. (2005). Ingeniería del software. In *Benet Campderrich Falgueras II Editorial UOC* (Séptima Ed). Addison-Wesley. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Ingeniería+del+software+Ingeniería+del+software#0

Sommerville, I. (2011). Software Engineering. In *Monthly Notices of …* (9 th., Vol. 291). Addison-Wesley. https://doi.org/10.1111/j.1365-2362.2005.01463.x

van der Aalst, L., Broekman, B., Koomen, T., & Vroon, M. (2010). *TMap Next, the Essentials of TMap*. Sogeti Nederland B.V.

Van Driel, E. (2010). *Software Control y Testing TMap Next® Pruebas impulsadas por los objetivos de negocio*. SOGETI España.

Van Veenendaal, E. (2018). *Test Maturity Model Integration (TMMi) Foundation*. http://www.tmmifoundation.org/downloads/tmmi/TMMi%5CnFramework.pdf%5Cnhttp://www.tmmifoundation.org/downloads/tmmi/TMMi;

Veenendaal, E. V. (2010). Standard glossary of terms used in Software Testing, Version 1.2. *International Software Testing Qualification Board*, *1*, 1–51.

Wallace, D.R. Fujii, R. U. (1989). Software Verification and Validation: An Overview. *IEEE Computer*, *6*(3), 10–17.