VALIDACIÓN AUTOMÁTICA DE SISTEMAS

H, Merlino[1,2], **E. Fernández** [2,1], **D. Rodríguez**[2], **P. Britos** [2,1], **R. García-Martínez**[2,1]

[1]Laboratorio de Sistemas Inteligentes, Facultad de Ingeniería, Universidad de Buenos Aires.
Paseo Colón 850 4to Piso. Ala Sur.
(1063) Capital Federal, ARGENTINA.
[2]Centro de Ingeniería de Software Ingeniería del Conocimiento. Escuela de Postgrado. ITBA
25 de Mayo 444 – 6to. Piso
Capital Federal, República Argentina
{hmerlino, rgm}@itba.edu.ar

Resumen

En el presente reporte se hace una introducción a la validación automática de sistemas, se detallan un conjunto de fundamentos sobre la importancia de la validación; a continuación se citan diferentes autores y sus propuestas sobre automatización; luego de esto se hace una reseña sobre algunas herramientas para realizar el proceso y por ultimo a modo de conclusión se da la visión del autor sobre la validación automática de sistemas.

Palabras Claves: Validación automática, automatización.

Abstract

This report describes an introduction to automatic validation systems, the fundamentals about validation's importance are detailed; first there are references about authors, its proposals and automation; secondly there is a summary about some tools required for the process in conclusion it is given the author's vision about.

Keywords: Automatic validation, validation

1. INTRODUCCIÓN

La validación automática de sistemas es el proceso por el cual los requisitos de sistemas son corroborados con poca o nula participación por parte del equipo de proyecto, lo que permite mejorar el producto obtenido. En términos generales esto se logra relacionando cada requisito de sistema a un paquete de pruebas.

La validación automática de software es una práctica que en los últimos tiempos la comunidad de informática le a prestado atención debido a los costos cada vez mas altos que tiene el proceso de complicación-depuración-integración-producción de los sistemas software.

Alguna de las razones por lo cual se hace menester hoy mas que nunca intentar llegar a un proceso de validación automática de sistemas es:

- ✓ *Requerimientos de usuarios cada vez más complejos.*
- ✓ *Diversidad de lenguajes con los que se desarrolla un mismo sistema.*
- ✓ *Diferentes plataformas donde se debe ejecutar un sistema.*
- ✓ *Distintos entornos de programación.*
- ✓ *Constante avance de las tecnologías y métodos de desarrollo.*
- ✓ *Alta rotación de personal.*

2. AUTOMATIZACIÓN DE PROCESOS DE PRUEBA

2.1 Introducción

Fewster [1], cita que la automatización de las pruebas representa un proceso diferente al del proceso de prueba. En consecuencia podemos decir que la calidad de la automatización de las pruebas es independiente de la calidad de la prueba. Para realzar un eficiente proceso de pruebas automatizado se debería determinar un buen proceso de integración de herramientas de prueba y construcción de software. En el siguiente diagrama se representa la evolución en el tiempo de al automatización de los procesos de prueba (Gráfico 1)

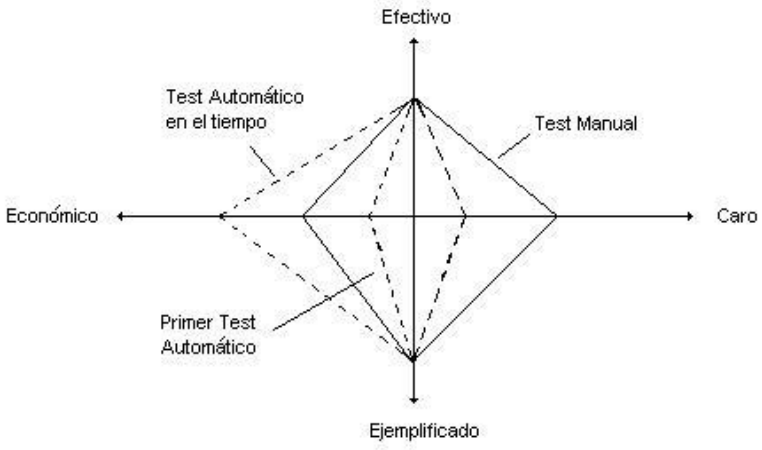

Gráfico 1. Evolución en el tiempo de al automatización

En el Gráfico 1 se observa como a medida que se consigue un grado mayor de automatización el proceso económica viable y efectivo en comparación a un proceso manual de validación.

A continuación se debería poder definir claramente que procesos de validación automatizar, en el presente trabajo y por lo extenso del tema en cuestión; se ha centrado la atención en los temas relacionados con la puesta en producción del software y la validación funcional de un sistema. Se ha dejado de lado la validación de requerimientos, documentación y arquitectura seleccionada, como así también selección de herramientas lenguajes y diseño arquitectónico.

La validación automática no siempre es recomendada, Fewster [1] detalla un conjunto de situaciones en donde la automatización no es más eficiente que las pruebas manuales, estos escenarios son:

- ✓ Las pruebas son ejecutadas rara vez.
- ✓ Cuando el software es muy cambiante.

También Fewster [1], detalla las cualidades de la automatización y los problemas comunes:

- ✓ Cualidades de la automatización
  - Poder ejecutar las pruebas existentes sobre una nueva versión del programa
  - Poder ejecutar mas pruebas en menor tiempo
  - Ciertas pruebas pueden ser muy difíciles de ejecutar manualmente
  - Mejor uso de los recursos
  - Consistencia entre los distintos ciclos de prueba
  - Poder reutilizar las pruebas
  - Incrementar la confianza en las pruebas.
- ✓ Problemas comunes de la automatización de las pruebas
  - Expectativas poco realistas
  - Poca experticia en las pruebas
  - Las expectativas que una prueba automática detectara muchos errores
  - Falso sentido de seguridad
  - Mantenimiento de las pruebas automatizadas
  - Problemas técnicos
  - Problemas de organización

2.2 Normas para realizar pruebas de sistemas

Para que las pruebas manuales o automáticas sean satisfactorias deben seguir algunas normas que han sido numeradas por varios autores, entre los cuales se detalla las dadas por Myers [2], Burstein [3].

Comunes a Myers [2] y Burstein [3]:

- ✓ Un aspecto necesario de los casos de prueba es la definición del resultado esperado.
- ✓ Las pruebas deben ser hechas por un grupo independiente de los desarrolladores
- ✓ Los casos de prueba deberían ser escritos para condiciones de pruebas inválidas e inexistentes, más que para condiciones conocidas y existentes.
- ✓ Los resultados de la prueba deben ser estudiados minuciosamente.
- ✓ Una prueba es buena cuando se logra encontrar defectos.

- ✓ La probabilidad de encontrar errores en una sección del programa es proporcional a la cantidad de errores ya encontrados en esa sección.
- ✓ El proceso de pruebas es una actividad creativa y un desafío intelectual.

Myers [2]:

- ✓ Examinar un programa para ver si este no hace lo que se supone que debería hacer es solo la mitad del camino; la otra mitad es revisar si hace algo que no se espera que haga.
- ✓ No desechar un caso de prueba al menos que el programa halla sido desechado.

Burstein [3]:

- ✓ La validación de software es un proceso en el cual se seleccionan un conjunto de casos de prueba con la intención de encontrar defectos en el programa probado.
- ✓ Las pruebas deben ser repetibles y reutilizables.
- ✓ Las pruebas deben ser planeadas.
- ✓ Las actividades de prueba deben ser integradas al ciclo de vida de desarrollo de sistemas.

2.3 Catalogación de errores comunes

Es necesario reconocer los distintos tipos de patrones de errores mas comunes que se presentan en el desarrollo de sistemas para minimizar sus efectos en la automatización; Jones [4] agrupa a los errores en categorías generales y aporta estadísticas de cada una de ellas y la relación existente entre los usuarios que utilizan los sistemas y los errores reportados.

Fuentes de errores requerimientos erróneos:

- ✓ Diseño defectuoso.
- ✓ Defectos de codificación.
- ✓ Defectos de documentación.
- ✓ Defectos mal arreglados.

Porcentajes de errores en las distintas etapas:

- ✓ Requerimientos 15%
- ✓ Diseño 40%
- ✓ Codificación 30%
- ✓ Documentación 5%
- ✓ Defectos mal arreglados 10 %

Relación que existe entre usuarios y reporte y cantidad de errores:

- ✓ El número de errores reportados es directamente proporcional a la cantidad de usuarios que utiliza el sistema
- ✓ Cuantos más defectos existen menor cantidad de ellos se encuentra.

2.4 Tipos de pruebas

Otro aspecto importante a considerar es el de reconocer los diversos tipos de prueba que se pueden llevar a cabo dentro de un proceso de validación automática. Para ello se citan algunos autores, según Black [5] define un conjunto de pruebas que se deberían realizar:

- ✓ **Prueba de unidad:** su objetivo es evaluar cada paquete o clase de sistemas, el grado de granularidad de la misma depende de lo que se deba probar.
- ✓ **Prueba de Componentes:** se realiza la evaluación de cada subsistema que conforma el sistema en su totalidad.
- ✓ **Prueba Alfa, Beta:** la prueba Alfa es hecha por el equipo de pruebas de sistema, es decir, es interna; la prueba Beta es realizada por usuarios potenciales.
- ✓ **Prueba de integración:** se evaluá si existen errores en la comunicación de los distintos subsistemas que conforman el sistema total.
- ✓ **Prueba de sistema:** se realiza la prueba funcional del sistema, es evaluar si cumple el objetivo para el que fue planeado
- ✓ **Prueba de usabilidad:** se evaluá cuan amigable es la interfaz de usuario.
- ✓ **Prueba piloto:** es una prueba conducida, guiada, en la cual participan los usuarios.

Myers [2] define en términos generales que áreas se deben probar:

- ✓ **Usabilidad:** se debe evaluar la apariencia del sistema, las fuentes, colores y gráficos que son utilizados.
- ✓ **Performance:** se debe probar el tiempo de carga, y cantidad de transacciones simultanees que soporta, en un tiempo de respuesta aceptable por el usuario.
- ✓ **Reglas de negocios:** validar que los procesos de negocios sean cumplidos.
- ✓ **Transacciones:** evaluar finalización y corrección de las transacciones hecha y la posibilidad de volver hacia atrás.
- ✓ **Integridad de Datos:** evaluar si las transacciones hechas mantienen la integridad de los datos en la base de datos.
- ✓ **Disponibilidad del sistema:** evaluar la respuesta ante fallas conocidas.
- ✓ **Arquitectura de red:** evaluar la conectividad y la carga del vínculo.

Además Myers [2] define los tipos de pruebas que son necesarios

- ✓ **Prueba de unidad de modulo:** El propósito de la prueba de unidad es el de comparar la funcionalidad de un modulo con la funcionalidad especificada definida en el modulo; el objetivo es intentar probar que el modulo contradice las especificaciones de diseño.
- ✓ **Prueba Incremental:** Es la prueba de la integración de distintos módulos del sistema a medida que se van realizando.
- ✓ **Pruebas Top-Down:** Se inicia con el primer modulo o el de mayor importancia de un programa y se va incrementados hasta probarlo completamente.
- ✓ **Pruebas Botton-Up:** A la inversa que el anterior.
- ✓ **Pruebas Funcionales:** Se intenta encontrar discrepancias entre lo que hace el programa y las especificaciones externas.
- ✓ **Prueba de Sistema:** Es probar el sistema con respecto a los objetivos originales.
- ✓ **Prueba de facilidades:** Es probar que cada funcionalidad es cumplida.
- ✓ **Prueba de Volumen:** Es probar el sistema con su carga máxima de datos.
- ✓ **Prueba de Carga:** Máximos usuarios trabajando al mismo tiempo.

Si nos centramos en el desarrollo Web deberían incluir [6] un conjunto de pruebas que se deberían agregar a las antes mencionadas:

- ✓ **Performance del Server:** es evaluar la capacidad del servidor Web y de aplicación.
- ✓ **Prueba de Seguridad:** realizar una prueba de penetración de sistema
- ✓ **Prueba de Memoria:** evaluar la cantidad de memoria utilizada por la aplicación para una carga máxima de usuarios.
- ✓ **Prueba de Firewall:** evaluar las reglas del mismo.

En la presente sección se han sentado las bases para la confección de un plan de pruebas el cual contemple la automatización del mismo.

3. PLAN DE PRUEBAS

Antes de realizar la automatización de la validación de sistemas es necesario definir y planificar los pasos a seguir, evaluar riesgos y contingencias del mismo; es por esto se citan las principales características que deberían tener un plan de pruebas; según Black [5] los pasos de un plan de pruebas deberían ser:

- ✓ *Planificar*: Es el proceso de entender el esfuerzo de la prueba, que se divide en: **(a)** contexto de la organización, **(b)** definir y priorizar riesgos, **(c)** estimar esfuerzo, **(d)** desarrollar un plan de tareas.
- ✓ *Preparar*: Es el proceso de especificar los recursos y las pruebas a realizar, esto se divide en: **(a)** capacitar de ser necesario a los recursos, **(b)** diseñar y evaluar el sistema de prueba.
- ✓ *Ejecutar*: Es el proceso de realizar la prueba, que se divide en: **(a)** ejecutar la prueba, **(b)** documentar el resultado.
- ✓ *Evaluar*: Es el proceso de hacer las correcciones para reiniciar el ciclo, esto se divide en: **(a)** adaptar y mejorar, **(b)** documentar errores, **(c)** comunicar la prueba, **(d)** ajustar cambios.

Para Stottlemyer [6] un plan de pruebas tiene como objetivo el desarrollo de una hoja de ruta para la realización de las pruebas de sistema. Este debe ser documentado y aprobado por los participantes del mismo. La estructura puede ser definida en: **(a)** nombre de proyecto de prueba, **(b)** propósito del documento, **(c)** equipo de prueba, **(d)** riesgos, **(e)** alcance de la prueba, **(f)** ambiente de prueba, **(g)** datos de prueba, **(h)** herramientas de prueba, **(i)** documentación, **(j)** seguimientos de problemas.

En el mismo se debería cuantificar: **(a)** tiempo de respuesta del sistema, **(b)** disponibilidad del sistema, **(c)** seguimiento de problemas encontrados, **(d)** definición de los entregables, **(e)** expectativas de la prueba, **(f)** documentación adecuada, **(g)** estrategias para llegar a los objetivos.

Por ultimo da una lista de tareas para una prueba pueda ser realizada con éxito se debería detallar: **(a)** quien hará la prueba, **(b)** porque se debe hacer esta prueba, **(c)** que se debería probar, **(d)** quienes escribirán los Scripts de prueba, **(e)** que desarrolladores participaran de la prueba, **(f)** que usuarios participaran de la prueba, **(g)** cuando se hará la prueba, **(h)** que documentación se entregar al equipo de auditoria, **(i)** quien seguirá los problemas detectados, **(j)** cual el ambiente de prueba.

Según Craig [7] contempla que los aspectos que debe contemplar un plan de pruebas son: **(a)** determinar la prioridad de pruebas y riesgos, **(b)** priorizar que probar, **(c)** determinar hasta donde probar, **(d)** planificar los riegos y las posibles contingencias.

Para Dustin [8] en un plan de pruebas se debe incluir los siguientes ítems: **(a)** cual es el motivo de la automatización, **(b)** detalle del método de selección de herramientas, **(c)** infamación de la prueba a realizar, **(d)** roles y responsabilidades, **(e)** proceso de diseño y desarrollo de la prueba, **(f)** necesidades de administración, **(g)** método de seguimiento de defectos, **(h)** método de seguimiento de la prueba y **(i)** métricas de la prueba.

3. Metodología del plan de pruebas automático

Existen trabajos en los cuales se hace referencia a metodologías para la automatización de la validación de las pruebas de sistemas, entre las mas sólidas se puede nombras a la propuesta en [8] es la denominada Ciclo de Vida Automatizada de las Pruebas (ATLM Automated Testing Life Cycle Methodology) en la misma se propone seis pasos:

- ✓ Decisión de automatizar la prueba
- ✓ Selección y evaluación de herramientas
- ✓ Recolección de información de las pruebas a realizar
- ✓ Planear, diseñar y desarrollar la prueba
- ✓ Ejecutar y administrar la prueba
- ✓ Analizar los resultados y sacar conclusiones.

ATLM cuenta con varias iteraciones para llegar al proceso final de validación automatizado.

Una metodología que no necesariamente debe ser automatizada es la conocida con el nombre de Modelo V [1] en donde se demuestra que actividades de prueba se deben realizar durante el ciclo de desarrollo de sistema (CDS). El factor mas importante a tener en cuenta en el Modelo V es el de identificar cuando escribir las pruebas y cuando ejecutarlas (Gráfico 2). El objetivo del mismo es relacionar que tipo de prueba debe ser realizada para cada etapa del CDS; a medida que se va realizando una nueva etapa del CDS se escribe la prueba relacionada con esa etapa para ser ejecutadas en orden inverso.

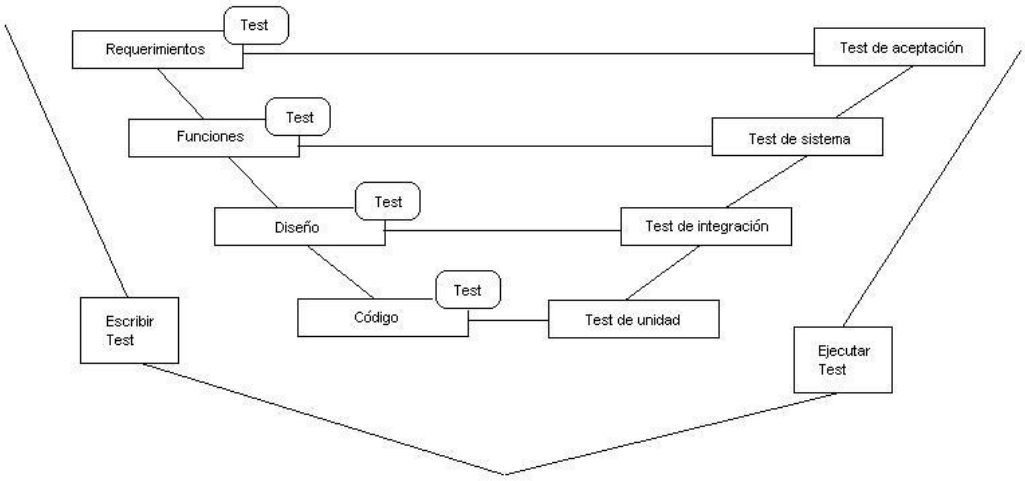

Gráfico 2: Modelo V

A lo largo del ciclo de vida deben ser confeccionadas y ejecutadas distintos tipos de prueba, las cuales son soportadas por distintas herramientas de prueba en el Gráfico 3 se da una breve guiá de las características deseables que debería tener la herramienta seleccionada.

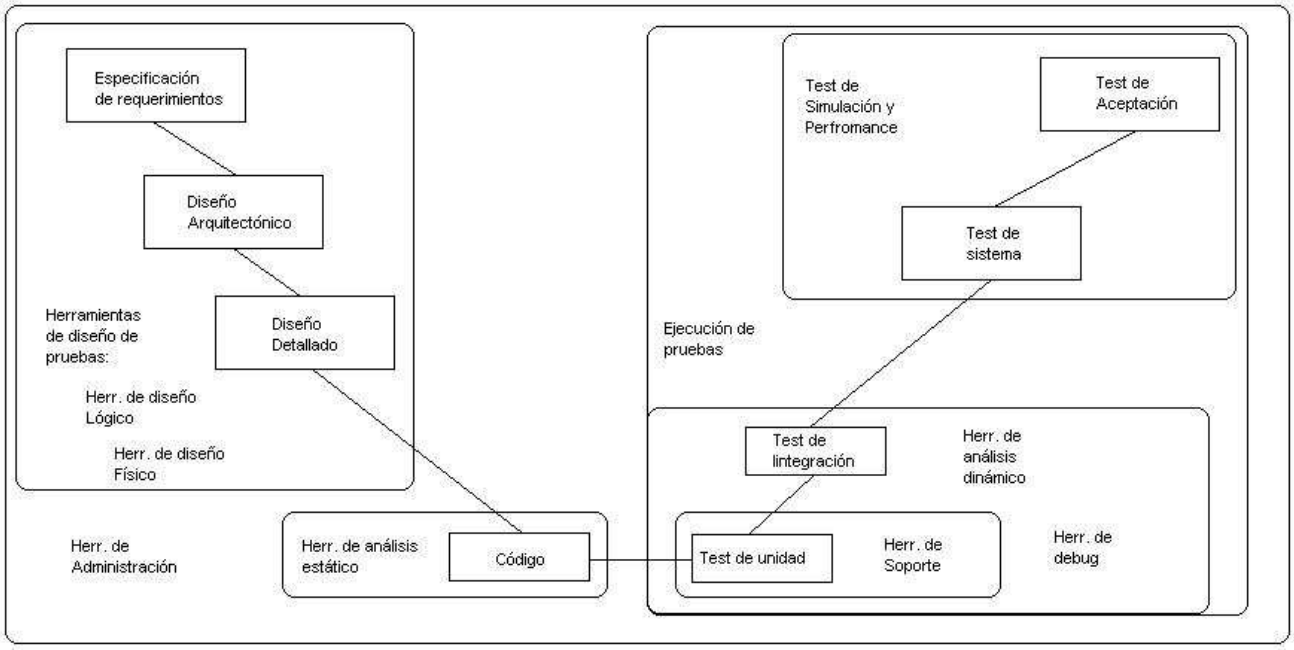

Gráfico 3. Tipo de pruebas

Además se hace hincapié en las actividades que se deben llevar a cabo a lo largo de todo el proceso de desarrollo (Gráfico 4)

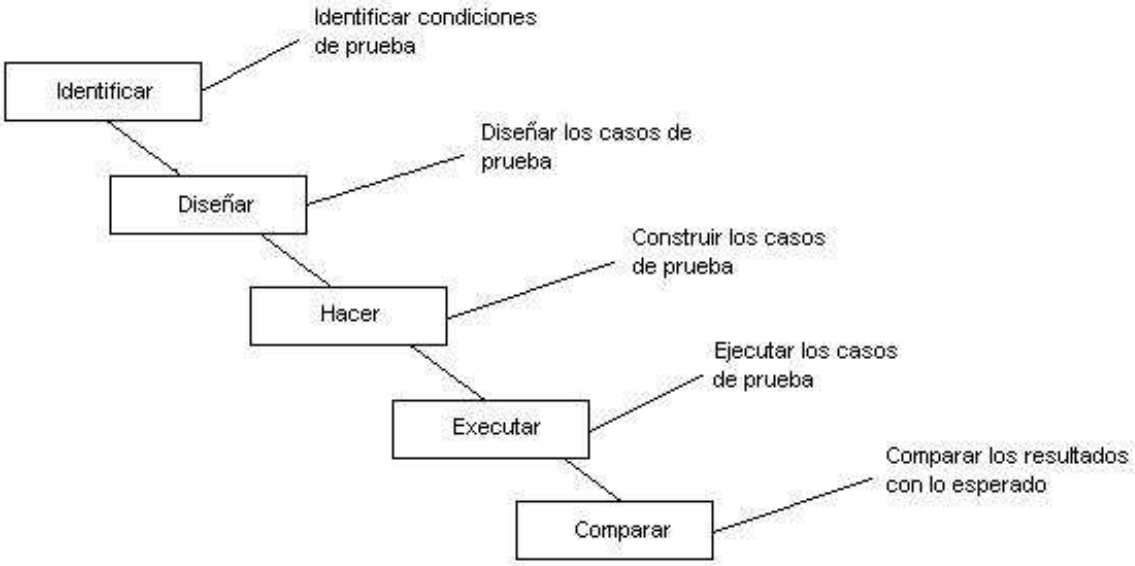

Gráfico 4. Proceso de desarrollo

4. HERRAMIENTAS PARA EL PROCESO DE AUTOMATIZACIÓN Y VALIDACIÓN

La presente lista de herramientas no pretende ser exhaustiva pero si representativa (tabla 1):

Automatización de la puesta en producción	
Producto	**Descripción**
CruiseControl	Framework para realizar el proceso de construcción continua, a este se le pueden adjuntar otras herramientas para complementar el proceso de validación automática, se lo puede considerar como la base principal donde se construirá todo el proceso de automatización.
Anthill	Framework para realizar le proceso de construcción continua y de similares características a CruiseControl.
Ant	Framework para realizar le proceso de construcción, muy configurable.
Validación funcional de sistemas	
Selenium	Framework para probar funcionalidad, imita lo que haría un usuario, principalmente orientado al explorador Mozilla.
Fitnesse	Herramienta que mejora la colaboración en el proceso de desarrollo de software orientada a definir un conjunto de pruebas de aceptación. La principal función de este paquete es la de hacer comprender a los usuarios, programadores y evaluadotes de calidad que hace el software y en forma automática comparar los que realmente hace y lo que debería hacer.
FIT	Herramienta que mejora la colaboración entre los distintos integrantes del desarrollo y sirve para la automatización de las pruebas de aceptación utilizando JUnit para la comprobación
iValidator	Framework basado en XML para la ejecución de escenarios complejos de prueba de funcionalidad de sistemas.
Watir	Framework para probar funcionalidad orientado al explorador de Microsoft
Clirr	Valida funcionalidad actual con versiones anteriores de la maquina virtual de Java.
Validación de unidad de sistemas	
JUnit	Framework de validación de unidad orientado al lenguaje Java.
Checkstyle	Herramienta para la validación de normas de programación.
JDepend	Herramienta para obtener métricas de extensibilidad, usabilidad y mantenibilidad de sistemas.
PathFinder	Herramienta que explora sistemáticamente todos los caminos de ejecución de un programa para detectar abrazos mortales u otras inconsistencias.
FindBugs	Herramienta que detecta patrones comunes de error en Java.
Carga de sistemas	
Grinder	Framework par pruebas de stress de un sistema principalmente orientado a sistemas Web. Este trabaja en la modalidad de Proxy Server con múltiples procesos que imitan ser usuarios

Tabla 1. Herramientas

5. CONCLUSIONES

Se pueden extraer algunas conclusiones sobre el presente trabajo; **(a)** la mejor manera de validar un sistema es relacionar cada requerimiento tanto sea funcional como técnico a un paquete de pruebas; **(b)** la selección de una herramienta de integración continua es la base de todo proceso de validación automática; **(c)** las pruebas no solo deben ser funcionales sino además, como mínimo, de carga de sistemas; **(d)** todo el proceso debe ser detallado en un plan donde se debe contemplar el resultado esperado de cada prueba. Para finalizar se puede mencionar que la validación automática de sistemas no es un proceso trivial, en el cual se debe invertir esfuerzo y recursos para su concreción, el mismo debe ser incluido en el plan de sistemas para su evaluación.

6. APÉNDICE A

6.1. Metodología propuesta

El método propuesto esta organizado en torno a una premisa:

"*Todo los equipos de desarrollo son distintos y enfrentan distintos desafíos*"

Que automatizar y que no, es el desafió al que debe enfrentarse el líder de proyecto, esto se debe pues en el equipo que lidera esta formado con individuos de diferentes intereses y capacidades.

Sobre la construcción del plan de sistemas se recomienda seguir los lineamientos de Stottlemyer [2001], sobre la metodología a utilizar **Modelo V** [1] es una metodología que se adapta en gran medida a la media de los sistemas comúnmente construidos por los ingenieros en software.

Con respecto a la automatización la recomendación es la implementación de las siguientes herramientas, se ha elegido un ambiente de desarrollo Web y desarrollado en Java a modo de ejemplo, utilizar **CruiseControl** como herramienta de integración continua, integrado al control de versiones **Sub-Version**. La carga de las versiones nuevas de software debería ser evaluada por **PathFinder**, **FindBugs** y **Checkstyle** antes de ser ingresadas en el repositorio de código de Sub-Version. Para la construcción de software para los distintos entornos, Prueba, Pre-Producción y Producción se utilizara **Clirr** y **Jdepend**, todos estos procesos son integrados a través de **Ant**.

Para las pruebas funcionales se debería utilizar **Selenium** para validar funcionalidad desde el explorador Mozilla y **Watir** para validar funcionalidad en el explorador de Microsoft. Para las pruebas de carga se utilizaría **Grinder**.

Acerca de la utilización de frameworks como **JUnit** para la prueba de unidad se le recomienda a los programadores utilizarlo, pero no se los obliga, la razón de esto es que escribir pruebas validas se debe tener un alto grado de experticia.

Sobre la validación de sub-módulos el mismo si debería ser realizado con herramientas como JUnit pero las mismas deben ser tareas incluirlas en le plan de pruebas.

REFERENCIAS

[1] Fewster, M. 1999. *Software Test Automation.* Addison-Wesley
[2] Myers, G. 2004. *The art of software testing.* John Wiley & Sons, Inc.
[3] Burstein, I. 2002. *Practical software testing.* Springer Verlag
[4] Jones, C. 1995. *Applied Software Measurement.* Springer Verlag
[5] Black, R. 2005. *Critical Testing Processes*. Addison-Wesley.
[6] Stottlemyer, D. 2001. *Automated Web Testing Toolkit.* John Wiley & Sons, Inc.
[7] Craig, R., Jaskiel, S.. 2002. *Systematic Software Testing*. Artech House.
[8] Dustin, E.. 2002. *Effective Software Testing: 50 Specific Ways to Improve your testing. Editorial*