Tabla de Contenidos

REVISO	APROBO
ANA BEIBA POVEDA ATUESTA Profesional especializado 30-06-2015	CELENIA LISSETT VARELA GOMEZ Jefe Oficina Tecnologias de la Información y las Comunicaciones 30-06-2015

Listado de Gráficas

Listado de Tablas

1. Resumen ejecutivo

El Ministerio de Agricultura y Desarrollo Rural requiere información de calidad y en tiempo real para poder tomar decisiones oportunas y acertadas. Para lograr esta información depende de sistemas informáticos, varios de los cuales están por construirse.

Para poder crear productos de software que realmente generen valor al Ministerio o a los ciudadanos es necesario entender primero realmente que es lo que estos necesitan y segundo ser capaz de satisfacer esas necesidades de forma oportuna y con calidad.

El desarrollo de software es una disciplina relativamente nueva, con muchas historias de éxito pero igualmente con muchas de oportunidades de mejora que apuntan en su mayoría al proceso que se sigue y las herramientas disponibles para soportar ese proceso.

Este documento detalla de las recomendaciones a seguir para la construcción de software a la medida en el marco del centro de excelencia del Ministerio de Agricultura y Desarrollo Rural.

2. Definición del ciclo de vida de aplicaciones

Los procesos alrededor de la construcción de productos de software se denominan procesos de ALM, “Application Lifecycle Management”, y típicamente persiguen varios objetivos:

- Proteger la propiedad intelectual – activos de software y/o de información
- Medir, mejorar y mantener la calidad de los productos de software
- Habilitar y mejorar la comunicación entre los miembros del equipo que produce el software
- Definir roles y sus responsabilidades
- Proveer trazabilidad entre los requerimientos y el trabajo para satisfacer y probar estos requerimientos
- Proveer consistencia, confiabilidad e integridad en los procesos de desarrollo de software

Definimos entonces ALM como el proceso continuo para la administrar una aplicación a través del gobierno, desarrollo y mantenimiento. ALM es la unión del negocio y la ingeniería de software facilitado por herramientas que integran manejo de requerimientos, pruebas, arquitectura, codificación, seguimiento y liberaciones.

a. Modelo de entrega continua

La estrategia de ALM a seguir en el Ministerio de Agricultura y Desarrollo Rural se basa en un modelo de entrega continua de valor que se centra en 4 aspectos fundamentales detallados a continuación:

Gráfica 1 - Modelo ALM para MADR

i. Agilidad

Dada la velocidad con la que cambia el entorno, el Ministerio necesita poder ser ágil para entregar resultados de forma rápida y continua.
Esto implica adoptar ciertas prácticas, desde interiorizar los cambios en procesos que permitan responder rápidamente a los cambios, hasta establecer colaboración entre desarrollo y operaciones, y adquirir una forma de pensar adaptativa y exploratoria. Las organizaciones ágiles se suelen caracterizar por:

- Tiempos cortos de entrega (Time to market)
- Todas las actividades se centran en generarle valor a la organización
- No se rechaza el cambio, al contrario, se promueve

ii. Velocidad

La velocidad se enfoca en el proceso que facilita la agilidad descrita anteriormente. La habilidad y velocidad con la que se entregan nuevas soluciones o nueva funcionalidad es clave para el éxito del Ministerio y sus programas.
Esto requiere que la organización como un todo colabore eficientemente y los procesos manuales se automaticen lo más posible a lo largo del ciclo de las aplicaciones. Este aspecto se enfoca en los procesos que materializan la agilidad, sus características son:

- Ciclos cortos en entrega
- Ambientes estables

- Procesos automatizados
- Foco en entregar funcionalidad sobre foco en documentación
- La estimación es una guía, no un contrato
- El avance del trabajo se mide en el valor que se logra para el negocio
- El manejo de requerimientos se implementa a través de historias de usuario y trabajo acumulado (Product Backlog)

iii. Calidad

La calidad del software impacta directamente tanto la agilidad del negocio como la satisfacción de sus usuarios. Para construir software de alta calidad es necesario incorporar ciclos cortos de retroalimentación. Estos ciclos miden la efectividad de las herramientas y procesos aplicados. A la vez, debe validarse que se están logrando los objetivos de negocio observando el comportamiento de los usuarios y capturando su retroalimentación lo que puede conducir a cambios en el software e incluso la revisión de los objetivos de negocio.

La implementación de ciclos cortos de retroalimentación y hacer de la calidad responsabilidad de todo el equipo produce software que realmente cumple las necesidades del usuario acompañado de muchos menos incidentes. Los atributos de software de calidad suelen ser:

- Menor cantidad de incidentes
- Incidentes de menor impacto
- Mejora en el tiempo de recuperación de incidentes
- Resiliencia a las fallas
- Manejo de fallas como equipo

Trabajo en equipo

Las soluciones de tecnología de alta calidad requieren trabajo en equipo. Los patrocinadores del negocio, desarrolladores, testers y administradores de infraestructura necesitan trabajar juntos, como una única entidad con un fin común. Es muy importante que todos asuman responsabilidad y se sientan dueños del proceso de desarrollo y entrega del software. Esto significa que si algo sale mal no se trata de buscar quien fue el que cometió un error sino trabajar juntos como equipo para resolver el problema y aprender cómo prevenirlo en el futuro, tratando cada inconveniente como una oportunidad de aprendizaje y de mejora del proceso. Las características del equipo son:

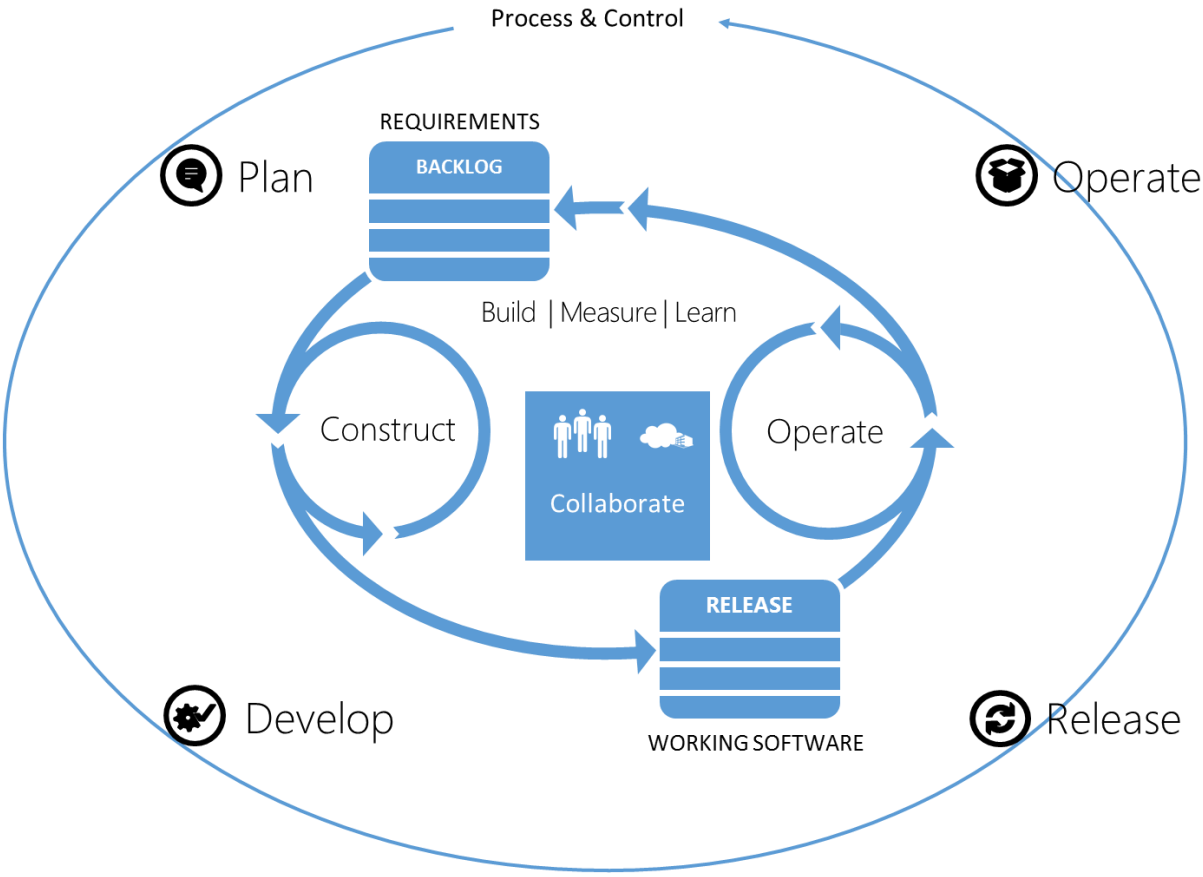
- Todos trabajan juntos como iguales
- El éxito es una responsabilidad compartida
- Los patrocinadores y el equipo de desarrollo trabajan juntos
- Los miembros del equipo se tratan con respeto

b. Fases del ciclo de vida

El ciclo de vida para una aplicación moderna es aquel que empodera al equipo para entregar valor continuamente, a un ritmo que balancea agilidad y calidad quitando del medio los silos tradicionales que

separan a los desarrolladores del grupo de operaciones, mejorando la comunicación y colaboración dentro del equipo y creando conexión entre las aplicaciones y los resultados de negocio que producen.

El ALM moderno comprende planear, desarrollar, liberar y operar. Estos pasos se repiten en un ciclo de entrega continua de valor. Dentro del ciclo, se construye el software, se mide su efectividad para entregar su objetivo esperado y aprender del comportamiento esperado, tanto para mejorar el software como para ajustar los objetivos de negocio que se persiguen o el proceso de desarrollo que se está siguiendo.



Gráfica 2 - Ciclo de ALM Moderno

El ALM moderno consiste en 4 fases de alto nivel:

c. Planear

Comienza con formular la hipótesis de negocio, especificando mediante elementos de trabajo pendiente las historias de usuario que se construirán y brindarán los criterios para medir el valor al negocio entregado por el software. En algunas ocasiones en la primera iteración de esta fase se desarrollan actividades de prototipado para bajar la incertidumbre y el riesgo al llevar desde un principio los requerimientos generales a un nivel más concreto para lograr un mejor entendimiento de los mismos y potencialmente tener mejores estimaciones. Esta práctica hace parte de la metodología ISD (Iterative

Software Development) donde se denomina Sketch y produce artefactos como prototipos y wireframes de las interfaces de usuario. Esta práctica se utilizará en los desarrollos de software a la medida que se realicen en el Ministerio de Agricultura y Desarrollo Rural en el marco del centro de excelencia.

Durante esta fase se producen y afinan iteración por iteración muchos artefactos que definen el qué y el cómo, a un alto nivel para toda la aplicación, y con un gran detalle para la iteración que inicia.

A continuación se describen los posibles artefactos a producir o actualizar en esta fase, es muy importante resaltar que no todos los artefactos aplican en todos los casos, algunos serán generales para cualquier proyecto del Ministerio de Agricultura y Desarrollo Rural y otros debido a las características del proyecto no aplicarán para el mismo, esto a criterio del equipo de desarrollo.

Artefacto	Descripción
Elementos de diseño	Es un documento de diseño técnico donde se describe el diseño de la aplicación mediante diagramas como diseños físicos, lógicos, conceptuales, de capas, de arquitectura, de despliegue, de flujo, de secuencia, de clases o de entidad relación, según sean relevantes para la construcción del sistema en cuestión. Este documento evolucionará junto con la aplicación y su nivel de detalle debe justificarse en la medida que realmente genere valor al equipo de desarrollo pues el objetivo de la metodología y el proyecto es entregar valor al ministerio y los ciudadanos con software funcionando, no llenar anaqueles con documentación exhaustiva.
Wireframes prototipo	Son bosquejos visuales de las interfaces de usuario final de la aplicación y los flujos de una pantalla a otra. La representación gráfica de las pantallas es un facilitador muy valioso para capturar y discutir los requerimientos con usuarios y otros stakeholders. En el centro de excelencia este artefacto resulta inicialmente del ejercicio de Sketch de ISD, pero también se produce continuamente como parte de los requerimientos que se capturen.
Lista de requerimientos	Este artefacto se explica en la sección 3.c.i por ser uno de los artefactos diferenciados en SCRUM. No se limita a requerimientos, pueden ser defectos, cambios y mejoras entre otras, en general es la lista de todo lo que hay por hacer en el producto de software. Cada elemento de esta lista además de la descripción tiene una prioridad y un esfuerzo estimado, y la lista está ordenada de manera que en cada iteración se tomen los elementos que están más alto en la lista. Físicamente esta lista y cada uno de sus elementos existen como una de las abstracciones que brinda la herramienta de ALM llamada “Product Backlog Item”. En cada iteración se revisa y detalla esta lista tal como se describe más adelante en este documento en los eventos de planeación, retrospectiva y revisión de la iteración.
Lista de requerimientos de la iteración	Este artefacto se detalla en la sección 3.c.iii por ser uno de los artefactos de SCRUM. Sale de la lista general de pendientes del producto y son aquellos elegidos por el dueño del producto para ser construidos en la iteración que comienza y que además tienen el suficiente nivel de detalle

	y entendimiento por parte del equipo.
Plan de pruebas	Son los casos de prueba que resultan de los criterios de aceptación de cada requerimiento y en los casos que aplique las pruebas que evalúen aspectos no funcionales como desempeño y carga. Físicamente esta lista, sus reportes de avance y reporte de resultados yacen en la herramienta de ALM.

d. Desarrollar

En cada iteración se eligen algunas funcionalidades priorizadas por el negocio y acordadas con el equipo para ser construidas y probadas.

Artefacto	Descripción
Código fuente	Cuando aplique, el código fuente producido por los desarrolladores, validado por los criterios descritos en la sección 6 de este documento
Pruebas unitarias	El código fuente va acompañado de código de prueba que se encarga de validar a nivel lógico y de negocio. Si por ejemplo dentro del código existe una sección de lógica de negocio que implementa las reglas para determinar si una familia es candidata para recibir subsidio de vivienda entonces debe existir otra sección de código con pruebas de esa lógica como "SiLaFamiliaEsSisben2EsCandidata" , "SiLaFamiliaEsSisben5YLaCabezaEsReinsertadoEsCandidata" , etc.
Pruebas de integración	Similares a las unitarias en cuanto a ser elementos de código pero no abarcan solamente lógica de negocio sino pueden probar aspectos de integración con otras secciones de código u otros sistemas.
Pruebas de sistema	Son pruebas que ejercitan el sistema como un todo. Es muy deseable que sean automatizadas pero dependiendo de las tecnologías involucradas su automatización puede requerir un significativo. La herramienta de ALM del Ministerio de Agricultura y Desarrollo Rural permite de forma relativamente sencilla hacer pruebas de sistema automatizadas para aplicaciones Web.
Pruebas de aceptación	Agrupan pruebas de sistema representando casos de uso completos y frecuentemente involucran también aspectos no funcionales como desempeño y carga. La herramienta de ALM del Ministerio de Agricultura y Desarrollo Rural permite de forma relativamente sencilla hacer pruebas de aceptación automatizadas para aplicaciones Web.
Scripts de instalación	Cuando apliquen, los scripts que se requieran para desplegar la aplicación en un ambiente específico
Artefactos de configuración	Cuando apliquen, los artefactos que configuran alguna herramienta o sistema de software para lograr el objetivo de negocio propuesto en la iteración. En casos como desarrollo para Sharepoint o Dynamics CRM muchos artefactos serán más de esta naturaleza que código como tal.
Guía, ayuda en línea o manual	Cuando apliquen, son los textos, ilustraciones o videos explicativos sobre el uso del software por parte de usuarios finales

de usuario final	
Elementos de diseño gráfico	Cuando apliquen, son artefactos que son parte de la aplicación pero no son código sino elementos de diseño gráfico como ilustraciones, fotografías, hojas de estilo en cascada (CSS) y demás artefactos similares

e. Liberar

Continuamente liberar versiones que agregan funcionalidad al negocio. Esta libración suele ocurrir por ambientes para dar espacio a pruebas automáticas y manuales.

Artefacto	Descripción
Software funcional	El software funcionando a disposición de usuarios finales o un grupo dedicado de pruebas
Manual de instalación	Cuando aplique, un documento técnico detallando los pasos necesarios para instalación y rollback de versión del producto. La intención en general es que en el Ministerio de Agricultura y Desarrollo Rural los despliegues ocurran de una forma altamente automatizada apoyados en la herramienta de ALM y otras herramientas, sin embargo en los casos donde esto no sea posible o su costo no se justifique se acudirá a la instalación manual que se describirá en este documento
Manual de operaciones	Este artefacto se puede actualizar tanto en esta fase como en la anterior y es un documento donde se describen las operaciones administrativas o de solución de problemas inesperados del software como creación y desactivación de usuarios, copias de seguridad, reinicio de la aplicación

f. Operar

Aprender del software, su comportamiento y el de sus usuarios para brindar retroalimentación a la primera fase.

Artefacto	Descripción
Retroalimentación	Esta viene de los usuarios finales y cualquier otro stakeholder del producto bien sea en forma de nuevos requerimientos, solicitudes de cambio o reportes de defectos que se introducirán a la lista de pendientes del producto tal como se describe en la sección 3.e.v
Indicadores de salud de la aplicación	Es la disponibilidad de contadores de desempeño, uso o diagnósticos de la aplicación en una consola propia de la aplicación o en el portal de Azure con *Application Insights* y/o centralizado en el ministerio a través de productos como System Center. Estos contadores muestran aspectos como cantidad de usuarios, cantidad de transacciones, carga en la infraestructura y otros aspectos

	operacionales de la aplicación que permitan tomar acciones informadas y oportunas para permitir a la aplicación satisfacer los objetivos de negocio.

3. Especificación del proceso de desarrollo

Los desarrollos de software a la medida que se ejecuten en el marco del centro de excelencia se rigen inicialmente por las generalidades de ALM descritas en el capítulo 0 y la particularidad de ISD - Sketch en la primera iteración de la fase de planeación. Excepto por esta particularidad el proceso de desarrollo en sí, llamado ADLM (Application Development Life Cycle Management), se regirá por el marco metodológico ágil SCRUM.

a. Definiendo SCRUM

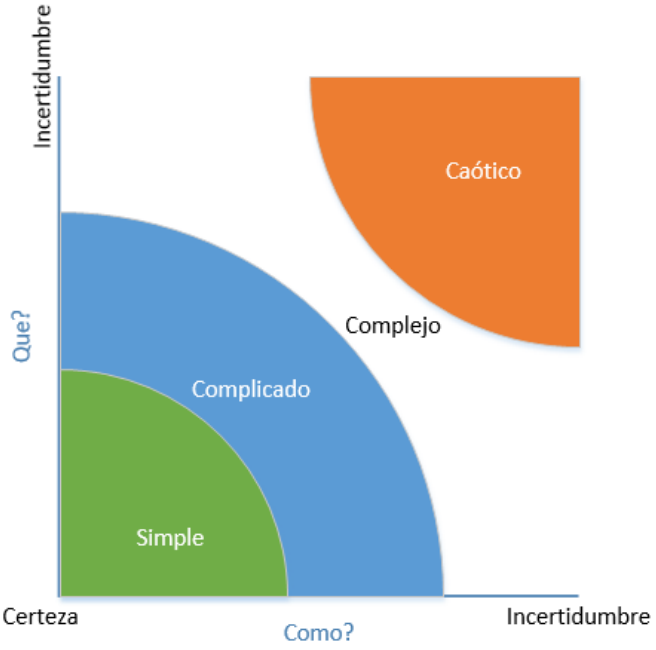
El Co-creador de SCRUM Ken Schawber lo define de esta manera: SCRUM es un marco de trabajo (Framework) para el desarrollo de productos complejos. Se basa en la teoría empírica de control de procesos. SCRUM emplea un proceso iterativo con enfoque incremental para optimizar la predictibilidad y controlar los riesgos.

La primera observación importante sobre esta definición es que SCRUM es un Framework. Es una estructura que puede acomodar dentro de sí otras técnicas, prácticas o herramientas conformando a la larga un proceso. Los frameworks no prescriben una manera de resolver problemas lo que contrasta con una metodología que es un sistema más completo de orientación prescrita, los comportamientos y las herramientas que trabajan juntos para conseguir algún resultado particular.

SCRUM es un framework ligero que puede coexistir limpiamente e incluso apoyar otras metodologías y técnicas de desarrollo. Cosas tales como KANBAN, PRINCE2, CMMI, Lean y programación extrema pueden funcionar muy bien conjuntamente con SCRUM.

b. Definiendo complejidad

SCRUM está especialmente afinado para abordar el trabajo que existe en un dominio complejo por lo que es necesario definir complejidad. Ralph Stacey, profesor de la Universidad de Hertfordshire utiliza la siguiente ilustración para definir complejidad:



Gráfica 3 - Complejidad de procesos

Cuando estamos hablando de la construcción de software hay dos aspectos esenciales a considerar: qué se va a construir y cómo se va a construir.

Si hay certeza de lo que va a construir y seguridad de cómo se va a construir, el trabajo de hacerlo entra en el dominio simple. Si en cambio, hay cierta incertidumbre de cómo se va a construir o lo que se construirá que es a menudo el caso en los proyectos de desarrollo de software se puede llamar a este trabajo complicado.

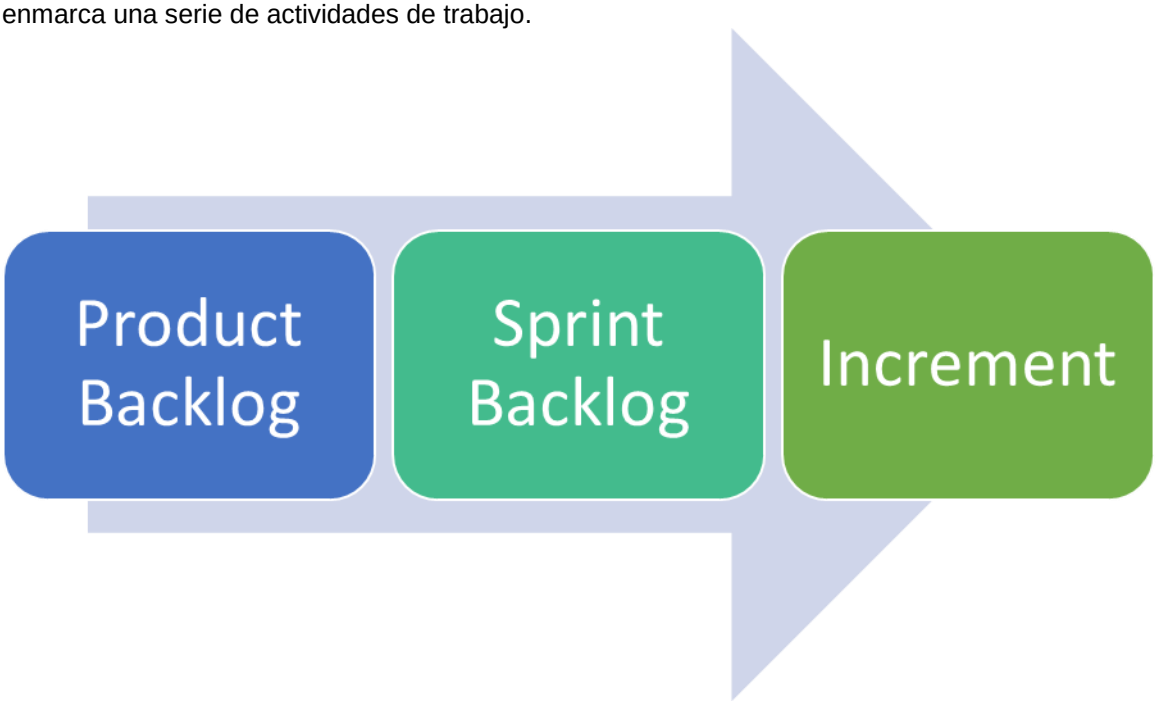
En el extremo opuesto de esta gráfica es el mundo en el que se sabe muy poco tanto sobre el qué y el cómo. Este es el reino del caos y en estos casos no hay mucho que hacer, no habrá Framework que pueda ayudar.

SCRUM vive justo en el medio de estos niveles de complejidad, es apropiado para esta zona de la complejidad. Es decir, "no necesariamente se sabe lo que se va a entregar o cómo se va a entregar". Las cosas cambiarán a medida que avance el proceso creativo.

Significa que el área del dominio de negocio puede ser un poco borrosa o en el área de la tecnología puede existir incertidumbre o tal vez no está complemente definido desde un principio como va a interactuar el equipo de trabajo. SCRUM es ideal para trabajar en el ámbito complejo de la matriz Stacey donde no se sabe exactamente desde un principio cuáles son los pasos que se tomarán para entregar el producto de software.

c. Artefactos de SCRUM

SCRUM sólo diferencia tres artefactos, el *Product Backlog*, *Sprint Backlog* y el incremento. *Backlog* se puede traducir como una lista de pendientes y *Sprint* como un periodo corto de tiempo en el que se enmarca una serie de actividades de trabajo.



Gráfica 4 - Artefactos de SCRUM

i. El product backlog

La lista de pendientes del producto es la única fuente de la verdad para cualquier cambio que se introduzca en el software. Es una lista ordenada con todo lo que el producto necesita y contiene un elemento por cada cambio que se planea hacer.

En esta lista puede haber características, funciones, defectos, requerimientos, mejoras y cambios. Cada elemento de la lista tiene como mínimo una descripción, orden y un esfuerzo estimado.

El dueño de esta lista es el dueño del producto (*Product Owner*), rol que se describirá más adelante en este documento. El product owner es el único responsable sobre los elementos que entran a la lista, como cambian, y como se ordenan. Esta lista se considera “viva”, porque está continuamente cambiando, ajustándose.

Al inicio de cada sprint el equipo SCRUM selecciona algunos elementos de la parte más alta de la lista (la lista está ordenada en base a riesgo, esfuerzo, que tan necesario es el elemento, todo a criterio del product owner) y esos son los elementos que se entregarán al final del sprint.

Gráfica 5 - El product backlog como lista ordenada

A menudo se incluyen elementos muy grandes en el Product Backlog que ciertamente no cabrían dentro de un Sprint. Estos elementos pueden descomponerse en elementos más pequeños dándoles a la vez mayor detalle para que estén listos para su ejecución y posiblemente suban en la lista ordenada. Esto es en todo caso responsabilidad y decisión del dueño del producto quien puede decidir no pasar estos elementos a la parte superior del product backlog lo que significa que posiblemente nunca se hagan.

No es raro que el product owner incluya cualquier petición del usuario como un elemento del Product Backlog pero lo priorice u ordene de forma muy baja. Esto puede conducir a elementos en el product backlog que nunca se trabajarán. No hay ninguna regla en SCRUM que rige esta decisión y no hay una sola respuesta para la forma correcta de manejar esta situación. El product owner, que es el responsable de esta lista de pendientes va a tener que hacer su trabajo y tomar esta decisión de acuerdo con el contexto del momento y el stakeholder que solicita la característica entre otras cosas. Este tema puede resultar delicado en el Ministerio de Agricultura y Desarrollo Rural, donde puede malinterpretarse como requerimientos incumplidos, y donde además se planea brindar un nivel de apertura donde el Product Backlog está disponible para cualquiera que lo quiera revisar y cualquiera puede insertar elementos para consideración del dueño del producto por lo que son factores críticos de éxito el entendimiento del Product Backlog y el buen manejo del mismo de parte del Product Owner.

ii. El incremento

El incremento es el resultado de un Sprint exitoso. El incremento del producto incluye todas las características de los elementos del Product Backlog que se hicieron en el último Sprint acumulativamente

junto con el trabajo de todos los Sprints anteriores, es decir, cada incremento que se libere de cada Sprint traerá consigo las características que ya estaban allí. El dueño del producto tiene la facultad de decidir qué hacer con el incremento, por ejemplo liberarlo a los usuarios. El equipo de desarrollo es responsable de que cualquier incremento es un producto entregable a los usuarios finales al asegurar que no hay nada pendiente en el incremento y para asegurarse de esto el Equipo SCRUM debe tener una definición de lo que significa "nada pendiente" o lo que en SCRUM se denomina la "definición de listo" (*Definition of done*). La "definición de listo" es una lista de verificación de los elementos que debe ser ciertos antes de poder llamar un elemento del product backlog como listo y presente en un incremento. Esta lista debe ser acordada y aceptada por todo el equipo de SCRUM.

iii. El Sprint Backlog

Es un plan con suficiente detalle que el equipo de desarrollo utiliza en todo el Sprint para orientar sus decisiones y evaluar su progreso hacia la creación del incremento ideal para ese Sprint. El Sprint Backlog lo crea el equipo de desarrollo y es para el equipo de desarrollo. La definición formal del Sprint Backlog es el conjunto de elementos del Product Backlog que han sido seleccionados para ser entregados en el Sprint actual, más un plan para convertir esos pendientes en un incremento al software ya liberado hasta ahora.

Una de las características del Sprint Backlog es que debe ser capaz de mostrar el trabajo que queda en el Sprint por lo que el trabajo pendiente se calcula al menos de forma diaria.

El Sprint Backlog también está ahí para garantizar que todo el equipo de desarrollo tenga visibilidad del trabajo comprometido para cumplir con la meta del Sprint actual.

Otra característica del Sprint Backlog es que cambiará constantemente durante el Sprint y esto ocurre porque el equipo de desarrollo aprende más acerca de su trabajo desde que comienza a trabajar en el primer día del Sprint, por lo tanto, el plan para la entrega de ese incremento va a cambiar para reflejar la nueva comprensión del trabajo comprometido y como se va a ejecutar ese trabajo.

Sólo el equipo de desarrollo puede cambiar el Sprint Backlog durante el Sprint y lo suele hacer constantemente. El equipo de desarrollo reordena el Sprint Backlog a medida que aprende sobre el trabajo necesario para lograr el objetivo del Sprint.

iv. Meta del sprint (Sprint Goal)

El Equipo SCRUM crea una meta para cada Sprint. Esta meta se crea normalmente durante el proceso de planificación de Sprint y es el resultado de un esfuerzo colaborativo entre todos los miembros del equipo de SCRUM luego de revisar todos los elementos del Backlog que se trabajarán durante el Sprint. La meta del Sprint ofrece el equipo de desarrollo el "porque" se está construyendo el incremento actual y como tal debe ser visitado con frecuencia por el Equipo de Desarrollo para asegurar foco en el trabajo correcto del Sprint.

v. Definición de listo (Definition of Done)

Si bien este no es un artefacto formal del Framework SCRUM juega un papel indispensable por lo que se abordará como parte del marco básico en sí. La definición de listo cambia de una organización a otra, incluso de un proyecto a otro. Es un acuerdo entre todo el equipo y consiste básicamente en una lista de chequeo que define un estándar de calidad que debe cumplir cualquier elemento del backlog para

considerarse como parte del incremento. Todos los miembros del equipo deben entender lo que significa "listo". La definición de listo no debe cambiar en medio de un Sprint. Finalizado cada Sprint, el equipo puede decidir ajustar la definición de listo, deseablemente incrementándola en el tiempo.

Como se indica la definición de listo es un acuerdo con el equipo por lo que no podría prescribirse en este documento, sin embargo una lista inicial candidata para discutir con el equipo del Ministerio de Agricultura y Desarrollo Rural incluirá al menos:

- El producto está desplegado en el ambiente de preproducción
- El producto es funcionalmente correcto
- El código cuenta con pruebas unitarias con cobertura de al menos el 70%
- El producto pasa todas las pruebas funcionales automáticas
- El producto pasa todas las pruebas no funcionales automáticas
- No hay defectos de severidad 1 y 2 abiertos
- El código no genera alertas de calidad de código bajo las reglas de análisis estático
- El código tuvo revisión de código y los comentarios de la revisión fueron resueltos
- El manual de usuario está actualizado y disponible con la nueva funcionalidad o la que cambió
- El manual de operaciones está actualizado y disponible en concordancia con el incremento liberado
- Los scripts de despliegue están actualizados y disponibles en concordancia con el incremento liberado
- Los artefactos de configuración están actualizados y disponibles en concordancia con el incremento liberado

d. Roles

SCRUM reconoce tres funciones y no presta atención a títulos como desarrollador, probador o gerente del proyecto. Los roles únicos reconocidos por SCRUM son el equipo de desarrollo, el propietario del producto (Product Owner) y el SCRUM Master. Juntas, estas tres unidades forman un equipo cohesivo. Esta tríada de competencias de desarrollo de software es lo que se conoce como un equipo SCRUM.

i. Dueño del producto (Product Owner)

El dueño del producto es el único rendidor de cuentas (Accountable) del producto y la forma en que gestiona su producto es a través del product backlog. El dueño del producto es responsable del product backlog en todo momento y no puede delegar esa responsabilidad. Es también responsable de garantizar que el trabajo que se realiza es el mayor valor en el momento. El dueño del producto no es un comité, siempre es una sola persona. Puede existir un comité de usuarios u otras partes interesadas que están constantemente en conversaciones con personas que se preocupan por el producto para que sea mejor, sin embargo, el dueño del producto es una sola persona. Esto no es una responsabilidad compartida. El propietario del producto no es un comité y no es un equipo.

Se propone y recomienda que en el Ministerio de Agricultura y Desarrollo Rural el dueño de cada producto sea el experto funcional de cada una de las iniciativas priorizadas en el centro de excelencia.

ii. Equipo de desarrollo (Development Team)

El equipo de desarrollo son las personas que realmente crean el incremento durante el Sprint. Todos en el equipo de desarrollo son parte de un equipo que crea software, que crea un producto como tal. SCRUM se refiere a cualquier miembro de este equipo como desarrollador aunque pueden existir especialidades por ejemplo en el diseño de interfaz de usuario, escritura técnica, o en las operaciones de TI. Los equipos de desarrollo se auto organizan, eligiendo qué y cómo hacerlo a partir de la planificación de Sprint. Los equipos de desarrollo de SCRUM se consideran de funciones cruzadas y cada competencia necesaria para ofrecer un incremento de producto terminado al final de cada Sprint está presente en el equipo. Es posible y frecuente aunque no recomendado compartir la capacidad de un individuo a través de múltiples equipos de desarrollo. Los equipos de Desarrollo lo conforman entre 6 y 3 personas de acuerdo a SCRUM. En la práctica este número puede llegar hasta 10.

En el Ministerio de Agricultura y Desarrollo Rural el equipo de desarrollo consistirá en su mayoría de los expertos suministrados por la fábrica de software y contará también con expertos del centro de excelencia cuando así se requiera, en particular el arquitecto de soluciones y posiblemente el arquitecto ESP participará activamente en cada sprint.

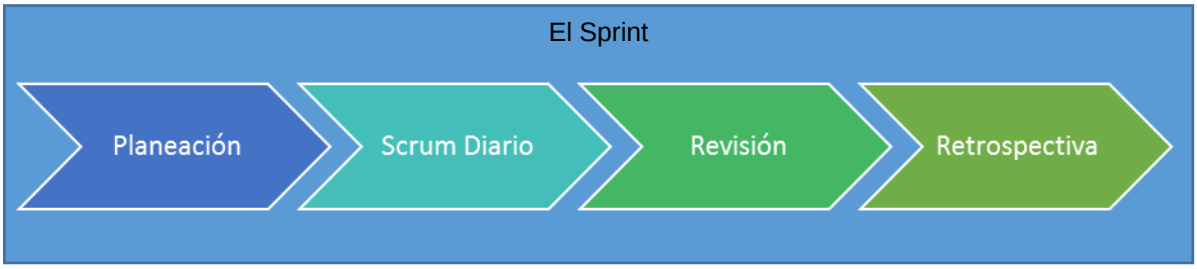
iii. SCRUM Master

Es el responsable de asegurar que los equipos de SCRUM se adhieren a la teoría, prácticas y reglas de SCRUM. El SCRUM Master actúa como líder de servicio para el equipo. No dirigen al equipo de desarrollo ni son dueños del producto. Es el responsable del entrenamiento, coaching, mentoring y lo que haga falta para comprender SCRUM y así lograr el máximo provecho de este. A menudo los SCRUM Masters facilitan eventos por lo que debe tener buenas habilidades al respecto. También ayudan a los que están fuera del equipo de SCRUM a comprender cómo trabajar con los que están en el Equipo SCRUM en formas que no socaven el foco de un Sprint. El SCRUM Master ayuda al dueño del producto en muchas maneras, incluyendo consejos de Investigación y de entrenamiento para una mejor gestión del Product Backlog. El SCRUM Master se asegurará de que cualquier persona que agrega elementos al Product Backlog sabe cómo crear elementos de productos claros y concisos. El SCRUM Master también ayuda al equipo de desarrollo mediante la facilitación de los eventos SCRUM, sin embargo no por ser facilitador los lidera, de hecho, se considera un buen SCRUM Master aquel que se asegura que el equipo de desarrollo es capaz de llevar a cabo el SCRUM diario sin su participación. Cuando el equipo de desarrollo se encuentra con un obstáculo o impedimento que en medio de un Sprint, pueden ir al SCRUM Master para ayudar a eliminar este obstáculo.

En el Ministerio de Agricultura y Desarrollo Rural el rol de SCRUM Master podrá ser provisto por la propia fábrica de software o en su defecto el centro de excelencia lo proveerá a través del líder de implementación o de otro rol.

e. Eventos

Los eventos o actividades en el marco SCRUM realmente definen cómo se utiliza SCRUM. Al ser un sistema de control de procesos empíricos, SCRUM anima a tomar decisiones basadas en la evidencia. En consecuencia, cada evento definido por SCRUM es una oportunidad para inspeccionar y adaptar algo.



Gráfica 6 - Eventos en SCRUM

i. El sprint

Está limitado a 30 días o menos, típicamente 2 semanas calendario. Su duración se decide con el equipo y entre menos se cambie mejor para afinar la estimación de esfuerzos.

Estos esfuerzos de desarrollo cortos limitan el riesgo al reducir la incertidumbre y permitir al equipo aprender rápidamente sobre lo que están desarrollando.

Provee igualmente una planeación y estimación más realista. Planear 2 o 4 semanas de trabajo suele ser más realista en proyectos complejos que hablar de meses o años.

Limitar los objetivos y el trabajo a realizar a cuestión de semanas permite al equipo enfocarse en el trabajo elegido para ese sprint.

ii. Velocidad (Velocity)

Es una medida histórica de la productividad del equipo que se registra en cada sprint y posteriormente se utiliza para estimar. En la medida que el equipo de desarrollo va madurando y ha ejecutado más sprints o proyectos esta medida es más realista haciendo las estimaciones más efectivas.

No se mide en horas sino en puntos de función, puntos de dificultad, tallas u otro valor numérico acordado que brinde una estimación del esfuerzo requerido por cada elemento del product backlog. Luego del sprint se mide cuántos de esos puntos se ejecutaron construyendo así el indicador de velocidad del equipo.

iii. Planeación del sprint (Sprint Planning)

Es una reunión donde participa todo el equipo de desarrollo, es la reunión que da inicio al Sprint y tiene una duración máxima de dos horas por cada semana de sprint. Es posible invitar a esta reunión stakeholders, expertos funcionales de los elementos del backlog que se han priorizado y otros interesados en el producto.

Como resultado de esta reunión se produce el sprint backlog y el sprint goal a partir del product backlog. Con el product backlog ordenado el equipo y el dueño del producto discuten sobre cuales elementos del

product backlog deben ejecutarse en el sprint, revisan estimaciones, verifican alcance, verifican que el equipo esté disponible hasta llegar a un acuerdo.

iv. SCRUM Diario (Daily SCRUM)

Es una reunión de y para el equipo de desarrollo que ocurre diariamente ojala siempre en el mismo sitio y con una duración no mayor a 15 minutos. En esta reunión el equipo planea en lo que va a trabajar ese día, inspecciona y adapta el sprint backlog, verifica su progreso para lograr el incremento e identifica y actúa sobre los impedimentos que puedan surgir.

v. Sprint Review

El equipo SCRUM comparte su trabajo con la organización o cualquier otro interesado en el producto. Este evento tiene una duración máxima de una hora por semana de duración del sprint. Durante la reunión el equipo recibe retroalimentación sobre el incremento, retroalimentación con la que se ajusta el product backlog bien sea ajustando elementos existentes, reordenándolos, o creando nuevos. Este evento ocurre el último día del sprint.

vi. Sprint Retrospective

En este evento participa todo el equipo SCRUM y típicamente lo facilita el SCRUM Master. El propósito de esta reunión es inspeccionar y adaptar el proceso como tal, que salió bien, que se puede mejorar y crear un plan para realizar esa mejora. En esta reunión se pueden definir cambios en la metodología, refinar o expandir la definición de listo, ajustar comportamientos y estándares que contribuyan a la calidad, agilidad, velocidad o trabajo en equipo.

4. Especificación de la herramienta de ALM

Los procesos de ALM en particular lo relacionado con el desarrollo y entrega de aplicaciones serán soportados por la herramienta de ALM Team Foundation Server bien sea en su versión como servicio en la nube (Visual Studio Online) o en su versión instalada en premisas (TFS).

1.1 Estado actual

El MADR cuenta actualmente con una instalación monolítica en premisas de TFS 2010

Machine Name	Roosevelt
Especificaciones	4GB RAM 4 Processors - Xeon CPU E5-2630 @2.3Ghz Windows 2008 R2 Enterprise Service Pack 1 64-bit OS
Disco	C: 60Gb / 9Gb Free
Versión de TFS	10.50.1617 TFS 2010
SharePoint	2010 instalado y configurado
Reporting Services	2008R2 instalado y configurado pero fallando
Build	Ausente
SQL Server	2008R2 instalado en la instancia por defecto en el mismo servidor de TFS

Tabla 1 - Instalación actual TFS

La instalación actual está 2 versiones atrás de la versión vigente del producto así como el sistema operativo y el motor de base de datos. Si bien es posible actualizarlos dado que esta instalación está realmente en desuso resultaría óptimo hacer una instalación limpia del producto junto con sus dependencias en sus últimas versiones liberadas.

a. Servicio en la nube vs instalación en premisas

VisualStudio Online (VSO) es la oferta de Software como servicio (SaaS) para Team Foundation Server. Dadas las necesidades del Ministerio de Agricultura y Desarrollo Rural es viable utilizar VSO aunque existen algunas diferencias claves de funcionalidad comparado con la instalación en servidores del Ministerio. A continuación se describen los retos y beneficios clave:

Beneficios

- Eliminar requerimientos de mantenimiento asociados con actualización, parchado, backups y monitoreo de la infraestructura de TFS
- Transferir la responsabilidad de disponibilidad y capacidad de TFS
- Facilidad de integración de terceros al no estar detrás del firewall del Ministerio

- Facilidad de integración de terceros al poder utilizar single-sign-on basado en Azure Active Directory

Retos

- Por ahora no es posible personalizar la plantilla de proceso
- Por ahora la plantilla de proceso y el portal del proyecto solamente están disponibles en idioma ingles
- Por ahora no está disponible la integración con Sharepoint
- Por ahora no cuenta con Data Warehouse ni reporting services

b. Análisis de capacidad para instalación en premisas

Si se utiliza TFS en premisas se debe tener en cuenta:

Capa de datos: Se recomienda quitar la capa de datos del servidor actual y aprovechar la infraestructura de SQL Server del Ministerio de Agricultura y Desarrollo Rural.

Capa de aplicación: Se recomienda al menos duplicar la capacidad del servidor tanto en RAM como en espacio físico en disco.

Build Services: Se requiere otra máquina de características al menos iguales al de la capa de aplicación para correr los servicios de compilación. Si bien es posible hacerlo en la misma máquina de la capa de aplicación la recomendación es hacerlo en máquinas independientes.

De utilizar el servicio en la nube los aspectos de capacidad para capa de datos y aplicación de TFS dejan de ser responsabilidad del Ministerio; si seguirá siendo necesario un servidor para los servicios de construcción para las aplicaciones que se publiquen en el centro de datos del Ministerio.

c. Estrategia de recuperación de desastres

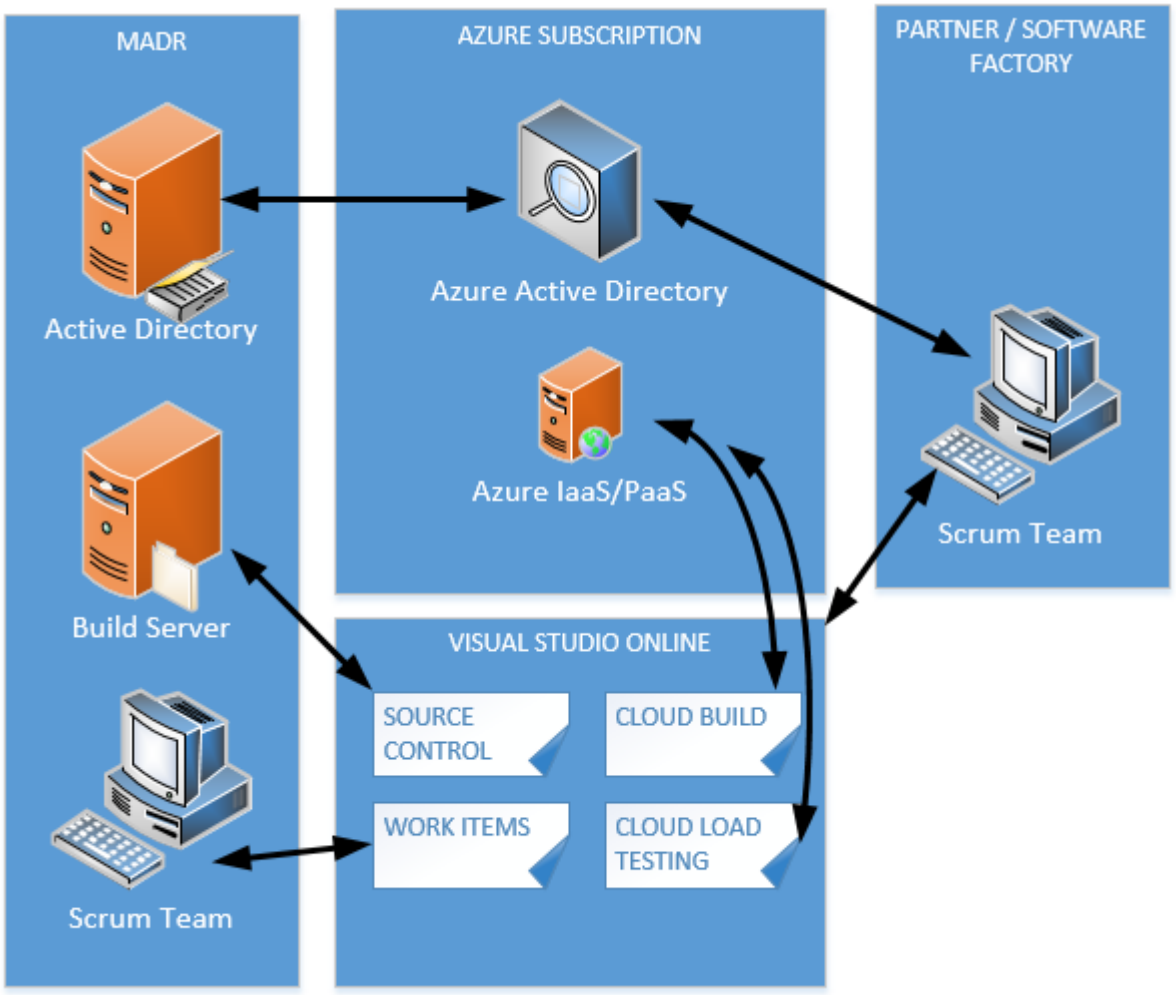
La información disponible al momento de elaborar este documento indica que actualmente no hay una estrategia de recuperación de desastres para TFS. De instalar TFS en premisas la estrategia de recuperación de desastres debe contemplar como mínimo un backup diario offsite de las bases de datos de TFS.

d. Estrategia de alta disponibilidad

Para una instalación en premisas de alta disponibilidad es necesario duplicar la infraestructura de la capa de aplicación, es decir contar con un segundo servidor. En todo caso para poder hablar de alta disponibilidad es necesario garantizar que todas las dependencias de TFS en premisas cuenten con mecanismos de alta disponibilidad incluyendo la infraestructura de base de datos, servicios de active directory y elementos de red.
De utilizar el servicio en la nube los aspectos de disponibilidad de TFS dejan de ser responsabilidad del Ministerio.

e. Arquitectura recomendada

La recomendación a partir de la información disponible y los requerimientos generales del Ministerio de Agricultura y Desarrollo Rural y el centro de excelencia es utilizar los servicios en la nube.



Gráfica 7 - Arquitectura recomendada

La autenticación de los usuarios del Ministerio de Agricultura y Desarrollo Rural ocurrirá por single-sign-on en Azure Active Directory el cual estará sincronizado con el directorio interno del Ministerio.

El Ministerio contará con un servidor de Build interno para liberar aplicaciones dentro de su propia infraestructura.

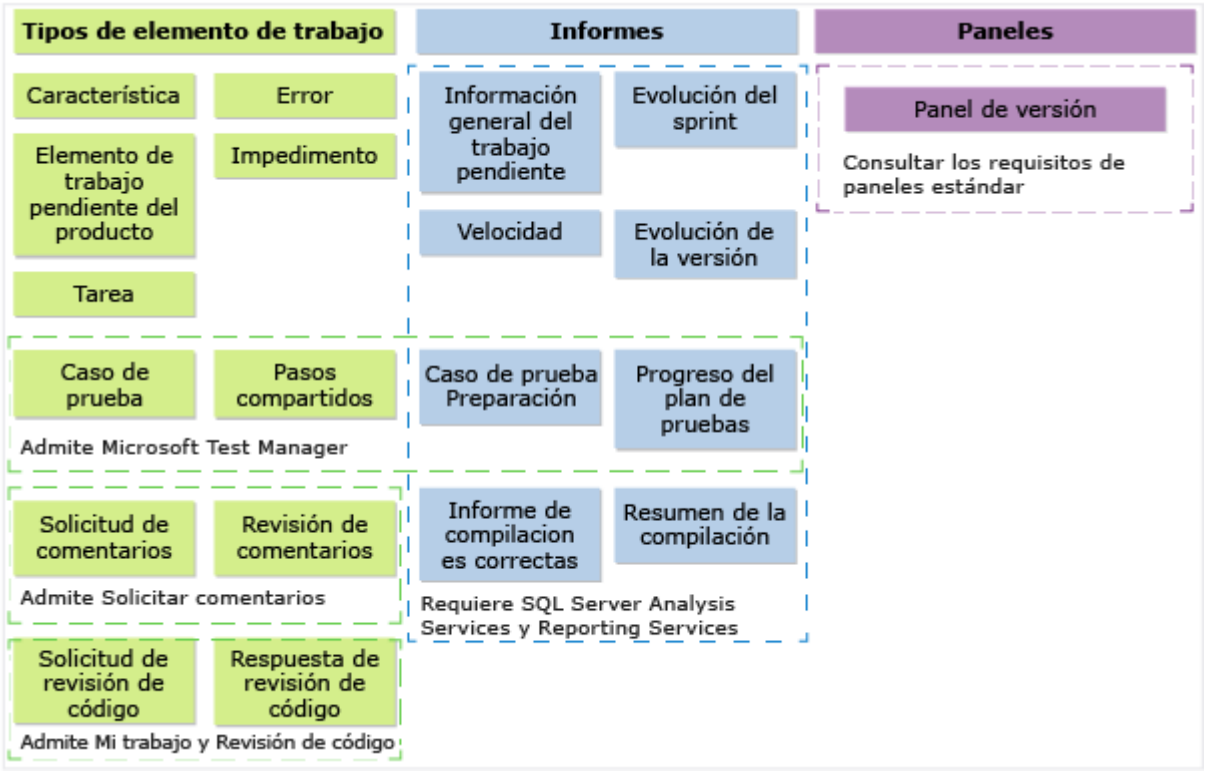
La(s) fábrica(s) de software se conectarán a VSO vía internet, sin ninguna dependencia del Ministerio.

En la gráfica se incluyen elementos de Azure destacando las facilidades para liberar aplicaciones en la nube y hacer pruebas de carga contra aplicaciones web.

f. Especificación de la plantilla de proceso

Para los proyectos de desarrollo del Ministerio de Agricultura y Desarrollo Rural liderados por el centro de excelencia se utilizará la plantilla de proceso SCRUM.

Esta plantilla proporciona todos los elementos de trabajo, consultas, informes y flujos necesarios para seguir el marco de trabajo SCRUM con todas las consideraciones indicadas en este documento.



Gráfica 8 - Elementos de trabajo en la plantilla SCRUM

La documentación en español de esta plantilla está disponible en:

https://msdn.microsoft.com/es-es/library/ff731587.aspx

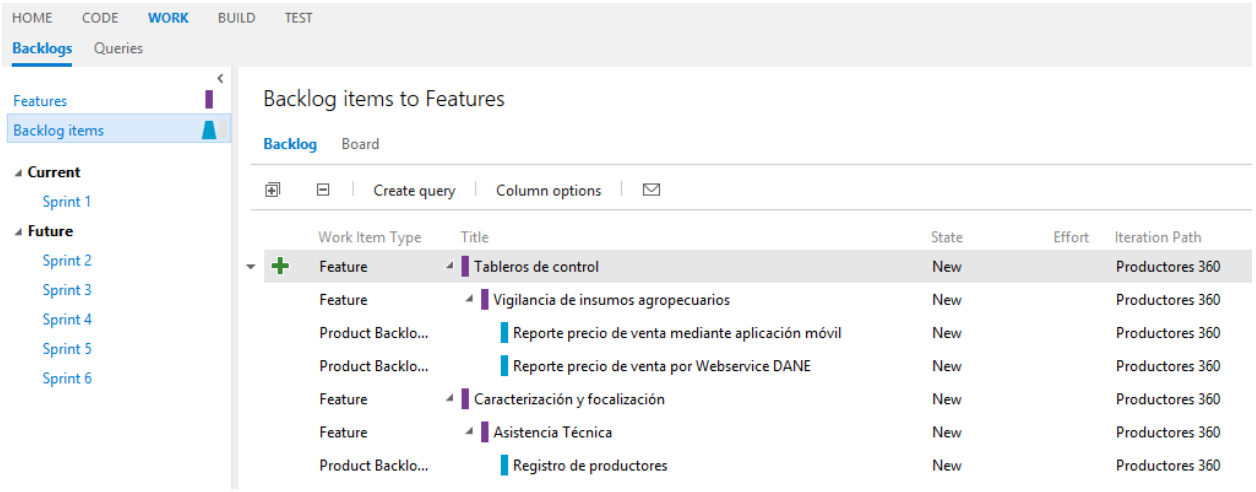
g. Planeación de colecciones de proyectos

TFS brinda la abstracción de colecciones de proyecto para separar proyectos y brindar aislamiento en términos de seguridad, escalabilidad y copias de seguridad.

No se espera tener requerimientos particulares en estos aspectos por lo que la recomendación para el Ministerio de Agricultura y Desarrollo Rural es utilizar una única colección de proyectos.

h. Estrategia de proyectos de equipo

Un Proyecto de equipo es un contenedor para datos relacionados a un mismo alcance. La palabra equipo sugiere que delimita a un equipo de trabajo, sin embargo este no es el caso. Dadas las características y la mecánica de trabajo que se prevé para el centro de excelencia del Ministerio de Agricultura y Desarrollo Rural la recomendación es utilizar un único proyecto de equipo que cobije todas las iniciativas del CoE facilitando así compartir recursos, consistencia en configuración y restricciones y vistas únicas de tareas y recursos.



Gráfica 9 - Proyecto de equipo productores 360

i. Estrategia de equipos, áreas e iteraciones

TFS brinda otra serie de abstracciones para brindar aislamiento o separación entre tareas, miembros del equipo y en general cualquier artefacto presente en TFS. En el Ministerio se van a utilizar estas abstracciones para modelar la separación que se requiera entre equipos de trabajo o entre iniciativas específicas.

5. Estrategia de Branching

a. Fundamentos

En el desarrollo de la presente sección se presentará una definición abstracta de cada uno de los conceptos comúnmente aceptados por la industria que se involucran en un proceso de gestión de código.

b. Vocabulario

Si bien la terminología puede variar de una organización a otra y según la herramienta de ALM utilizada, el vocabulario generalmente aceptado es el siguiente:

Término	Descripción
Rama Desarrollo (Dev)	Cambios para la siguiente versión
Integración hacia adelante (FI)	Consolida de la rama padre a las hijas.
Hot Fix	Un cambio implementado para corregir un defecto que bloquea el uso de la aplicación o no permite dar servicio a la misma.
Rama Main	Esta rama es la unión entre las ramas de Desarrollo y Release. Esta rama debe representar una vista estable del producto que puede ser compartida con QA o equipos externos.
Rama Release	Una rama en donde se corrigen defectos críticos antes de ser liberados en el producto. Luego que el producto es liberado esta rama debería configurarse como solo lectura.
Integración hacia atrás (RI)	Integra desde la rama hija hacia la rama padre.
Service Pack (SP)	Una colección de Hot Fixes y/o funcionalidades destinadas a la versión previa del producto.
Mecanismos de distribución	Es la forma como el producto se libera al cliente (ej.: versión final del release, hotfixes y/o service packs).

Tabla 2 - Vocabulario

c. Branches (Ramas)

Un branch (o rama) representa un compartimiento de código almacenado, en el cual se puede evolucionar el mismo (cambiar y/o generar nuevo código) sin que esto afecte a otras ramas o compartimientos.

Normalmente se crean ramas para generar aislamiento entre equipos que están trabajando en versiones distintas de un determinado componente o aplicación que hasta incluso pueden tener una fecha de implementación diferente.

La utilización de ramas nos asegura que el código generado por cada equipo no será compartido de forma automática.

d. Merging (Consolidación)

Es el nombre que recibe la acción mediante la cual se integran los cambios de una determinada "rama" (o branch) en otra.

Esta acción cobra sentido, ya que como se mencionó en el concepto anterior en algunos casos puede ser necesario generar aislamiento entre equipos de desarrollo para aumentar la productividad o poder separar los cambios para poder generar versiones de una determinada aplicación o componente en momentos diferentes en la escala de tiempo. Pero aun así las ramas forman parte de una misma aplicación o componente, y en algún momento esos cambios tienen que consolidarse.

Tipos de Branches

Hay tres tipos generales de ramas, Desarrollo (Dev), Main y Release. Independientemente del tipo de rama se deben aplicar las siguientes consideraciones para todas las ramas:

- Construir el código diariamente para dar una cadencia diaria de trabajo al equipo.
- Implementar build de integración continua para identificar problemas de calidad.
- Flujo de código, el movimiento de cambios entre las ramas padres e hijas, es un concepto que todos los miembros del equipo deben considerar y entender.

Por cada tipo de rama hay una serie de consideraciones adicionales que son detalladas a continuación.

Main

- Cada rama que se va a ser consolidada posteriormente debe tener un camino hacia Main (Es la línea base del control de versiones).
- Si hay fallas en el build de Main deben ser corregidas inmediatamente.
- No se deben hacer modificaciones directamente en la rama Main (Es decir Check-In/Check-out o Commit).
- Los equipos de QA deben poder obtener el build desde la rama Main en cualquier momento para propósitos de testeo.

MAIN es la unión entre las ramas de Desarrollo y Release. Los cambios en Main serán integrados en Desarrollo, de manera que es crítico que los build de Main sean de alta calidad. Esto significa que como mínimo Main debe permanecer construible en todo momento.

Desarrollo

Antes de crear una nueva rama de Desarrollo debemos asegurarnos de lo siguiente:

- Seleccionar la rama padre: si los cambios están orientados al desarrollo de la siguiente versión del producto el padre de la nueva rama debe ser Main.
- Abrir la nueva rama desde un padre que se encuentre en buen estado (una versión estable que se pueda compilar y construir). Comenzar la rama desde el último build exitoso de la rama padre.
- Integrar frecuentemente. El objetivo debería ser no más de 1 ó 2 días fuera de sincronización con Main. Esto reduce la complejidad de consolidaciones grandes.
- Consolidar desde el hijo hacia al padre basado en calidad. Los requerimientos mínimos para hacer una integración con la rama padre es que el build sea exitoso.

Release

Una estrategia de ramas de Release exitosa permite los siguientes 3 escenarios:

- Los desarrolladores solo necesitan hacer check-in una vez de acuerdo al mecanismo de distribución para el cual se implemente el cambio.
- Se crea una ruta de integración natural para volver a Main, ya que se crea una estructura jerárquica de las ramas basada en los mecanismos de distribución.
- Reduce el riesgo de regresiones. Creando la relación de ramas padre/hijo entre las ramas Main→SP→Hotfix los cambios son naturalmente consolidados en las versiones futuras, reduciendo los riegos de regresiones de defectos en futuras versiones.

e. Planes de "Branching" (Ramificación)

Los elementos introducidos en esta sección en los planes de ramificación son típicamente aditivos, es decir, comenzando con el plan básico permite la transición a los planes estándar y avanzado, si los requerimientos de ramificación se vuelven más complejos. Hay dos razones comunes para incrementar el número de ramas y por consiguiente la complejidad:

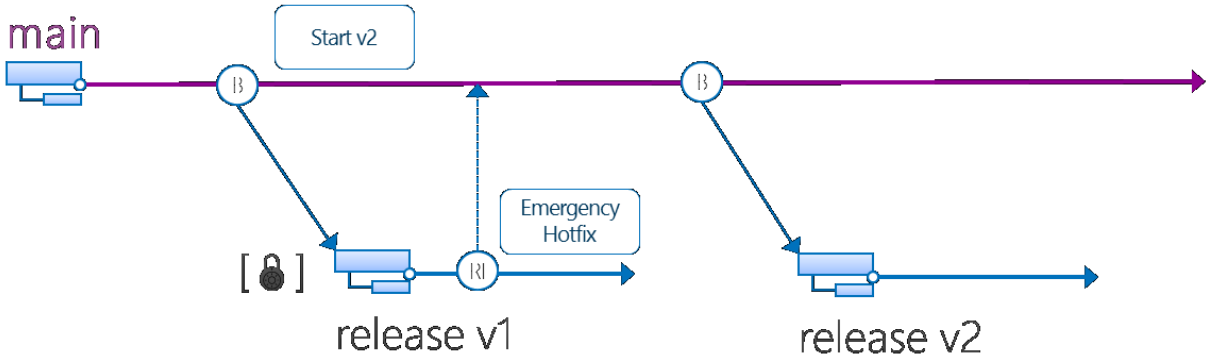
1. La necesidad de mecanismos de distribución adicionales (ej.: Service Pack y Hotfixes) requerirá un plan de ramificación que soporte la concurrencia del desarrollo para cada uno de estos.
2. El desarrollo de requerimientos adicionales tales como la necesidad de separar funcionalidades dentro de sus propias ramas.

i. Plan de Ramificación Básico

El plan de ramificación básico con las ramas Main, Dev y Release permite el desarrollo concurrente para la siguiente versión del producto, una rama estable para propósitos de test (Main) y una rama para la corrección de errores (Release). Las diferentes áreas de desarrollo son soportadas creando múltiples ramas de desarrollo desde la rama Main.

Este será el plan recomendado para el Ministerio de Agricultura y Desarrollo Rural.

Las versiones adicionales son soportadas creando ramas para cada versión del producto. Cada rama Release es hija de la rama Main y además son pares con otras ramas destinadas a otras versiones (ej.: la rama Release 2.0 es par de la rama Release 3.0 y ambas son hijas de la rama Main). Si solo se quiere soportar una versión en producción a la vez, se debe considerar una sola rama Release y corregir los errores que pudiesen surgir directamente en esa rama. Una vez que la rama Release es creada, la rama Main y las ramas de Desarrollo pueden comenzar a integrar los cambios aprobados para la siguiente versión de producción.



Gráfica 10 - Plan de ramificación básico

Escenarios de uso

1. Cuando solo se tiene una versión que va ser liberada al cliente.
2. El modelo de servicio al cliente es actualizar la siguiente versión.
3. No se necesita soporte de múltiples versiones del producto en producción a la vez.
4. Cualquier corrección hecha en la rama de Release incluirá todas las correcciones previas hechas en dicha rama.

Consideraciones

En las ramas de Desarrollo (Dev) de la siguiente versión:

a. El trabajo en las ramas de Desarrollo se puede separar por funcionalidad, organización o colaboración temporaria.

b. Cada rama de Dev debe ser una rama completa de Main.

c. En las ramas de Dev se debe construir el build de la misma forma en que se hace en Main.

d. Se debe integrar frecuentemente desde Main a Dev si hay cambios en Main.

e. Se debe integrar de Dev a Main basado en los criterios seleccionados por el equipo (ej.: fin de la versión, release internos de calidad, etc.).

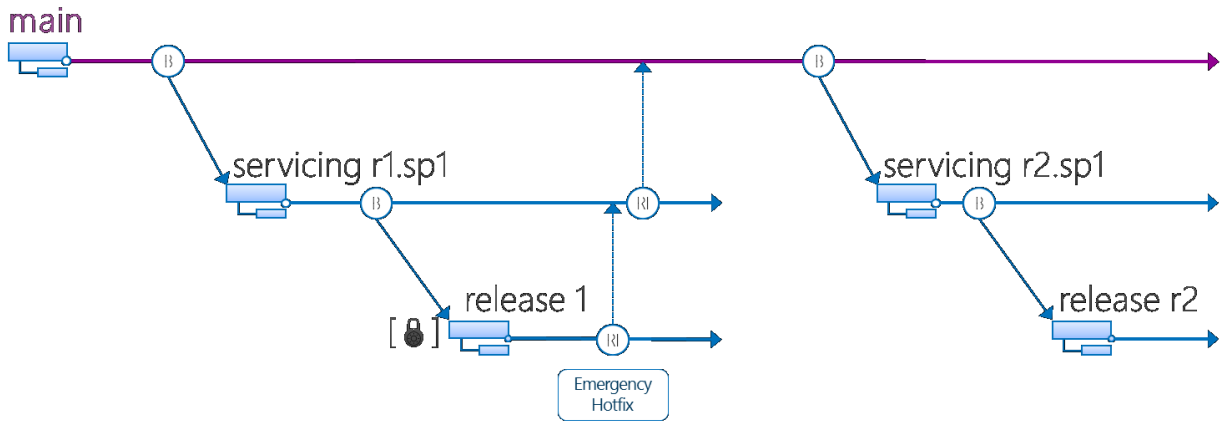
f.

En las ramas de Release en donde se libera la versión a producción:

a. Release debe ser una rama hija de Main.
b. El nivel de acceso de los desarrolladores a la rama Release debe ser de solo lectura, de esta manera se evitan los cambios accidentales en la rama.
c. Los cambios de la rama Release se consolidan con Main en una sola dirección. Una vez que se crea la rama Release, Main puede tomar cambios para la siguiente versión que pueden no estar aprobados para la rama de Release.
d. Crear una nueva rama de Release para las subsecuentes versiones si se requiere tal nivel de separación.

ii. Plan de Ramificación Estándar

Cada vez que se agregan mecanismos de distribución adicionales al plan básico, puede ser necesario crear ramas adicionales en el área de producción/Release para permitir el desarrollo concurrente. El plan de ramificación estándar introduce una nueva rama de Release para soportar mecanismos de distribución adicionales. La mayoría de las organizaciones llaman a esto ramas de servicio, dichas ramas permiten la corrección de defectos y service packs. Con este plan se puede desarrollar la siguiente versión del producto en Desarrollo y Main mientras el desarrollo de Service Packs es soportado en la rama Service Pack. Otros mecanismos de distribución tales como hotfixes de emergencia son soportados directamente en la rama del Release. Una vez que la siguiente versión esta lista para Release se puede elegir crear dos nuevas ramas para Service Pack y Release. Esto permite un alto grado de paralelismo en el desarrollo.



Gráfica 11 - Plan estándar

Todas las guías aplicadas al plan básico también aplican a este plan.

Escenarios de uso

1. Se tienen múltiples mecanismos de distribución (ej.: el Release y los Service Packs adicionales para el Release).
2. Se quiere permitir el desarrollo concurrente de service packs y la siguiente versión del producto.

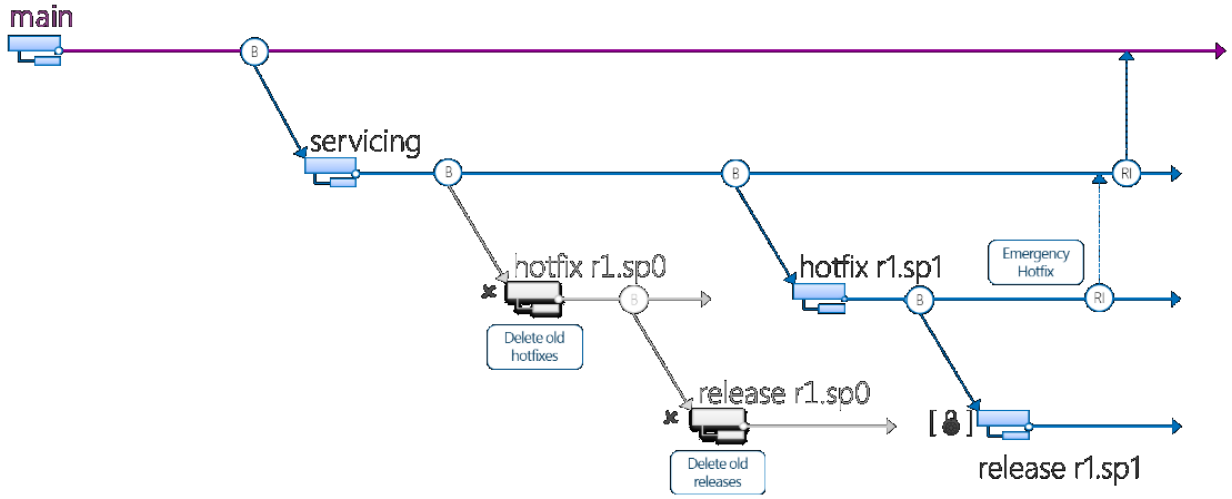
3. Se requiere tener una vista exacta del código fuente que se encuentra en producción dentro de la rama Release.

Consideraciones

1. La rama de Release se encuentra salvaguardada, mientras la rama de SP sirve para trabajar.
2. El árbol de ramas de Release (ej.: SP y Release) son ramificados desde Main al mismo tiempo para crear la relación padre/hijo Main→SP→Release.
3. El producto se libera desde la rama Release y luego dicha rama debería ser configurada como solo lectura.
4. Los cambios en el servicio son actualizados en la rama Service Pack.
5. Los cambios en las ramas de SP se consolidan en un solo sentido hacia Main (SP-> Main).
6. Duplicar el árbol del Release para las versiones subsecuentes si se requiere ese tipo de separación.

iii. Plan de Ramificación Avanzado

El plan de ramificación avanzado es para productos que deben soportar más de un mecanismo de distribución y escenarios de servicio. El plan permite el desarrollo concurrente del Release, Service Pack, hotfixes y la siguiente versión. Esto se logra agregando la rama Hotfix entre las ramas Service Pack y Release, de esta manera se agrega un nuevo nivel de desarrollo en paralelo ya que se puede desarrollar en la rama de Desarrollo tanto como corregir defectos críticos en la rama de Service Pack. Se pueden crear nuevas ramas de Release tanto como eliminar las ramas de Release obsoletas, también se puede administrar los permisos para asegurar que las diferentes rama de Release sean de solo lectura. Este plan soporta múltiples versiones en producción a la vez debido a que se pueden tener múltiples ramas de Release. Se debe tener en cuenta que se pueden crear ramas de Desarrollo adicionales para soportar un nivel de separación mayor en la parte de desarrollo (siempre y cuando sea necesario).



Gráfica 12 - Plan avanzado

Todas las guías aplicadas a los planes básico y estándar también aplican a este plan.

Consideraciones adicionales

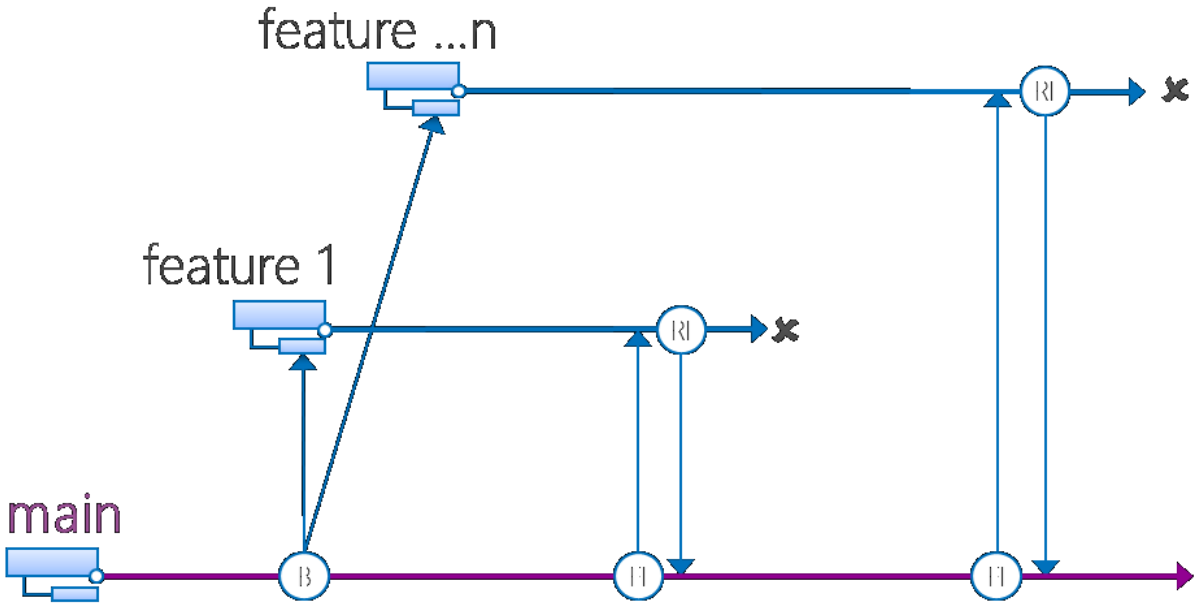
1. La rama de Release se encuentra salvaguardada, mientras que las ramas de Hotfix y SP sirven para corregir defectos o trabajar en nuevas funcionalidades.
2. El árbol de ramas de Release (ej.: SP, Hotfix y Release) es ramificado desde Main al mismo tiempo para crear la relación padre/hijo Main→SP→Hotfix→Release.
3. Luego de la liberación del nuevo release los permisos de acceso de la rama Release deberían ser configurados como solo lectura. Esto asegura que el código fuente representa la versión exacta del producto entregado e impide cambios accidentales en dicha rama.
4. Los cambios en las ramas Hotfix y SP se consolidan en un solo sentido a través de las ramas intermediarias a Main (HOTFIX→SP→MAIN).
5. Como los hotfixes se integran desde la rama Hotfix a la rama SP cualquier release desde SP incluirá todos los hotfixes además de cualquier cambio hecho directamente en la rama SP.

Todos los planes anteriores cubren la mayoría de las actividades de desarrollo de software. Usando estos planes como base permitirá a los equipos gastar más tiempo en el desarrollo de código de alta calidad y asegurar que cada rama tenga un rol especifico.

iv. Plan de Ramificación por Funcionalidad

En muchos proyectos de software se puede definir claramente el conjunto de funcionalidades que se desea implementar en el producto. Inclusive puede haber equipos separados dedicados que desarrollan diferentes funcionalidades que van a consolidarse en un único producto. Cuando se tiene esta clara separación en el proceso, se debe considerar el plan de ramificación por funcionalidad. La ramificación por funcionalidad puede ser también un plan lógico de ramas para grandes productos que inherentemente pueden tener muchas funcionalidades y varios equipos trabajando en el producto. La idea con este plan es que se puede separar cada funcionalidad en su propia rama, esto da mucha flexibilidad en términos de cuando un release es liberado a producción, tanto como para ayudar a prevenir situaciones en donde una rama está lista para producción en contraposición con funcionalidades que no lo están. A medida que las ramas de funcionalidades se vuelven lo suficientemente estable para release, dichas ramas se deben consolidar con Main. De esta manera se pueden eliminar las ramas por funcionalidad o crear nuevas ramas a medida que se necesite.

Como se muestra en el siguiente diagrama, cuando una funcionalidad se encuentra estable y lista para ser promovida a Main se puede integrar desde Main a la rama del release correspondiente. Esto creará un solo conjunto de cambios en Main junto con los cambios en la rama del release, de esta manera se puede hacer un roll back fácilmente de ser necesario.



Gráfica 13 - Plan por funcionalidad

Cuando se está trabajando con una estrategia de ramificación por funcionalidad hay que tener en cuenta que mientras otras funcionalidades se están consolidando con la rama Main, también se introducen posibles complejidades y conflictos en la consolidación con las ramas de las funcionalidades que están bajo desarrollo, ya que van cambiando más y más de la base original en Main a medida que pasa el tiempo. Una rama de funcionalidad no tiene por qué estar completa antes de volver a integrar con la rama Main. De hecho se debería considerar integrar con la rama Main frecuentemente de manera de mantener los cambios de integración de tamaño pequeño y por ello menos complejos. Como un ejemplo, considere si la “Feature 2” del diagrama se encuentra en desarrollo y la “Feature 3” fue recientemente integrada a la rama Main como finalizada, se podría hacer una integración desde la rama Main a la rama de la “Feature 2”, lo cual incluiría los cambios liberados en la rama “Feature 3”. Estos cambios pueden ser estabilizados en la rama para la “Feature 2”, e incluso puede ayudar a evitar reinventar la rueda en algunos casos. Si es posible se debe mantener el ciclo de vida del desarrollo de las funcionalidades corto, e integrar con la rama Main frecuentemente.

Escenarios de uso

1. Permite el desarrollo de múltiples funcionalidades en paralelo.
2. Permite una clara separación de las diferentes funcionalidades.
3. Permite el desarrollo de un gran producto con muchas funcionalidades que necesitan ser desarrolladas por separado.
4. Permite el desarrollo de funcionalidades en paralelo que no necesariamente se corresponden con un mismo ciclo de release.

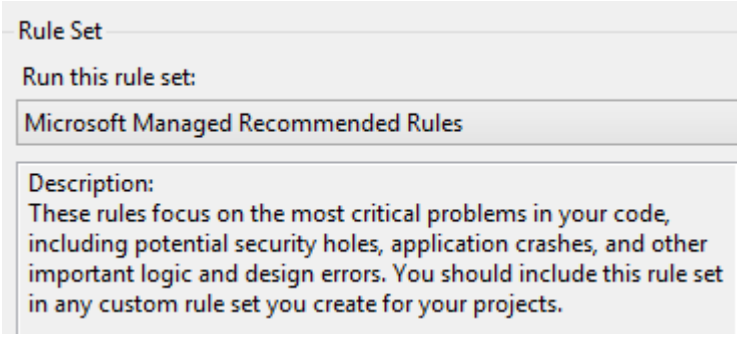
En los casos en los que se necesite poder hacer roll-back fácilmente de ciertas funcionalidades de una versión.

6. Políticas aceptación de código

En los proyectos de desarrollo de software a la medida del centro de excelencia del Ministerio de Agricultura y Desarrollo Rural se utilizarán políticas a nivel de infraestructura de TFS para exigir validaciones sobre el código antes que este sea introducido al control de versiones.
En particular:

- Cualquier adición o cambio en el código tiene relación a un Product Backlog Item
- Todo checkin debe hacer build localmente
- Todo checkin debe pasar las pruebas unitarias
- El análisis de métricas de código no debe arrojar alertas de complejidad ciclomática https://msdn.microsoft.com/en-us/library/ms182212.aspx
- El análisis de métricas de código no debe arrojar alertas de acoplamiento excesivo https://msdn.microsoft.com/en-us/library/bb397994.aspx
- Todo checkin debe pasar las pruebas de análisis estático de código sin generar alertas

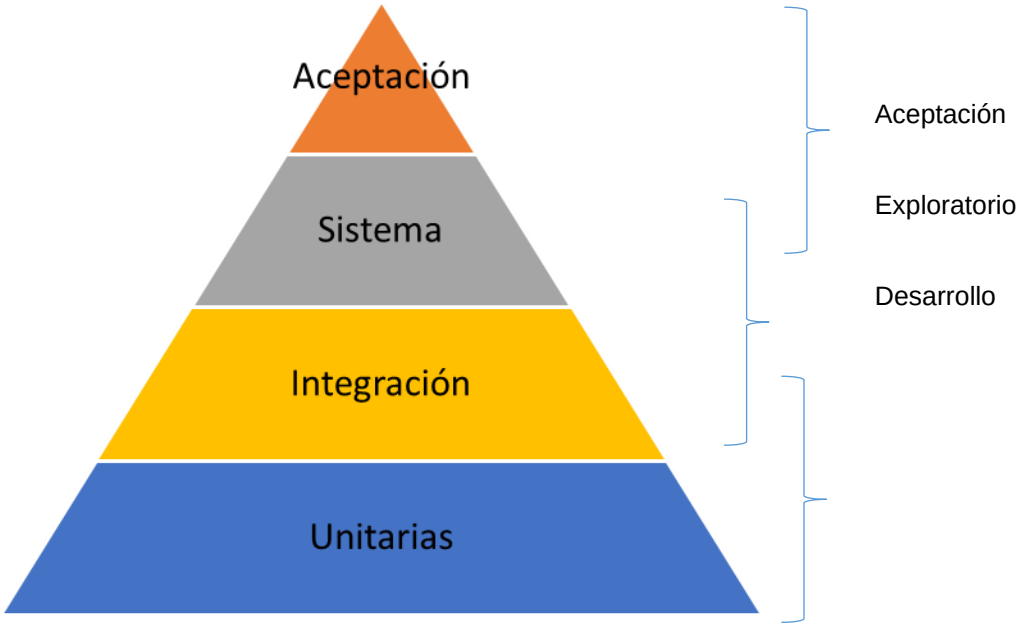
Inicialmente las reglas de análisis estático se limitarán a las reglas recomendadas:



Gráfica 14 - Reglas recomendadas

7. Estrategia de pruebas

a. Categorías de pruebas



La estrategia general para entregar código de excelente calidad tiene como base el uso de pruebas unitarias.

Desarrollo	Son pruebas elaboradas y ejecutadas por los desarrolladores
Aceptación	Con estas pruebas se verifica que una característica del producto está "lista"
Exploratorio	Es probar la aplicación sin seguir un guion o caso de prueba específico

Tabla 3 - Pruebas unitarias

b. Tipos de prueba

Pruebas unitarias	Aíslan y verifican unidades discretas de lógica.

	Los desarrolladores las escriben y ejecutan mientras desarrollan el producto.
Pruebas de integración	Similares a las unitarias pero no necesariamente enfocadas en código aislado.
Pruebas del sistema	Múltiples pasos que representan un escenario realista del uso de la aplicación como un todo. Se suelen automatizar mediante simulaciones de la interacción del usuario.
Pruebas de aceptación	Verifican que la aplicación satisface las necesidades de sus usuarios. En su alcance se incluyen además de la verificación funcional aspectos de calidad como usabilidad y desempeño.

Tabla 4 - Tipos de prueba

c. Estrategia

Como se propuso en la “definición de listo” se espera tener un alto nivel de cobertura de código, mayor al 50%, en las pruebas unitarias. En los casos donde el código que se esté construyendo consiste principalmente en lógica de negocio se recomienda abordar la construcción del mismo bajo el esquema de Test Driven Developement (TDD), donde primero se construye la prueba antes que el código. Con este esquema el codificador logra un entendimiento mucho más profundo del problema que está resolviendo y deja en el camino pruebas unitarias confiables y completas.

Respecto a las pruebas de sistema y de aceptación la solución de ALM brinda una serie de facilidades de automatización de las que el Ministerio de Agricultura y Desarrollo Rural debe tomar ventaja. La idea acá es que luego de ejecutar una prueba manual esta se automatice con ayuda de la herramienta de forma que luego esta misma prueba haga parte del proceso de build del software, validándolo de forma funcional en cada incremento.

En cuanto a pruebas de aspectos no funcionales la solución de ALM brinda mecanismos para automatizar y ejecutar pruebas de carga. Para otros aspectos no funcionales como seguridad, usabilidad y confiabilidad es necesario ejecutar pruebas manuales o acudir a herramientas de terceros. En la medida que surjan requerimientos no funcionales en este sentido será necesario detallar como probarlos manteniendo como premisa que sean pruebas repetibles y automatizables.

8. Historial de Cambios

Fecha	Versión	Descripción
30 junio de 2015	1	Inicial.