Herramientas de Pruebas

INTRODUCCIÓN A LAS TÉCNICAS DE PRUEBAS

Introducción

- El software generado en la fase de implementación no puede ser "entregado" al cliente para que lo utilice, sin practicarle antes una serie de pruebas.
- La fase de pruebas tienen como objetivo encontrar defectos en el sistema final debido a la omisión o mala interpretación de alguna parte del análisis o el diseño. Los defectos deberán entonces detectarse y corregirse en esta fase del proyecto.
- En ocasiones los defectos pueden deberse a errores en la implementación de código (errores propios del lenguaje o sistema de implementación), aunque en esta etapa es posible realizar una efectiva detección de los mismos, estos deben ser detectados y corregidos en la fase de implementación.
- La prueba puede ser llevada a cabo durante la implementación, para verificar que el software se comporta como su diseñador pretendía, y después de que la implementación esté completa.

Introducción

- Esta fase tardía de prueba comprueba la conformidad con los requerimientos y asegura la fiabilidad del sistema.
- Las distintas clases de prueba utilizan **clases de datos de prueba** diferentes:
  - La **prueba estadística**.
  - La **prueba de defectos**.
  - La **prueba de regresión**.

Prueba de defectos

- La **prueba de defectos** intenta incluir áreas donde el programa no está de acuerdo con sus especificaciones.
- Las pruebas se diseñan para revelar la presencia de defectos en el sistema.
- Cuando se han encontrado defectos en un programa, éstos deben ser localizados y eliminados. A este proceso se le denomina **depuración**.
- La prueba de defectos y la depuración son consideradas a veces como parte del mismo proceso. En realidad, son muy diferentes, puesto que la prueba establece la existencia de errores, mientras que la depuración se refiere a la localización y corrección de estos errores.

Prueba estadística

- La **prueba estadística** se puede utilizar para probar el rendimiento del programa y su confiabilidad.
- Las pruebas se diseñan para reflejar la frecuencia de entradas reales de usuario.
- Después de ejecutar las pruebas, se puede hacer una estimación de la confiabilidad operacional del sistema.
- El rendimiento del programa se puede juzgar midiendo la ejecución de las pruebas estadísticas.

Prueba de regresión

- Durante la depuración se debe generar hipótesis sobre el comportamiento observable del programa para probar entonces estas hipótesis esperando provocar un fallo y encontrar el defecto que causó la anomalía en la salida.
- Después de descubrir un error en el programa, debe corregirse y volver a probar el sistema.
- A esta forma de prueba se le denomina **prueba de regresión**.
- La prueba de regresión se utiliza para comprobar que los cambios hechos a un programa no han generado nuevos fallos en el sistema.

Conflicto de intereses

- Aparte de esto, en cualquier proyecto software existe un **conflicto de intereses** inherente que aparece cuando comienza la prueba:
  - Desde un punto de vista psicológico, el análisis, diseño y codificación del software son tareas **constructivas**.
    - El ingeniero de software crea algo de lo que está orgulloso, y se enfrenta a cualquiera que intente sacarle defectos.
    - La prueba, entonces, puede aparecer como un intento de “romper" lo que ha construido el ingeniero de software.
  - Desde el punto de vista del constructor, la prueba se puede considerar (psicológicamente) **destructiva**.
    - Por tanto, el constructor anda con cuidado, diseñando y ejecutando pruebas que demuestren que el programa funciona, en lugar de detectar errores.
    - Desgraciadamente los errores seguirán estando, y si el ingeniero de software no los encuentra, lo hará el cliente.

Desarrollador como probador

- A menudo, existen ciertos **malentendidos** que se pueden deducir equivocadamente de la anterior discusión:
  1. El desarrollador del software no debe entrar en el proceso de prueba.
  2. El software debe ser "puesto a salvo" de extraños que puedan probarlo de forma despiadada.
  3. Los encargados de la prueba sólo aparecen en el proyecto cuando comienzan las etapas de prueba.
- Cada una de estas frases es incorrecta.
- El **desarrollador** siempre es responsable de probar las unidades individuales (módulos) del programa, asegurándose de que cada una lleva a cabo la función para la que fue diseñada.
- En muchos casos, también se encargará de la prueba de integración de todos los elementos en la estructura total del sistema.

Grupo independiente de prueba

- Sólo una vez que la arquitectura del software esté completa, entra en juego un **grupo independiente de prueba**, que debe eliminar los problemas inherentes asociados con el hecho de permitir al constructor que pruebe lo que ha construido. Una prueba independiente elimina el conflicto de intereses que, de otro modo, estaría presente.
- En cualquier caso, el desarrollador y el grupo independiente deben trabajar estrechamente a lo largo del proyecto de software para asegurar que se realizan pruebas exhaustivas.
- Mientras se dirige la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

Principios fundamentales

- Hay 6 principios fundamentales respecto a las metodologías de pruebas que deben quedar claros desde el primer momento aunque volveremos a ellos continuamente:
  - Las pruebas exhaustivas no son viables
  - Ejecución de pruebas bajo diferentes condiciones
  - El proceso de pruebas no puede demostrar la ausencia de defectos
  - Las pruebas no garantizan ni mejoran la calidad del software
  - Las pruebas tienen un coste
  - Inicio temprano de pruebas

Las pruebas exhaustivas no son viables

- Es imposible, inviable, crear casos de prueba que cubran todas las posibles combinaciones de entrada y salida que pueden llegar a tener las funcionalidades (salvo que sean triviales).
- Por otro lado, en proyectos cuyo número de casos de uso o historias de usuario desarrollados sea considerable, se requeriría de una inversión muy alta en cuanto a recursos y tiempo necesarios para cubrir con pruebas todas las funcionalidades del sistema.
- Por lo tanto es conveniente realizar un análisis de riesgos de todas las funcionalidades y determinar en este punto cuales serán objeto de prueba y cuales no, creando pruebas que cubran el mayor número de casos de prueba posibles.

Ejecución de pruebas bajo diferentes condiciones

- El plan de pruebas determina la condiciones y el número de ciclos de prueba que se ejecutarán sobre las funcionalidades del negocio.
- Por cada ciclo de prueba, se generan diferentes tipos de condiciones, basados principalmente en la variabilidad de los datos de entrada y en los conjuntos de datos utilizados.
- No es conveniente, ejecutar en cada ciclo, los casos de prueba basados en los mismos datos del ciclo anterior, dado que con mucha probabilidad, se obtendrán los mismos resultados.
- Ejecutar ciclos bajo diferentes tipos de condiciones, permitirá identificar posibles fallos en el sistema que antes no se detectaron y no son fácilmente reproducibles.

El proceso no puede demostrar la ausencia de defectos

- Independientemente de la rigurosidad con la que se haya planeado el proceso de pruebas de un producto, nunca será posible garantizar al ejecutar este proceso, la ausencia total de defectos (es inviable una cobertura del 100%).
- Una prueba se considera un éxito si detecta un error. Si no detecta un error no significa que no haya error, significa que no se ha detectado.
- Un proceso de pruebas riguroso puede garantizar una reducción significativa de los posibles fallos y/o defectos del software, pero nunca podrá garantizar que el software no fallará en producción.

Las pruebas no garantizan ni mejoran la calidad del software

- Las pruebas ayudan a **mejorar la percepción** de la calidad permitiendo la eliminación de los defectos detectados.
- La calidad del software viene determinada por las metodologías y buenas practicas empleadas en el desarrollo del mismo.
- Las pruebas **permiten medir la calidad** del software, lo que permite, a su vez, mejorar los procesos de desarrollo que son los que conllevan la mejora de la calidad y permiten garantizar un nivel determinado de calidad.

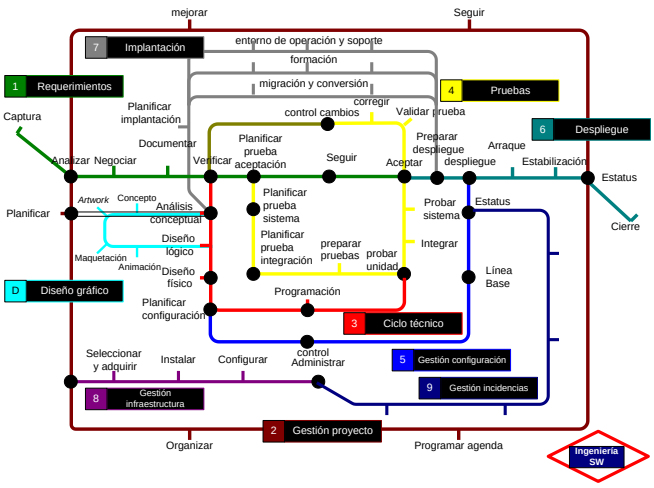
Las pruebas tienen un coste

- Aunque exige dedicar esfuerzo (coste para las empresas) para crear y mantener los test, los beneficios obtenidos son mayores que la inversión realizada.
- La creciente inclusión del software como un elemento más de muchos sistemas productivos y la importancia de los "costes" asociados a un fallo del mismo están motivando la creación de pruebas minuciosas y bien planificadas.
- No es raro que una organización de desarrollo de software gaste el 40 por 100 del esfuerzo total de un proyecto en la prueba.
- En casos extremos, la prueba del software para actividades críticas (por ejemplo, control de tráfico aéreo, o control de reactores nucleares) puede costar ¡de 3 a 5 veces más que el resto de los pasos de la ingeniería del software juntos!
- El coste de hacer las pruebas es siempre inferior al coste de no hacer las pruebas.

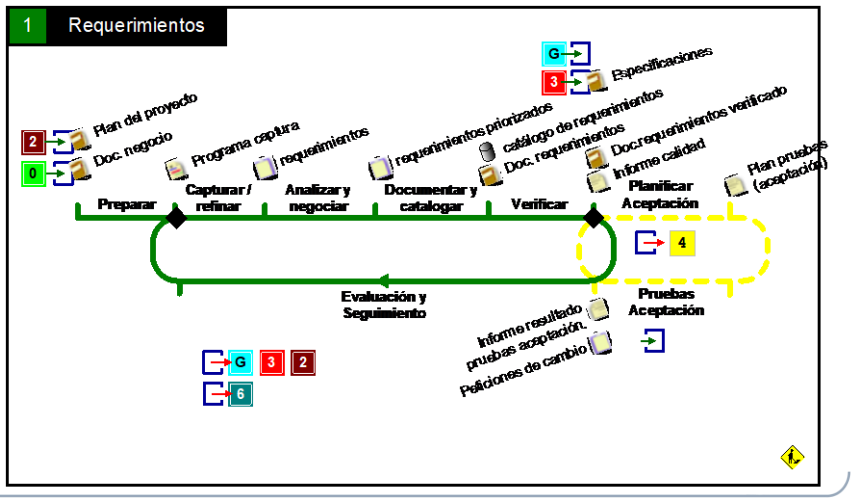
Inicio temprano de pruebas

- Si bien la fase de pruebas es la última del ciclo de vida, las actividades del proceso de pruebas deben ser incorporadas desde la fase de especificación, incluso antes de que se ejecutan las etapas de análisis y diseño.
- De esta forma los documentos de especificación y de diseño deben ser sometidos a revisiones y validaciones, lo que ayudará a detectar problemas en la lógica del negocio mucho antes de que se escriba una sola línea de código.
- Cuanto mas temprano se detecte un defecto, ya sea sobre los entregables de especificación, diseño o sobre el producto, menor impacto tendrá en el desarrollo y menor será el costo dar solución a dichos defectos.

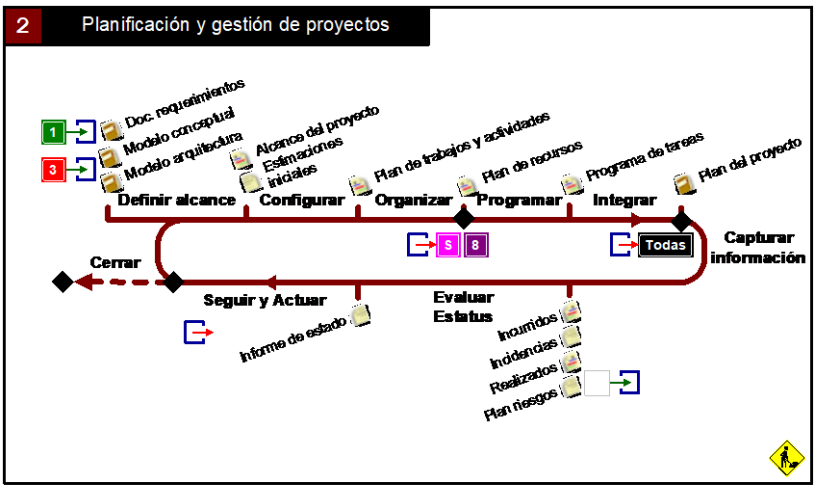
Integración en el ciclo de vida
Símil del mapa del Metro



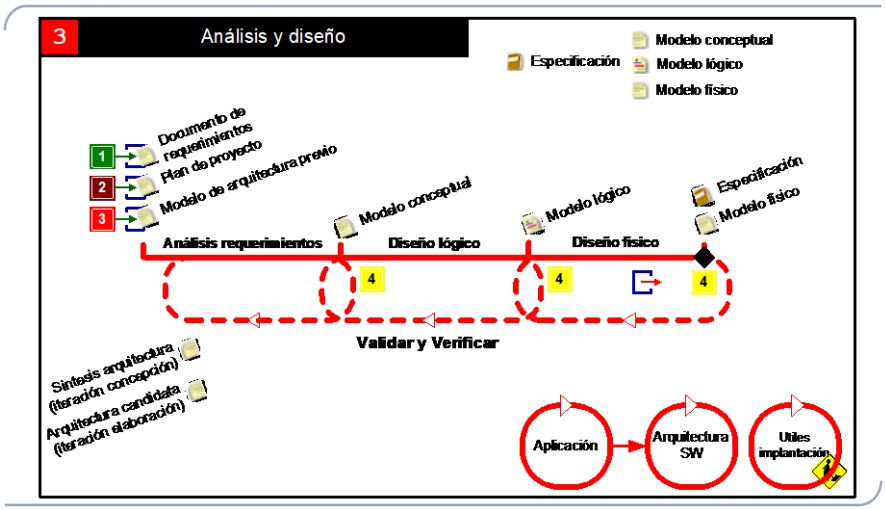
Línea 1: Requerimientos



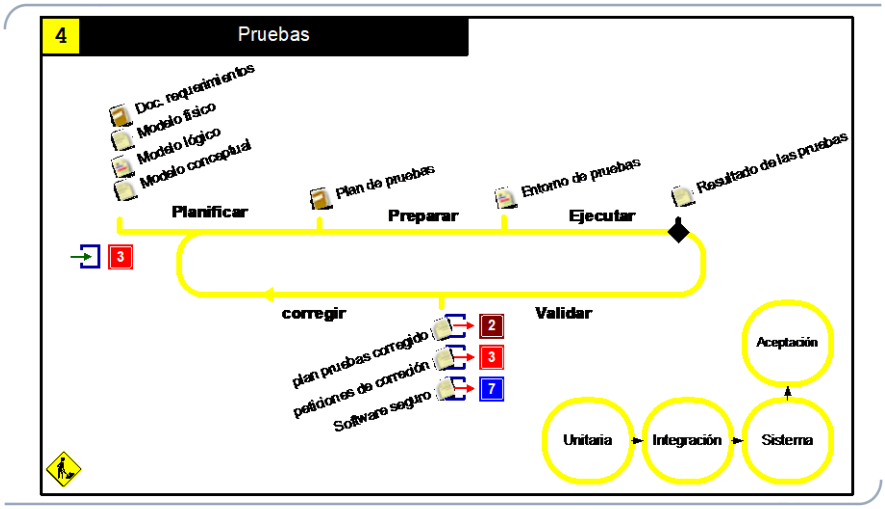
Línea 2: Planificación y gestión



Línea 3: Análisis y diseño



Línea 4: Pruebas

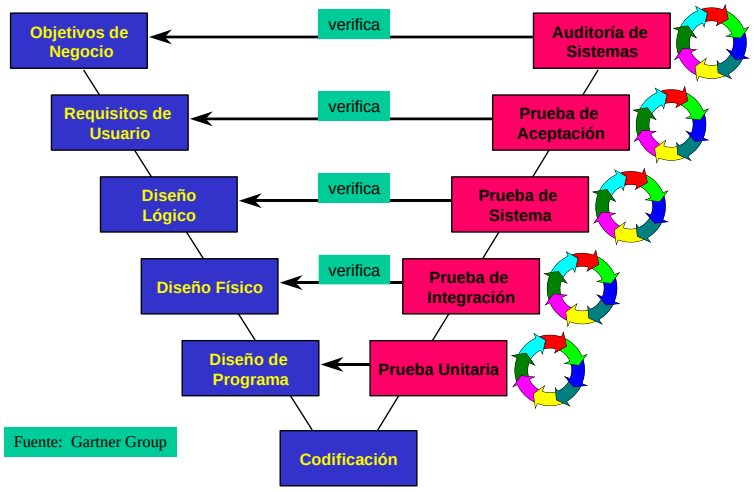


EL PROCESO DE PRUEBAS

El proceso

- Excepto para programas pequeños, los sistemas no deberían probarse como un único elemento indivisible.
- Los sistemas grandes se construyen a partir de subsistemas que se construyen a partir de módulos, compuestos de funciones y procedimientos.
- El proceso de prueba debería trabajar por etapas, llevando a cabo la prueba de forma incremental a la vez que se realiza la implementación del sistema, siguiendo el modelo en V.

Proceso de pruebas: Ciclo en V



Ciclo en V

- El modelo en V establece una simetría entre las fases de desarrollo y las pruebas.
- Las principales consideraciones se basan en la inclusión de las actividades de planificación y ejecución de pruebas como parte del proyecto de desarrollo.
- Inicialmente, la ingeniería del sistema define el papel del software y conduce al análisis de los requisitos del software, donde se establece el campo de información, la función, el comportamiento, el rendimiento, las restricciones y los criterios de validación del software. Al movernos hacia abajo, llegamos al diseño y, por último, a la codificación, el vértice de la V.
- Para desarrollar las pruebas seguimos el camino ascendente por la otra rama de la V.
- Partiendo de los elementos más básicos, probamos que funcionan como deben (lo que hacen, lo hacen bien). Los combinamos y probamos que siguen funcionando como deben. Para terminar probamos que hacen lo que deben (que hacen todo lo que tienen que hacer).

Niveles de pruebas

- **Pruebas Unitarias**: verifican la funcionalidad y estructura de cada componente individualmente, una vez que ha sido codificado.
- **Pruebas de Integración**: verifican el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente a través de sus interfaces, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.
- **Pruebas de Regresión**: verifican que los cambios sobre un componente de un sistema de información no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.
- **Pruebas del Sistema**: ejercitan profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- **Pruebas de Aceptación**: validan que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema, que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.

Pruebas Unitarias

- Las pruebas unitarias tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente, una vez que ha sido codificado.
- Con las pruebas unitarias verificas el diseño de los programas, vigilando que no se producen errores y que el resultado de los programas es el esperado.
- Estas pruebas las efectúa normalmente la misma persona que codifica o modifica el componente y que, también normalmente, genera un juego de ensayo para probar y depurar las condiciones de prueba.
- Las pruebas unitarias constituyen la prueba inicial de un sistema y las demás pruebas deben apoyarse sobre ellas.

Pruebas de Integración

- Las pruebas de integración te permiten verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.
- Se trata de probar los caminos lógicos del flujo de los datos y mensajes a través de un conjunto de componentes relacionados que definen una cierta funcionalidad.
- En las pruebas de integración examinas las interfaces entre grupos de componentes o subsistemas para asegurar que son llamados cuando es necesario y que los datos o mensajes que se transmiten son los requeridos.
- Debido a que en las pruebas unitarias es necesario crear módulos auxiliares que simulen las acciones de los componentes invocados por el que se está probando, y a que se han de crear componentes "conductores" para establecer las precondiciones necesarias, llamar al componente objeto de la prueba y examinar los resultados de la prueba, a menudo se combinan los tipos de prueba unitarias y de integración.

Pruebas de Integración

- Los **tipos fundamentales de integración** son los siguientes:
  - **Integración incremental:** combinas el siguiente componente que debes probar con el conjunto de componentes que ya están probados y vas incrementando progresivamente el número de componentes a probar.
  - **Integración no incremental:** pruebas cada componente por separado y, luego, los integras todos de una vez realizando las pruebas pertinentes. Este tipo de integración se denomina también Big-Bang (gran explosión).
- Con el tipo de prueba incremental lo más probable es que los problemas te surjan al incorporar un nuevo componente o un grupo de componentes al previamente probado. Los problemas serán debidos a este último o a las interfaces entre éste y los otros componentes.

Pruebas de Regresión

- El objetivo de las pruebas de regresión es eliminar el efecto onda, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.
- Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora.
- No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.
- Normalmente, este tipo de pruebas implica la repetición de las pruebas que ya se han realizado previamente, con el fin de asegurar que no se introducen errores que puedan comprometer el funcionamiento de otros componentes que no han sido modificados y confirmar que el sistema funciona correctamente una vez realizados los cambios.

Pruebas de Regresión

- Los pruebas de regresión deben su nombre al momento en que se ejecutan, no a su formato ni a otras características.
- Las pruebas de regresión **pueden incluir**:
  - La repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada.
  - La revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.
  - La obtención impresa del diccionario de datos de forma que se compruebe que los elementos de datos que han sufrido algún cambio son correctos.
- El **responsable** de realizar las pruebas de regresión será el equipo de desarrollo junto al técnico de mantenimiento, quién a su vez, será responsable de especificar el plan de pruebas de regresión y de evaluar los resultados de dichas pruebas.

Pruebas del Sistema

- Las pruebas del sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- Son pruebas de integración del sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.
- Una vez que se han probado los componentes individuales y se han integrado, se prueba el sistema de forma global. En esta etapa pueden distinguirse diferentes tipos de pruebas, cada uno con un objetivo claramente diferenciado.

Pruebas del Sistema

- **Pruebas funcionales**: dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario del sistema.
- **Pruebas de humo**: son un conjunto de pruebas aplicadas a cada nueva versión, su objetivo es validar que las funcionalidades básicas de la versión se cumplen según lo especificado. Impiden la ejecución el plan de pruebas si detectan grandes inestabilidades o si elementos clave faltan o son defectuosos.
- **Pruebas de comunicaciones**: determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Asimismo, se han de probar las interfaces hombre-máquina.
- **Pruebas de rendimiento**: consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- **Pruebas de volumen**: consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- **Pruebas de sobrecarga o estrés**: consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.

Pruebas del Sistema

- **Pruebas de disponibilidad de datos**: consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- **Pruebas de configuración**: consisten en comprobar todos y cada uno de los dispositivos, en sus propiedades mínimo y máximo posibles.
- **Pruebas de usabilidad**: consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados.
- **Pruebas extremo a extremo (e2e)**: consisten en interactuar con la aplicación como un usuario regular lo haría, cliente-servidor, y evaluando las respuestas para el comportamiento esperado.
- **Pruebas de operación**: consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y re-arranque del sistema, etc.
- **Pruebas de entorno**: consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de seguridad**: consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

Pruebas de Aceptación

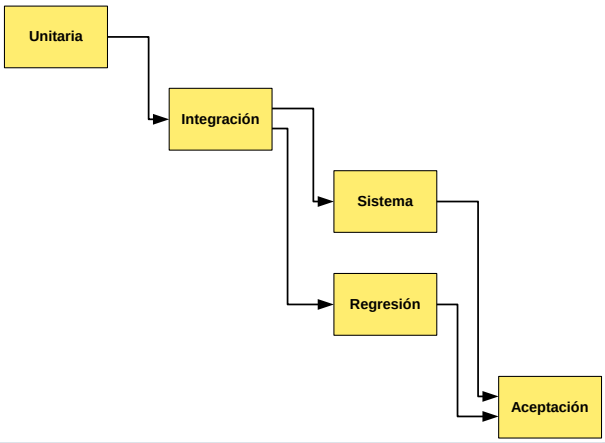
- El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.
- Las pruebas de aceptación son preparadas por el usuario del sistema y el equipo de desarrollo, aunque la ejecución y aprobación final corresponde al usuario.
- Estas pruebas van dirigidas a comprobar que el sistema cumple los requisitos de funcionamiento esperado recogidos en el catálogo de requisitos y en los criterios de aceptación del sistema de información, y conseguir la aceptación final del sistema por parte del usuario.

Pruebas de Aceptación

- Previamente a la realización de las pruebas, el responsable de usuarios revisa los criterios de aceptación que se especificaron previamente en el plan de pruebas del sistema y dirige las pruebas de aceptación final.
- La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados.
- Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta, a su vez, los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.
- La formalidad de estas pruebas dependerá en mayor o menor medida de cada organización, y vendrá dada por la criticidad del sistema, el número de usuarios implicados en las mismas y el tiempo del que se disponga para llevarlas cabo, entre otros.

Niveles de pruebas y orden de ejecución.

- De tal forma que la secuencia de pruebas es:



Tipos de Pruebas

- Las actividades de las pruebas pueden centrarse en comprobar el sistema en base a un objetivo o motivo especifico:
  - Una función a realizar por el software.
  - Una característica no funcional como el rendimiento o la fiabilidad.
  - La estructura o arquitectura del sistema o el software.
  - Los cambios para confirmar que se han solucionado los defectos o localizar los no intencionados.
- Las pruebas se pueden clasificar como:
  - Pruebas funcionales
  - Pruebas no funcionales
  - Pruebas estructurales
  - Pruebas de mantenimiento

Prueba exploratoria

- Incluso los esfuerzos de automatización de pruebas más diligentes no son perfectos. A veces se pierden ciertos casos extremos en sus pruebas automatizadas. A veces es casi imposible detectar un error en particular escribiendo una prueba unitaria. Ciertos problemas de calidad ni siquiera se hacen evidentes en sus pruebas automatizadas (como en el diseño o la usabilidad).
- Las pruebas exploratorias es un enfoque de prueba manual que enfatiza la libertad y creatividad del probador para detectar problemas de calidad en un sistema en ejecución. Simplemente tome un tiempo en un horario regular, arremangue e intente romper su aplicación. Use una mentalidad destructiva y encuentre formas de provocar problemas y errores en su aplicación. Documente todo lo que encuentre para más adelante. Tenga cuidado con los errores, los problemas de diseño, los tiempos de respuesta lentos, los mensajes de error faltantes o engañosos y todo lo que pueda molestarlo como usuario de su software.
- La buena noticia es que se puede automatizar la mayoría de sus hallazgos con pruebas automatizadas. Escribir pruebas automatizadas para los errores que se detectan asegura que no habrá regresiones de ese error en el futuro. Además, ayuda a reducir la causa raíz de ese problema durante la corrección de errores.

Pruebas de mutaciones

- Los pruebas de mutaciones son las pruebas de las pruebas unitarias.
- El objetivo es tener una idea de la calidad de las pruebas en cuanto a fiabilidad.
- Su funcionamiento es relativamente sencillo: la herramienta que se utilice debe generar pequeños cambios en el código fuente. A estos pequeños cambios se les conoce como mutantes. Una vez introducidos estos mutantes en el código, se lanzan todos los tests.
- Si los test unitarios fallan, es que han sido capaces de detectar ese cambio de código. En este caso el mutante se considera eliminado.
- Si, por el contrario, los test unitarios pasan, el mutante sobrevive y la fiabilidad (y calidad) de los tests unitarios queda en entredicho.
- Los test de mutaciones presentan reportes del porcentaje de mutantes detectados: cuanto más se acerque este porcentaje al 100%, mayor será la calidad de nuestros test unitarios.

Aprender con pruebas unitarias

- La incorporación de código de tercero es complicado, hay que aprenderlo primero e integrarlo después. Hacer las dos cosa a la vez es el doble de complicado.
- Utilizar pruebas unitarias en el proceso de aprendizaje (*pruebas de aprendizaje* según Jim Newkirk) aporta importantes ventajas:
  - La inmediatez de las pruebas unitarias y sus entornos.
  - Realizar “pruebas de concepto” para comprobar si el comportamiento se corresponde con lo que hemos entendido, permitiéndonos clarificarlo.
  - Experimentar para encontrar los mejores escenarios de integración.
  - Permite saber si un fallo es nuestro, de la librería o del uso inadecuado de la librería.

Pruebas de aprendizaje

- Las pruebas de aprendizaje no suponen un coste adicional, es parte del coste de aprendizaje que, en todo caso, lo minoran.
- Es mas, las pruebas de aprendizaje son rentables. Ante la aparición de nuevas versiones del código ajeno, ejecutar la batería de pruebas de aprendizaje valida el impacto de la adopción de la nueva versión: detecta cambios relevantes, efectos negativos en las integraciones, ...
- Las pruebas de aprendizaje no sustituyen al conjunto de pruebas que respaldan los limites establecidos.

Principios F.I.R.S.T.

- El principio FIRST fue definido por Robert Cecil Martin en su libro Clean Code. Este autor, entre otras muchas cosas, es conocido por ser uno de los escritores del Agile Manifesto, escrito hace más de 15 años y que a día de hoy se sigue teniendo muy en cuenta a la hora de desarrollar software.
  - Fast: Los tests deben ser rápidos, del orden de milisegundos, hay cientos de tests en un proyecto.
  - Isolated/Independent (Aislado/Independiente). Los tests no deben depender del entorno ni de ejecuciones de tests anteriores.
  - Repeatable. Los tests deben ser repetibles y ante la misma entrada de datos, los mismos resultados.
  - Self-Validating. Los tests tienen que ser autovalidados, es decir, NO debe de existir la intervención humana en la validación
  - Thorough and Timely (Completo y oportuno). Los tests deben de cubrir el escenario propuesto, no el 100% del código, y se han de realizar en el momento oportuno

Principios F.I.R.S.T.

- El principio FIRST fue definido por Robert Cecil Martin en su libro Clean Code. Este autor, entre otras muchas cosas, es conocido por ser uno de los escritores del Agile Manifesto, escrito hace más de 15 años y que a día de hoy se sigue teniendo muy en cuenta a la hora de desarrollar software.
  - Fast: Los tests deben ser rápidos, del orden de milisegundos, hay cientos de tests en un proyecto.
  - Isolated/Independent (Aislado/Independiente). Los tests no deben depender del entorno ni de ejecuciones de tests anteriores.
  - Repeatable. Los tests deben ser repetibles y ante la misma entrada de datos, los mismos resultados.
  - Self-Validating. Los tests tienen que ser autovalidados, es decir, NO debe de existir la intervención humana en la validación
  - Thorough and Timely (Completo y oportuno). Los tests deben de cubrir el escenario propuesto, no el 100% del código, y se han de realizar en el momento oportuno

Pirámide de pruebas

METODOLOGÍA

Introducción

- Durante las fases anteriores se han dedicado a construir el sistema, partiendo de las especificaciones han realizado el análisis, el diseño y la implementación.
- En la fase de pruebas te dedicas a destruir el sistema. El objetivo es identificar el mayor número de posibles causas de fallo y probar si se producen errores.
- En la fase de pruebas **estudiaremos toda la documentación anteriormente generada**: el modelo de casos de uso con el documento de requisitos adicionales, el modelo de análisis, el modelo de diseño, el modelo de implementación y el documento de descripción de arquitectura.
- La calidad de dicha documentación es fundamental para la fase de pruebas, incidiendo directamente en la calidad del proceso de pruebas.

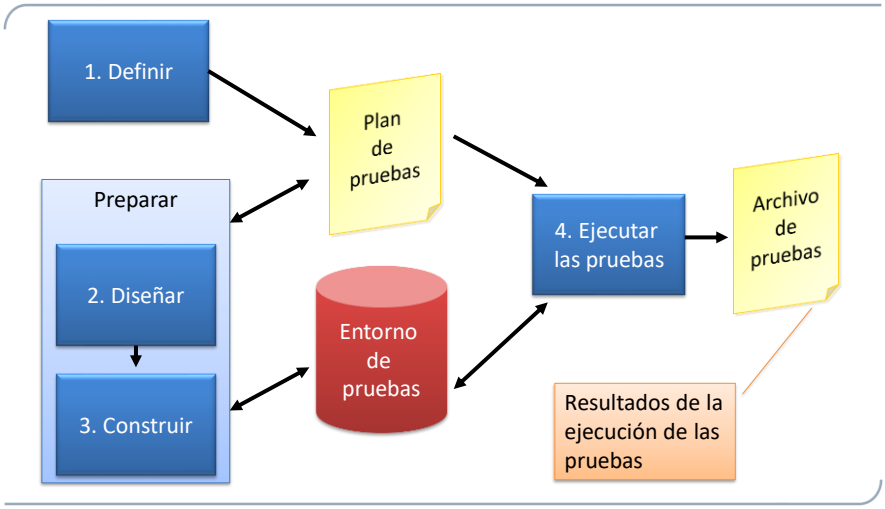
Introducción

- Una baja calidad en la documentación o la ausencia de algunos documentos degrada la validez del proceso de pruebas llegando incluso a imposibilitar dicho proceso.
- El **modelo de pruebas** describe cómo se prueban los componentes ejecutables del modelo de implementación con pruebas de integración y de sistema, así como aspectos específicos como puede ser la consistencia de la interfaz de usuario, si el manual del usuario cumple su cometido, y otros.
- A la fase de prueba se llega después de completar la fase de implementación, es cometido de dicha fase haber realizado las pruebas de unidad.

Elementos

- Como vimos en su momento, la base de la fase de pruebas son los **casos de prueba.** Recuerda que un caso de prueba específica una forma de probar el sistema, incluyendo la entrada con la que se ha de probar, los resultados que se esperan obtener y las condiciones bajo las que ha de probarse.
- Un **procedimiento de prueba** especifica cómo realizar uno o varios casos de prueba o partes de éstos.
- En el procedimiento documentas los pasos que deben darse para cada uno de los casos de prueba. Los procedimientos de prueba pueden reutilizarse para varios casos de prueba similares. Así mismo, un caso de prueba puede estar incluido en varios procedimientos de prueba.

Procedimiento de pruebas



Elementos

- Un **componente de prueba** automatiza uno o varios procedimientos de prueba o partes de éstos.
- Los componentes de prueba se diseñan e implementan de forma específica para proporcionar las entradas, controlar la ejecución e informar de la salida de los elementos a probar.
- Los componentes de prueba son necesarios puesto que la mayoría de las veces el elemento que estas probando no dispone de un interfaz de usuario que te permita la comunicación directa con él.

Elementos

- Un **defecto** es una anomalía del sistema, como por ejemplo un síntoma de un fallo software o un problema descubierto en una revisión. Los defectos deben estar documentados indicando síntomas, posibles causas y consecuencias.
- Existen **herramientas específicas de prueba de software** que permiten establecer los procedimientos de pruebas, ejecutar los procedimientos y evaluar los resultados guardando un registro de los mismos. Si utilizas dichas herramientas, en muchos casos, evitarás la necesidad de desarrollar componentes de prueba.
- Por lo tanto, el modelo de prueba está compuesto de casos de prueba, procedimientos de prueba y componentes de prueba. El modelo se organiza mediante los planes de prueba.

Plan de pruebas

- La prueba de sistemas es cara.
- La creciente inclusión del software como un elemento más de muchos sistemas productivos y la importancia de los "costes" asociados a un fallo del mismo están motivando la creación de pruebas minuciosas y bien planificadas.
- No es raro que una organización de desarrollo de software gaste el 40 por 100 del esfuerzo total de un proyecto en la prueba.
- En casos extremos, la prueba del software para actividades críticas (por ejemplo, control de tráfico aéreo, o control de reactores nucleares) puede costar ¡de 3 a 5 veces más que el resto de los pasos de la ingeniería del software juntos!

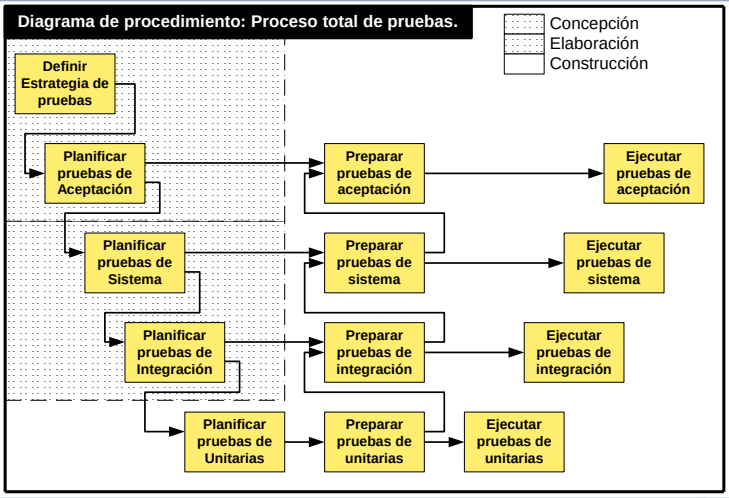
Plan de pruebas

- Se necesita una planificación cuidadosa para obtener lo máximo del proceso de prueba y controlar los costes.
- El propósito del plan de pruebas es explicitar el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas.
- Puede haber un plan global que explicite el énfasis a realizar sobre los distintos tipos de pruebas (verificación, integración).

Características del plan de pruebas

- Este plan debería contemplar cantidades significativas de eventualidades, de forma que errores en el diseño o la implementación se puedan solucionar y parte del personal que realiza las pruebas se pueda dedicar a otras actividades.
- Como otros planes, el plan de prueba no es un documento estático. Debe revisarse regularmente puesto que la prueba es una actividad dependiente del avance de la implementación. Si parte del sistema a probar está incompleto, el proceso de prueba del sistema no puede comenzar.
- Debes contar con un plan de pruebas global y tantos planes de pruebas como sean necesarios para cubrir el alcance del plan global.
- La planificación de las pruebas comienza desde las fases iniciales del desarrollo tal y como muestra el siguiente diagrama:

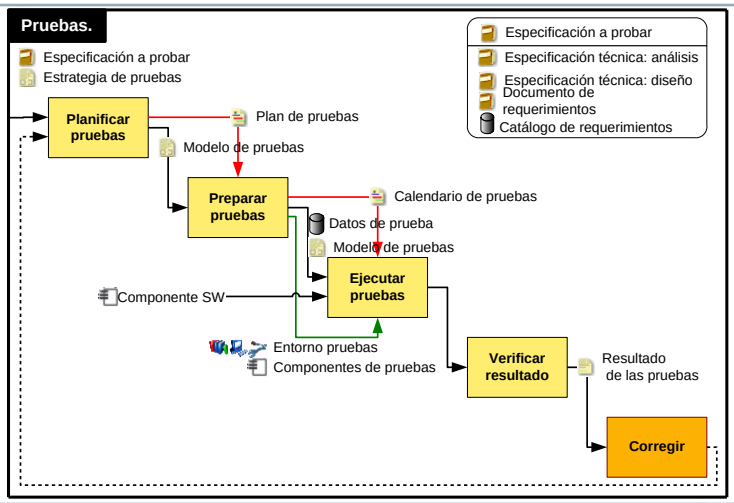
Proceso total de pruebas



Metodología

- El primer paso que debes dar es la **planificación de la prueba**, pasando a continuación a su **diseño**.
- Para diseñar la prueba debes identificar y diseñar los casos de prueba de integración, los casos de prueba de sistema, los casos de prueba de regresión y los procedimientos de prueba.
- Una vez concluido el diseño, debes **implementar los componentes de pruebas necesarios**.
- Por último realizas las **pruebas de integración** y la **prueba de sistema**.
- Concluyes el proceso con la **evaluación de la prueba**.

Metodología



Diseñar la prueba

- Para diseñar la prueba empiezas por identificar y describir los casos de prueba de cada componente.
- La selección de las técnicas de pruebas depende factores adicionales como pueden ser requisitos contractuales o normativos, documentación disponible, tiempo, presupuesto, conocimientos, experiencia, …
- Cuando dispongas de los casos de prueba, identificas y estructuras los procedimientos de prueba describiendo cómo ejecutar los casos de prueba.

Patrones

- Los casos de prueba se pueden estructurar siguiendo diferentes patrones:
  - ARRANGE-ACT-ASSERT: Preparar, Actuar, Afirmar
  - GIVEN-WHEN-THEN: Dado, Cuando, Entonces
  - BUILD-OPERATE-CHECK: Generar, Operar, Comprobar
- Aunque con diferencias conceptuales, todos dividen el proceso en tres fases:
  - Una fase inicial donde montar el escenario de pruebas que hace que el resultado sea predecible.
  - Una fase intermedia donde se realizan las acciones que son el objetivo de la prueba.
  - Una fase final donde se comparan los resultados con el escenario previsto. Pueden tomar la forma de:
    - Aserción: Es una afirmación sobre el resultado que puede ser cierta o no.
    - Expectativa: Es la expresión del resultado esperado que puede cumplirse o no.

Simulación de objetos

- Las dependencias con sistemas externos afectan a la complejidad de la estrategia de pruebas, ya que es necesario contar con sustitutos de estos servicios externos durante el desarrollo. Ejemplos típicos de estas dependencias son Servicios Web, Sistemas de envío de correo, Fuentes de Datos o simplemente dispositivos hardware.
- Estos sustitutos, muchas veces son exactamente iguales que el servicio original, pero en otro entorno o son simuladores que exponen el mismo interfaz pero realmente no realizan las mismas tareas que el sistema real, o las realizan contra un entorno controlado.
- Para poder emplear la técnica de simulación de objetos se debe diseñar el código a probar de forma que sea posible trabajar con los objetos reales o con los objetos simulados:
  - Doble herencia
  - IoC: Inversión de Control (Inversion Of Control)
  - DI: Inyección de Dependencias (Dependency Injection)
  - Objetos Mock

Programar para las interfaz

- Reutilizar la implementación de la clase base es la mitad de la historia.
- Ventajas:
  - Reducción de dependencias.
  - El cliente desconoce la implementación.
  - La vinculación se realiza en tiempo de ejecución.
  - Da consistencia (contrato).
- Desventaja:
  - Indireccionamiento.

Objetos mock

- La forma de establecer los valores esperados y "memorizar" el valor con el que se ha llamado al simulador para posteriormente verificarlo se ha generalizado dando lugar a un marco de trabajo que permite definir objetos simulados sin necesidad de crear explícitamente el código que verifica cada uno de los valores.
- Los MockObjects son objetos que siempre realizan las mismas tareas:
  - Implementan un interfaz dado
  - Permiten establecer los valores esperados (tanto de entrada como de salida)
  - Permiten establecer el comportamiento (para lanzar excepciones en casos concretos)
  - Memorizan los valores con los que se llama a cada uno de sus miembros
  - Permiten verificar si los valores esperados coinciden con los recibidos

Inversión de Control

- Inversión de control (Inversion of Control en inglés, IoC) es un concepto junto con unas técnicas de programación:
  - en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales,
  - en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.
- Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones.
- En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.

Inyección de Dependencias

- Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de arquitectura orientado a objetos, en el que se inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto, básicamente recomienda que las dependencias de una clase no sean creadas desde el propio objeto, sino que sean configuradas desde fuera de la clase.
- La inyección de dependencias (DI) procede del patrón de diseño más general que es la Inversión de Control (IoC).
- Al aplicar este patrón se consigue que las clases sean independientes unas de otras e incrementando la reutilización y la extensibilidad de la aplicación, además de facilitar las pruebas unitarias de las mismas.
- Desde el punto de vista de Java, un diseño basado en DI puede implementarse mediante el lenguaje estándar, dado que una clase puede leer las dependencias de otra clase por medio del API Reflection de Java y crear una instancia de dicha clase inyectándole sus dependencias.
- Hay frameworks que simplifican la aplicación del patrón DI como Spring y Google Guice.

Dobles de prueba

- La regla de oro de las pruebas unitarias, es que una unidad (unit) tiene que ser testeada sin utilizar ninguna de sus dependencias.
- Siguiendo la misma regla de oro, las pruebas de integración y sistema deben estar aisladas de sus dependencias salvo cuando se estén probando dichas dependencias. Así mismo, el resultado de las pruebas debe ser previsible.
- Entre las ventajas de esta aproximación se encuentran:
  - Devuelven resultados determinísticos
  - Permiten crear o reproducir determinados estados (por ejemplo errores de conexión)
  - Obtienen resultados mucho mas rápidamente y a menor coste, incluso offline.
  - Permiten el inicio temprano de las pruebas incluso cuando las dependencias todavía no están disponibles.
  - Permiten incluir atributos o métodos exclusivamente para el testeo.

Arrange (Preparar)

- **Fixture**: Es el término se utiliza para hablar de los datos de contexto de las pruebas, aquellos que se necesitan para construir el escenario que requiere la prueba.
- **Dummy**: Objeto que se pasa como argumento pero nunca se usa realmente. Normalmente, los objetos dummy se usan sólo para rellenar listas de parámetros.
- **Fake**: Objeto que tiene una implementación que realmente funciona pero, por lo general, usa una simplificación que le hace inapropiado para producción (como una base de datos en memoria por ejemplo).
- **Stub**: Objeto que proporciona respuestas predefinidas a llamadas hechas durante los tests, frecuentemente, sin responder en absoluto a cualquier otra cosa fuera de aquello para lo que ha sido programado. Los stubs pueden también grabar información sobre las llamadas (**spy**).
- **Mock**: Objeto preprogramado con expectativas que conforman la especificación de cómo se espera que se reciban las llamadas. Son más complejos que los stubs aunque sus diferencias son sutiles.

Desarrollo Guiado por Pruebas (TDD)

- El Desarrollo Guiado por Pruebas, es una técnica de programación (definida por KentBeck); consistente en desarrollar primero el código que pruebe una característica o funcionalidad deseada antes que el código que implementa dicha funcionalidad.
- El objetivo a lograr es que no exista ninguna funcionalidad que no esté avalada por una prueba.
- Lo primero que hay que aprender de TDD son sus reglas básicas:
  - No añadir código sin escribir antes una prueba que falle
  - Eliminar el Código Duplicado empleando Refactorización

Ritmo TDD

- TDD invita a seguir una serie de tareas ordenadas, que a menudo se denomina ritmo TDD, y que se basa en los siguientes pasos:
  1. Escribir una prueba que demuestre la necesidad de escribir código.
  2. Escribir el mínimo código para que el código de pruebas compile
  3. Implementar exclusivamente la funcionalidad demandada por las pruebas
  4. Mejorar el código (Refactoring) sin añadir funcionalidad
  5. Volver al primer paso
- Este ritmo permite formalizar las tareas que se han de realizar para conseguir un código fácil de mantener, bien diseñado y que se puede probar automáticamente.

Beneficios de TDD

- Reducen el número de errores y bugs ya que éstos, aplicando TDD, se detectan antes incluso de crearlos.
- Facilitan entender el código y que, eligiendo una buena nomenclatura, sirven de documentación.
- Facilitan mantener el código:
  - Protege ante cambios, los errores que surgen al aplicar un cambio se detectan (y corrigen) antes de subir ese cambio.
  - Protegen ante errores de regresión (rollbacks a versiones anteriores).
  - Dan confianza.
- Facilitan desarrollar ciñéndose a los requisitos.
- Ayudan a encontrar inconsistencias en los requisitos
- Ayudan a especificar comportamientos
- Ayudan a refactorizar para mejorar la calidad del código (Clean code)
- A medio/largo plazo aumenta (y mucho) la productividad.

Desarrollo Dirigido por Tests de Aceptación (ATDD)

- El Desarrollo Dirigido por Test de Aceptación (ATDD), técnica conocida también como Story Test-Driven Development (STDD), es una variación del TDD pero a un nivel diferente.
- Las pruebas de aceptación o de cliente son el criterio escrito de que un sistema cumple con el funcionamiento esperado y los requisitos de negocio que el cliente demanda. Son ejemplos escritos por los dueños de producto. Es el punto de partida del desarrollo en cada iteración.
- ATDD/STDD es una forma de afrontar la implementación de una manera totalmente distinta a las metodologías tradicionales. Cambia el punto de partida, la forma de recoger y formalizar las especificaciones, sustituye los requisitos escritos en lenguaje natural (nuestro idioma) por historias de usuario con ejemplos concretos ejecutables de como el usuario utilizara el sistema, que en realidad son casos de prueba. Los ejemplos ejecutables surgen del consenso entre los distintos miembros del equipo y el usuario final.
- La lista de ejemplos (pruebas) de cada historia, se escribe en una reunión que incluye a dueños de producto, usuarios finales, desarrolladores y responsables de calidad. Todo el equipo debe entender qué es lo que hay que hacer y por qué, para concretar el modo en que se certifica que el sotfware lo hace.

Desarrollo Dirigido por Tests de Aceptación (ATDD)

- El algoritmo o ritmo es el mismo de tres pasos que en el TDD practicado exclusivamente por desarrolladores pero a un nivel superior.
- En ATDD hay dos prácticas claves:
  - Antes de implementar (fundamental lo de antes de implementar) una necesidad, requisito, historia de usuario, etc., los miembros del equipo colaboran para crear escenarios, ejemplos, de cómo se comportará dicha necesidad.
  - Después, el equipo convierte esos escenarios en pruebas de aceptación automatizadas. Estas pruebas de aceptación típicamente se automatizan usando Selenium o similares, "frameworks" como Cucumber, etc.

Desarrollo Dirigido por Comportamiento (BDD)

- El Desarrollo Dirigido por Comportamiento (Behaviour Driver Development) es una evolución de TDD (Test Driven Development o Desarrollo Dirigido por Pruebas), el concepto de BDD fue inicialmente introducido por Dan North como respuesta a los problemas que surgían al enseñar TDD.
- En BDD también vamos a escribir las pruebas antes de escribir el código fuente, pero en lugar de pruebas unitarias, lo que haremos será escribir pruebas que verifiquen que el comportamiento del código es correcto desde el punto de vista de negocio. Tras escribir las pruebas escribimos el código fuente de la funcionalidad que haga que estas pruebas pasen correctamente. Después refactorizamos el código fuente.
- Partiremos de historias de usuario, siguiendo el modelo "Como [rol] quiero [característica] para [los beneficios]". A partir de aquí, en lugar de describir en 'lenguaje natural' lo que tiene que hacer esa nueva funcionalidad, vamos a usar un lenguaje ubicuo (un lenguaje semiformal que es compartido tanto por desarrolladores como personal no técnico) que nos va a permitir describir todas nuestras funcionalidades de una única forma.

BDD

- Para empezar a hacer BDD sólo nos hace falta conocer 5 palabras, con las que construiremos sentencias con las que vamos a describir las funcionalidades:
  - Feature (característica): Indica el nombre de la funcionalidad que vamos a probar. Debe ser un título claro y explícito. Incluimos aquí una descripción en forma de historia de usuario: “Como [rol] quiero [característica] para [los beneficios]”. Sobre esta descripción empezaremos a construir nuestros escenarios de prueba.
  - Scenario: Describe cada escenario que vamos a probar.
  - Given (dado): Provee el contexto para el escenario en que se va a ejecutar el test, tales como el punto donde se ejecuta el test, o prerequisitos en los datos. Incluye los pasos necesarios para poner al sistema en el estado que se desea probar.
  - When (cuando): Especifica el conjunto de acciones que lanzan el test. La interacción del usuario que acciona la funcionalidad que deseamos testear.
  - Then (entonces): Especifica el resultado esperado en el test. Observamos los cambios en el sistema y vemos si son los deseados.

ATDD

- El algoritmo o ritmo es el mismo de tres pasos que en el TDD practicado exclusivamente por desarrolladores pero a un nivel superior.
- En ATDD hay dos prácticas claves:
  - Antes de implementar (fundamental lo de antes de implementar) una necesidad, requisito, historia de usuario, etc., los miembros del equipo colaboran para crear escenarios, ejemplos, de cómo se comportará dicha necesidad.
  - Después, el equipo convierte esos escenarios en pruebas de aceptación automatizadas. Estas pruebas de aceptación típicamente se automatizan usando Selenium o similares, “frameworks” como Cucumber, etc.

Data Driven Testing (DDT)

- Se basa en la creación de tests para ejecutarse en simultáneo con sus conjuntos de datos relacionados en un framework. El framework provee una lógica de test reusable para reducir el mantenimiento y mejorar la cobertura de test. La entrada y salida (del criterio de test) pueden ser resguardados en uno o más lugares del almacenamiento central o bases de datos, el formato real y la organización de los datos serán específicos para cada caso.
- Todo lo que tiene potencial de cambiar (también llamado "variabilidad," e incluye elementos como el entorno, puntos de salida, datos de test, ubicaciones, etc) está separado de la lógica del test (scripts) y movido a un 'recurso externo'. Esto puede ser configuración o conjunto de datos de test. La lógica ejecutada en el script está dictada por los valores.
- Los datos incluyen variables usadas tanto para la entrada como la verificación de la salida. En casos avanzados (y maduros) los entornos de automatización pueden ser obtenidos desde algún sistema usando los datos reales o un "sniffer", el framework DDT por lo tanto ejecuta pruebas sobre la base de lo obtenido produciendo una herramienta de test automáticos para regresión.

SERVICIOS WEB

¿Qué es una API?

- API es el acrónimo de Application Programming Interface, que es un intermediario de software que permite que dos aplicaciones se comuniquen entre sí. A lo largo de los años, lo que es una API a menudo se ha descrito como cualquier tipo de interfaz de conectividad genérica para una aplicación. Más recientemente, sin embargo, una API moderna ha adquirido algunas características que las hacen mas valiosas y útiles:
  - Las API modernas se adhieren a los estándares (servicios web generalmente HTTP y REST), que son amigables para los desarrolladores, de fácil acceso y comprensibles ampliamente
  - Se tratan más como productos que como código. Están diseñados para el consumo de audiencias específicas, están documentados y están versionadas de manera que los usuarios puedan tener ciertas expectativas sobre su mantenimiento y ciclo de vida.
  - Debido a que están mucho más estandarizados, tienen una disciplina mucho más sólida para la seguridad y la gobernanza, además de monitorear y administrar el rendimiento y la escalabilidad.
  - Como cualquier otra pieza de software producido, una API moderna tiene su propio ciclo de vida de desarrollo de software (SDLC) de diseño, prueba, construcción, administración y control de versiones.

¿Qué es un Servicio Web?

- Componentes que permiten a las aplicaciones compartir datos siguiendo un modelo de programación distribuida basados en estándares abiertos de tipo texto que permite interactuar entre si a sistemas heterogéneos.

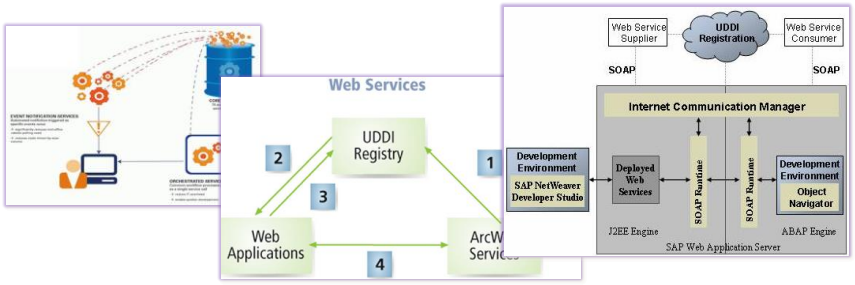
- Según Microsoft:
  - "… es una entidad programable que proporciona un elemento de funcionalidad determinado, como lógica de aplicación … (accesible) mediante estándares de Internet … como XML y HTTP."

- Según Sun:
  - "… es un componente software con las siguientes características:
  - 1) Es accesible a través del interfaz SOAP,
  - 2) Su interfaz se describe en un documento WSDL."

- Según Wikipedia:
  - "… es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones."

¿Qué es un Servicio Web?

- Pero esto no es todo, aun podemos definirlo de mas formas
  - De otra forma:
    - "A través de una interfaz estándar … permite el funcionamiento de una serie de sistemas heterogéneos como un conjunto integrado."
  - Simplificando:
    - "Sistema diseñado para soportar interoperabilidad entre máquinas sobre una red."



¿Qué es REALMENTE un Servicio Web?

- Es un «elemento» accesible a través de Internet, mediante la invocación remota
- Se deben de alojar en un Servidor Web. (que soporte Web Services)
- Debe ser:
  - Autocontenido
  - Autodescrito
  - Modular e
  - Independiente de Plataforma
- Utilizar estándares (XML, SOAP, RPC) para poder ofrecer sus servicios

¿Qué es REALMENTE un Servicio Web?

- Un servicio web es:
  - Una aplicación remota
  - Descrito usando WSDL (Web Service Description Language)
  - Accediendo mediante SOAP (Simple Object Access Protocol)
  - Todo esto bajo una serie de reglas/estándares definidas por WS-I Basic Profile
- El WS-I (Web Services Organización de Integración) es un grupo de proveedores (Microsoft, IBM, BEA, Sun Microsystems, Oracle, HP, y otros) que se han unido para garantizar que los servicios web son interoperables en todas las plataformas.
- Para ello, han creado una recomendación llamado Basic Profile 1.1, que define un conjunto de reglas para el uso de XML, SOAP y WSDL para crear servicios web interoperables.

¿Qué es REALMENTE un Servicio Web?

- Los servicios Web son sistemas que se comunican principalmente a través de HTTP.
- Pero puede comunicarse a través de otros protocolos de también populares y con posibilidades de ser tratado (inspeccionado, transformado, persistido, etc) como XML o JSON
- Normalmente, las cargas útiles de comunicaciones de servicios Web suelen ser generadas en XML.
- También es posible la comunicación de datos binarios compactos entre Web Services.

Estilos arquitectónicos

- Precursores:
  - RPC: Llamadas a Procedimientos Remotos
  - Binarios: CORBA, Java RMI, .NET Remoting
  - XML-RPC: Precursor del SOAP
- Actuales:
  - Servicios Web XML o Servicios SOAP
  - Servicios Web REST o API REST
    - WebHooks
  - Servicios GraphQL
  - Servicios gRPC

AÑO	Descripción
1976	Aparición de RPC (Remote Procedure Call) en Sistema Unix
1990	Aparición de DCE (Distributed Computing Environment) que es un sistema de software para computación distribuida, basado en RPC.
1991	Aparición de Microsoft RPC basado en DCE para sistemas Windows.
1992	Aparición de DCOM (Microsoft) y CORBA (ORB) para la creación de componentes software distribuidos.
1997	Aparición de Java RMI en JDK 1.1
1998	Aparición de XML-SOAP
1999	Aparición de SOAP 1.0, WSDL, UDDI
2000	Definición del REST
2012	Propuesta de GraphQL por Facebook
2015	Desarrollo de gGRPC por Google

Estilos arquitectónicos

- Basados en Recursos: Los servicios REST / Hypermedia exponen documentos que incluyen tanto identificadores de datos como de acciones (enlaces y formularios). REST es un estilo arquitectónico que separa las preocupaciones del consumidor y del proveedor de la API al depender de comandos que están integrados en el protocolo de red subyacente. REST (Representational State Transfer) es extremadamente flexible en el formato de sus cargas útiles de datos, lo que permite una variedad de formatos de datos populares como JSON y XML, entre otros.

- Basados en Procedimientos: Las llamadas a procedimiento remoto, o RPC, generalmente requieren que los desarrolladores ejecuten bloques específicos de código en otro sistema: operaciones. RPC es independiente del protocolo, lo que significa que tiene el potencial de ser compatible con muchos protocolos, pero también pierde los beneficios de usar capacidades de protocolo nativo (por ejemplo, almacenamiento en caché). La utilización de diferentes estándares da como resultado un acoplamiento más estrecho entre los consumidores y los proveedores de API y las tecnologías implicadas, lo que a su vez sobrecarga a los desarrolladores involucrados en todos los aspectos de un ecosistema de APIs impulsado por RPC. Los patrones de arquitectura de RPC se pueden observar en tecnologías API populares como SOAP, GraphQL y gRPC.

- Basados en Eventos/Streaming: a veces denominadas arquitecturas de eventos, en tiempo real, de transmisión, asíncronas o push, las APIs impulsadas por eventos no esperan a que un consumidor de la API las llame antes de entregar una respuesta. En cambio, una respuesta se desencadena por la ocurrencia de un evento. Estos servicios exponen eventos a los que los clientes pueden suscribirse para recibir actualizaciones cuando cambian los valores del servicio. Hay un puñado de variaciones para este estilo que incluyen (entre otras) reactivo, publicador/suscriptor, notificación de eventos y CQRS.

Servicios SOAP

- SOAP es un protocolo de comunicación web altamente estandarizado basado en formatos de tipo texto en XML. Publicado por Microsoft en 1999, un año después de XML-RPC, SOAP heredó mucho de él.
- Basado en operaciones, de tipos texto, en formato XML y comunicaciones síncronas.
- Ventajas:
  - Independiente del lenguaje y la plataforma: La funcionalidad incorporada para crear servicios basados en web permite a SOAP manejar las comunicaciones y hacer que las respuestas sean independientes del lenguaje y la plataforma.
  - Vinculado a una variedad de protocolos de transporte: SOAP es flexible en términos de protocolos de transferencia para adaptarse a múltiples escenarios.
  - Manejo de errores incorporado: La especificación de la API SOAP permite devolver el mensaje Retry XML con el código de error y su explicación.
  - Varias extensiones de seguridad: Integrado con los protocolos WS-Security, SOAP cumple con una calidad de transacción de nivel empresarial. Proporciona privacidad e integridad dentro de las transacciones al tiempo que permite el cifrado a nivel de mensaje.

Servicios SOAP

- Inconvenientes:
  - Solamente acepta formato XML: Los mensajes SOAP contienen una gran cantidad de metadatos y solo admiten estructuras XML detalladas para solicitudes y respuestas.
  - De peso pesado: Debido al gran tamaño de los archivos XML, los servicios SOAP requieren un gran ancho de banda hasta para la información mas nimia.
  - Conocimientos estrictamente especializados: La creación de servidores de API SOAP requiere un conocimiento profundo de todos los protocolos involucrados y sus reglas altamente restringidas o disponer de framework que simplifiquen su utilización que no están disponibles en todas las plataformas y lenguajes.
  - Actualización de mensajes tediosa: Al requerir un esfuerzo adicional para agregar o eliminar las propiedades del mensaje, el esquema SOAP rígido ralentiza la adopción.

Servicios REST

- REpresentational State Transfer es un estilo arquitectónico autoexplicativo basado en el uso del protocolo HTTP e hipermedia y establece un conjunto de restricciones arquitectónicas y destinado a una amplia adopción. RESTful hace referencia a un servicio web que implementa la arquitectura REST.
- Restringe la forma de usar de las URLs, los métodos (verbos) de HTTP, sus encabezados (Accept, Content-type, …) y sus códigos de estado.
- Basado en recursos, independiente de formato (texto o binario) y comunicaciones síncronas.
- Ventajas:
  - Soporte de múltiples formatos: La capacidad de admitir múltiples formatos (a menudo JSON y XML) para almacenar e intercambiar datos es una de las razones por las que REST es actualmente una opción predominante para crear APIs públicas.
  - Documentación mínima, basados en convenios y el descubrimiento hipermedia.
  - Mínima curva de aprendizaje: usa tecnologías ampliamente conocidas: HTTP, URL, MIME, …

Servicios REST

- Inconvenientes:
  - Uso de HTTP: Aunque es la infraestructura mas ampliamente difundida y utilizada, está restringido a ella.
  - Grandes cargas útiles: REST devuelve una gran cantidad de metadatos enriquecidos para que el cliente pueda comprender todo lo necesario sobre el estado de la aplicación solo a partir de sus respuestas.
  - Sin estructura REST única: No existe una forma exacta y correcta de crear una API REST. Cómo modelar los recursos y qué recursos modelar dependerá de cada escenario. Esto hace que REST sea simple en teoría, pero difícil en la práctica.
  - Problemas de recuperación excesiva o insuficiente: Devolver todo suele conllevar demasiados datos y el convenio no estable mecanismos de filtrado, paginación o proyecciones. El uso de hipermedia para obtener datos relacionados requiere solicitudes adicionales y encadenadas.
  - Convenio genérico: En muchos casos no acorde con las reglas de negocio (create o replace, delete, …) lo que acaba requiriendo documentación.

WebHooks

- Los webhooks son eventos que desencadenan acciones. Su nombre se debe a que funcionan como «ganchos» de los programas en Internet y casi siempre se utilizan para la comunicación entre sistemas. Son la manera más sencilla de obtener un aviso cuando algo ocurre en otro sistema y para el intercambio de datos entre aplicaciones web.
- Un webhook es una retro llamada HTTP, una solicitud HTTP POST insertada en una página web, que interviene cuando ocurre algo (una notificación de evento a través de HTTP POST).
- Los webhooks se utilizan para las notificaciones en tiempo real (con los datos del evento en JSON o XML) a una determinada dirección http:// o https://, que puede:
  - almacenar los datos del evento en JSON o XML
  - generar una respuesta que permita actualizarse al sistema donde se produce el evento
  - ejecutar un proceso en el sistema receptor del evento (Ej: enviar un correo electrónico)
- Los webhook están pensados para su utilización desde páginas web y sus diferentes consumidores: navegadores, correo electrónico, webapps, …
- Pueden considerarse una versión especializada y simplificada de los servicios REST (solo POST).

Servicios GraphQL

- GraphQL es una sintaxis que describe cómo obtener la descripción del modelo de datos y cómo realizar una solicitud de datos precisa, consultas y mutaciones, pensada para los modelos de datos con muchas entidades complejas que hacen referencia entre sí. Fue propuesta por Facebook en 2012, publicada en 2015 y posteriormente pasada a open source en 2018.
- Basado en consultas, en formato JSON y comunicaciones síncronas.
- Ventajas:
  - Un esquema de GraphQL establece una fuente única de información, ofrece una forma de unificar todo en un único servicio.
  - Las llamadas a GraphQL se gestionan mediante HTTP POST en un solo recorrido de ida y vuelta. Los clientes obtienen lo que solicitan sin que se genere una sobrecarga.
  - Los tipos de datos bien definidos reducen los problemas de comunicación entre el cliente y el servidor. Un cliente puede solicitar una lista de los tipos de datos disponibles y esto es ideal para la generación automática de documentación.
  - GraphQL permite que las APIs de las aplicaciones evolucionen sin afectar a las consultas actuales.
  - GraphQL no exige una arquitectura de aplicación específica. Puede incorporarse sobre una API de REST actual y funcionar con las herramientas de gestión de APIs actuales. Hay muchas extensiones open source de GraphQL que ofrecen características que no están disponibles con las APIs de REST.

Servicios GraphQL

- Inconvenientes:
  - GraphQL intercambia complejidad por flexibilidad. Tener demasiados campos anidados en una solicitud puede provocar una sobrecarga del sistema. Además, delega gran parte del trabajo de las consultas de datos en el servidor, lo cual representa una mayor complejidad para los desarrolladores de servidores.
  - GraphQL no indica cómo almacenar los datos ni qué lenguaje de programación utilizar, requiere disponer de framework que simplifiquen su utilización que no están disponibles en todas las plataformas.
  - Como GraphQL no utiliza la semántica de almacenamiento en caché HTTP, requiere un esfuerzo de almacenamiento en caché personalizado.
  - GraphQL presenta una curva de aprendizaje elevada para desarrolladores que tienen experiencia con las APIs de REST.

Servicios gRPC

- gRPC es un marco de llamada a procedimiento remoto (RPC) de alto rendimiento e independiente de lenguaje. Desarrollado por Google en el año 2015, y luego convertido en código abierto. Como GraphQL, es una especificación que se implementa en una variedad de lenguajes.
- Basado en contratos, en el formato binario Protocol Buffers y comunicaciones síncronas/asíncronas.
- Las principales ventajas de gRPC son:
  - Marco de RPC moderno, ligero y de alto rendimiento.
  - La especificación gRPC es preceptiva con respecto al formato que debe seguir un servicio gRPC, formaliza un contrato independiente de lenguaje.
  - Es compatible con el intercambio de datos bidireccional y asíncrono al estar basado en HTTP/2, así como con la compresión, multiplexación y streaming.
  - Uso reducido de red al usar un formato de mensaje binario eficaz que genera cargas de mensajes pequeñas.

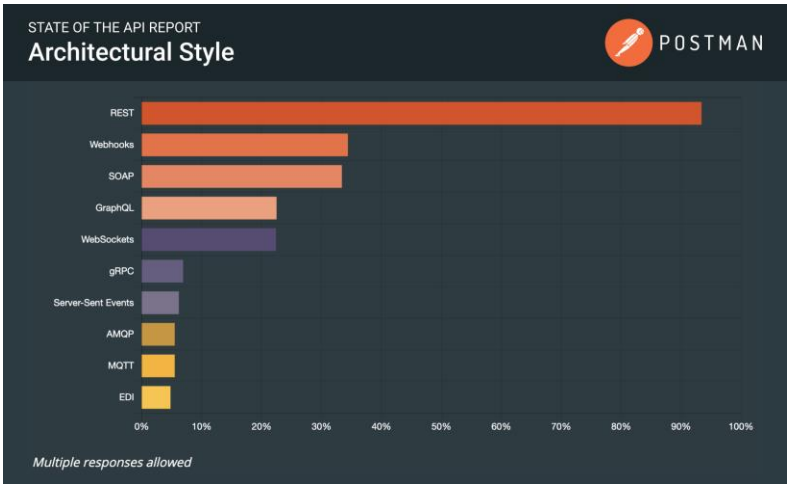
Servicios gRPC

- Inconvenientes:
  - Tanto el cliente como el servidor deben admitir la misma especificación de búfer de protocolo, requiere un estricto control de versiones. gRPC es un formato muy particular que proporciona una ejecución ultrarrápida a expensas de la flexibilidad.
  - Basado en HTTP/2 que no tiene soporte universal para interacciones cliente-servidor de cara al público general en Internet y una compatibilidad limitada con exploradores.
  - gRPC es una especificación, por lo que requiere disponer de framework que permitan su utilización tanto en el cliente como en le servidor y que no están disponibles en todas las plataformas y lenguajes.
  - El proceso de serialización/deserialización para su conversión a binario es costoso en términos de CPU.
  - gRPC presenta una curva de aprendizaje mas empinada que la de REST.

Protocolos y Estándares

- Síncronos:
  - SOAP: Basado en operaciones, de tipos texto, en formato XML y comunicaciones síncronas.
  - REST: Basado en recursos, independiente de formato (texto/binario) y comunicaciones síncronas.
  - GraphQL: Basado en consultas, en formato JSON y comunicaciones síncronas.
  - gRPC: Basado en contratos, en el formato binario Protocol Buffers y comunicaciones síncronas/asíncronas.
- Asíncronos:
  - WebSockets: protocolo estándar abierto, elemental, texto y binario.
  - STOMP (Simple/Streaming Text Oriented Messaging Protocol): protocolo estándar abierto, basado en texto, soluciones simples y ligero.
  - AMQP (Advanced Message Queuing Protocol): protocolo estándar abierto, binario, encolamiento, P2P y Pub/Sub, exactitud y seguridad
  - MQTT (Message Queue Telemetry Transport): protocolo estándar abierto, binario, P2P, liviano, soluciones simples y seguridad
  - JMS (Java Message Service): API, binario Java, P2P y Pub/Sub

Estilos arquitectónicos mas utilizados

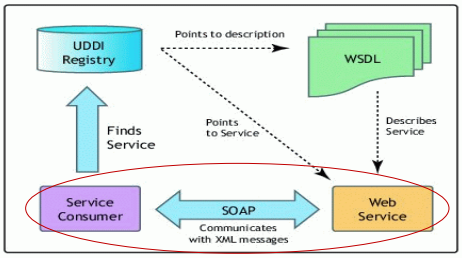


https://www.postman.com/state-of-api/api-technologies/#api-technologies

TECNOLOGÍAS DE SERVICIOS WEB

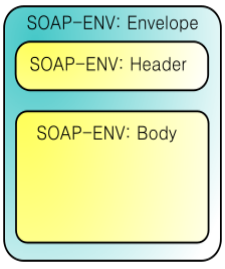
El protocolo SOAP

- Simple Object Access Protocol (SOAP) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.
- Fundamentalmente, SOAP es la capa de transporte por defecto para los servicios Web.



Elementos SOAP

- Envelope
  - Identifica el documento XML como un mensaje SOAP.
  - En el cual define qué hay en el mensaje y cómo procesarlo.
  - Marca el inicio y el final del mensaje SOAP
- Header (opcional)
  - Cuando se utiliza el protocolo SOAP sobre HTTP, las cabeceras HTTP proporcionan información sobre el tipo de contenido, longitud del contenido y el destinatario del mensaje.
- Body
  - Contiene información de la llamada y respuesta.
  - El cuerpo es un elemento obligatorio que contiene la información para el destinatario del mensaje



Ejemplo SOAP

Como ejemplo se muestra la forma en que un cliente solicitaría información de un producto a un proveedor de servicios Web:

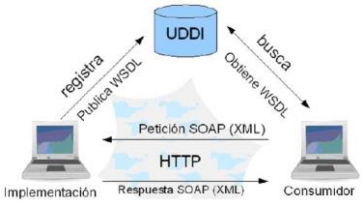
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productId>827635</productId>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

Y esta sería la respuesta del proveedor:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Toptimate 3-Piece Set</productName>
         <productId>827635</productId>
         <description>3-Piece luggage set.  Black Polyester.</description>
         <price>96.50</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```


Descripción de servicios Web con WSDL

- Web Services Description Language (WSDL) es una tecnología que se utiliza para describir la interfaz pública de un servicio Web.
- WSDL es un estándar desarrollado por el W3C que describe:
  - Lo que hace un servicio
  - Cómo invocar sus operaciones
  - Y dónde encontrarlo.



- Está basado y creado en lenguaje XML.

Elementos de un documento WSDL

- Tipos de Datos <types>
  - Define los tipos de datos usados en los mensajes.
  - Se utilizan los tipos definidos en la especificación de esquemas XML.
- Mensajes <message>
  - Se utilizan para definir el formato de los datos intercambiados entre un consumidor de servicios Web y el proveedor de servicios.
- Tipos de Puerto <portType>:
  - Se utiliza para especificar las operaciones del servicio Web.
- Bindings <binding>:
  - Describe el protocolo, formato de datos y la seguridad de un elemento <portType>
  - Los Bindings estándar son HTTP o SOAP, o uno propio.
- Servicios <service>:
  - Conjunto de direcciones y puertos para poder interactuar con los servicios.

Ejemplo WSDL

```
<portType name="CreditCheckEndpointBean">
        <operation name="CreditCheck">
                <input message="tns:CreditCheck"/>
                <output message="tns:CreditCheckResponse"/>
        </operation>
</portType>
```

```
<binding name="CreditCheckEndpointBeanPortBinding"   type="tns:CreditCheckEndpointBean">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="CreditCheck">
        <soap:operation soapAction=""/>
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
    </operation>
</binding>
```

```
<service name="CreditService">
        <port name="CreditCheckEndpointBeanPort"  binding="tns:CreditCheckEndpointBeanPortBinding">
                <soap:address location="http://localhost:64082/CreditService/CreditCheckEndpointBean"/>
        </port>
</service>
```

REST (REpresentational State Transfer)

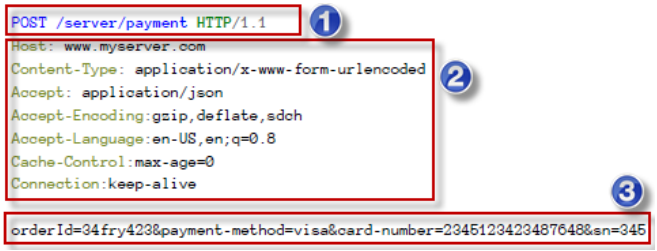
- En 2000, Roy Fielding propuso la transferencia de estado representacional (REST) como enfoque de arquitectura para el diseño de servicios web. REST **es un estilo de arquitectura** para la creación de sistemas distribuidos basados en hipermedia. REST es independiente de cualquier protocolo subyacente y no está necesariamente unido a HTTP. Sin embargo, en las implementaciones más comunes de REST se usa HTTP como protocolo de aplicación, y esta guía se centra en el diseño de API de REST para HTTP.
- Originalmente se basaba en lo que ya estaba disponible en HTTP:
  - URL como identificadores de recursos
  - HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT + PATCH (HTTP1.1)
  - HTTP permite transmitir en el encabezado la información de comportamiento: Content-type, Accept, Authorization, Cache-control, …
  - HTTP utiliza códigos de estado en la respuesta para indicar como se ha completado una solicitud HTTP específica: respuestas informativas (1xx), respuestas satisfactorias (2xx), redirecciones (3xx), Errores en la petición (4xx) y errores de los servidores (5xx).

Estilo de arquitectura REST

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP
- Request: Método /uri?parámetros
  - GET: Recupera el recurso (200)
    - Todos: Sin identificador
    - Uno: Con identificador
  - POST: Crea o reemplaza un nuevo recurso (201)
  - PUT: Crea o reemplaza el recurso identificado (200, 204)
  - DELETE: Elimina el recurso (204)
    - Todos: Sin identificador
    - Uno: Con identificador
- Accept: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- Content-type: Indica en que formato está codificado el cuerpo, utilizando MIME
- HTTP Status Code: Código de estado con el que el servidor informa del resultado de la petición.

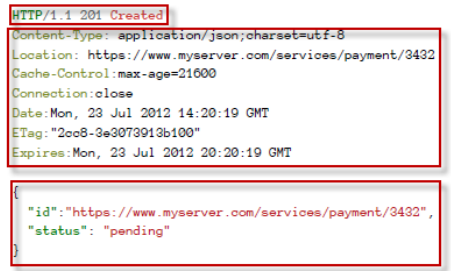
Petición HTTP

- Cuando realizamos una petición HTTP, el mensaje consta de:
  - Primera línea de texto indicando la versión del protocolo utilizado, el verbo y el URI
    - El verbo indica la acción a realizar sobre el recurso web localizado en la URI
  - Posteriormente vendrían las cabeceras (opcionales)
  - Después el cuerpo del mensaje, que contiene un documento, que puede estar en cualquier formato (XML, HTML, JSON → Content-type)



Respuesta HTTP

- Los mensajes HTTP de respuesta siguen el mismo formato que los de envío.
- Sólo difieren en la primera línea
  - Donde se indica un código de respuesta junto a una explicación textual de dicha respuesta.
  - El código de respuesta indica si la petición tuvo éxito o no.



Recursos

- Un recurso es cualquier elemento que dispone de un URI correcto y único.
- Es cualquier cosa que sea direccionable a través de internet.
- Estos recursos pueden ser manipulados por clientes y servidores.
- En REST todos los recursos comparten una interfaz única y constante. (http://...)
- Todos los recursos tienen las mismas operaciones (CRUD)
  - CREATE, READ, UPDATE, DELETE
- Las URI es el único medio por el que los clientes y servidores pueden realizar el intercambio de representaciones.
- Normalmente estos recursos son accesibles en una red o sistema.
- Para que un URI sea correcto, debe de cumplir los requisitos de formato, REST no indica de forma específica un formato obligatorio.
  - <esquema>://<host>:puerto/<ruta><querystring><fragmento>
- Los URI asociados a los recursos pueden cambiar si modificamos el recurso (nombre, ubicación, características, etc)

Tipos MIME

- Otro aspecto muy importante es la posibilidad de negociar distintos formatos (representaciones) a usar en la transferencia del estado entre servidor y cliente (y viceversa). La representación de los recursos es el formato de lo que se envía un lado a otro entre clientes y servidores. Como REST utiliza HTTP, podemos transferir múltiples tipos de información.
- Los datos se transmiten a través de TCP/IP, el navegador sabe cómo interpretar las secuencias binarias (CONTENT-TYPE) por el protocolo HTTP.
- La representación de un recurso depende del tipo de llamada que se ha generado (Texto, HTML, PDF, etc).
- En HTTP cada uno de estos formatos dispone de su propio tipos MIME, en el formato <tipo>/<subtipo>.
  - application/json application/xml text/html text/plain image/jpeg
- Para negociar el formato:
  - El cliente, en la cabecera ACCEPT, envía una lista priorizada de tipos MIME que entiende.
  - Tanto cliente como servidor indican en la cabecera CONTENT-TYPE el formato MIME en que está codificado el body.
- Si el servidor no entiende ninguno de los tipos MIME propuestos (ACCEPT) devuelve un mensaje con código 406 (incapaz de aceptar petición).

Métodos HTTP

HTTP	REST	Descripción
GET	RETRIEVE	Sin identificador: Recuperar el estado completo de un recurso (HEAD + BODY) Con identificador: Recuperar el estado individual de un recurso (HEAD + BODY)
HEAD		Recuperar la cabecera del estado de un recurso (HEAD)
POST	CREATE or REPLACE	Crea o modifica un recurso (sin identificador)
PUT	CREATE or REPLACE	Crea o modifica un recurso (con identificador)
DELETE	DELETE	Sin identificador: Elimina todo el recurso Con identificador: Elimina un elemento concreto del recurso
CONNECT		Comprueba el acceso al host
TRACE		Solicita al servidor que introduzca en la respuesta todos los datos que reciba en el mensaje de petición
OPTIONS		Devuelve los métodos HTTP que el servidor soporta para un URL específico
PATCH	REPLACE	HTTP 1.1 Reemplaza parcialmente un elemento del recurso

Códigos HTTP (status)

status	statusText	Descripción
100	Continue	Una parte de la petición (normalmente la primera) se ha recibido sin problemas y se puede enviar el resto de la petición
101	Switching protocols	El servidor va a cambiar el protocolo con el que se envía la información de la respuesta. En la cabecera Upgrade indica el nuevo protocolo
200	OK	La petición se ha recibido correctamente y se está enviando la respuesta. Este código es con mucha diferencia el que mas devuelven los servidores
201	Created	Se ha creado un nuevo recurso (por ejemplo una página web o un archivo) como parte de la respuesta
202	Accepted	La petición se ha recibido correctamente y se va a responder, pero no de forma inmediata
203	Non-Authoritative Information	La respuesta que se envía la ha generado un servidor externo. A efectos prácticos, es muy parecido al código 200
204	No Content	La petición se ha recibido de forma correcta pero no es necesaria una respuesta
205	Reset Content	El servidor solicita al navegador que inicialice el documento desde el que se realizó la petición, como por ejemplo un formulario
206	Partial Content	La respuesta contiene sólo la parte concreta del documento que se ha solicitado en la petición

Códigos de redirección

status	statusText	Descripción
300	Multiple Choices	El contenido original ha cambiado de sitio y se devuelve una lista con varias direcciones alternativas en las que se puede encontrar el contenido
301	Moved Permanently	El contenido original ha cambiado de sitio y el servidor devuelve la nueva URL del contenido. La próxima vez que solicite el contenido, el navegador utiliza la nueva URL
302	Found	El contenido original ha cambiado de sitio de forma temporal. El servidor devuelve la nueva URL, pero el navegador debe seguir utilizando la URL original en las próximas peticiones
303	See Other	El contenido solicitado se puede obtener en la URL alternativa devuelta por el servidor. Este código no implica que el contenido original ha cambiado de sitio
304	Not Modified	Normalmente, el navegador guarda en su caché los contenidos accedidos frecuentemente. Cuando el navegador solicita esos contenidos, incluye la condición de que no hayan cambiado desde la última vez que los recibió. Si el contenido no ha cambiado, el servidor devuelve este código para indicar que la respuesta sería la misma que la última vez
305	Use Proxy	El recurso solicitado sólo se puede obtener a través de un proxy, cuyos datos se incluyen en la respuesta
307	Temporary Redirect	Se trata de un código muy similar al 302, ya que indica que el recurso solicitado se encuentra de forma temporal en otra URL

Códigos de error del navegador

status	statusText	Descripción
400	Bad Request	El servidor no entiende la petición porque no ha sido creada de forma correcta
401	Unauthorized	El recurso solicitado requiere autorización previa
402	Payment Required	Código reservado para su uso futuro
403	Forbidden	No se puede acceder al recurso solicitado por falta de permisos o porque el usuario y contraseña indicados no son correctos
404	Not Found	El recurso solicitado no se encuentra en la URL indicada. Se trata de uno de los códigos más utilizados y responsable de los típicos errores de *Página no encontrada*
405	Method Not Allowed	El servidor no permite el uso del método utilizado por la petición, por ejemplo por utilizar el método GET cuando el servidor sólo permite el método POST
406	Not Acceptable	El tipo de contenido solicitado por el navegador no se encuentra entre la lista de tipos de contenidos que admite, por lo que no se envía en la respuesta
407	Proxy Authentication Required	Similar al código 401, indica que el navegador debe obtener autorización del proxy antes de que se le pueda enviar el contenido solicitado
408	Request Timeout	El navegador ha tardado demasiado tiempo en realizar la petición, por lo que el servidor la descarta

Códigos de error del navegador

status	statusText	Descripción
409	Conflict	El navegador no puede procesar la petición, ya que implica realizar una operación no permitida (como por ejemplo crear, modificar o borrar un archivo)
410	Gone	Similar al código 404. Indica que el recurso solicitado ha cambiado para siempre su localización, pero no se proporciona su nueva URL
411	Length Required	El servidor no procesa la petición porque no se ha indicado de forma explícita el tamaño del contenido de la petición
412	Precondition Failed	No se cumple una de las condiciones bajo las que se realizó la petición
413	Request Entity Too Large	La petición incluye más datos de los que el servidor es capaz de procesar. Normalmente este error se produce cuando se adjunta en la petición un archivo con un tamaño demasiado grande
414	Request-URI Too Long	La URL de la petición es demasiado grande, como cuando se incluyen más de 512 bytes en una petición realizada con el método GET
415	Unsupported Media Type	Al menos una parte de la petición incluye un formato que el servidor no es capaz procesar
416	Requested Range Not Suitable	El trozo de documento solicitado no está disponible, como por ejemplo cuando se solicitan bytes que están por encima del tamaño total del contenido
417	Expectation Failed	El servidor no puede procesar la petición porque al menos uno de los valores incluidos en la cabecera Expect no se pueden cumplir

Códigos de error del servidor

status	statusText	Descripción
500	Internal Server Error	Se ha producido algún error en el servidor que impide procesar la petición
501	Not Implemented	Procesar la respuesta requiere ciertas características no soportadas por el servidor
502	Bad Gateway	El servidor está actuando de proxy entre el navegador y un servidor externo del que ha obtenido una respuesta no válida
503	Service Unavailable	El servidor está sobrecargado de peticiones y no puede procesar la petición realizada
504	Gateway Timeout	El servidor está actuando de proxy entre el navegador y un servidor externo que ha tardado demasiado tiempo en responder
505	HTTP Version Not Supported	El servidor no es capaz de procesar la versión HTTP utilizada en la petición. La respuesta indica las versiones de HTTP que soporta el servidor

Estilo de arquitectura

- Request: Método /uri?parámetros
  - GET: Recupera el recurso (200)
    - Todos: Sin identificador
    - Uno: Con identificador
  - POST: Crea o reemplaza un nuevo recurso (201)
  - PUT: Crea o reemplaza el recurso identificado (200, 204)
  - DELETE: Elimina el recurso (204)
    - Todos: Sin identificador
    - Uno: Con identificador
- Accept: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- Content-type: Indica en que formato está codificado el cuerpo, utilizando MIME
- HTTP Status Code: Código de estado con el que el servidor informa del resultado de la petición.

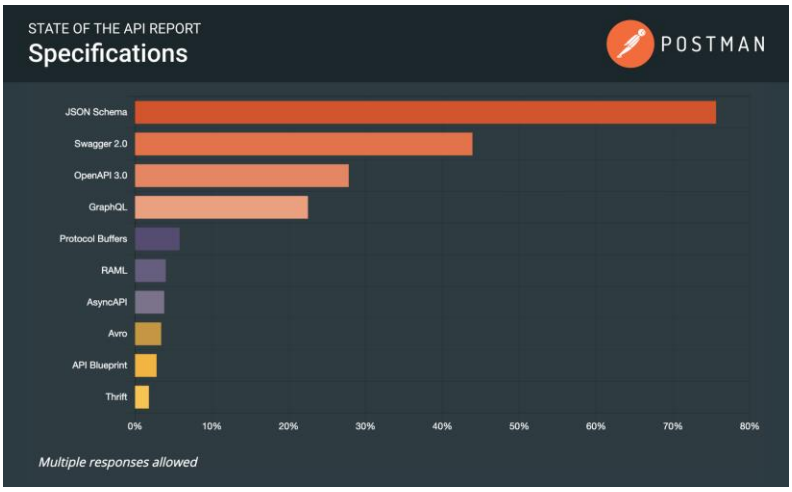
Peticiones REST

- Request: GET /users
- Response: 200
  - content-type:application/json
  - BODY
- Request: GET /users/11
- Response: 200
  - content-type:application/json
  - BODY
- Request: POST /users
  - BODY
- Response: 201
  - content-type:application/json
  - BODY
- Request: PUT /users
  - BODY
- Response: 200
  - content-type:application/json
  - BODY
- Request: DELETE /users/11
- Response: 204 no content

Documentar servicios Rest

- Web Application Description Language (WADL): Especificación de W3C, que la descripción XML legible por máquina de aplicaciones web basadas en HTTP (normalmente servicios web REST). Modela los recursos proporcionados por un servicio y las relaciones entre ellos. Está diseñado para simplificar la reutilización de servicios web basados en la arquitectura HTTP existente de la web. Es independiente de la plataforma y del lenguaje, tiene como objetivo promover la reutilización de aplicaciones más allá del uso básico en un navegador web.
- RAML: RESTful API Modeling Language es una forma práctica de describir un API RESTful de una manera que sea muy legible tanto para humanos como para maquinas.
- Open API (anteriormente Swagger): Especificación para describir, producir, consumir y visualizar servicios web RESTful. Es el más ampliamente difundido y cuenta con un ecosistema propio.
- JSON Schema: JSON Schema es una especificación para definir, anotar y validar las estructuras de datos JSON.
- api-spec-converter: Conversor entre formatos de descripción de API

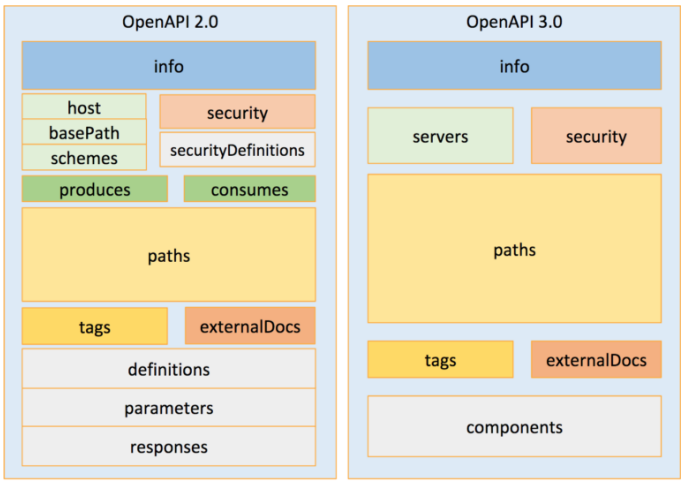
Especificaciones mas utilizadas



OpenAPI

- OpenAPI es un estándar para definir contratos de Api. Los cuales describen la interfaz de una serie de servicios que vamos a poder consumir por medio de una signatura. Conocido previamente como Swagger, ha sido adoptado por la Linux Foundation y obtuvo el apoyo de compañías como Google, Microsoft, IBM, Paypal, etc. para convertirse en un estándar para las API REST.
- Las definiciones de OpenAPI se pueden escribir en JSON o YAML. La versión actual de la especificación es la 3.0.1 y orientada a YAML y la versión previa la 2.0, que es idéntica a la especificación 2.0 de Swagger antes de ser renombrada a “Open API Specification”.
- Actualmente nos encontramos en periodo de transición de la versión 2 a la 3, sin soporte en muchas herramientas.

Cambio de versión



Sintaxis

- YAML es un lenguaje de serialización de datos similar a XML pero que utiliza el sangrado para indicar el anidamiento, estableciendo la estructura jerárquica, y evitar la necesidad de tener que cerra los elementos.
- El sangrado utiliza espacios en blanco, no se permite el uso de caracteres de tabulación.
- Los miembros de las listas van entre corchetes ([ ]) y separados por coma espacio (,), o uno por línea con un guion (-) inicial.
- Los vectores asociativos se representan usando los dos puntos seguidos por un espacio, "clave: valor", bien uno por línea o entre llaves ({ }) y separados por coma seguida de espacio (,).
- Un valor de un vector asociativo viene precedido por un signo de interrogación (?), lo que permite que se construyan claves complejas sin ambigüedad.
- Los valores sencillos (o escalares) por lo general aparecen sin entrecomillar, pero pueden incluirse entre comillas dobles ("), o apostrofes (').
- Los comentarios vienen encabezados por la almohadilla (#) y continúan hasta el final de la línea.
- Es sensible a mayúsculas y minúsculas, todas las propiedades (palabras reservadas) de la especificación deben ir en minúsculas y terminar en dos puntos (:).
- Las propiedades requieren líneas independiente, su valor puede ir a continuación en la misma línea (precedido por un espacio) o en múltiples líneas (identadas)
- Las descripciones textuales pueden ser multilínea y admiten el dialecto CommonMark de Markdown para una representación de texto enriquecido. HTML es compatible en la medida en que lo proporciona CommonMark (Bloques HTML en la Especificación 0.27 de CommonMark).

Estructura básica

```
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in ...
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Estructura básica (cont)

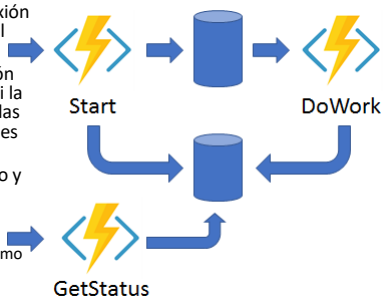
```
components:
  schemas:
    User:
      properties:
        id:
          type: integer
        name:
          type: string
      # Both properties are required
      required:
        - id
        - name
  securitySchemes:
      BasicAuth:
        type: http
        scheme: basic
security:
  - BasicAuth: []
```


WebHooks

- Los webhooks son eventos HTTP que desencadenan acciones. Su nombre se debe a que funcionan como «enganches» de los programas en Internet y casi siempre se utilizan para la comunicación entre sistemas. Son la manera más sencilla de obtener un aviso cuando algo ocurre en otro sistema y para el intercambio de datos entre aplicaciones web, permitiendo las llamada asíncronas en HTTP.
- Un webhook es una retro llamada HTTP, una solicitud HTTP GET/POST insertada en una página web, que interviene cuando ocurre algo (una notificación de evento a través de HTTP GET/POST).
- Los webhooks se utilizan para las notificaciones en tiempo real (con los datos del evento como parámetros o en cuerpo en JSON o XML) a una determinada dirección http:// o https://, que puede:
  - almacenar los datos del evento en JSON o XML
  - generar una respuesta que permita actualizarse al sistema donde se produce el evento
  - ejecutar un proceso en el sistema receptor del evento (Ej: enviar un correo electrónico)
- Los webhook están pensados para su utilización desde páginas web y sus diferentes consumidores: navegadores, correo electrónico, webapps, …
- Pueden considerarse una versión especializada y simplificada de los servicios REST (solo GET/POST).

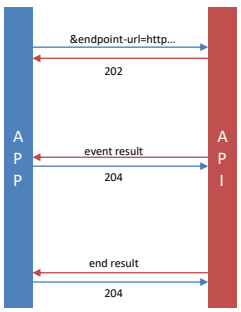
WebHooks

- El patrón Async HTTP APIs soluciona el problema de coordinar el estado de las operaciones de larga duración con los clientes externos. Una forma habitual de implementar este patrón es que un punto de conexión HTTP desencadene la acción de larga duración. A continuación, el cliente se redirige a un punto de conexión de estado al que sondea para saber el estado actual del proceso y cuando finaliza la operación.
- Un endpoint HTTP desencadena el proceso con la acción de larga duración y devuelve inmediatamente un 202 si la petición es correcta y, opcionalmente, como carga útil las URLs de los endpoints de estado y control. Las peticiones incorrectas recibirán el 4XX apropiado.
- El cliente sondea periódicamente el endpoint de estado y obtiene:
  - 202 con el estado actual del proceso como carga útil mientras el proceso este en marcha.
  - 200 con el estado final (correcto o fallido) del proceso como carga útil cuando haya finalizado la operación.
- El endpoint HTTP desencadenador puede suministrar endpoints adicionales para pausar, reanudar, cancelar/terminar, reiniciar o enviar eventos al proceso en marcha.



WebHooks

- El patrón Reverse API soluciona el problema del sondeo periódico, las API inversas invierten esta situación, para que sea la API invocada la que notifique automáticamente cuando se ha producido un determinado evento. El cliente debe crear su propia API para recibir las notificaciones.
- Al realizar la petición al endpoint HTTP desencadenador, se suministra un endpoint propio para que el proceso desencadenado pueda notificar cuándo ocurre algo de interés. Le da la vuelta a la comunicación, el cliente pasa a ser servidor y el servidor a cliente. El desencadenador devuelve inmediatamente un 202 si la petición es correcta o el 4XX apropiado se es incorrecta.
- Este esquema de funcionamiento tiene muchas ventajas en ambos extremos:
  - Ahorro de recursos y tiempo: con el sondeo se harán muchas llamadas "para nada", que no devolverán información relevante. Los dos extremos gastan recursos para hacer y responder a muchas llamadas que no tienen utilidad alguna (no nos llame, ya les llamaremos).
  - Eliminación de los retrasos: la aplicación usuaria recibirá una llamada en el momento exacto en el que se produce y no tendrá retrasos al próximo sondeo.
  - Velocidad de las llamadas: generalmente la llamada que se hace a un webhook es muy rápida porque solo se envía una pequeña información sobre el evento y se suele procesar asíncronamente. Muchas veces ni siquiera se espera por el resultado: se hace una llamada del tipo "fire and forget" (o sea, dispara y olvídate), pues se trata de notificar el evento y listo.



https://www.soapui.org

SOAPUI

Contenidos

- Instalación
- Estructura de proyectos
- Preparación de la prueba
- Pruebas funcionales: Pasos, Aserciones, Propiedades
- Ejecución
- Pruebas de carga
- Service Mocking
- Informes
- Data Driven Testing
- TestRunner Command-Line

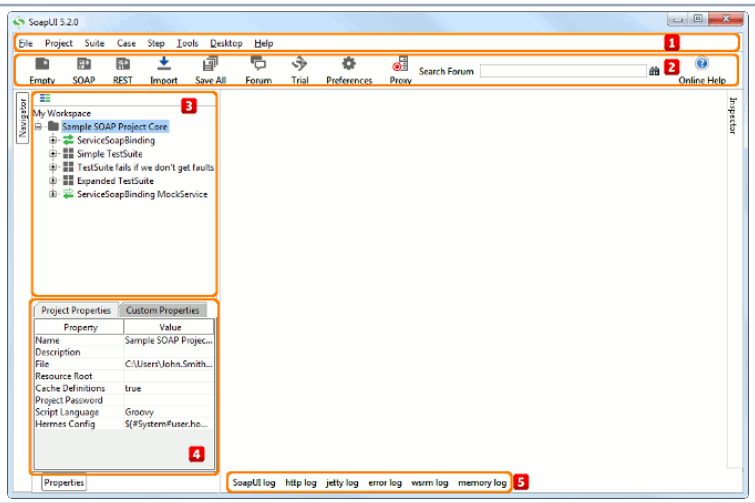
Introducción

- SoapUI es una herramienta para probar servicios web que pueden ser servicios web SOAP, servicios web RESTful u otros servicios basados en HTTP.
- SoapUI es una herramienta de código abierto y completamente gratuita con una versión comercial, ReadyAPI, que tiene una funcionalidad adicional para empresas con servicios web de misión crítica.
- SoapUI se considera el estándar de facto para las Pruebas de servicios API. Esto significa que hay mucho conocimiento en la red sobre la herramienta y blogs para obtener más información sobre el uso de SoapUI en la vida real.
- SoapUI permite realizar pruebas funcionales, pruebas de rendimiento, pruebas de interoperabilidad, pruebas de regresión y mucho más. Su objetivo es que la prueba sea bastante fácil de comenzar, por ejemplo, para crear una Prueba de carga, simplemente hay que hacer clic derecho en una prueba funcional y ejecutarla como una prueba de carga.
- SoapUI puede simular servicios web (mocking). Se puede grabar pruebas y usarlas más tarde.
- SoapUI puede crear apéndices de código desde el WSDL. Incluso puede crear especificaciones REST (WADL) a partir de la comunicación grabada.

Instalación

- SoapUI está basado en Java, por lo que se ejecuta en la mayoría de los sistemas operativos. Ha sido probado en varias versiones de Windows, así como en Mac y múltiples dialectos de Linux.
- SoapUI requiere una versión 1.8+ de JRE (Java Runtime Environment), se recomienda al menos 1 GB de memoria y aproximadamente 100 MB de espacio en disco.
- Para descargar e instalar accedemos a su página oficial:
  - https://www.soapui.org/downloads/soapui.html
- Si se está instalando con el instalador o las distribuciones independientes, el JRE está incluido y no es necesario en el sistema. De lo contrario, hay que asegúrese de que esté instalado y la variable de entorno JAVA_HOME esté configurada.

Interfaz de SoapUI



Log

- Las diferentes pestañas de log disponibles con las siguientes:
  - SoapUI log: Notificaciones generales y mensajes.
  - Http log: Muestra los datos enviados y recibidos por http. Deshabilitado durante las pruebas de stress.
  - Jetty log: Con las notificaciones de estado del mock-service.
  - Script log: Los scripts lanzan estos mensajes usando el objeto log disponible (está deshabilitado durante las pruebas de stress pero puede ser habilitado desde /File/Preferencess/UI settings).
  - Error log: Es un log con información sobre los errores ocurridos durante la ejecución. No solo errores de SoapUI, sino que pueden ser producidos por algún servicio o servidor que no esté disponible.
  - Memory Log: Muestra información del uso de la memoria.
- SoapUI utiliza log4j para crear los logs, es posible adaptar la configuración de log4j, renombrando el archivo log4j.xml, llamándolo “soapui-log4j.xml” y posteriormente moviéndolo al directorio bin de soapUI.

Espacio de trabajo

- En SoapUI, el trabajo se organiza en espacios de trabajo y proyectos. Un proyecto puede contener cualquier número de pruebas funcionales, pruebas de carga y simulaciones de servicio requeridas para los propósitos de prueba.
- SoapUI ofrece dos formatos de proyecto:
  - Proyectos independientes (predeterminado): los almacena como un único archivo XML que contiene todos los artefactos del proyecto, como interfaces, pruebas, servicios simulados, scripts, etc.
  - Proyectos compuestos (ReadyAPI): para equipos de prueba, esto permite que varias personas trabajen en el proyecto al mismo tiempo.

Proyectos

- Los proyectos SOAP se pueden crear desde un archivo WSDL o una llamada de servicio individual. Se puede usar estos proyectos para probar cada aspecto de los servicios SOAP, verificar que los servicios sean compatibles con los estándares de uso común como WS-Security, WS-Addressing y MTOM, crear pruebas funcionales, de carga, y mucho más.
- Los proyectos REST se pueden crear desde un archivo WADL o, directamente, URI y sus parámetros. Se puede usar estos proyectos para probar servicios RESTful, crear varias solicitudes y verificar la información que recibe, probar una multitud de métodos y operaciones, etc.
- Se pueden importar proyectos existentes locales, compuestos, comprimidos o remotos. Al importar, SoapUI lo comprueba para verificar que sea coherente y que tenga todas las dependencias externas necesarias disponibles. Este proceso se llama resolución .

Propiedades del proyecto

- La propiedad de raíz de recursos controla cómo SoapUI maneja las rutas para los recursos del proyecto.
  - Cadena Literal: Usa este camino absoluto.
  - $ {projectDir}: Resuelve archivos relativos a la carpeta del proyecto.
  - $ {workspaceDir}: Resuelve archivos relativos a la carpeta que contiene el archivo del espacio de trabajo
- Se pueden crear propiedades personalizadas.
- Se puede usar la propiedad del proyecto Contraseña del proyecto para cifrar todo el contenido del archivo del proyecto (el icono del proyecto contendrá un pequeño carácter E para indicar que se ha cifrado). Al abrir el proyecto SoapUI pedirá dicha contraseña. Para eliminar el cifrado, hay que borrar el valor de la contraseña del proyecto y guardar el proyecto. SoapUI no proporciona ningún medio para recuperar un archivo de proyecto cifrado si se ha perdido la contraseña.

Estructura del proyecto

- Interfaces
  - SOAP
  - REST
- Suite de pruebas (TestSuite)
  - Casos de prueba
    - Pasos de la prueba (pruebas funcionales)
    - Pruebas de carga
    - Pruebas de seguridad
- Simulaciones de servicio
  - SOAP MockServer
  - REST MockServer

Preparación de la prueba

- Importar definición y configurar endpoints:
  - Añadir WSDL (http://www.dneonline.com/calculator.asmx?wsdl)
  - Añadir WADL
  - Importar Swagger (https://petstore.swagger.io/v2/swagger.json)
  - Compatible con el descubrimiento REST
- Explorar el servicio:
  - Cada servicio basado en WSDL expone una serie de operaciones que tienen un formato de mensaje de solicitud y respuesta (ambos opcionales).
  - Cada servicio REST expone una serie de URLs y el método de solicitud determina la operación a realizar. Las básicas son: POST, GET, PUT y DELETE; aunque SoapUI también admite solicitudes HEAD, OPTIONS, TRACE, PATCH, PROPFIND, LOCK, UNLOCK, COPY y PURGE.
- Preparar y probar peticiones al servicio
  - Para invocar una operación, puede agregar cualquier número de objetos de solicitud a una operación en el árbol del navegador.

Explorar el servicio

- Si al crear el proyecto se activa "Create Request" muestrea las posibles peticiones extraídas de las definiciones.
- Todas las peticiones se pueden completar con:
  - Auth: permite establecer el mecanismo de autenticación y la identidad con la que se realizará la petición.
  - Encabezados HTTP: permite agregar los valores a los encabezado de la solicitud
  - Adjuntos: permite agregar ficheros que se enviaran adjuntos con la solicitud
- Las peticiones SOAP adicionalmente se pueden completar con:
  - WS-A: las propiedades utilizadas para agregar encabezados WS-A a Request / MockResponse de acuerdo con la especificación WS Addressing.
  - WS-RM: las propiedades utilizadas para configurar y usar una secuencia WS-RM para la solicitud de acuerdo con la especificación de WS Reliable Messaging.

Parámetros REST

- Tipos de parámetros:
  - CONSULTA (QUERY): aparecen en la URL después del signo de interrogación (?) después del nombre del recurso (nombre=valor).
  - ENCABEZAMIENTO (HEADER): se pasan en los encabezados de las solicitudes salientes (encabezado: valor).
  - PLANTILLA (TEMPLATE): estos parámetros aparecen en la ruta encerrados entre {}.
  - MATRIX: estos parámetros también van en la URL de la solicitud. Residen entre la ruta del recurso y los parámetros QUERY, y están separados de la ruta del recurso por un punto y coma (;).
  - PLAIN: estos parámetros están presentes en el editor de solicitudes, pero SoapUI no los incluye en las solicitudes.
- Nivel de parámetro:
  - El nivel de RECURSO significa que el parámetro que cree se agregará a todos los elementos de método y solicitud en el elemento de recurso.
  - El nivel METHOD significa que el parámetro se agregará a los elementos de la solicitud debajo del elemento del método solamente. No afectará al recurso ni a otros elementos del método.

Opciones de parámetros

- SoapUI utiliza las siguientes opciones para los parámetros para configurar la solapa Forms y seleccionar el valor de una lista desplegable (solo en ReadyAPI):
  - Required: Establece si los pasos de prueba deben especificar el valor del parámetro o si puede omitirlo
  - Type: Especifica el tipo de datos del parámetro.
  - Options: Una lista de posibles valores para el parámetro que aparecen en la lista desplegable.
  - Description: Un texto libre que describe el parámetro y proporcionar una pista rápida para el parámetro en los editores de pasos de prueba.
  - Disable Encoding: Si los valores de un parámetro van en una URL y contienen algunos símbolos especiales como espacios o barras diagonales, SoapUI de forma predeterminada reemplazará estos símbolos con sus códigos (como %20 o %2F). En determinados casos, es posible que se desee desactivar dicha codificación marcando la casilla.

Representaciones

- Establece los diferentes formatos MIME en el que se puede solicitar u obtener las peticiones REST:
  - Type: Tipo de representación, puede tomar los siguientes valores:
    - REQUEST: formato del cuerpo de la solicitud
    - RESPONSE: formato del cuerpo de las respuestas correctas (2xx)
    - FAULT: formato del cuerpo de las respuestas fallidas (4xx y 5xx)
  - Media-Type: Formato MIME
  - Status Codes: Código numérico del estado HTTP (no aplicable en el tipo REQUEST)

PRUEBAS FUNCIONALES

Pruebas funcionales

- En SoapUI, las pruebas funcionales se pueden usar para validar requisitos funcionales, tanto para invocar los Servicios Web propios (="pruebas unitarias") como para una secuencia de peticiones (= "pruebas de integración"). Además, es posible añadir lógica a las pruebas mediante scripts de Groovy y JavaScript, lo que permite, por ejemplo, interactuar con una base de datos o realizar un flujo de pruebas complejo.
- Las pruebas funcionales, en SoapUI, se pueden usar para una variedad de propósitos:
  - Pruebas Unitarias: valida que cada operación del Servicio Web funciona como se indica
  - Pruebas de compatibilidad: valida que el resultado devuelto por el Servicio Web es compatible con su definición
  - Prueba de procesos: valida que una secuencia de invocaciones de Servicios Web ejecuta un proceso de negocio requerido
  - Pruebas guiadas por datos: valida que cualquiera de los anteriores funciona con requerimientos de datos de entrada procedentes de fuentes externas (por ejemplo, una base de datos u otro servicio Web)

Pruebas funcionales

- SoapUI soporta pruebas funcionales de Servicios Web suministrando un caso de prueba con un número de pasos que pueden ser ejecutados en secuencia. En la actualidad, hay seis tipos de pasos que proporcionan muchas posibilidades de prueba. Los casos de prueba están organizados en un grupo de pruebas y en un mismo proyecto se pueden crear varios grupos de pruebas.
- SoapUI estructura las pruebas funcionales en tres niveles: TestSuites, TestCases y TestSteps.
- Un TestSuite es una colección de TestCases que se pueden usar para agrupar pruebas funcionales en unidades lógicas. Se puede crear cualquier número de TestSuites dentro de un proyecto de SoapUI para admitir escenarios de pruebas masivas.
- Un TestCase es una colección de TestSteps que se ensamblan para probar algunos aspectos específicos de sus servicios. Se puede agregar cualquier número de TestCases a el TestSuite que lo contenga e incluso modularizarlos para llamarse entre sí para escenarios de prueba complejos.
- TestSteps son los "bloques de construcción" de las pruebas funcionales en SoapUI. Se agregan a un TestCase y se utilizan para controlar el flujo de ejecución y validar la funcionalidad de los servicios que se probarán.

Tipos de Paso

- SOAP Request: Envía una petición SOAP y permite que la respuesta sea validada usando una variedad de aserciones
- REST Request: Ejecuta una Request Rest definida en el proyecto
- HTTP Request: Ejecuta una llamada http
- JDBC Request: Ejecuta una petición a una BBDD
- AMF Request: Ejecuta una petición AMF (es el formato de mensajería Adobe ActionScript utilizado por las aplicaciones Flash / Flex para interactuar con un servidor back-end).
- GraphQL Request: Ejecuta una consulta o una mutación en GraphQL.
- Properties: Se utiliza para definir propiedades globales que pueden ser leídas desde una fuente externa.
- Property Transfer: Utilizadas para transferir los valores de propiedad entre dos TestSteps
- Run TestCase Step: Ejecuta otro TestCase dentro de uno ya existente
- Conditional Goto: Permite cualquier número de saltos condicionales en la ruta de ejecución del TestCase.
- Delay Step: Pausa la ejecución de una TestCase durante un número especificado de milisegundos.
- Groovy Script: Ejecuta un script Groovy que puede hacer más o menos "cualquier cosa"
- Mock Response: Escucha un petición que se valida y devuelve una respuesta mock
- Manual TestStep: Se utiliza cuando se necesita interacción manual del tester dentro de la prueba
- Message Queuing Telemetry Transport: Receive MQTT Message, Publish using MQTT, Drop MQTT Connection

TestRequests

- Los TestRequests son una de las principales características cuando se trabaja con SoapUI. Extiende peticiones estándar con la posibilidad de añadir cualquier número de aserciones que se aplicarán a la respuesta recibida por la petición. Esto es, comprueba que la respuesta contenga lo que se espera que contenga.
- Los TestRequests son enviados manualmente a través de las acciones de envío de los editores o cuando se ejecuta el TestCase que contiene la petición.
- La respuesta de la petición es validada contra las aserciones de peticiones y el icono de la petición cambia para reflejar el resultado de la validación; verde significa que todas las validaciones fueron bien y rojo que algunas fallaron. Un icono con el fondo gris indica que la petición todavía no ha sido enviada para validar, un fondo blanco indica que los TestRequests carecen de aserciones.

Petición

- SOAP Request:
  - Vinculada a operación del interfaz contiene el fragmento SOAP de la petición en XML para su edición.
- REST Request:
  - Vinculada a un método del interfaz contiene los parámetros de solicitud HTTP para la asignación de sus valores.
- HTTP Request:
  - Permite crear una petición a una URL usando el método indicado y parámetros de solicitud HTTP para la asignación de sus valores.

Petición

- Todas las peticiones se pueden completar con:
  - Auth: permite establecer el mecanismo de autenticación y la identidad con la que se realizará la petición.
  - Encabezados HTTP: permite agregar los valores a los encabezado de la solicitud
  - Adjuntos: permite agregar ficheros que se enviaran adjuntos con la solicitud
- Las peticiones SOAP adicionalmente se pueden completar con:
  - WS-A: las propiedades utilizadas para agregar encabezados WS-A a Request / MockResponse de acuerdo con la especificación WS Addressing.
  - WS-RM: las propiedades utilizadas para configurar y usar una secuencia WS-RM para la solicitud de acuerdo con la especificación de WS Reliable Messaging.

Respuesta

- Una vez ejecutada manualmente la petición se mostrara la respuesta. Admite diferentes formatos: XML, JSON, HTML, Raw.
- Así mismo permite inspeccionar:
  - Encabezados HTTP: muestra los encabezado HTTP de la respuesta.
  - Adjuntos: muestra los adjuntos de la respuesta asociada.
  - Información SSL: muestra la respuesta detallada SSL para la respuesta actual.
  - WS-A: muestra las propiedades del encabezado WS-Addressing.
  - WSS: muestra las propiedades del encabezado WS-Security

Aserciones

- Las aserciones se utilizan para validar el mensaje recibido por un TestStep durante la ejecución, generalmente comparando partes del mensaje (o el mensaje completo) con algún valor esperado.
- Se puede agregar cualquier cantidad de aserciones a una muestra de TestStep, cada una validando algún aspecto o contenido diferente de la respuesta.
- Después de que se ejecuta una muestra TestStep, todas las aserciones se aplican a la respuesta recibida y si alguna de ellas falla, el TestStep se marca como fallido en la vista del TestCase y se muestra una entrada FALLO correspondiente en el Registro de ejecución de prueba.
- Las aserciones se pueden deshabilitar para que no se apliquen.
- Las aserciones se dividen en varias categorías para facilitar la gestión. Solo se habilitan las categorías que contienen aserciones aplicables para un tipo específico de muestra.

Categoría: contenido de propiedad

- Contains: busca la existencia de un token de cadena en el valor de la propiedad, admite expresiones regulares. Aplicable a cualquier propiedad.
- Not Contains: busca la inexistencia de un token de cadena en el valor de la propiedad, admite expresiones regulares. Aplicable a cualquier propiedad.
- XPath Match: utiliza una expresión XPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga XML.
- XQuery Match: utiliza una expresión XQuery para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga XML.

Categoría: contenido de propiedad

- JsonPath Count: usa una expresión JsonPath para contar las ocurrencias de un elemento. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath Existence Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Si los valores coinciden con las afirmaciones pasadas supera la prueba, de lo contrario falla. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath RegEx Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con una expresión regular especificada. Aplicable a cualquier propiedad que contenga JSON.

Categoría: Cumplimiento, estado y estándares

- HTTP Download all resource: descarga todos los recursos referidos como un documento HTML (imágenes, scripts, etc.) y valida que estén disponibles. Aplicable a cualquier propiedad que contenga HTML.
- Valid HTTP Status Codes: comprueba que se recibió un resultado HTTP con un código de estado en la lista de códigos definidos. Aplicable a cualquier TestStep que recibe mensajes HTTP.
- Invalid HTTP Status Codes: comprueba que se recibió un resultado HTTP con un código de estado que no figura en la lista de códigos definidos. Aplicable a cualquier TestStep que recibe mensajes HTTP
- SOAP Response: valida que la última respuesta recibida es una respuesta SOAP válida. Aplicable solo a los pasos de solicitud de prueba SOAP.
- SOAP Request: valida que la última solicitud recibida es una solicitud SOAP válida. Aplicable solo a MockResponse TestSteps.
- Not SOAP Fault: valida que el último mensaje recibido no es un fallo SOAP. Aplicable a SOAP TestSteps.

Categoría: Cumplimiento, estado y estándares

- SOAP Fault: valida que el último mensaje recibido es un fallo SOAP. Aplicable a la solicitud SOAP TestSteps
- Schema Compliance: valida que el último mensaje recibido sea compatible con la definición de esquema WSDL o WADL asociada. Aplicable a SOAP y REST TestSteps.
- WS-Addressing Request: valida que la última solicitud recibida contiene encabezados de direccionamiento WS válidos. Aplicable solo a MockResponse TestSteps.
- WS-Addressing Response: valida que la última respuesta recibida contiene encabezados de WS-Addressing válidos. Aplicable solo a los pasos de solicitud de prueba SOAP.
- WS-Security Status: valida que el último mensaje recibido contenía encabezados WS-Security válidos. Aplicable a SOAP TestSteps.

Otras categorías

- Script
  - Script Assertion: ejecuta una secuencia de comandos personalizada para realizar validaciones arbitrarias. Aplicable solo a TestSteps (es decir, no a las propiedades).
- SLA
  - Response SLA: valida que el último tiempo de respuesta recibido estuvo dentro del límite definido. Aplicable a Script TestSteps y TestSteps que envían solicitudes y reciben respuestas.
- JMS
  - JMS Status: valida que la solicitud JMS del TestStep objetivo se ejecutó correctamente. Aplicable a Request TestSteps con un punto final JMS.
  - JMS Timeout: valida que la instrucción JMS del TestStep de destino no tardó más de la duración especificada. Aplicable a Request TestSteps con un punto final JMS.
- JDBC
  - JDBC Status: valida que la instrucción JDBC del TestStep objetivo se ejecutó correctamente. Aplicable solo a JDBC TestSteps.
  - JDBC Timeout: valida que la instrucción JDBC del TestStep de destino no tardó más de la duración especificada. Aplicable solo a JDBC TestSteps.
- Security
  - Sensitive Information Exposure: comprueba que el último mensaje recibido no expone información confidencial sobre el sistema de destino. Aplicable a REST, SOAP y HTTP TestSteps.

Verificación SOAP

- SOAP Response
- Schema Compliance
- XPath Match
  - XPath Expression:

    declare namespace ns1='http://tempuri.org/';

    //ns1:AddResult
- XQuery Math
  - XQuery Expression:

    count(//Row)
- SOAP Fault

Verificación REST

- JsonPath Count
  - JsonPath Expression: $[*]
- JsonPath RegEx Match:
  - JsonPath Expression: $[0]
- JsonPath Existence Match
  - JsonPath Expression: $.id
- JsonPath RegEx Match:
  - JsonPath Expression: $.id
  - Regular Expression: \d
- Valid HTTP Codes:
  - Codes: 200, 201, 204
- Schema Compliance

Propiedades

- Las propiedades son un aspecto central de las pruebas más avanzadas con SoapUI.
- En lo que respecta a las propiedades de pruebas funcionales, se utilizan para parametrizar la ejecución y la funcionalidad de las pruebas, por ejemplo:
  - Las propiedades se pueden utilizar para mantener los puntos finales de los servicios, lo que facilita el cambio de los puntos finales reales utilizados durante la ejecución de la prueba.
  - Las propiedades se pueden usar para mantener las credenciales de autenticación, lo que facilita su administración en un lugar central o en un archivo externo.
  - Las propiedades se pueden usar para transferir y compartir identificadores de sesión durante la ejecución de la prueba, por lo que múltiples pasos de prueba o casos de prueba pueden compartir las mismas sesiones.
- Las propiedades se pueden definir en varios niveles en SoapUI:
  - En el nivel Proyecto, TestSuite y TestCase
  - En un Properties TestStep
  - En un DataGen TestStep
  - Como parte de una configuración de TestStep (DataSource TestStep y DataSink TestStep)
- Además, la mayoría de los otros TestSteps exponen propiedades para lectura y escritura, por ejemplo:
  - Script TestStep expone una propiedad "Result" que contiene el valor de cadena devuelto por el último script ejecutado.
  - Todos los pasos de Request exponen una propiedad con la última respuesta recibida.
- Las propiedades pueden leerse y escribirse fácilmente desde scripts y también transferirse entre TestSteps con Property-Transfer TestStep

Properties TestStep

- El Properties TestStep se utiliza para definir propiedades personalizadas que se utilizarán dentro de un TestCase.
- Sus principales ventajas sobre la definición de propiedades en el nivel TestCase son:
  - Se pueden organizar las propiedades en varios Properties TestStep (si se tienen muchas de ellas).
  - Se puede especificar los nombres de los archivos de origen y destino que se usarán para leer y escribir las propiedades contenidas cuando se ejecute TestStep. En formato .properties:

    Propiedad1=Valor1

    Propiedad2=Valor2

Property Transfer TestStep

- Los Property Transferes son TestStep que transfieren propiedades entre los contenedores de propiedades dentro del mismo alcance que el Property Transfer TestStep (es decir, que contenga su TestCase, su TestSuite, el proyecto y las propiedades del destino).
- El paso puede contener un número arbitrario de "transferencias" especificando una propiedad de origen y destino con expresiones opcionales de XPath/XQuery.
- Property Transferes utiliza el mismo motor de Saxon XPath/XQuery utilizado para las aserciones XPath y XQuery.
- Tras la ejecución de un TestCase, cada transferencia se realiza mediante la selección de las propiedades especificadas por el paso de origen, por la propiedad y expresión XPath opcional y copiando su valor a la propiedad especificada por el paso de destino pudiendo usar una expresión XPath opcional.
- Si las expresiones XPath están especificadas, SoapUI tratará de sustituir el nodo destino con el nodo de origen si son del mismo tipo. Si no (por ejemplo, cuando se asigna texto () a un @atributo), SoapUI hará todo lo posible para copiar el valor.
- El origen de la transferencia puede ser la respuesta de un paso anterior.
- El destino puede ser el contenido de un paso posterior, pero ser mas cómodo el uso de la expansión de propiedades.

Propiedades de Expansión

- SoapUI proporciona una sintaxis común para insertar dinámicamente ("expandir") valores de propiedad durante el procesamiento. La sintaxis es la siguiente:
  - ${[scope]propertyName[#xpath-expression]}
- donde alcance puede ser uno de los siguientes valores literales:
  - #Project#: hace referencia a una propiedad del Proyecto
  - #TestSuite#: hace referencia a una propiedad en el TestSuite que lo contiene
  - #TestCase#: hace referencia a una propiedad en el TestCase que lo contiene
  - #MockService#: hace referencia a una propiedad en el MockService que lo contiene
  - #Global#: hace referencia a una propiedad global en todos los proyectos. En File>Preferences>Global Properties.
  - #System#: hace referencia a una propiedad del sistema. En Help>System properties.
  - #Env#: hace referencia a una variable de entorno
  - [Nombre de TestStep]#: hace referencia a una propiedad TestStep
- Si no se especifica ningún alcance, la propiedad se resuelve de la siguiente manera:
  - Mira en el contexto actual (por ejemplo, TestRunContext) una propiedad con el nombre coincidente
  - Busca una propiedad global coincidente
  - Comprueba si hay una propiedad del sistema coincidente

Propiedades de Expansión

- Si la expansión de la propiedad incluye además una expresión XPath, se usará para seleccionar el valor correspondiente del valor de la propiedad referenciada (que debe contener XML):
  - ${Search Request#Response#//ns1:Item[1]/n1:Author[1]/text()}
- SoapUI 2.5 introdujo la posibilidad de escribir scripts directamente dentro de una PropertyExpansion; si se prefija el contenido con un '=' el contenido restante hasta la llave de cierre se evaluará como un script y se insertará su resultado:
  - ${=(int)(Math.random()*1000)}
- Dependiendo del contexto de la expansión, las variables relevantes estarán disponibles para acceder al modelo de objeto SoapUI.
  - $ {= request.name}
- Las siguientes variables están (casi) siempre disponibles en estos scripts:
  - log: un Logger log4j que registra en la ventana de registro
  - modelItem: el modelItem actual (por ejemplo, Request, MockResponse, etc.).
  - context: el contexto de ejecución actual (por ejemplo, un TestCase o MockService)

JDBC Request

- SoapUI 3.5 presento un nuevo TestStep para recuperar datos de una base de datos utilizando JDBC. El resultado está formateado como XML y se puede afirmar o procesar de la manera estándar (XPath , XQuery, …), pero también hay dos aserciones adicionales disponibles:
  - Afirmación de estado JDBC
  - La aserción JDBC Timeout verificará que el SQL se ejecutó dentro del tiempo de espera configurado.
- Para utilizar JDBC TestStep se deberá agregar el controlador JDBC a la carpeta soapui_home/bin/ext y reiniciar la aplicación.
- Para configurar la conexión:
  - Driver: com.mysql.jdbc.Driver
  - Connection String: jdbc:mysql://localhost:3306/sakila?user=root&password=root
- La consulta SQL, los parámetros pueden ser referenciados mediante la expansión de propiedades:
  - SELECT count(*) FROM `sakila`.`actor` WHERE `actor_id`=:id

https://www.soapui.org/docs/jdbc/reference/jdbc-drivers/

Run TestCase TestStep

- Si los escenarios de prueba se vuelven cada vez más complejos, es posible que se desee compartir una serie de TestSteps entre diferentes TestCases, tal vez para configurar algunos requisitos previos (inicio de sesión, etc.) o para realizar ciertos pasos en diferentes contextos.
- Run TestCase TestStep presenta un enfoque flexible para la modularización; ejecuta el TestCase objetivo configurado opcionalmente, pasando y recuperando propiedades antes y después de la ejecución, por ejemplo, se podría usar esto para llamar a una secuencia compleja de TestSteps para realizar y validar un procedimiento de inicio de sesión, pasando las credenciales requeridas y recibiendo el sessionid resultante.

Delay TestStep

- Los TestStep de retardo se pueden insertar en cualquier posición en un TestCase para pausar la ejecución de un TestCase durante un número determinado de milisegundos.
- Ejemplos de escenarios:
  - Repetir varias veces el TestCase o determinados TestStep después de una demora de tiempo.
  - Demorar los siguientes pasos para dar tiempo a que un proceso ajeno se complete.

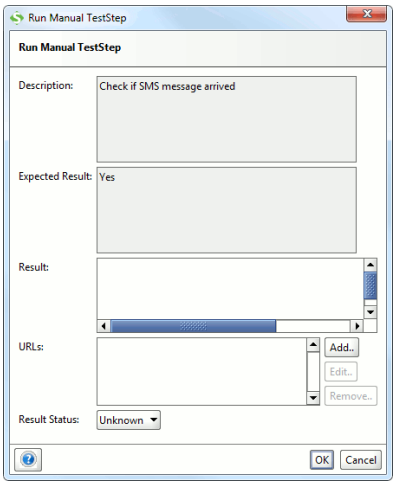
Conditional Goto TestStep

- Los pasos de condicional Goto evalúan un número arbitrario de condiciones XPath del mensaje de respuesta de la petición anterior y transfieren la ejecución del TestCase al TestStep asociado con la primera condición que se evalúe a verdadero.
- Esto permite la ejecución condicional de rutas de TestCase, donde el resultado de algunas peticiones controla cómo moverse por el TestCase.
- Si no hay condiciones que coincidan con la respuesta actual, la ejecución del TestCase continúa después del paso Goto de forma habitual.
- Ejemplos de escenarios:
  - Ramificaciones que dependen de los resultados devueltos por una petición
  - Reiniciar después de un largo retraso en las pruebas de vigilancia
  - En varias ocasiones esperar y chequear el valor de estado antes de continuar (por ejemplo en un proceso por lotes)
- Las condiciones usan el mismo motor que Saxon XPath utilizado para las aserciones XPath (una condición debe devolver un valor booleano para ser válida).
- Para un comportamiento más complejo se usa el paso Groovy Script.

Manual TestStep

- Manual TestStep se utiliza cuando se necesita interacción manual del tester dentro de la prueba. Algunos ejemplos:
  - Iniciar dispositivos como teléfonos movil
  - Realizar una acción en el sitio web o en el servidor
  - Verificar manualmente la base de datos / registros
- Para definirlo se puede especificar una descripción y el resultado esperado. Cuando la ejecución del caso de prueba alcanza un paso de prueba Manual, se muestra una ventana emergente con la descripción y el resultado esperado y permite agregar los detalles de los resultados reales y especificar el botón de opción Pasar-Fallar-N/A para establecer el estado del paso de prueba. Al hacer clic en continuar, SoapUI va al siguiente paso.
- Estos pasos de prueba manuales, se omiten cuando se inicia a través de testrunner ya que no hay forma de interactuar con ventanas emergentes.

Manual Test



Script Step

- SoapUI ofrece amplias opciones para la creación de secuencias de comandos, utilizando Groovy o Javascript (desde SoapUI 3.0) como su lenguaje de secuencias de comandos, que se utiliza para establecer el nivel del proyecto en la pestaña de detalles del proyecto en la parte inferior izquierda.
- Todos los scripts tienen acceso a una serie de variables específicas de la situación, que siempre incluyen un objeto de registro para iniciar sesión en Groovy Log y un objeto de contexto para realizar PropertyExpansions específicas de contexto o manejo de propiedades, si corresponde.
- Una variable específica de contexto siempre está disponible para acceder directamente al modelo de objetos SoapUI.

Script Step

- Los scripts se pueden usar en los siguientes lugares en SoapUI:
  - Como parte de un TestCase con Script TestStep, que permite que las pruebas realicen prácticamente cualquier funcionalidad deseada
  - Antes y después de ejecutar un TestCase o TestSuite para inicializar y limpiar antes o después de ejecutar sus pruebas.
  - Al iniciar/detener un MockService para inicializar o limpiar el estado de MockService
  - Para proporcionar despacho dinámico de MockOperation o crear contenido dinámico de MockResponse
  - Al abrir / cerrar un proyecto, para inicializar o limpiar configuraciones relacionadas con el proyecto
  - Como un DataSource o DataSink dinámico con los pasos de prueba de DataSource / DataSink correspondientes
  - Para crear aserciones arbitrarias con la aserción de script
  - Para extender SoapUI y agregar funcionalidad arbitraria al núcleo SoapUI.

SOAP Mock Response

- El paso SOAP Mock Response escucha una solicitud SOAP y devuelve una respuesta preconfigurada antes de continuar. La solicitud entrante se puede validar con las mismas aserciones de las solicitudes SOAP, específicamente con SOAP Request y Schema Compliance se puede validar la petición entrante.
- Cuando la ejecución de un caso de prueba llega al paso de prueba SOAP Mock Response, la ejecución de la prueba se detiene y espera una solicitud al recurso especificado en el puerto configurado con la ruta especificada. Una vez que se recibe una solicitud, SOAP Mock Response devuelve el resultado y continua con siguiente paso en el caso de prueba.
- Dichos servicios mock temporales pueden capturar solicitudes enviadas por webhooks. De esta manera, se puede configurar rápidamente un servicio temporal para capturar una sola solicitud, o hacer que esto sea parte de la prueba sin configurar una aplicación o una API virtual separada para manejar la solicitud.

PRUEBAS DE CARGA

Pruebas de carga

- Las pruebas de rendimiento son una de las tareas más comunes en las pruebas de servicios web. Lo que se llama una prueba de carga en SoapUI es en realidad una prueba de rendimiento.
- Las pruebas de rendimiento crean o simulan artificialmente la carga y miden cómo la maneja el entorno.
- Esto significa que no necesariamente tiene que ser el rendimiento de un sistema bajo una carga alta, sino también el rendimiento bajo la carga base o la carga esperada.
- En SoapUI, se crean las pruebas de carga sobre la base de pruebas funcionales existentes. Esto ayuda a crear rápida y fácilmente pruebas de rendimiento para el servicio web. Luego se puede validar el rendimiento del servicio web utilizando diferentes estrategias de carga, verificar que su funcionalidad no se rompa bajo carga, ejecutar varias pruebas de carga simultáneamente para ver cómo se afectan entre sí y mucho más.

Pruebas de carga

- Pruebas de línea de base
  - Técnicamente hablando, esto se puede definir como pruebas de rendimiento puro. Las pruebas de línea de base examinan cómo funciona un sistema bajo la carga esperada o normal y crea una línea de base con la que se pueden comparar otros tipos de pruebas.
  - Objetivo: encontrar métricas para el rendimiento del sistema bajo carga normal.
- Prueba de carga
  - La prueba de carga incluye aumentar la carga y ver cómo se comporta el sistema bajo una carga mayor. Durante las pruebas de carga, puede monitorear los tiempos de respuesta, el rendimiento, las condiciones del servidor y más. Sin embargo, el objetivo de las pruebas de carga no es romper el entorno de destino.
  - Objetivo: encontrar métricas para el rendimiento del sistema bajo alta carga.
- Prueba de esfuerzo o estrés
  - La prueba de esfuerzo consiste en aumentar la carga hasta encontrar el volumen de carga donde el sistema realmente se rompe o está cerca de romperse.
  - Objetivo: encontrar el punto de ruptura del sistema.

Pruebas de carga

- Pruebas de remojo
  - Para encontrar inestabilidades del sistema que ocurren con el tiempo, debe ejecutar pruebas durante un período prolongado. Para eso están las pruebas de remojo; ejecute pruebas de carga o incluso pruebas de línea de base durante un largo período de tiempo y vea cómo el entorno de destino maneja los recursos del sistema y si funciona correctamente. El defecto más común encontrado en las pruebas de remojo son las pérdidas de memoria. El escenario más común para las pruebas de remojo es activar una serie de pruebas cuando sale de su oficina un viernes y lo deja correr durante el fin de semana.
  - Objetivo: Asegurarse de que no surja ningún comportamiento no deseado durante un largo período de tiempo.
- Pruebas de escalabilidad
  - El propósito de las pruebas de escalabilidad es verificar si su sistema se adapta adecuadamente a la carga cambiante. Por ejemplo, un mayor número de solicitudes entrantes debería causar un aumento proporcional en el tiempo de respuesta. El aumento no proporcional indica problemas de escalabilidad.
  - Objetivo: buscar métricas y verificar si el rendimiento del sistema cambia adecuadamente a la carga.

Factores que afectan al rendimiento

- Si observamos características únicas del rendimiento del servicio web, se destacan dos aspectos: una gran cantidad de procesamiento XML / JSON en el lado del servidor (análisis y serialización XML / JSON) y procesamiento de cargas (cuerpos de solicitud y respuesta). Las razones por las cuales esto falla pueden ser múltiples; puede estar en la plataforma en forma de, por ejemplo, debilidades en el servidor de aplicaciones, y en la implementación en forma de WSDL innecesariamente complejos. También podría ser que el código está haciendo una solicitud a una base de datos que responde lentamente.
- Un factor más que afecta el rendimiento con frecuencia es la seguridad. Aquellos que realizan pruebas de rendimiento web saben que los sitios HTTPS tienen un rendimiento considerablemente menor que sus análogos HTTP, y en las pruebas de servicios web, podemos agregar una capa de WS-Security a la capa de seguridad HTTP, disminuyendo aún más el rendimiento.
- La complejidad de analizar las cargas útiles de XML / JSON significa que tenemos que poner un enfoque adicional en las pruebas de escalabilidad, mientras que el problema de la seguridad significa que tendremos que enfocarnos un poco más en hacer pruebas de solicitudes que son seguras. Si todo el servicio web es seguro, esto significa que la prueba de carga es más importante, especialmente si se utiliza WS-Security y el manejo de tokens.
- También significa que tenemos que examinar la especificación API de cerca. Si las solicitudes y respuestas son complejas o grandes, o si incluyen archivos adjuntos grandes, debemos centrarnos en enfatizar esa complejidad y ver cómo se comporta bajo carga.

LoadTest

- En SoapUI, los LoadTests se basan en los TestCases funcionales existentes. Un LoadTest ejecuta el TestCase repetidamente durante el tiempo especificado con un número deseado de subprocesos (o usuarios virtuales). Los LoadTests se muestran en el navegador como elementos secundarios del TestCase que usan. Se puede crear cualquier número de LoadTests para un TestCase.
- Las diferentes estrategias de carga disponibles en SoapUI permiten simular varios tipos de carga a lo largo del tiempo, lo que permite probar fácilmente el rendimiento de sus servicios de destino en una serie de condiciones. Dado que SoapUI también le permite ejecutar múltiples LoadTests simultáneamente, se puede usar una combinación de LoadTests para afirmar aún más el comportamiento de los servicios.
- Si queremos simular varios clientes consumiendo el servicio, vamos a la opción threads, por defecto aparecen 5 pero se puedan variar ese número a discreción, aunque mientras más hilos más consumo del CPU. Si queremos especificar una demora entre una petición y otra vamos a la opción delay (1000 milisegundos por defecto) y la cantidad aleatoria de la demora que se desea aleatorizar (es decir, establecerlo en 0.5 dará como resultado retrasos entre 0,5 y 1 segundo)
- Si queremos determinar el tiempo de duración del servicio vamos a la opción Limit y aquí se muestra que solo durará 60 segundos pero pueden cambiarla para especificar la cantidad de peticiones por hilo por ejemplo.

Ejecución de LoadTest

- SoapUI permite ejecutar el LoadTest con tantos subprocesos (o usuarios virtuales) como el hardware pueda administrar, dependiendo principalmente de la memoria, la CPU, el tiempo de respuesta del servicio objetivo y otros factores. Establezca el valor deseado e inicie LoadTest con el botón Ejecutar en la barra de herramientas del editor LoadTest.
- El TestCase subyacente se clona internamente para cada subproceso configurado y se inicia en su propio contexto; scripts, transferencias de propiedades, etc. El TestCase accederá a una "copia" única de TestCase y TestSteps que evita problemas de subprocesos en el nivel TestStep y TestCase (pero no más arriba en la jerarquía).
- A medida que se ejecuta LoadTest, la tabla de estadísticas de LoadTest se actualiza continuamente con los datos recopilados cada vez que finaliza un TestCase ejecutado, lo que le permite controlar de forma interactiva el rendimiento de sus servicios de destino mientras se ejecuta LoadTest. Si el TestCase contiene un bucle o TestSteps de ejecución prolongada, las estadísticas pueden tardar un tiempo en actualizarse (ya que se recopilan cuando finaliza TestCase), en este caso, seleccione la opción "Estadísticas TestStep" en el cuadro de diálogo Opciones de LoadTest para actualizarse estadísticas en el nivel TestStep en lugar del nivel TestCase (esto requiere un poco más de procesamiento interno, por eso está desactivado de forma predeterminada).

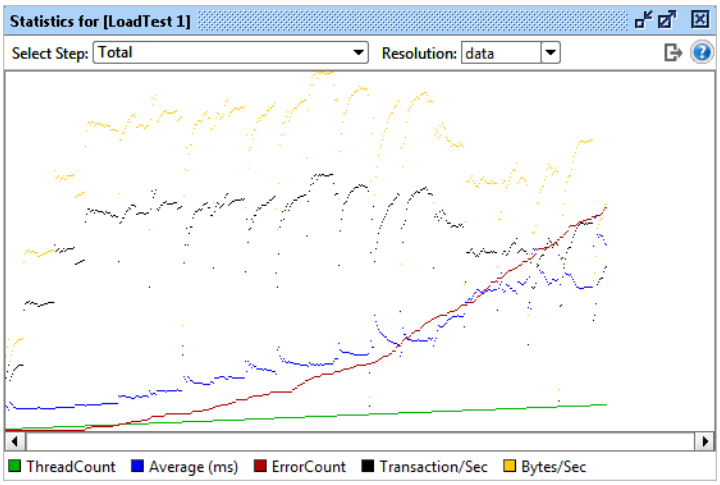
Estadísticas

- Tiempos mínimos (min), máximos (max), medios (avg) y de la última petición (last), todos en milisegundos.
- Número de ejecuciones (cnt), total de bytes (bytes), número errores (err) y el ratio de errores (rat).
- TPS (transacciones por segundo) y BPS (bytes por segundo) se pueden calcular de dos maneras diferentes (Opciones de LoadTest);
  - Según el tiempo real transcurrido (predeterminado):
    - TPS: cnt/segundos pasados, es decir, un TestCase que se ejecutó durante 10 segundos y manejó 100 solicitudes obtendrá un TPS de 10
    - BPS: bytes/tiempo transcurrido, es decir, un TestCase que se ejecutó durante 10 segundos y manejó 100000 bytes obtendrá un BPS de 10000.
  - Basado en el tiempo promedio de ejecución:
    - TPS: (1000/avg) * número de hilos, por ejemplo avg = 100 ms con diez hilos dará un TPS de 100
    - BPS: (bytes/cnt) * TPS, es decir, el número promedio de bytes por solicitud * TPS.
- Se pueden exportar a ficheros .csv

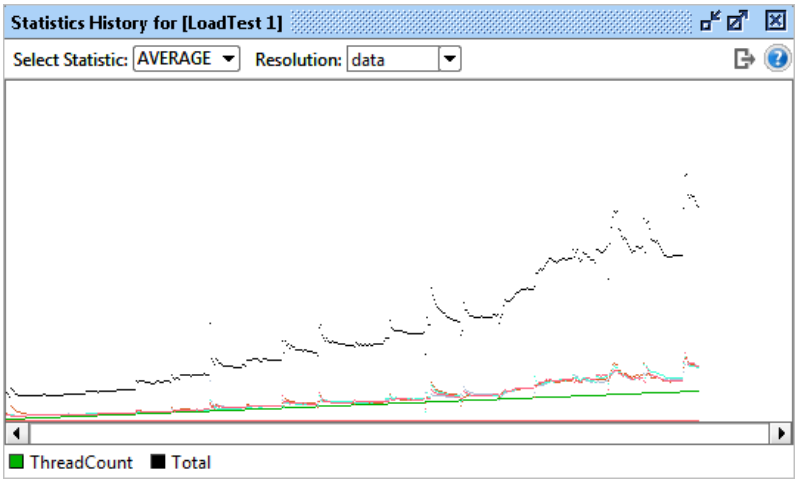
Gráficos estadísticos

- Hay dos tipos de gráficos disponibles en la barra de herramientas de LoadTest durante la ejecución; estadísticas e historial de estadísticas.
- El objetivo principal de estos es visualizar estadísticas seleccionadas a lo largo del tiempo para poder detectar cambios repentinos e inesperados. La visualización de estadísticas es relativa (no absoluta) y, por lo tanto, los gráficos no son muy útiles para analizar datos exactos.
- Ambos gráficos tienen una configuración de Resolución que controla con qué frecuencia se actualiza el gráfico; establecer esto en "datos" actualizará el gráfico con el mismo intervalo que la Tabla de estadísticas (que son los datos subyacentes para el gráfico, de ahí el nombre). Alternativamente, si desea una de las resoluciones fijas, seleccione estas.
- El gráfico de estadísticas muestra todas las estadísticas relevantes para un paso seleccionado o el caso de prueba completo a lo largo del tiempo, lo que le permite ver cómo cambian los valores cuando, por ejemplo, aumenta el número de subprocesos.
- El gráfico de historial de estadísticas muestra un valor estadístico seleccionado para todos los pasos que le permite compararlos y ver si la distribución de cualquier valor entre los pasos de prueba cambia con el tiempo.
- Sus datos se pueden exportar a ficheros .csv

Gráficos estadísticos



Gráficos estadísticos



Validando el rendimiento

- Al igual que puede agregar aserciones funcionales a ciertos TestSteps en un TestCase (para afirmar sus resultados), se pueden agregar aserciones a un LoadTest para validar tanto el rendimiento como la funcionalidad durante la ejecución de LoadTest. Las aserciones configuradas se aplican continuamente a los resultados de LoadTest y puede fallar el LoadTest al igual que su contraparte funcional.
- Las aserciones disponibles son:
  - Step Average: valida que el valor promedio de un TestStep o todo el TestCase no excede el límite especificado.
  - Step TPS: valida el valor de TPS (transacción por segundo) para el TestStep o TestCase correspondiente.
  - Step Maximum: valida el valor tiempo máximo para el TestStep o TestCase correspondiente.
  - Step Status: comprueba que el estado de ejecución subyacente del TestStep o TestCase sea exitoso; de lo contrario, la aserción falla.
  - Max Errors: compruebe que el número de fallos para el TestStep o TestCase correspondiente no exceda el valor configurado.

SERVICE MOCKING

SOAP Service Mocking

- La funcionalidad de SOAP Service Mocking en SoapUI permite crear una simulación compatible con los estándares de un servicio basado en WSDL solo a partir de su contrato WSDL, que en SoapUI se denomina "MockService". Se puede ejecutar directamente desde dentro de SoapUI, con la utilidad de línea de comandos o con un contenedor de servlet estándar.
- Los escenarios de uso son muchos:
  - Prototipos rápidos de servicios web: generar una implementación simulada completa estática de un WSDL en segundos y agregar funcionalidad dinámica usando Groovy. Esto permite implementar y probar clientes mucho más rápido que si se hubiera tenido que esperar a que se desarrollara la solución real.
  - Prueba o desarrollo del cliente: crear implementaciones simuladas de las operaciones deseadas y configurar una serie de respuestas alternativas (incluidos scripts, archivos adjuntos y encabezados http personalizados). Los clientes pueden desarrollarse contra el MockService y probarse sin acceso a los servicios en vivo. Las respuestas pueden ciclarse, aleatorizarse o seleccionarse con la expresión XPath de la solicitud entrante
  - Desarrollo dirigido por pruebas: crear pruebas funcionales y de carga en SoapUI contra un MockService antes o durante la implementacíon los servicios reales
- Un MockService cumple con los estándares aceptados WSDL, SOAP o HTTP y un cliente debe poder usarlo como si fuera un servicio real.

SOAP Service Mocking

- Un MockService puede simular cualquier número de contratos WSDL y la funcionalidad de scripting incorporada le permite simular básicamente cualquier tipo de comportamiento deseado; respuestas fijas, errores aleatorios, resultados dinámicos, etc. Incluso es posible implementar un servicio completo en SoapUI y desplegarlo en un contenedor de servlet estándar utilizando la funcionalidad DeployAsWar (aunque no se recomienda para uso en producción).
- MockServices proporciona la simulación del servicio al exponer un número arbitrario de MockOperations que, a su vez, pueden contener un MockRequest Dispatcher y cualquier número de mensajes MockResponse.
- Cuando el MockService recibe una solicitud SOAP y se envía a una MockOperation específica y se selecciona la MockResponse correspondiente en función del MockRequest Dispatcher configurado.
- Cada MockOperation se corresponde a una Operación WSDL de un Servicio WSDL del proyecto de prueba, y los mensajes MockResponse contenidos se crean a partir del esquema asociado con la operación. Esto no significa que un MockResponse deba cumplir con el esquema asociado, puede devolver cualquier mensaje de texto o XML arbitrario y, por ejemplo, se puede configurar para que devuelva un mensaje de respuesta para una operación completamente diferente, lo que permite probar en los clientes la 'capacidad para manejar mensajes de respuesta no válidos e inesperados.

SOAP Service Mocking

- El MockResponse es el mensaje que MockService debe devolver al cliente que realiza la llamada. MockResponse puede contener contenido personalizado, encabezados y archivos adjuntos, lo que le permite simular cualquier tipo de respuesta HTTP válida (o no válida), y las posibilidades de secuencias de comandos adicionales permiten agregar fácilmente contenido dinámico a la respuesta saliente.
- Los MockResponse se pueden generar desde los servicios reales.
- Una vez que MockService ha recibido una solicitud y se ha enviado a una MockOperation, SoapUI debe seleccionar el método de respuesta correcto y devolverlo al cliente. Hay varios métodos de envío diferentes:
  - SECUENCIA: este es el método de envío más simple; las respuestas se seleccionan y devuelven en el orden en que se agregaron al MockOperation.
  - ALEATORIO: casi tan simple, este despachador selecciona qué respuesta usar al azar, con el tiempo todas las respuestas se habrán devuelto la misma cantidad de veces.
  - QUERY-MATCH: es bastante versátil, puede devolver diferentes respuestas en función del contenido de la solicitud.
  - XPATH: similar a QUERY_MATCH pero no tan potente, solo permite una condición.
  - SCRIPT: la opción más versátil y más difícil de dominar, se crea un script que debería devolver el nombre de MockResponse para usar.

SOAP Service Mocking

- Al hacer clic en el botón Ejecutar, MockService se inicia inmediatamente dentro de SoapUI, se puede consultar jetty log.
- El MockService ahora está listo para manejar las solicitudes SOAP entrantes en la ruta y el puerto configurados. Estas se enviarán a la correspondiente MockOperation en función de su contenido, si no hay disponible ninguna MockOperation coincidente, se devolverá un fallo SOAP estándar.
- El registro de mensajes en la parte inferior muestra todos los mensajes que ha recibido el MockService desde que se inició por última vez y permite consultar, a través del visor de mensajes, la petición recibida así como la respuesta emitida.
- Además, también expone el WSDL de MockServic en la URL estándar de WSDL; por ejemplo, introduciendo http://localhost:8088/mockSampleServiceBinding?WSDL en un navegador se mostrará el WSDL.

REST Service Mocking

- De forma similar a los SOAP Service Mocking, la función de simulación del servicio REST permite simular un servicio REST creando un servicio simulado. Luego se puede ejecutar directamente desde SoapUI o usar la aplicación de línea de comandos mockservicerunner.bat
- Un servicio simulado simula un servicio en vivo al exponer un cierto número de acciones simuladas (método y URL). Las acciones simuladas (Mock Action), a su vez, contienen una serie de respuestas simuladas.
- Las respuestas simuladas permiten establecer el HTTP Status Code, los valores de las cabeceras, el contenido (response body) y su media-type (content-type).
- El MockRequest Dispatcher esta limitado a SECUENCIA y SCRIPT.
- Los REST Service Mocking se ejecutan y supervisan de la misma forma que los SOAP Service Mocking.

OTROS TIPOS DE PRUEBAS

Seguridad

- Las funciones de prueba de seguridad introducidas en SoapUI 4.0 facilitan enormemente la validación de la seguridad funcional de los servicios de destino, lo que permite evaluar la vulnerabilidad del sistema para los ataques de seguridad comunes. Esto es especialmente crítico si el sistema está disponible públicamente, pero incluso si ese no es el caso, garantizar un entorno completamente seguro es igualmente importante.
- Los siguientes análisis de seguridad están disponibles actualmente:
  - Inyección SQL: intenta aprovechar la mala codificación de integración de bases de datos
  - Inyección XPath: intenta explotar el procesamiento XML incorrecto dentro de su servicio de destino
  - Escaneo de límites: intenta aprovechar el mal manejo de valores que están fuera de los rangos definidos
  - Tipos no válidos: intenta aprovechar el manejo de datos de entrada no válidos
  - XML con formato incorrecto: intenta aprovechar el mal manejo de XML no válido en su servidor o en su servicio
  - XML Bomb: intenta aprovechar el mal manejo de solicitudes XML maliciosas (tenga cuidado)
  - Adjunto malicioso: intenta aprovechar el mal manejo de los archivos adjuntos
  - Cross Site Scripting: intenta encontrar vulnerabilidades de cross-site scripting
  - Secuencia de comandos personalizada: le permite utilizar una secuencia de comandos para generar valores de fuzzing de parámetros personalizados

Data Driven Testing

- Las pruebas basadas en datos son aquellas que almacenan los datos de la prueba (entrada, salida esperada, etc.) en algún almacenamiento externo (base de datos, hoja de cálculo, archivos xml, etc.) y luego usa esos datos de forma iterativa en las pruebas cuando se ejecutan.
- Las pruebas basadas en datos solo están disponibles en ReadyAPI y crearlas es realmente fácil.
- El DataSource TestStep permite la lectura de los datos de prueba en propiedades SoapUI estándar a partir de una serie de fuentes externas (archivos de Excel, propiedades XML, fuentes JDBC, archivos/directorios, etc.), estas propiedades se pueden utilizar en los siguientes TestSTEPS (solicitudes, aserciones, consultas XPath, scripts, etc.), ya sea a través de Transferencias de propiedad o Expansiones de propiedad. Finalmente un DataSource Loop TestStep permite recorrer la fuente de datos para aplicar los TestSteps anteriores para los datos de cada fila/registro del DataSource.
- Esto se reduce a la siguiente configuración:
  1. DataSource: lee datos de prueba en propiedades de una fuente externa
  2. TestSteps (cualquier número): prueba la funcionalidad del servicio utilizando las propiedades DataSource disponibles en (1)
  3. DataSource Loop: realiza un bucle de regreso a (2) para cada fila en (1)

EJECUCIÓN E INFORMES

Ejecución

- Ejecución de prueba
  - Todas las vistas anteriores tienen una barra de herramientas en la parte superior con botones que ejecutan los elementos de prueba contenidos.
- Salida de la Prueba
  - Todas las vistas anteriores también contienen un registro de ejecución en la parte inferior que muestra información continua sobre los pasos de prueba ejecutados y su estado.
- Informes
  - ReadyAPI también agrega un botón "Crear informe" a la barra de herramientas superior, lo que le permite exportar los resultados de la ejecución actual a un documento, por ejemplo, un PDF.

TestRunner Command-Line

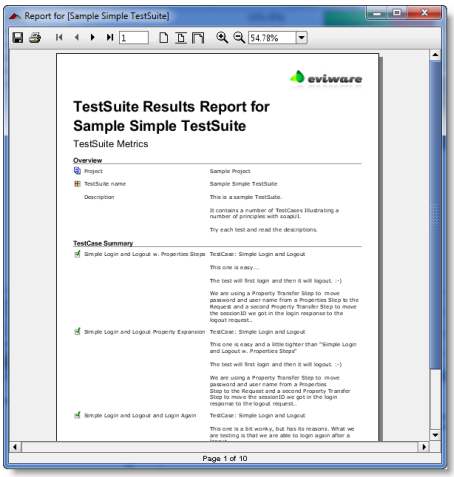
- Una vez creado, es posible que se desee ejecutar los Tests desde la línea de comandos, tal vez como parte de una compilación de integración continua o para monitorear el rendimiento diario de sus servicios.
- SoapUI proporciona utilidades de línea de comandos y complementos para que maven haga esto.
- Las utilidades están disponible en la carpeta SoapUI\bin (.bat o .sh):
  - testrunner.bat
  - loadtestrunner.bat
  - securitytestrunner.bat
  - mockservicerunner.bat
- Se necesitan muchos argumentos relacionados con informes, propiedades, etc., que pueden hacer que la creación de la lista de argumentos inicial sea bastante tediosa. Afortunadamente, la interfaz de usuario incluye un asistente para iniciar LoadTestRunner, que ayudará a evaluar la salida, así como a crear una cadena de invocación de línea de comandos.

https://www.soapui.org/test-automation/running-from-command-line/functional-tests.html

Informes

- ReadyAPI proporciona tres tipos de informes que se pueden generar desde el interior de la interfaz de usuario a nivel de Proyecto, TestSuite, TestCase y LoadTest:
  - Informes imprimibles: se pueden imprimir o guardar como PDF, HTML, RTF, Excel, etc. y son totalmente personalizables tanto a nivel global como de proyecto, lo que permite crear y personalizar cualquier tipo de informe que se pueda necesitar.
  - Exportación de datos: permite la exportación de datos de informes subyacentes en formato XML y CSV. Esto es útil si se desea importar datos de informes en otras herramientas para informes personalizados o integraciones.
  - Informes HTML: brinda una descripción general simplificada de los resultados de las pruebas funcionales en formato HTML (no disponible en el nivel de LoadTest).
- Se puede incluir fácilmente cualquier dato deseado en los informes generados, ya que puede exportar datos personalizados tanto en informes imprimibles como de exportación de datos a través del SubReport DataSink.

Informes



ANEXOS

Lenguaje eXtensible de Marcas

XML

INTRODUCCIÓN

Introducción

- Originalmente el XML (Lenguaje de Marcas eXtensible) surgió para solucionar un gran contrasentido.
- Con la Web, una gran parte del conocimiento humano se encuentra almacenada en sistemas informáticos, que requiere de equipos informáticos para su consulta, pero que no puede ser utilizada mediante procesos automáticos, dado que el formato utilizado es para representar gráficamente la información y no permite identificar los datos contenidos en dicha información.
- El XML fue desarrollado el Consorcio World Wide Web (W3C), en 1996, aprobándose la especificación 1.0 del XML en febrero de 1998.
- La aparición del XML supuso el primer gran paso hacia la "Web Semántica" que separa los contenidos de su representación visual.
- La flexibilidad inherente del XML le llevó rápidamente a transcender los objetivos iniciales quedando como el formato más válido para utilizar con la información que debe ser transmitida o almacenada de forma independiente a la plataforma, idioma o aplicación que la maneje.

Objetivos

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínima, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácilmente creables.
- La concisión en las marcas XML es de mínima importancia.

Características

- XML es un método para introducir datos estructurados en un fichero de texto
- XML se parece al HTML pero no es HTML
- XML es texto, pero no para ser leído
- XML es una familia de tecnologías
- XML es prolijo, pero eso no es un problema
- XML es nuevo, pero no tan nuevo
- XML es gratis, independiente de la plataforma y ampliamente distribuida

Documentos bien formados

- La sintaxis del XML es muy sencilla pero muy tajante.
- Se dice que un documento XML está bien formado si es sintácticamente correcto, respeta estrictamente la sintaxis.
- En XML un documento puede tener un formato libre o formal.
- Si el formato es libre no tendremos ninguna limitación, siempre que respetemos la sintaxis.
- Si queremos utilizar un formato estricto que limite y establezca los elementos disponibles, debemos declarar (o utilizar si ya existe) dicho formato y asociarlo al documento.
- Dicho formato se denomina "tipo de documento" o "vocabulario XML".

Documentos válidos

- El formato se puede definir mediante DTD (Definición de Tipo de Documento) o XSD (Esquemas de Documentos XML).
- Los DTD vienen recogidos en la especificación del XML y son los que veremos en el curso, aunque los XSD, al aparecer más tarde, son más flexibles y precisos.
- Si el documento tiene asociado tipo de documento, un documento bien formado es válido si cumple las restricciones especificadas en su tipo de documento.
- Los documentos bien formados sin formato asociado siempre son válidos.
- El estándar define que un documento XML que NO esté bien formado o NO sea válido NO SE PUEDE PROCESAR y debe ser rechazado obligatoriamente.
- Al software que manipula el documento XML se le denomina procesador XML y puede estar dentro de otra aplicación. Al software que verifica el documento XML se le denomina Parse y puede estar dentro del procesador XML.

Algunas Tecnologías XML

- Definición de tipos de datos:
  - DTD: Definición de tipo de documento
  - XSD: Esquemas de documento
- XSL: Lenguaje de estilo extensible (Visualización)
  - XSL: Lenguaje de objetos de formato.
  - XSLT: Lenguaje de transformación de XSL.
  - FO: Objetos de Formato
  - XPath: Lenguaje de direccionamiento de fragmento.
- XLL: Lenguaje de enlazado
  - XLink: Vinculación de documentos
  - XPointer: Vinculación de sectores de documento
- XHTML: Hibridación con HTML
- Análisis y manipulación
  - DOM: Modelo de Objetos de Documentos
  - SAX: API simples de XML (orientada a eventos)
  - JAXP: API de Java para XML
- Servicios Web:
  - SOAP: Protocolo de acceso a objetos simple
  - WSDL: Lenguaje de descripción de servicios Web
  - UDDI: Descripción, descubrimiento e integración universales
- Otros:
  - XQuery: Lenguaje de consulta de XML
  - XInclude: Inclusión de documentos
  - XForms: Definición de Formularios

DOCUMENTOS

Estructura

- Un documento XML es un objeto de datos.
- Para convertir una información en un documento XML debemos marcar (etiquetar) dónde comienza y dónde termina cada dato contenido en dicha información, utilizando unas reglas sintácticas muy sencillas pero muy tajantes.
- La sencillez de la sintaxis XML hace que sea muy fácil de aprender y de utilizar.
- Debido a esto, la creación de software que pueda leer o manipular XML queda simplificado al máximo (se dice que es un lenguaje hecho por programadores para programadores).
- Por lo tanto, un documento XML es un conjunto de datos denominados elementos (también conocidos como nodos o etiquetas).
- Un elemento puede tener características o propiedades que le cualifique, se denominan atributos y permiten asociar pares nombre igual a valor.
- Un elemento puede estar compuesto por otros elementos, creando así una estructura jerárquica.
- En conclusión, el XML nos dice que podemos estructurar cualquier información en un árbol compuesto de datos.

Sintaxis

- Los documentos XML utilizan una sintaxis simple y autodescriptiva.
- La tipográfica es importante, dado que es sensible a mayúsculas y minúsculas sobre todo en los nombres cualificados.
- Todos los elementos deben tener obligatoriamente una etiqueta de comienzo y una etiqueta de cierre.
- Las etiquetas de comienzo y de cierre deben estar dentro del mismo elemento, es decir, todos los elementos deben estar correctamente anidados.
- Todos los documentos XML deben tener obligatoriamente un elemento raíz que debe ser único y contenga al resto de los elementos, también se denomina elemento documento.
- En caso de aparecer, los atributos obligatoriamente deben tener valor, dicho valor debe encontrarse delimitado por apostrofes o comillas.

Sintaxis

- En XML se preserva por defecto el espacio blanco.
- En XML, el espacio en blanco consiste de uno o más caracteres de espacio (SP: #x20), retornos de carro (CR: #x0D), avances de línea (LF: #x0A) o tabuladores (TAB: #x09).
- En XML las diferentes marcas de fin de línea (LF, CR, CR+LF) se convierten al carácter LF.

Nombres cualificados

- Se denominan nombres cualificados a los nombres que siguen las reglas establecidas en el XML.
- Los nombres cualificados se utilizan para nombrar a los elementos, las entidades y los atributos.
- Las reglas a cumplir son:
  - Deben de empezar por una letra, _ (subrayado) o : (dos puntos).
  - Puede ir seguida cualquier combinación de letras, dígitos, guiones (-), subrayados (_), punto (.) o dos puntos (:).
  - No deben comenzar con las letras XML, en cualquiera de sus combinaciones de mayúsculas y minúsculas.
  - No pueden contener espacios en blanco. En caso de contener varias palabras, se recomienda el uso del subrayado como separador o escribir en minúsculas con las iniciales en mayúsculas, lo que facilita la legibilidad del nombre.
- No se recomienda el uso de los guiones (-), puntos (.) o dos puntos (:) dado que pueden tener significado en otros lenguajes que usan el XML.
- No tienen longitud máxima, pero deberían ser autodescriptivos, cortos y simples.

Formato

- En un nivel lógico, un documento contiene:
  - Elementos (datos)
  - Atributos (propiedades)
  - Entidades (componentes)
  - Comentarios (ignorados)
  - Prólogo (propiedades del documento)
  - Declaraciones (definen los elementos)
  - Instrucciones de proceso (destinados a la aplicación que trata el documento, pero no son parte de la información)
- Formato de un documento XML
  - Al escribir un documento XML se debe seguir un determinado orden, aunque algunas partes son opcionales (mostradas en cursiva).

Prologo
Declaraciones
Instrucciones de proceso y comentarios
<ElementoRaiz *atributos* >
… Elementos del documento …
</ElementoRaiz>

Elementos

- Un elemento representa un dato de la información, que puede estar, a su vez, compuesto por otros datos.
- Son extensibles: se pueden crear y ampliar de una forma flexible.
- Tienen un nombre cualificado.
- Mediante etiquetas se marca el comienzo y el final del elemento, formando parte del elemento dichas etiquetas.
- Los elementos pueden contener: un valor simple, a otros elementos, estar mezclado (valor simple con otros elementos) o estar vacío.
- En caso de contener a otros elementos se establece una relación de jerárquica de padre e hijos y hermanos.
- El orden de los elementos influye estableciendo una secuencia: primero/último y anterior/posterior.
- Un elemento puede contener atributos, en cuyo caso se expresan en la etiqueta de comienzo, separados por espacios.

Elementos

- Por lo tanto, un elemento se define:
  <nombre atributos> … Contenido … </nombre>

- Los elementos vacíos se pueden expresar de dos formas distintas:
  - Con etiquetas normales sin nada (ni siquiera espacio en blanco) entra los paréntesis angulados de la etiqueta de comienzo y la de cierre
    <nombre atributos></nombre>
  - O con la abreviatura:
    <nombre atributos />

- Por ejemplo, si queremos marcar como se llama la empresa, lo expresaríamos de la siguiente forma:
  <RazonSocial>ACME</RazonSocial>

Atributos

- Los atributos son parejas de nombre-valor que contiene información descriptiva adicional acerca de un elemento. Tienen un nombre cualificado.
- Los atributos deben ir dentro de la etiqueta de comienzo, a continuación del nombre del elemento y separados por espacios en blanco
- Puede haber atributos obligatorios y opcionales, pero siempre que aparecen obligatoriamente deben tener valor.
- El valor debe encontrarse delimitado por apostrofes o comillas, permitiendo el uso del otro delimitador dentro del valor.

  <empresa codigo='1775' nombre="Gino's" />
- Los atributos son similares a los subelementos en el sentido que permiten expresar los datos que componen a un elemento pero con una serie de matices importantes que les diferencian.
- Los atributos:
  - Son menos flexibles por lo que son más difíciles de extender y manipular.
  - No pueden aparecer varias veces para el mismo elemento.
  - El valor no puede ser una estructura.
  - El orden no influye.

Entidades

- Una entidad es un nemotécnico que será reemplazado por su correspondiente valor.
- El nombre de la entidad se encuentra delimitado con una marca de inicio: el ampersand (&), y una marca de final: el punto y coma (;):

  &entidad;

- El valor de reemplazo de una entidad puede ser:
  - La ubicación de un fichero, permitirnos elaborar un documento XML en "trozos"; es decir, tendremos un único documento XML que, físicamente, se encontrará dividido en varios ficheros. Permitiendo a su vez compartir partes de documento entre varios documentos.
  - Un texto, actuando como abreviaturas textos que se usen mucho.
  - Una referencia a carácter.
  - Un valor predefinido.

Referencias a caracteres

- Las referencias carácter son un tipo de entidades que nos permiten insertar, por su código, los caracteres especiales que, al no ser imprimibles o no estar en el teclado, no podrían escribirse directamente.
- Es necesario encontrar el valor del código del carácter deseado en el juego de caracteres definido para el documento.
- El valor puede expresarse en decimal o en hexadecimal, pero la sintaxis difiere:
  - Decimal: &#...Valor…;
  - Hexadecimal: &#x...Valor…;

Entidades predefinidas

- La sintaxis del XML utiliza una serie de caracteres como delimitadores y no pueden ser utilizados en el contenido de los elementos, dado que crearía confusión.
- Para solucionar el conflicto, el XML cuenta con una serie de entidades predefinidas para sustituir a los caracteres delimitadores.

Entidades predefinidas	**Símbolo**	**Descripción**
&lt;	<	menor que
&gt;	>	mayor que
&amp;	&	ampersand
&apos;	'	apostrofe
&quot;	"	comillas

Texto en bruto

- Para simplificar la introducción de fragmentos de texto que usan frecuentemente los caracteres delimitadores y evitar su sustitución por entidades predefinidas, el XML cuenta con las secciones de texto en bruto. Mediante unas marcas especiales se marca el trozo del documento en el que queremos que se ignoren los caracteres con significado para el XML y los trate como un bloque.
- Dichas marcas especiales son un conjunto de caracteres que deben escribirse "tal cual", en el mismo orden y sin espacios entre ellos.
- La marca de comienzo es: <![CDATA[y la de final de bloque:]]>
  - Por ejemplo:

```
<![CDATA[En la empresa Walter & Co puede ser:
 uno < otro
ó
este > aquel.]]>
```


Comentarios

- Los documentos XML pueden contar con comentarios en cualquier lugar, para que si es una persona la que lee el documento, en vez de una aplicación (a las que fundamentalmente están destinados los documentos XML), esta entere mejor. El XML permite a las aplicaciones que ignoren los comentarios y no los procesen.
- Los comentarios están encerrados entre senda etiquetas <!-- y --> y pueden contener cualquier tipo de datos, en una o varias líneas, excepto el literal '- -'.
- Por ejemplo:

```
<!- -
Esto es un comentario de
varias líneas.
 - ->
```

Prologo

- Los documentos XML pueden, y deben, empezar con una declaración de XML que específica tres propiedades: la versión de XML usada, el juego de caracteres empleado y si el documentos esta aislado.
- Las propiedades que usan sus valores por defecto no es necesario incluirlas. Si todas las propiedades tienen los valores por defecto, no es obligatorio empezar con la declaración, aunque se recomienda hacerlo.
- En caso de aparecer la declaración, el primer carácter de la declaración debe ser el primer carácter del documento (estrictamente).

- La sintaxis con sus valores por defecto es:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
```


Prologo

- Los atributos del prologo son:
  - version: El número de versión "1.0" debe ser usado para indicar conformidad con esta versión de la especificación (es la versión que se utiliza actualmente).
  - encoding: Juego de caracteres utilizado en el documento. El XML utiliza por defecto el Unicode (UTF-8 y UTF-16), pero para soportar múltiples idiomas, permite el uso de otros sistemas de codificación (ISO-XXXX-X, EBCDIC, EUC-XX, windows--XXXX) si se marca correctamente. Esta propiedad permite que los documentos XML sean multiplataforma, dado que una de las principales incompatibilidades entre las plataformas son los diferentes juegos de caracteres empleados por cada una.
  - standalone: Un documento se encuentra aislado si no requiere de marcas externas para ser procesado. Los posibles valores son: 'yes' y 'no'.

Declaraciones

- El documento XML puede tener tipo, en cuyo caso esta sujeto a una serie de restricciones que se declaran a través de los DTD, Definición de Tipo de Documento (que veremos más adelante).
- Las declaraciones se pueden realizar dentro del propio documento (declaración interna) o fuera del documento (declaración externa).
- Para realizar una declaración interna usaremos:

```
<!DOCTYPE ElementoRaiz [
… Instrucciones DTD …
]>
```

- Si la declaración es externa indicaremos donde esta el documento con la definición:

```
<!DOCTYPE ElementoRaiz SYSTEM "fichero.dtd">
```


Declaraciones

- Si la declaración utiliza un estándar público:

```
<!DOCTYPE ElementoRaiz PUBLIC "identificador público"
"dirección (URI) del recurso">
```

- Se pueden realizar declaraciones mixtas, tomar una declaración externa y completarla con una declaración interna (en caso de conflicto priman las internas sobre las externas).

```
<!DOCTYPE ElementoRaiz SYSTEM "fichero.dtd" [
… Instrucciones DTD …
]>
```

Instrucciones de proceso

- Las instrucciones de procesamiento permiten a los documentos incluir instrucciones para las aplicaciones que los van a utilizar, y no son parte de los datos de carácter del documento.
- La instrucción están encerrados entre senda etiquetas <? y ?> y comienza con un objetivo usado para identificar la aplicación para la que está dirigida, continuando con una serie de pares nombre=valor con los parámetros pasados a la aplicación.
- En los ejemplos indicamos al navegador que hoja de estilos y que transformación debe realizar para mostrar el documento XML:

```
<?xml-stylesheet type='text/css' href='HojaDeEstilos.css'?>
<?xml-stylesheet type='text/xsl' href='FicheroDeTransformacion.xsl'?>
```

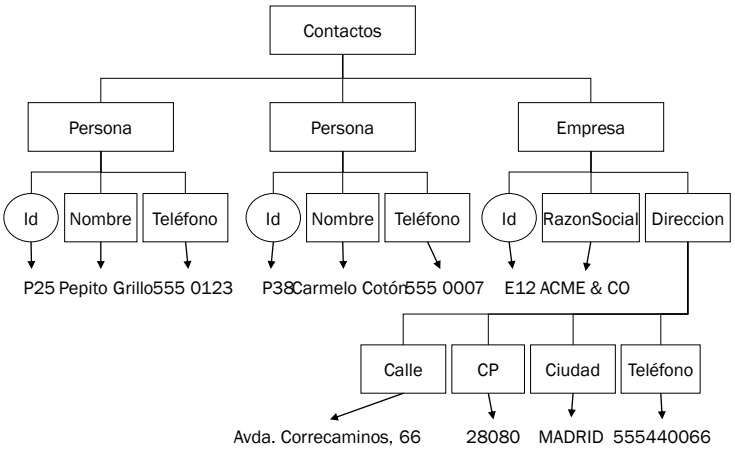

Conclusión

- Un documento XML es un objeto de datos.
- Es posible ver todo documento XML como un árbol, dado que tiene estructura jerárquica:
  - La raíz es el elemento raíz
  - Los nodos internos son elementos o atributos
  - Los nodos externos (hojas) son texto o elementos vacíos.
  - Los hijos de un nodo elemento pueden ser elementos, texto o atributos
  - Los hijos de un nodo atributo sólo pueden ser texto
  - El orden de los hijos de un nodo es importante y debe ser conservado

Agenda

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Ejemplo de documento XML -->
<!DOCTYPE Contactos SYSTEM "./contactos.dtd">
<?xml-stylesheet type='text/xsl' href='./Agenda.xsl'?>
<Contactos>
  <Persona Id="P23">
    <Nombre>Pepito Grillo</Nombre>
    <Telefono>555 0123</Telefono>
  </Persona>
  <Persona Id="P38">
    <Nombre>Carmelo Cotón</Nombre>
    <Telefono>555 0007</Telefono>
  </Persona>
  <Empresa Id="E12">
    <RazonSocial>ACME &amp;amp; CO</RazonSocial>
    <Direccion>
      <Calle>Avda. Correcaminos, 66</Calle>
      <CP>28080</CP>
      <Ciudad>MADRID</Ciudad>
      <Telefono>555440066</Telefono>
    </Direccion>
  </Empresa>
</Contactos>
```


Árbol de nodos



contactos.dtd

```
<!ELEMENT Contactos (Persona | Empresa)*>
<!ELEMENT Persona (Nombre,Telefono)>
<!ATTLIST Persona Id ID #REQUIRED>
<!ELEMENT Empresa (RazonSocial,Direccion*)>
<!ATTLIST Empresa Id ID #REQUIRED>
<!ELEMENT Direccion (Calle, CP, Ciudad, Telefono)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Telefono (#PCDATA)>
<!ELEMENT RazonSocial (#PCDATA)>
<!ELEMENT Calle (#PCDATA)>
<!ELEMENT CP (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
```


Definición de Tipo de Documento

DTD

Definición de Tipo de Documento (DTD)

- El XML no tiene elementos predefinidos. Cuando nos ponemos a escribir un documento XML, podemos utilizar nuestros propios elementos, pero si es necesario restringir a un formato más estricto debemos declarar y utilizar un DTD.
- Una DTD (Document Type Definition) es un conjunto de reglas para definir un documento XML y etiquetarlo adecuadamente.
- Define los elementos, atributos, entidades y notaciones que pueden utilizarse para construir un tipo de documentos, así como su estructura y las reglas para su utilización. La definición utiliza una sintaxis especializada (tienen su origen en SGML).
- El uso de una DTD permite fijar un formato común de datos para su utilización en el intercambio de datos entre diversos orígenes y destinos independientes entre si. Tanto emisor como receptor utilizan la misma DTD, que permite la validación, por lo que están de acuerdo y no se producen conflictos, es por ello por lo que se dice que tienen carácter contractual (comprometen a ambas partes).

DTD

- Mediante las DTD's se comprueba la validez de un documento, verificando que cumple las restricciones en ellas expresadas. En caso de no cumplirla, el documento debe ser rechazado sin tratamiento alguno.
- El XML es un metalenguaje, lenguaje para definir a otros lenguajes. La definición de un nuevo lenguaje se realiza mediante una DTD, por ello las DTD's se conocen también como "vocabularios XML".
- La Definición de Tipo de Documento se realiza en un fichero o conjuntos de ficheros con extensión .dtd (por defecto).
- Dentro del documento Definición de Tipo de Documento se pueden declarar:
  - Tipos Elemento
  - Lista de Atributos
  - Entidades
  - Notaciones
- Se pueden introducir secciones condicionales dentro del documento.

Declaraciones de tipo elemento

- Determinan las reglas que deben cumplir los elementos para que puedan aparecer en el documento. Cada elemento debe ser declarado una única vez.
- Para declarar un elemento debemos definir su nombre, contenido y frecuencia con la siguiente sintaxis:

<!ELEMENT NombreElemento Contenido>

- El NombreElemento sigue las reglas de composición de nombres cualificados anteriormente expuestas.
- El contenido del elemento puede ser: EMPTY, ANY, PCDATA, subelementos o mixto.
- Cuando el elemento siempre está vacío, dado que la información se suministra a través de atributos, se marca con EMPTY.

<!ELEMENT NombreElemento EMPTY>

Declaraciones de tipo elemento

- Cuando un elemento solo va a contener datos de carácter, se utiliza la declaración (#PCDATA):

  <!ELEMENT NombreElemento (#PCDATA)>

- Para los elementos compuestos de otros elementos (subelementos o elementos hijos) se debe acompañar de la lista de subelementos encerrada entre paréntesis y su frecuencia.

  <!ELEMENT NombreElemento (lista de subelementos)frecuencia>

- La lista de subelementos está compuesta por uno o varios subelementos, indicando la composición, el orden y la frecuencia de los hijos.
- Los elementos de la lista se separan por comas cuando son secuenciales y por pipe (|) cuando son alternativos (uno u otro pero no los dos a la vez).
- Los subelementos de la lista pueden ser a su vez una lista de subelementos, en cuyo caso se encierran entre paréntesis para delimitarlos.

Declaraciones de tipo elemento

- La frecuencia indica las veces que debe o puede aparecer un elemento. Por defecto, si no se indica nada, debe aparecer obligatoriamente una única vez. Para el resto de los casos se utilizan los siguientes indicadores a continuación del elemento o lista y sin espacio entre medias:
  - ? Opcional simple, puede aparecer pero solo una vez.
  - * Opcional múltiple, puede aparecer una o varias veces.
  - + Obligatorio múltiple, debe aparecer al menos una vez pero puede aparecer varias veces.
- Por ejemplo:

  <!ELEMENT Agenda (Contacto)*>

  <!ELEMENT Contacto (Codigo, ((Nombre, Apellido?) | RazonSocial), Telefono+, Direccion*)>
- Algunos elementos pueden tener contenido mixto, mezclando datos de carácter con subelementos, pero con algunas limitaciones: una única aparición del #PCDATA y los subelementos pueden ser restringidos, pero no su orden o su frecuencia.

  <!ELEMENT NombreElemento (#PCDATA | subelemento | …)*>

Declaraciones de Lista de Atributos

- La declaración de lista de atributos especifica el nombre, tipo de datos, y valor por defecto (si existe) de un atributo asociado con un tipo de elemento dado.

  <!ATTLIST NombreElemento DefinicionesDeAtributos>

- Una instrucción ATTLIST puede contener una o varias (separadas por espacios) definiciones de atributos de un elemento. Para definir un atributo es necesario indicar:

  NombreAtributo Tipo ValorPorDefecto

- El NombreAtributo sigue las reglas de composición de nombres cualificados anteriormente expuestas.

Declaraciones de Lista de Atributos

El ValorPorDefecto indica si el atributo debe aparecer siempre (es obligatorio) o es opcional. Si es opcional indicaremos que valor tiene por defecto si no aparece. Las opciones son:

#REQUIRED	Requerido, es obligatorio que aparezca con valor, por lo tanto no es necesario que tenga valor por defecto.
#IMPLIED	Opcional sin valor por defecto, si no aparece queda vacío.
#FIXED '*valor*' #FIXED "*valor*"	Opcional con valor fijo, puede aparecer en cuyo caso obligatoriamente solo puede tomar el valor expresado, se suministra para clarificar contenidos: *<!ATTLIST DeclaracionRenta Ejercicio CDATA #FIXED "2005">*
'*valor*' "*valor*"	Opcional, en caso de omitirse toma el valor expresado como valor por defecto.

Declaraciones de Lista de Atributos

- Los atributos en XML son de tres tipos:
  - –Cadena (CDATA)
    <!ATTLIST Factura Fecha CDATA #REQUIRED>
  - –Enumerados (lista de valores entre paréntesis, separados por | y sin comillas ni apostrofes)
    <!ATTLIST Factura FormaPago (cheque|efectivo|tarjeta) 'efectivo'>
  - –Simbólicos (con significado):

ID	Identificador o clave del elemento, debe contener un valor único, siguiendo las reglas de composición de nombres cualificados, que no esté repetido en todo el documento. Solo un atributo del elemento puede ser de tipo ID y tiene que tener como valor por defecto #REQUIRED o #IMPLIED.
IDREF	Valor referencia (clave ajena), el atributo solo puede tomar como valor un identificador (valor usado en un atributo de tipo ID de otro elemento)

Declaraciones de Lista de Atributos

IDREFS	Lista de IDREF, puede contener varios valores IDREF separados por espacios.
ENTITY	Entidad, valor que hace referencia al nombre de una entidad no procesada declarada en el DTD.
ENTITIES	Lista de ENTITY, puede contener varios valores ENTITY separados por espacios.
NMTOKEN	Cadena que sigue las reglas de composición de nombres cualificados.
NMTOKENS	Lista de NMTOKEN, puede contener varios valores NMTOKEN separados por espacios.

- A modo de ejemplo podemos ver diferentes tipos de atributos asociados a las facturas:
  - <!ATTLIST Factura Numero ID #REQUIRED
    Cliente IDREF #REQUIRED
    Pedidos IDREFS #IMPLIED>

Declaraciones de Lista de Atributos

- Existen un par de atributos especiales predefinidos que peden asociarse a todos los elementos y sirven para indicar el idioma utilizado en el contenido y si se desea preservar el espacio (tantos caracteres como aparezcan en el contenido o resumidos en un único carácter).
  - xml:lang="XX" ó xml:lang="XX-SS"
    - Donde XX representa el código del idioma y el SS, es opcional, representa el subcódigo del idioma o código de país según indican los estándares IANA, ISO 639 e ISO 3166.
  - xml:space (default|preserve) 'preserve'

Declaraciones de Entidades y Notaciones

- La declaración de una entidad asocia un nombre (nemotécnico) al contenido por el cual queremos que se reemplace dicho nombre.
- La sintaxis de la declaración está en función su tipo. El tipo se establece dependiendo de:
  - Si la entidad debe ser analizada o no (verificar la sintaxis del contenido), una entidad se puede clasificar como analizable o no analizable.
  - Si el contenido de la entidad es interno o externo es decir de si su contenido aparece en la declaración o en un fichero, la entidad se puede clasificar como interna o externa.
  - Si se utiliza dentro del DTD o en el documento se clasifica como parámetro o general.
- No todas las combinaciones de las diferentes características producen tipos válidos.
- Las notaciones se utilizan en las entidades de tipo general externa no analizable.

Declaraciones de Entidades y Notaciones

- Entidad general interna (siempre analizable).
  - Sustituyen el nombre por una cadena.
    <!ENTITY nemotécnico "texto de reemplazo">
  - Este tipo de entidades es muy utilizado también cuando es necesario emplear algún carácter especial del juego de caracteres (referencia a carácter):
    <!ENTITY copyright "©">
- Entidad general externa analizable.
  - En este caso el contenido es encuentra almacenado en un fichero externo.
    - <!ENTITY nemotécnico SYSTEM "dirección (URI) del recurso">
  - En el caso de el fichero se encuentre estandarizado podemos utilizar su identificador público (unívoco).
    - <!ENTITY nemotécnico PUBLIC "identificador público" "dirección (URI) del recurso">

Declaraciones de Entidades y Notaciones

- Entidad general externa no analizable.
  - Es un recurso externo cuyos contenidos pueden o no ser texto, y si lo son, tal vez no sean XML. Cada entidad sin procesar requiere una notación que identifique su contenido e indique como tratarlo (texto, gráfico, binario…).
  - Solo pueden ser usadas como valor de atributos de tipo ENTITY o ENTITIES.

    <!ENTITY nemotécnico SYSTEM "dirección (URI) del recurso" NDATA NombreNotacion>

  - Para declarar la notación utilizaremos:

    <!NOTATION NombreNotacion SYSTEM "dirección (URI) del recurso">

    <!NOTATION NombreNotacion PUBLIC "identificador público" "dirección (URI) del recurso">

Declaraciones de Entidades y Notaciones

- Entidad parámetro interna (siempre analizable).
  - Las entidades que sólo pueden utilizarse en la DTD se denominan entidades parámetro y su sintaxis difiere ligeramente (usa el símbolo %) de la sintaxis de las globales. Nos ayudarán a organizar y agilizar la creación de las DTD, haciéndola más eficiente y consistente.
  - Están diseñadas para contener listas de atributos y modelos de contenido como texto de reemplazo. La definimos una vez y luego hacemos referencia a ella tantas veces como sea necesario. De esta manera si necesitamos modificar estos elementos, tendremos que hacerlo sólo una vez, en un único lugar, y no tener que ir elemento por elemento realizando la modificación. Se las considera segmentos de código reutilizable.
  - Dichas entidades se declaran:

    <!ENTITY % parámetro "Declaración de listas de atributos o contenido">
  - Para hacer referencia a los parámetros la marca de inicio es el símbolo porciento (%) en vez del ampersand (&) de las globales:

    <!ENTITY % persona "nombre, inicial?, apellidos, telefono*">

    <!ELEMENT profesor (%persona; titulo, conocimientos+, salario)>

    <!ELEMENT alumno (%persona; nota)>

Declaraciones de Entidades y Notaciones

- Entidad parámetro externo (siempre analizable).
- En este caso, la definición es encuentra almacenada en un fichero externo, esto permite compartirlas entre varios DTD's.

<!ENTITY % parámetro SYSTEM "dirección (URI) del recurso">

<!ENTITY % parámetro PUBLIC "identificador público" "dirección (URI) del recurso">

Secciones Condicionales

- Para facilitar el versionado, las DTD's permiten marcar zonas incluidas, participan en la definición, y zonas ignoradas, aunque aparecen no son parte de la definición.
- En la DTD podemos conservar un parte obsoleta pero que no queremos borrar, o mostrar lo que se va a incluir en la próxima versión, pero que no queremos que se use en la versión actual. Para marcar que ignore una zona utilizaremos la sintaxis:

```
<![IGNORE[
… Todas las declaraciones se ignoran …
]]>
```

- Para marcar que incluya una zona utilizaremos la sintaxis:

```
<![INCLUDE[
… Todas las declaraciones se ignoran …
]]>
```

Secciones Condicionales

- Por defecto, todo está incluido por lo que no es necesario utilizar INCLUDE. Se suministra para ser usado en combinación con las entidades parámetro. Al declarar entidad parámetro, como contenido podemos utilizar INCLUDE o IGNORE a conveniencia.
- En la versión actual de la DTD queremos disponer de facturas:

```
<!ENTITY % opcional 'INCLUDE' >
… Otras declaraciones …
<![%opcional;[
<!ELEMENT Factura (LineaFactura+)>
<!ATTLIST Factura Numero ID #REQUIRED
Cliente IDREF #REQUIRED
Pedidos IDREFS #IMPLIED>
]]>
```

- En la siguiente versión ya no son necesarias las facturas pero, como es posible que puedan volver a ser necesarias, no las queremos borrar. La solución más simple es cambiar la definición de la entidad. Basta sustituir la anterior con:

```
<!ENTITY % opcional 'IGNORE' >
```


contactos.dtd

```
<!ELEMENT Contactos (Persona | Empresa)*>
<!ELEMENT Persona (Nombre,Telefono)>
<!ATTLIST Persona Id ID #REQUIRED>
<!ELEMENT Empresa (RazonSocial,Direccion*)>
<!ATTLIST Empresa Id ID #REQUIRED>
<!ELEMENT Direccion (Calle, CP, Ciudad, Telefono)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Telefono (#PCDATA)>
<!ELEMENT RazonSocial (#PCDATA)>
<!ELEMENT Calle (#PCDATA)>
<!ELEMENT CP (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
```

Agenda

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Ejemplo de documento XML -->
<!DOCTYPE Contactos SYSTEM "./contactos.dtd">
<?xml-stylesheet type='text/xsl' href='./Agenda.xsl'?>
<Contactos>
  <Persona Id="P23">
    <Nombre>Pepito Grillo</Nombre>
    <Telefono>555 0123</Telefono>
  </Persona>
  <Persona Id="P38">
    <Nombre>Carmelo Cotón</Nombre>
    <Telefono>555 0007</Telefono>
  </Persona>
  <Empresa Id="E12">
    <RazonSocial>ACME &amp;amp; CO</RazonSocial>
    <Direccion>
      <Calle>Avda. Correcaminos, 66</Calle>
      <CP>28080</CP>
      <Ciudad>MADRID</Ciudad>
      <Telefono>555440066</Telefono>
    </Direccion>
  </Empresa>
</Contactos>
```


Espacios de nombres (Namespaces)

- Un espacio de nombre XML es una colección de nombres, identificados por una URI (que contiene la DTD o esquema), que se utiliza en los documentos XML para identificar y validar los nombres de los elementos y atributos del documento.
- El uso de espacios de nombres:
  - Permite utilizar varios vocabularios XML (DTD) dentro de un mismo documento XML
  - Resuelve las ambigüedades y colisiones entre las diversas definiciones
  - Permite la modularidad y reutilización de las definiciones
  - Permite la creación de definiciones universales
- Las aplicaciones que soporten al espacio deben reconocer y estar de acuerdo con sus declaraciones y prefijos.
- Se pueden declarar varios espacios por elemento (asociados a un prefijo), en cuyo caso, la definición es la unión de todas las definiciones.
- Se considera que la declaración del espacio de nombres se aplica al elemento en que está especificada y a todos los elementos pertenecientes al contenido del dicho elemento (subelementos).

Espacios de nombres

- Se pueden anidar declaraciones de espacios de nombres mediante su declaración en los elementos contenidos, en cuyo caso, un subelemento puede incluso anular un espacio recibido mediante la declaración de otros espacio de nombres con el mismo prefijo o del espacio de nombres por defecto.
- Los espacios de nombres deben ir dentro de la etiqueta de comienzo de los elementos, a continuación del nombre del elemento y separados por espacios en blanco, utilizando el atributo especial xmlns que debe ir siempre en minúsculas, asociados a un prefijo.

  <Elemento xmlns:prefijo="dirección (URI) de la definición" atributos>
- El prefijo es un nombre cualificado (aunque puede comenzar por _) asociado al espacio de nombres. Los prefijos permiten resolver los conflictos que se producen cuando dos subelementos se llaman igual en dos espacios distintos.
- Se puede declarar un espacio de nombres por defecto para el elemento suprimiendo el prefijo.

  <Elemento xmlns="dirección (URI) de la definición" atributos>

Espacios de nombres

- Para indicar que un subelemento pertenece a un determinado espacio de nombres, se precede el nombre del elemento, en la etiqueta de comienzo y en la de cierre, por el prefijo del espacio al que pertenece seguido de dos puntos. Esto solo es obligatorio cuando sea necesario resolver los conflictos.

```
<Empresa xlmns:contacto='…URI…' xlmns:servidor='…URI…'>
        <RazonSocial>RTVE</RazonSocial>
        <contacto:Direccion>
            <Calle>Prado del Rey, 32</Calle>
            <CP>28050</CP>
            <Ciudad>MADRID</Ciudad>
            <Telefono>912345678</Telefono>
        </contacto:Direccion>
        <servidor:Direccion>
            <url> http://www.rtve.com</url>
            <ip>192.168.0.1</ip>
        </servidor:Direccion>
</Empresa>
```

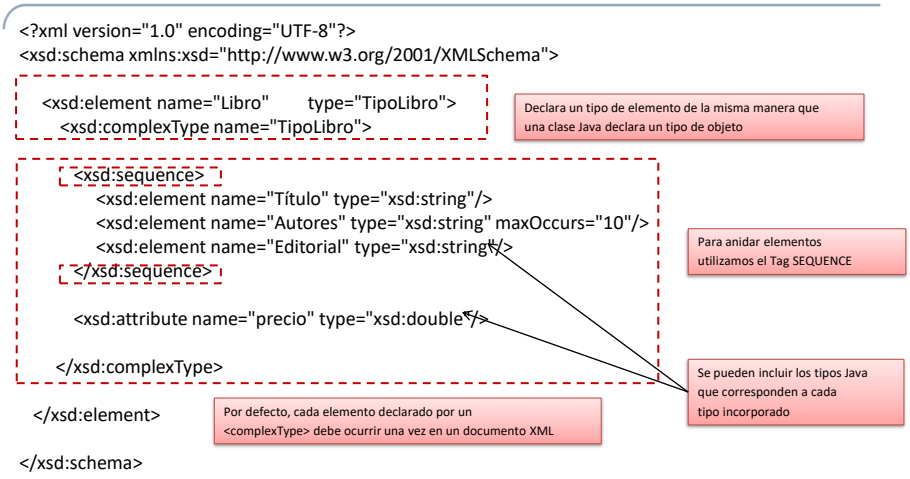
- Si los espacios de nombres se declaran en el elemento raíz, el efecto es similar que la declaración DOCTYPE del documento.

ESQUEMAS DE DOCUMENTOS XML (XSD)

Esquemas de Documentos XML (XSD)

- Es una tecnología XML con las misma finalidad que las DTD's, describir los elementos y atributos que se pueden utilizar para construir documentos XML y las reglas de utilización.
- El XSD surgió para suplir las carencias del DTD's, aportando importantes mejoras entre las que destacan:
  - Permiten asociar tipos de datos con los elementos
  - Permiten limitar los valores aceptados en los elementos.
  - Permite crear nuevos tipos de datos tanto simples como compuestos.
- El XSD es un "vocabulario XML", es decir, está definido mediante una DTD, siendo los esquemas, a su vez, documentos XML.
- Los ficheros resultantes tienen extensión .xsd
- Se diseñan alrededor de los espacios de nombres (namespaces).
- Para ampliar la información se puede consultar la especificación de los mismos en:
  http://www.w3c.es/Traducciones/es/TR/2001/REC-xmlschema-0-20010502/

Esquemas de Documentos XML (XSD)



Esquema

- Un esquema XSD es un documento XML con la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 :
</xsd:schema>
```

- Donde aparecen los siguientes elementos:
  - La etiqueta xml define la versión de XML utilizada y la codificación de caracteres del documento.
  - La etiqueta raíz xsd:schema define el espacio de nombres del esquema XSD así como una serie de atributos opcionales que podéis ampliar aquí.
  - La etiqueta xsd:element que contiene la definición del elemento raíz del documento XML al que se le va a aplicar el esquema y que explicaremos más adelante.
- El documento XML debe referenciar el esquema XSD con el que se va a validar utilizando la siguiente cabecera:

```
<?xml version="1.0" encoding="UTF-8"?>
<raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mi-esquema.xsd">
  ...
</raiz>
```

- Donde la etiqueta raíz del documento define los atributos:
  - xmlns:xsi para declarar el espacio de nombres del esquema XSD.
  - xsi:noNamespaceSchemaLocation para vincular el documento XML con el esquema local XSD.

Elementos de los esquemas XML

- Elementos de nivel superior
- Atributos
- Definiciones de tipo simple
- Definiciones de tipo complejo
- Partículas
- Espacios de nombres y documentos XML múltiples
- Restricciones de identidad
- Objetos de esquemas con nombre

Elementos de nivel superior

<xsd:element>: Declara un elemento.

<xsd:attribute>: Declara un atributo.

<xsd:simpleType>: Define un tipo simple que determina las restricciones e información acerca de los valores de atributos o elementos con contenido de texto únicamente.

<xsd:complexType>: Define un tipo complejo, que determina el conjunto de atributos y el contenido de un elemento.

<xsd:group>: Agrupa un conjunto de declaraciones de elementos de forma que puedan incorporarse como un grupo en definiciones de tipos complejos.

<xsd:attributeGroup>: Agrupa un conjunto de declaraciones de atributos para que puedan ser incorporadas como grupo en definiciones de tipos complejos.

<xsd:annotation>: Define una anotación.

<xsd:import>: Identifica un espacio de nombres a cuyos componentes de esquema se hace referencia desde el esquema contenedor.

<xsd:include>: Incluye el documento de esquema especificado en el espacio de nombres de destino del esquema contenedor.

<xsd:notation>: Contiene la definición de una notación para describir el formato de datos que no son de XML en un documento XML.

xsd:element

- Este componente permite declarar los elementos del documento XML.
- Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:
  - name: Indica el nombre del elemento. Obligatorio si el elemento padre es <xsd:schema>.
  - ref: Indica que la declaración del elemento se encuentra en otro lugar del esquema. No se puede usar si el elemento padre es <xsd:schema>. No puede aparecer junto con name.
  - type: Indica el tipo de dato que almacenará el elemento. No puede aparecer junto con ref.
  - default: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
  - fixed: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual.
  - minOccurs: Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xsd:schema>. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.
  - maxOccurs: Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xsd:schema>. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.

```
<xsd:element name="nombre" type="xsd:string" default="TicArte" minOccurs="1" maxOccurs="unbounded" />
<xsd:element ref="contacto" minOccurs="1" maxOccurs="unbounded" />
```


xsd:attribute

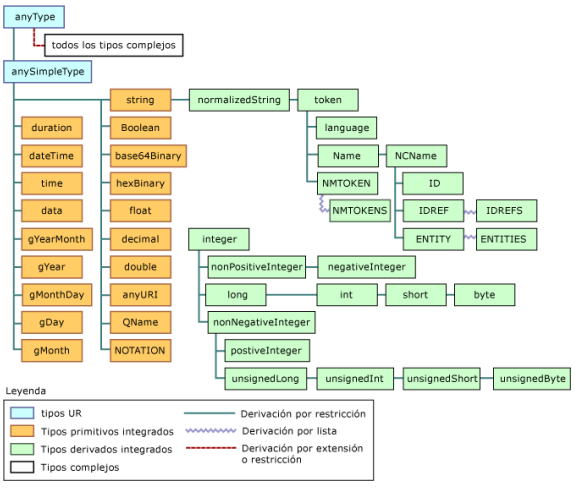
- Este componente permite declarar los atributos de los elementos del documento XML. Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:
  - name: Indica el nombre del atributo.
  - ref: Indica que la declaración del atributo se encuentra en otro lugar del esquema. No puede aparecer junto con name.
  - type: Indica el tipo de dato que almacenará el atributo. No puede aparecer junto con ref.
  - use: Indica si la existencia del atributo es opcional (optional), obligatoria (required) o prohibida (prohibited). Por defecto opcional.
  - default: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
  - fixed: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual.

```
<xsd:attribute name="moneda" type="xsd:string" default="euro" />
<xsd:attribute ref="moneda" use="required" />
```

Tipos de datos

- Tipos de datos simples (xsd:simpleType): Se dividen en los siguientes.
  - Tipos de datos predefinidos.
    - Primitivos: Descienden directamente de xsd:anySimpleType.
    - No primitivos: Descienden de alguno de los primitivos.
  - Tipos de datos definidos con nuestras propias restricciones y basados en los tipos de datos predefinidos.
- Tipos de datos complejos (xsd:complexType): Se dividen en los siguientes.
  - Elementos dentro de otros elementos.
  - Elementos que tienen atributos.
  - Elementos mixtos que tienen datos y otros elementos.
  - Elementos vacíos.

Tipos de datos



Tipos de datos primitivos

Tipo de Dato	Descripción
string	Cadena de caracteres.
boolean	Valores booleanos que pueden ser **true** o **false**.
decimal	Números reales.
float	Números en punto flotante de 32 bits.
double	Números en punto flotante de 64 bits.
duration	Duración representada en "Años + Meses + Días + Horas + Minutos + Segundos" (P10Y5M4D10H12M50S).
dateTime	Fecha y hora en formato CCYY-MM-DDThh:mm:ss donde CC es el siglo, YY el año, MM el mes, DD el día, precedido del carácter (-) indica un número negativo. La T es el separador de hh horas, mm minutos, y ss segundos. Puede seguirse de una Z para indicar la zona UTC (Universal Time Coordinated).
time	Hora en formato hh:mm:ss.sss.

Tipos de datos primitivos

Tipo de Dato	Descripción
gYearMonth	Sólo el año y mes en formato Gregoriano CCYY-MM.
gYear	Sólo el año en formato Gregoriano CCYY.
gMonthDay	Sólo el mes y el día en formato Gregoriano --MM-DD.
gDay	Sólo el día en formato Gregoriano ---DD.
gMonth	Sólo el mes en formato Gregoriano --MM--.
hexBinary	Secuencia de dígitos hexadecimales.
base64Binary	Secuencia de dígitos hexadecimales en base 64.
anyURI	Cualquier identificador URI.
QName	Cualquier nombre cualificado del espacio de nombres
NOTATION	Tipo de datos para atributo compatible con DTD.

Tipos de datos derivados

Tipo de dato	Descripción
normalizedString	Representa cadenas normalizadas de espacios en blanco. Este tipo de datos se deriva de string.
token	Representa cadenas convertidas en tokens. Este tipo de datos se deriva de normalizedString.
language	Representa identificadores de lenguaje natural (definidos por RFC 1766). Este tipo de datos se deriva de token.
ID	Representa el tipo de atributo ID definido en la recomendación de XML 1.0. El IDno debe incluir un signo de dos puntos (NCName) y debe ser único en el documento XML. Este tipo de datos se deriva de NCName.
IDREF	Representa una referencia a un elemento que tiene un atributo ID que coincide con el ID especificado. IDREF debe ser un NCName y tener un valor de un elemento o atributo de tipo ID dentro del documento XML. Este tipo de datos se deriva de NCName.
IDREFS	Representa el tipo de atributo IDREFS. Contiene un conjunto de valores de tipo IDREF.

Tipos de datos derivados

Tipo de dato	Descripción
ENTITY	Representa el tipo de atributo ENTITY definido en la recomendación de XML 1.0. Es una referencia a una entidad sin analizar con un nombre que coincide con el especificado. ENTITY debe ser un NCName y declararse en el esquema como nombre de entidad sin analizar. Este tipo de datos se deriva de NCName.
ENTITIES	Representa el tipo de atributo ENTITIES. Contiene un conjunto de valores de tipo ENTITY.
NMTOKEN	Representa el tipo de atributo NMTOKEN. NMTOKEN es un juego de caracteres de nombres (letras, dígitos y otros caracteres) en cualquier combinación. A diferencia de Name y NCName, NMTOKEN, no tiene restricciones del carácter inicial. Este tipo de datos se deriva de token.
NMTOKENS	Representa el tipo de atributo NMTOKENS. Contiene un conjunto de valores de tipo NMTOKEN.
Name	Representa nombres en XML. Name es un token que empieza con una letra, carácter de subrayado o signo de dos puntos, y continúa con caracteres de nombre (letras, dígitos y otros caracteres). Este tipo de datos se deriva detoken.

Tipos de datos derivados

Tipo de dato	Descripción
NCName	Representa nombres sin el signo de dos puntos. Este tipo de datos es igual que Name, excepto en que no puede empezar con el signo de dos puntos. Este tipo de datos se deriva de Name.
integer	Representa una secuencia de dígitos decimales con un signo inicial (+ o -) opcional. Este tipo de datos se deriva dedecimal.
positiveInteger	Representa un número entero mayor que cero. Este tipo de datos se deriva de nonNegativeInteger.
nonPositiveInteger	Representa un número entero menor o igual que cero. nonPositiveInteger consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de integer.
negativeInteger	Representa un número entero menor que cero. Consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de nonPositiveInteger.
nonNegativeInteger	Representa un número entero mayor o igual que cero. Este tipo de datos se deriva de integer.

Tipos de datos derivados

Tipo de dato	Descripción
long	Representa un entero con un valor mínimo de -9223372036854775808 y un valor máximo de 9223372036854775807.Este tipo de datos se deriva de integer.
unsignedLong	Representa un entero con un valor mínimo de cero y un valor máximo de 18446744073709551615. Este tipo de datos se deriva de nonNegativeInteger.
int	Representa un entero con un valor mínimo de -2147483648 y un valor máximo de 2147483647. Este tipo de datos se deriva de long.
unsignedInt	Representa un entero con un valor mínimo de cero y un valor máximo de 4294967295. Este tipo de datos se deriva de unsignedLong.
short	Representa un entero con un valor mínimo de -32768 y un valor máximo de 32767. Este tipo de datos se deriva de int.
unsignedShort	Representa un entero con un valor mínimo de cero y un valor máximo de 65535. Este tipo de datos se deriva de unsignedInt.
byte	Representa un entero con un valor mínimo de -128 y un valor máximo de 127. Este tipo de datos se deriva de short.
unsignedByte	Representa un entero con un valor mínimo de cero y un valor máximo de 255. Este tipo de datos se deriva de unsignedShort.

Tipos de datos simples definidos

- Los tipos de datos definidos son generados por el usuario a partir de un tipo de dato predefinido, aplicándoles restricciones si se desea.
- Pueden usarse directamente en la definición de un elemento, en lugar de usar el atributo type:

```
<xsd:element name="edad" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:integer">
   <xsd:minInclusive value="16"/>
   <xsd:maxInclusive value="67"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

- También pueden definirse asignándoles un nombre y pudiéndose usar en cualquier elemento del documento mediante el atributo type. El orden de definición es indiferente:

```
<xsd:simpleType name="calificacion">
 <xsd:restriction base="xsd:string">
  <xsd:minLength value="0"/>
  <xsd:maxLength value="10"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:element name="nota" type="calificacion"/>
```

- Y se pueden construir tipos de datos extendiendo otros tipos de datos ya existentes.

Definiciones de tipo simple

<xsd:list>: Define una colección de una única definición simpleType.

```
<xs:simpleType name='listOfIntegers'>
 <xs:list itemType='integer'/>
</xs:simpleType>
```

<xsd:union>: Define una colección de varias definiciones simpleType.

```
<xs:attribute name="fontsize">
 <xs:simpleType>
  <xs:union memberTypes="fontbynumber fontbystringname" />
 </xs:simpleType>
</xs:attribute>
```

<xsd:restriction>: Define restricciones en una definición.

Restricciones o facetas (simpleType)

- Rango de números: Se utiliza en los tipos de datos numéricos y de fecha/hora. Define el valor mínimo y máximo, tanto inclusive como exclusive.

```
<xsd:element name="edad" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:integer">
   <xsd:minInclusive value="16"/>
   <xsd:maxExclusive value="67"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

- Dígitos: Se utiliza en los tipos de datos numéricos. Define el número máximo de dígitos permitidos.

```
<xsd:element name="telefono" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:integer">
   <xsd:totalDigits value="9"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```


Restricciones o facetas (simpleType)

- Longitud: Se utiliza en tipos de datos de texto. Define el número exacto de caracteres permitidos, o el mínimo y máximo de ellos.

```
<xsd:element name="clave" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:length value="5"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:restriction base="xsd:string">
    <xsd:minLength value="5"/>
    <xsd:maxLength value="8"/>
</xsd:restriction>
```

- Plantilla de caracteres: Se utiliza en tipos de datos de texto. Especifica un patrón o expresión regular que debe cumplir el contenido del elemento.

```
<xsd:element name="iniciales" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="[A-Z][A-Za-z][A-Za-z]"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

Restricciones o facetas (simpleType)

- Tratamiento de espacios en blanco: Se utiliza en tipos de datos de texto. Especifica cómo se tratan los espacios en blanco (que incluyen también saltos de línea y tabuladores). Los puede respetar (preserve), los puede reducir a uno (collapse) y los puede reemplazar (replace).

```
<xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="preserve"/>
</xsd:restriction>
```

- Lista de valores: Se utiliza en todos los tipos de datos. Define una lista de valores permitidos en el elemento.

```
<xsd:element name="genero" minOccurs="1" maxOccurs="1">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="Masculino"/>
   <xsd:enumeration value="Femenino"/>
   <xsd:enumeration value="Otros"/>
   <xsd:enumeration value=""/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```


Tipos de datos complejos

- Dentro de los tipos de datos complejos, podemos definir dos tipos de contenidos que pueden ir entre las etiquetas de apertura y cierre de los elementos:
  - Contenido simple (xsd:simpleContent): Cuando el elemento declarado sólo tienen contenido textual, sin elementos descendientes.
  - Contenido complejo (xsd:complexContent): Cuando el elemento declarado tiene elementos descendientes, pudiendo tener o no contenido textual.
- Las partículas permiten establecer las características de los elementos que van a utilizarse dentro de tipo de contenido complejo. Todas tienen atributos minOccurs y maxOccurs.

Partículas

<xsd:sequence>: Requiere que los elementos del grupo aparezcan en la secuencia especificada dentro del elemento que los contiene.

```
<xs:complexType name="Address">
 <xs:sequence>
  <xs:element name="street" type="xs:string"/>
  <xs:element name="city"   type="xs:string"/>
  <xs:element name="state"  type="xs:string"/>
  <xs:element name="zip"    type="xs:decimal"/>
 </xs:sequence>
</xs:complexType>
```

<xsd:choice>: Permite que uno y solo uno de los elementos contenidos en el grupo seleccionado esté presente en el elemento contenedor.

```
<xs:complexType name="chadState">
 <xs:choice minOccurs="1" maxOccurs="1">
  <xs:element ref="selected"/>
  <xs:element ref="unselected"/>
  <xs:element ref="dimpled"/>
 </xs:choice>
</xs:complexType>
```


Partículas

<xsd:all>: Permite que los elementos del grupo aparezcan o no en cualquier orden en el elemento contenedor.

```
<xs:complexType name="myComplexType">
 <xs:all>
  <xs:element ref="thing1"/>
  <xs:element ref="thing2"/>
  <xs:element ref="thing3"/>
 </xs:all>
</xs:complexType>
```

<xsd:any>: Permite que cualquier elemento de los espacios de nombres especificados aparezca en el elemento sequence o choice contenedor.

```
<xs:element name='htmlText'>
 <xs:complexType>
  <xs:sequence>
   <xs:any namespace='https://www.w3.org/1999/xhtml'
       minOccurs='1' maxOccurs='unbounded' processContents='lax'/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

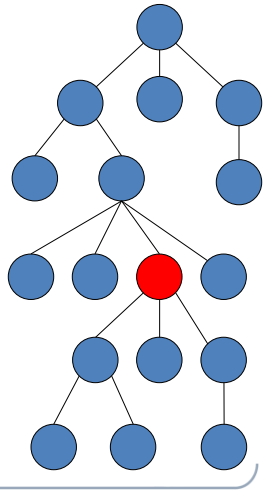
XPath

XPath

- XPath (Lenguaje de Caminos XML) es un lenguaje para direccionar partes de un documento XML, diseñado para ser utilizado por XSLT, XPointer, XQuery y otras tecnologías. Permite buscar dentro de un documento XML (caminos de localización) o comprobar si un componente cumple una condición (patrones).
- XPath opera sobre un documento XML considerándolo como un árbol de nodos. Hay siete tipos de nodo:
  - Raíz
    - Instrucción de procesamiento
    - Comentario
    - Elemento
      - Atributo
      - Texto (valor del elemento)
      - Espacio de nombres
- El nodo raíz es la raíz del árbol y corresponde con el elemento raíz del documento. El nodo raíz tiene también como hijos los nodos instrucción de procesamiento y comentario correspondientes a las instrucciones de procesamiento y comentarios que aparezcan en el prólogo y tras el fin del elemento de documento.

XPath

- Hay un nodo elemento por cada elemento en el documento. Si el elemento tiene atributos, cada atributo se convierte en un nodo atributo y tienen un valor de cadena. Si el elemento tiene espacio de nombres se convierte en un nodo espacio de nombres. El contenido del elemento se convierte en un nodo texto. Los nodos elementos se encuentra ordenados dentro de la jerarquía.
- Un nodo dentro de la jerarquía que va a ser usado como punto de partida, al que se aplica el camino, se denomina nodo contextual. Dicho nodo tiene un padre, nodo en el que está definido, y ancestros, su padre y el padre de su padre y así sucesivamente hasta el nodo raíz.
- Los nodos pueden tener hijos, nodos definidos directamente en el nodo, y descendientes, sus hijos y los hijos de sus hijos y así sucesivamente hasta el último nivel.
- Los nodos pueden tener hermanos, el resto de los hijos de su padre, en cuyo caso se encuentran ordenados. De tal forma que tienen hermanos que le preceden, a su izquierda en el árbol (definidos antes que el en el documento) y hermanos que le siguen, a su derecha en el árbol.



XPath

- Las construcciones sintácticas básicas en XPath son las expresiones. La evaluación de una expresión produce uno de los cuatro posibles resultados:
  - conjunto de nodos (una colección desordenada de nodos sin duplicados)
  - boolean (verdadero o falso)
  - número (un número en punto flotante)
  - cadena (una secuencia de caracteres UCS)
- Un camino de localización consiste en una secuencia de uno o más pasos de localización separados por /. La evaluación de expresiones tiene lugar respecto a un contexto: el camino de localización relativo parte del nodo actual y el camino de localización absoluto parte del nodo raíz.
- Un paso de localización tiene tres partes:
  1. un eje, que especifica la relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual,
  2. una prueba de nodo, que especifica el tipo de nodo y el nombre de los nodos seleccionados por el paso de localización, y
  3. cero o más predicados, que usan expresiones arbitrarias para refinar aún más el conjunto de nodos seleccionado por el paso de localización.
- La notación de un paso es:
  - Eje::prueba de nodo[predicados]

XPath

- Los posibles ejes son:
  - parent: el padre del nodo contextual, si lo hay
  - ancestor: los ancestros del nodo contextual; los ancestros del nodo contextual consisten en el padre del nodo contextual y el padre del padre, etc; así, el eje ancestor siempre incluirá al nodo raíz, salvo que el nodo contextual sea el nodo raíz
  - child: los hijos del nodo contextual
  - descendant: los descendientes del nodo contextual; un descendiente es un hijo o el hijo de un hijo, etc; de este modo el eje descendant nunca contiene nodos atributo o espacio de nombres
  - following: todos los nodos del mismo documento que el nodo contextual que están después de este según el orden del documento, excluyendo los descendientes y excluyendo nodos atributo y nodos espacio de nombres
  - preceding: todos los nodos del mismo documento que el nodo contextual que están antes de este según el orden del documento, excluyendo los ancestros y excluyendo nodos atributo y nodos espacio de nombres
  - following-sibling: todos los siguientes hermanos del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombres, el eje following-sibling está vacío
  - preceding-sibling: todos los hermanos precedentes del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombres, el eje preceding-sibling está vacío
  - attribute: los atributos del nodo contextual; el eje estará vacío a no ser que el nodo contextual sea un elemento
  - namespace: los nodos espacio de nombres del nodo contextual; el eje estará vacío a no ser que el nodo contextual sea un elemento
  - self: simplemente el propio nodo contextual
  - descendant-or-self: el nodo contextual y sus descendientes
  - ancestor-or-self: el nodo contextual y sus ancestros; así, el eje ancestor-or-self siempre incluirá el nodo raíz

XPath

- La prueba de nodo puede ser: el nombre exacto del nodo buscado, * para cualquier nodo del tipo principal de nodo, processing-instruction('literal') o el tipo de nodo a seleccionar:
  - comment() Comentario
  - text() Texto
  - node() Nodo
- Un predicado es una expresión (encerrada entre corchetes) que filtra un conjunto de nodos con respecto a un eje para producir un nuevo conjunto de nodos. La expresión debe dar un resultado lógico, que combina nombres de nodos, variables y funciones mediante el uso de operadores.

XPath: Selectores y abreviaturas

/	Separador de nivel
//	Separador descendente recursivo de niveles
.	Nivel actual
..	Nivel superior
*	Comodín
[ ]	Filtros e índices (Valor numérico: 1..n)
( )	Grupos
|	Unión
@	Prefijo selector de atributo
$	Prefijo de referencia a variable
‘	Delimitador de valor constante
“	Delimitador de valor constante
any	El primero
all	Todos

XPath: Operadores

and	Y lógico
or	O lógico
not( )	Negación
=	Igual
!=	Distinto
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
+	Suma
-	Resta
*	Multiplicación
div	División en coma flotante
mod	Resto entero de la división

XPath: Funciones

- Cada función en la biblioteca se especifica utilizando un prototipo de función, que da el tipo devuelto, el nombre de la función y el tipo de los argumentos. Si un tipo de argumento es seguido por un signo de interrogación, entonces el argumento es opcional; en otro caso, el argumento es obligatorio.
- Funciones de conjuntos de nodos
  - Number last(): devuelve un número igual al tamaño contextual del contexto de evaluación de la expresión.
  - Number position(): devuelve un número igual a la posición contextual del contexto de evaluación de la expresión.
  - Number count(node-set): devuelve el número de nodos en el conjunto de nodos argumento.
  - Node-set id(object): selecciona elementos mediante su identificador.
  - String local-name(node-set?): devuelve la parte local del nombre expandido del nodo, en el conjunto de nodos argumento, que es el primero en orden de documento.
  - String namespace-uri(node-set?): devuelve el URI del espacio de nombres del nombre expandido del nodo, en el conjunto de nodos argumento, que es el primero en orden de documento.
  - String name(node-set ?): devuelve una cadena conteniendo un QName que representa el nombre expandido del nodo, en el conjunto de nodos argumento, que es el primero en orden de documento.

XPath: Funciones

- Funciones de cadenas
  - String string(object?): convierte un objeto en cadena.
  - String concat(string, string, string*): devuelve la concatenación de sus argumentos.
  - Boolean starts-with(string, string): devuelve verdadero si la primera cadena argumento empieza con la segunda cadena argumento, y devuelve falso en otro caso.
  - Boolean contains(string, string): devuelve verdadero si la primera cadena argumento contiene a la segunda cadena argumento, y devuelve falso en otro caso.
  - String substring-before(string, string): devuelve la subcadena de la primera cadena argumento que precede a la primera aparición de la segunda cadena argumento en la primera cadena argumento, o la cadena vacía si la primera cadena argumento no contiene a la segunda cadena argumento
  - String substring-after(string, string): devuelve la subcadena de la primera cadena argumento que sigue a la primera aparición de la segunda cadena argumento en la primera cadena argumento, o la cadena vacía si la primera cadena argumento no contiene a la segunda cadena argumento.
  - String substring(string, number, number?): devuelve la subcadena del primer argumento que comienza en la posición especificada en el segundo argumento y tiene la longitud especificada en el tercer argumento. Si no se especifica el tercer argumento, devuelve la subcadena que comienza en la posición especificada en el segundo argumento y continúa hasta el final de la cadena.
  - Number string-length(string?): devuelve el número de caracteres en la cadena. Si se omite el argumento, toma por defecto el nodo contextual convertido en cadena, es decir, el valor de cadena del nodo contextual.
  - String normalize-space(string?): devuelve la cadena argumento con el espacio en blanco normalizado mediante la eliminación del que se encuentra al principio y al final y la substitución de secuencias de caracteres de espacio en blanco por un solo espacio.
  - String translate(string, string, string): devuelve la cadena primer argumento con las apariciones de caracteres del segundo argumento substituidas por los caracteres en las posiciones correspondientes de la tercera cadena argumento.

XPath

- Funciones Booleanas
  - Boolean boolean(object): convierte su argumento en booleano.
  - Boolean not(boolean): devuelve verdadero si su argumento es falso, y falso en otro caso.
  - Bolean true(): devuelve verdadero.
  - Bolean false(): devuelve falso.
  - Boolean lang(string): devuelve verdadero o falso dependiendo de si el lenguaje del nodo contextual tal como se especifica por los atributos xml:lang es el mismo que, o es un sublenguaje de, el lenguaje especificado por la cadena argumento.
- Funciones numéricas
  - Number number(object?): convierte su argumento en un número.
  - Number sum(node-set): devuelve la suma, a lo largo de todos los nodos del conjunto de nodos argumento, del resultado de convertir los valores de cadena de los nodos en números.
  - Number floor(number): devuelve el mayor (más cercano al infinito positivo) número que no sea mayor que el argumento y que sea entero.
  - Number ceiling(number): devuelve el menor (más cercano al infinito negativo) número que no sea menor que el argumento y que sea entero.
  - Number round(number): devuelve el número que esté más próximo al argumento y que sea entero.

XPath: Ejemplos

- **Selecciona el elemento raíz AAA**
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
/AAA
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- **Selecciona todos los BBB que son hijos del elemento AAA**
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
/AAA/BBB
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- **Selecciona todos los BBB que son hijos de DDD**
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//DDD/BBB
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- **Selecciona el segundo hijo de AAA y el último hijo BBB de DDD**
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
/AAA/*[2] | //DDD/BBB[last()]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- **Selecciona todos los atributos 'A'**
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//@A
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- **Selecciona los BBB que contienen un atributo 'A'**
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//BBB[@A]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- **Selecciona los BBB que contienen un atributo 'nombre'**
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//BBB[@nombre]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- **Selecciona los BBB que no contienen atributos**
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//BBB[not(@*)]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- **Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'**
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//*[@A='id2']
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- **Selecciona elementos con dos hijos**
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//*[count(*)=2]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- **Selecciona todos los ancestros de E1234**
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//E1234/ancestor::*
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- **Selecciona los elementos cuyo nombre se inicie con la letra C**
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//*[starts-with(name(),'C')]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- **Selecciona elementos cuyo nombre contenga mas de tres caracteres**
- Selecciona los elementos pares
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//*[string-length(name()) > 3]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- **Selecciona los elementos pares**
- Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'

```
//*[position() mod 2 = 0 ]
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```


XPath: Ejemplos

- Selecciona el elemento raíz AAA
- Selecciona todos los BBB que son hijos del elemento AAA
- Selecciona todos los BBB que son hijos de DDD
- Selecciona el segundo hijo de AAA y el último hijo BBB de DDD
- Selecciona todos los atributos 'A'
- Selecciona los BBB que contienen un atributo 'A'
- Selecciona los BBB que contienen un atributo 'nombre'
- Selecciona los BBB que no contienen atributos
- Selecciona los elementos cuyo atributo 'A' tiene por valor 'id2'
- Selecciona elementos con dos hijos
- Selecciona todos los ancestros de E1234
- Selecciona los elementos cuyo nombre se inicie con la letra C
- Selecciona elementos cuyo nombre contenga mas de tres caracteres
- Selecciona los elementos pares
- **Selecciona los elementos que contengan un elemento cuyo contenido sea 'XY'**

```
//*[text()='XY']/parent::*
```

```
<AAA>
    <BBB A = "id1"/>
    <CCC/>
    <BBB nombre = "xxx"/>
    <DDD>
        <BBB A = "id2"/>
        <BBB/>
    </DDD>
    <EEE>
        <DDD A = "id2">
            <BBB/>
            <BBB/>
            <E1234/>
        </DDD>
        <C123>XY</C123>
    </EEE>
</AAA>
```

JavaScript Object Notation
http://tools.ietf.org/html/rfc4627

JSON

Introducción

- JSON (JavaScript Object Notation) es un formato sencillo para el intercambio de información.
- El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- La notación de objetos mediante JSON es una de las características principales de JavaScript y es un mecanismo definido en los fundamentos básicos del lenguaje.
- En los últimos años, JSON se ha convertido en una alternativa al formato XML, ya que es más fácil de leer y escribir, además de ser mucho más conciso.
- No obstante, XML es superior técnicamente porque es un lenguaje de marcado, mientras que JSON es simplemente un formato para intercambiar datos.
- La especificación completa del JSON es la RFC 4627, su tipo MIME oficial es application/json y la extensión recomendada es .json.

Estructuras

- JSON está constituido por dos estructuras:
  - Una colección de pares de nombre/valor. En los lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
  - Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como tablas, arreglos, vectores, listas o secuencias.
- Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Sintaxis

- Un array es un conjunto de valores separados por comas (,) que se encierran entre corchetes […]
- Un objeto es un conjunto de pares nombre:valor separados por comas (,) que se acotan entre llaves { … }
- Los nombre son cadenas, entre comillas dobles (").
- El separador entre el nombre y el valor son los dos puntos (:)
- El valor debe ser un objeto, un array, un número, una cadena o uno de los tres nombres literales siguientes (en minúsculas):
  - true, false o null
- Se codifica en Unicode, la codificación predeterminada es UTF-8.

Valores numéricos

- La representación de números es similar a la utilizada en la mayoría de los lenguajes de programación.
- Un número contiene una parte entera que puede ser prefijada con un signo menos opcional, que puede ser seguida por una parte fraccionaria y / o una parte exponencial.
- La parte fraccionaria comienza con un punto (como separador decimal) seguido de uno o más dígitos.
- La parte exponencial comienza con la letra E en mayúsculas o minúsculas, lo que puede ser seguido por un signo más o menos, y son seguidas por uno o más dígitos.
- Los formatos octales y hexadecimales no están permitidos. Los ceros iniciales no están permitidos.
- No se permiten valores numéricos que no se puedan representar como secuencias de dígitos (como infinito y NaN).

Valores cadena

- La representación de las cadenas es similar a las convenciones utilizadas en la familia C de lenguajes de programación.
- Una cadena comienza y termina con comillas (").
- Se pueden utilizar todos los caracteres Unicode dentro de las comillas con excepción de los caracteres que se deben escapar: los caracteres de control (U + 0000 a U + 001F) y los caracteres con significado.
- Cuando un carácter se encuentra fuera del plano multilingüe básico (U + 0000 a U + FFFF), puede ser representado por su correspondiente valor hexadecimal. Las letras hexadecimales A-F puede ir en mayúsculas o en minúsculas.
- Secuencias de escape:
  - \\, \/, \", \n, \r, \b, \f, \t
  - \u[0-9A-Fa-f]{4}

Objeto con anidamientos

```
{
      "Image": {
          "Width":  800,
          "Height": 600,
          "Title":  "View from 15th Floor",
          "Thumbnail": {
              "Url":    "/image/481989943",
              "Height": 125,
              "Width":  "100"
              },
          "IDs": [116, 943, 234, 38793]
          }
  }
```


Array de objetos

```
[
    {
       "precision": "zip",
       "Latitude":  37.7668,
       "Longitude": -122.3959,
       "City":      "SAN FRANCISCO",
       "State":     "CA",
       "Zip":       "94107"
    },
    {
       "precision": "zip",
       "Latitude":  37.371991,
       "Longitude": -122.026020,
       "City":      "SUNNYVALE",
       "State":     "CA",
       "Zip":       "94085"
    }
]
```

JSON en JavaScript

- El Standard Built-in ECMAScript Objects define que todo interprete de JavaScript debe contar con un objeto JSON como miembro del objeto Global.
- El objeto debe contener, al menos, los siguientes miembros:
  - **JSON.parse** (Función): Convierte una cadena de la notación de objetos de JavaScript (JSON) en un objeto de JavaScript.
  - **JSON.stringify** (Función): Convierte un valor de JavaScript en una cadena de la notación de objetos JavaScript (JSON).

https://ietf-wg-jsonpath.github.io/draft-ietf-jsonpath-base/draft-ietf-jsonpath-base.html

JSONPath

JSONPath

- JSONPath es una forma estandarizada para consultar elementos de un objeto JSON. JSONPath utiliza expresiones de ruta similares a XPath para desplazarse por elementos, elementos anidados y matrices en un documento JSON.
- JSONPath solo cubre las partes esenciales de XPath 1.0.
- Las expresiones JSONPath siempre se refieren a una estructura JSON de la misma manera que las expresiones XPath se usan en combinación con un documento XML. Dado que una estructura JSON suele ser anónima y no necesariamente tiene un "objeto miembro raíz", JSONPath asume el nombre abstracto $ asignado al objeto o matriz de nivel externo.
- Las expresiones JSONPath pueden usar la notación de puntos
  - $.store.book[0].title
- o la notación corchetes
  - $['store']['book'][0]['title']

JSONPath

- JSONPath permite el símbolo comodín * para nombres de miembros e índices de matriz.
  - $.store.*
- Toma prestado el operador descendiente '..' de E4X y la sintaxis de recorte de matriz [start:end:step] de ECMASCRIPT 4.
  - $..author
  - $..book[0:]
- Las expresiones JSONPath distinguen entre mayúsculas y minúsculas, incluidos los nombres y valores de propiedades. Los literales cadena deben estar entre comillas simples.
- Las expresiones del lenguaje de script subyacente (<expr>) se pueden usar como una alternativa a los nombres o índices explícitos, usando el símbolo '@' para el objeto actual.
  - $.store.book[(@.length-1)].title
- Las expresiones de filtro son compatibles a través de la sintaxis ?(<boolean expr>). No todos los filtros están disponibles en todos los motores.
  - $.store.book[?(@.price < 10)].title

Expresiones JSONPath

Expresión	Descripción
$	El objeto raíz o matriz.
.property	Selecciona la propiedad especificada en un objeto principal.
..property	Descenso recursivo: busca el nombre de propiedad especificado de forma recursiva y devuelve una matriz de todos los valores con este nombre de propiedad. Siempre devuelve una lista , incluso si solo se encuentra una propiedad.
*	Comodín selecciona todos los elementos de una matriz o las propiedades de un objeto, independientemente de sus nombres o índices.
['property']	Selecciona la propiedad especificada en un objeto principal. Asegúrese de poner comillas simples alrededor del nombre de la propiedad.
[n]	Selecciona el n-ésimo elemento de una matriz. Los índices están basados en 0.

Expresiones JSONPath

Expresión	Descripción
[index1, index2,…]	Selecciona elementos de matriz con los índices especificados. Devuelve una lista.
[start:end] [start:] [:end]	Selecciona elementos de matriz desde el índice de inicio y hasta el índice final, pero sin incluirlo. Si se omite end, selecciona todos los elementos desde start hasta el final de la matriz. Si se omite start, selecciona los primeros elementos de la matriz hasta end.
[-n:]	Selecciona los últimos n elementos de la matriz. Devuelve una lista .
[(expression)]	Se pueden usar expresiones de script en lugar de nombres o índices de propiedad explícitos.
[?(expression)]	Expresión de filtro. Selecciona todos los elementos de un objeto o matriz que coincidan con el filtro especificado. Devuelve una lista.
@	Se utiliza en expresiones de filtro para hacer referencia al nodo actual que se está procesando.

Filtros JSONPath

Operador	Descripción
==	Igual a.
!=	No igual a.
>	Mas grande que.
>=	Mayor qué o igual a.
<	Menos que.
<=	Menos que o igual a.
=~	Coincide con una expresión regular de JavaScript: [?(@.description =~ /cat.*/i)] (con /i no distingue entre mayúsculas y minúsculas)
!	Se utiliza para negar un filtro: [?(!@.isbn)]
&&	AND lógico, utilizado para combinar varias expresiones de filtro:
||	OR lógico, utilizado para combinar varias expresiones de filtro:

Filtros JSONPath

Operador	Descripción
in	Comprueba si el valor del lado izquierdo está presente en la lista del lado derecho: [?(@.size in ['M', 'L'])][?('S' in @.sizes)]
nin	Contrario de in. Comprueba que el valor del lado izquierdo no está presente en la lista del lado derecho.
subsetof	Comprueba si la matriz del lado izquierdo es un subconjunto de la matriz del lado derecho. El orden real de los elementos de la matriz no importa. Una matriz vacía del lado izquierdo siempre coincide.
contains	Comprueba si una cadena contiene la subcadena especificada (distingue entre mayúsculas y minúsculas) o si una matriz contiene el elemento especificado.
size	Comprueba si una matriz o cadena tiene la longitud especificada.
empty true	Coincide con una matriz o cadena vacía.
empty false	Coincide con una matriz o cadena no vacía.

XPath vs JSONPath

XPath	JSONPath	Descripción
/	$	el objeto / elemento raíz
.	@	el objeto / elemento actual
/	. or []	operador hijo
..	n/a	operador padre
//	..	descenso recursivo
*	*	comodín. Todos los objetos / elementos independientemente de sus nombres.
@	n/a	acceso al atributo. Las estructuras JSON no tienen atributos.
[]	[]	operador de subíndice. XPath lo usa para iterar sobre colecciones de elementos y para predicados. En Javascript y JSON es el operador de matriz nativo.
|	[,]	el operador de unión en XPath da como resultado una combinación de conjuntos de nodos. JSONPath permite nombres alternativos o índices de matriz como un conjunto.
n/a	[start:end:step]	operador de recorte de matriz.
[]	?()	aplica una expresión de filtro
n/a	()	expresión de script, utilizando el motor de script subyacente.

XPath vs JSONPath

XPath	JSONPath	Resultado
/store/book/author	$.store.book[*].author	los autores de todos los libros en la tienda
//author	$..author	todos los autores
/store/*	$.store.*	todas las cosas en la tienda
/store//price	$.store..price	el precio de todo en la tienda
//book[3]	$..book[2]	el tercer libro
//book[last()]	$..book[(@.length-1)] $..book[-1:]	el último libro en orden.
//book[position()<3]	$..book[0,1] $..book[:2]	los dos primeros libros
//book[isbn]	$..book[?(@.isbn)]	filtrar todos los libros con número isbn
//book[price<10]	$..book[?(@.price<10)]	filtrar todos los libros más baratos que 10
//*	$..*	todos los elementos en el documento XML o todos los miembros de la estructura JSON.