

CONTORNOS DE DESENVOLVEMENTO

TEMA 1: DESENVOLVEMENTO DE SOFTWARE



XUNTA DE GALICIA
CONSELLERÍA DE EDUCACIÓN
E ORDENACIÓN UNIVERSITARIA



Unión Europea
Fondo Social Europeo

Índice

1.	O software e os proxectos de desenvolvemento de software.....	3
1.1	Software	3
1.2	Hardware e Software	7
1.3	Clasificación de software	7
1.4	Desenvolvemento de software.....	8
1.5	Paradigma de ciclo de vida clásico ou modelo en cascada	9
1.6	Modelo en espiral	14
1.7	Programación eXtrema.....	15
1.8	Métrica v.3.....	16
1.9	Metodoloxías áxiles : Scrum	18
1.10	Licencias de software	29
2.	Linguaxes de programación e ferramentas de desenvolvemento	30
2.1	Clasificación das linguaxes informáticas.....	30
2.2	Clasificación das linguaxes de programación	33
2.3	Linguaxes máis difundidas	39
2.4	Proceso de xeración de código.....	44
2.5	Frameworks.....	47

Traballo derivado por: Patricia González Pardo

Con Licenza Creative Commons BY-NC-SA (recoñecemento - non comercial - compartir igual) a partir de material propio e dos documentos orixinais:

© Xunta de Galicia. Consellería de Cultura, Educación e Ordenación Universitaria.

Autores: María del Carmen Fernández Lameiro

Licenza Creative Commons BY-NC-SA (recoñecemento - non comercial - compartir igual).

Para ver unha copia desta licenza, visitar a ligazón <http://creativecommons.org/licenses/by-nc-sa/3.0/es/>.

1. O software e os proxectos de desenvolvemento de software

1.1 Software

A Real Academia Galega define a informática como a "Ciencia do tratamento automático da información por medio de máquinas electrónicas". Este tratamento necesita de:

- Soporte físico ou hardware formado por todos os compoñentes electrónicos tanxibles involucrados no tratamento. Segundo a Real Academia Galega é máis aconsellable dicir soporte físico que hardware e defíneo como a maquinaria ou elementos que compoñen un ordenador.
- Soporte lóxico ou software que fai que o hardware funcione e está formado por todos os compoñentes intanxibles involucrados no tratamento: programas, datos e documentación. Segundo a Real Academia Galega é máis aconsellable dicir soporte lóxico que software e defíneo como o conxunto de ordes e programas que permiten utilizar un ordenador.
- Equipamento humano ou persoal informático que manexa o equipamento físico e o lóxico para que todo o tratamento se leve a cabo.



O concepto de programa informático está moi ligado ao concepto de algoritmo. Un algoritmo é un conxunto de instrucións ou regras ben definidas ordenadas y finitas que permite resolver un problema.

Algoritmo para cambiar a roda dun coche.

Datos: roda pinchada e localización do gato, da roda de reposto y da chave inglesa

PASO 1. Afrouxar os parafusos da roda pinchada coa chave inglesa

PASO 2. Colocar o gato mecánico no seu sitio

PASO 3. Levantar o gato ata que a roda pinchada poida xirar libremente

PASO 4. Quitar los parafusos

PASO 5. Quitar a roda pinchada

PASO 6. Poñer roda de reposto

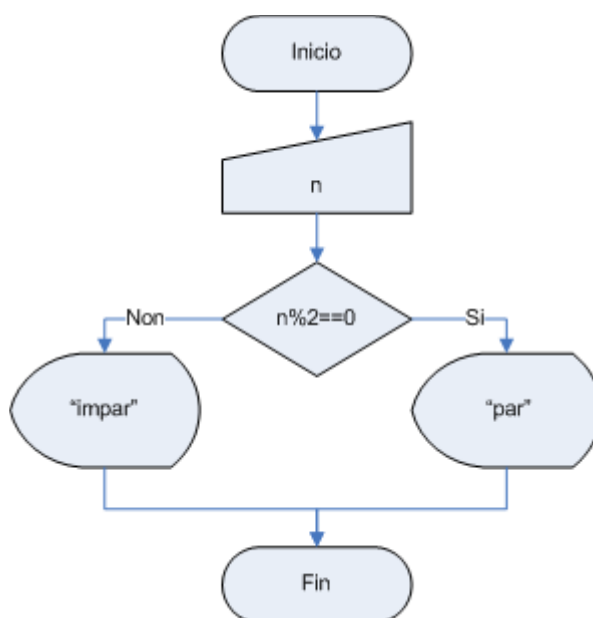
PASO 7. Poñer os parafusos e apertalos lixeiramente

PASO 8. Baixar o gato ata que se poida liberar

PASO 9. Sacar o gato do seu sitio

PASO 10. Apertar os parafusos coa chave inglesa

Exemplo de algoritmo representado mediante un diagrama de fluxo que permite ver se un número tecleado é par ou impar e que considera o 0 como número par:



O obxectivo final é **Resolver un problema**. Como resólvelo? Empregando un algoritmo (as instrucións) e os **datos** dese problema.

Os ordenadores non entenden a linguaxe natural: Como lle decimos ao ordenador o que ten que facer?: Escribindo un programa nunha linguaxe de programación determinado, para implementar o *algoritmo*.

O algoritmo pode escribirse nunha linguaxe de programación utilizando algunha ferramenta de edición, dando lugar a un código que deberá de gravarse nunha memoria externa non volátil para que perdure no tempo. Exemplo de código escrito en linguaxe Java que se corresponde co anterior algoritmo:

```
package parimpar;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int n;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Teclee un número inteiro:");
        n = teclado.nextInt();
        if(n%2==0){
            System.out.println(n + " é par");
        }
    }
}
```

```

    }
    else{
        System.out.println (n + " é impar");
    }
}
}

```

O código terá que sufrir algunhas transformacións para que finalmente se obteña un programa que poida ser executado nun ordenador. Para a execución é preciso que o programa estea almacenando na memoria interna e volátil do ordenador; así o procesador pode ir collendo cada orde que forma o programa, resolvéndoa e controlando a súa execución. Se é preciso, accede á memoria interna para manipular variables, ou aos periféricos de entrada, saída ou almacenamento, fai cálculos ou resolve expresións lóxicas ata que finaliza con éxito a última instrución ou se atopa cun erro irresoluble.

Na execución do programa correspondente ao código exemplo anterior, o procesador necesitaría un número enteiro subministrado a través do teclado (dispositivo ou periférico de entrada), obtería o resultado dunha operación aritmética (resto da división enteira de n entre 2), resolvería unha expresión lóxica (descubrir se o resto anterior é 0), e executaría a instrución alternativa para que no caso de que o resto sexa 0 saia pola pantalla (dispositivo ou periférico de saída) que n é par ou que é impar. Exemplo de execución do código anterior e aspecto do resultado da execución na pantalla de texto cando se teclea o número 11:

```

--- exec-maven-plugin:1.5.0:exec (default-cli) @ ejercicios ---
Teclee un número enteiro:11
11 é impar
-----
BUILD SUCCESS
-----
Total time: 6.113 s
Finished at: 2020-07-29T10:33:00+02:00
Final Memory: 7M/27M
-----

```

Os programas informáticos son imprescindibles para que os ordenadores funcionen e son conxuntos de instrucións que unha vez executadas realizarán unha ou varias tarefas. O software é un conxunto de programas e neste concepto inclúense tamén os datos que se manexan e a documentación deses programas.

Pseudocódigo e organigramas

Antes da programación do algoritmo, este pódese representar de forma máis “natural” para entendela ben e que o proceso de programación sexa logo máis sinxelo e sen erros. Para iso podemos empregar *pseudocódigo* ou organigramas.

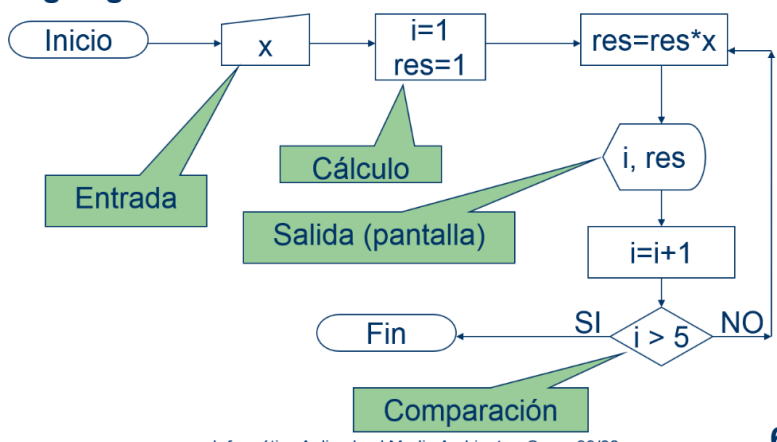
O pseudocódigo é una descrición verbal, en linguaxe case natural, que mostra cada paso a paso o proceso a seguir.

Ejemplo: programa para escribir 5 primeras potencias de un número (pseudocódigo):

```
1 programa Potencias;  
2 leer(x)  
3 i = 1; res = 1;  
4 res = res*x;  
5 escribir(x " elevado a " i " es " res);  
6 i = i + 1  
7 si (i > 5) entonces  
    terminar  
8 ir al paso 4  
9 fin.
```

Os organigramas son unha descrición gráfica do problema, no que cada símbolo representa unha acción (entradas, saídas, comparacións, etc.

Organigramas. Símbolos



PSeInt é unha ferramenta gratuíta que permite escribir pseudocódigo sinxelo ou debuxar organigramas, sendo unha forma sinxela de acercarnos á programación. Entre as súas principais funcións cabe destacar:

- Edición en forma de pseudocódigo
- Edición en forma de organigrama
- Execución do código escrito
- Exportación a distintas linguaxes como Java, C, etc.

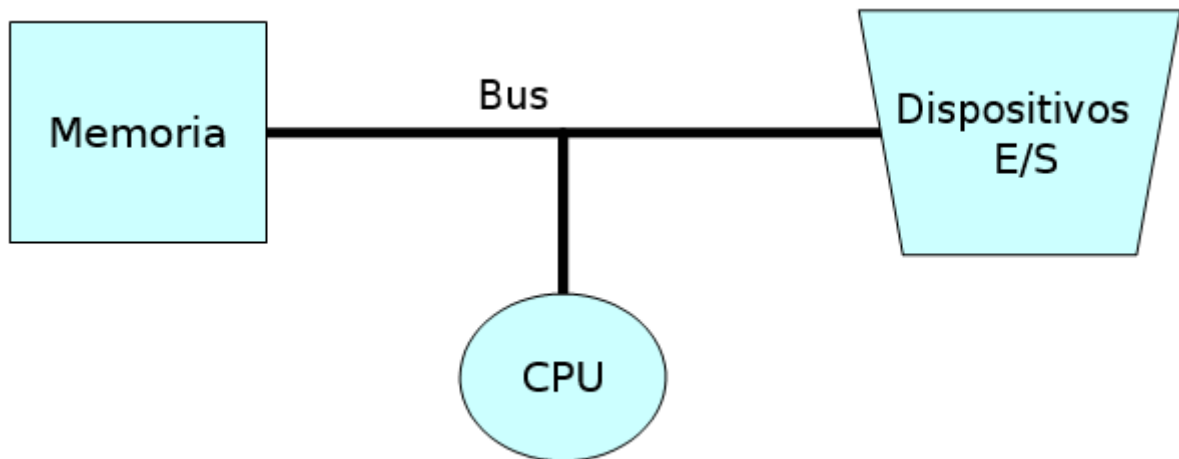


Tarefa 1.1. Traballando con PSeInt.

- Descarga PSeInt para Windows na súa versión portable. Descomprimeo e execútao (wxPSeInt.exe).
- Abre o arquivo PasarEuroADolar.psc e comenta o que fai cada instrución.
- Abre o arquivo NumeroMayor.psc e comenta o que fai cada instrución.
- Abre o arquivo adivinarNumero.psc e comenta o que fai cada instrución.
- Crea un pseudocódigo chamado PasarDolarAEuro.psc que faga o inverso ao do apartado b)
- Crea un diagrama de fluxo (Archivo > Editar diagrama de flujo) dun algoritmo ao que se lle introducen dous números e informa cal dos dous é maior ou se son iguais.
- Crea un arquivo .jpg co diagrama de fluxo do algoritmo de adivinarNumero.psc.
- Exporta os algoritmos dos apartados previos a distintas linguaxes (C, Java, JS, Python) e fai un pequeno resumo sobre as diferenzas máis visibles (por exemplo como é una asignación de variable, como se mostran por pantalla os resultados, que tipos de datos ten a linguaxe, etc.)

1.2 Hardware e Software

El 99% dos ordenadores actuais teñen a seguinte arquitectura:



Datos e instrucións comparten memoria: Denominada arquitectura de Von Neumann, aínda que foi proposta inicialmente por Eckert e Mauchly.

Compoñentes

- Unidade central de proceso (CPU): Encárgase fundamentalmente de executar as instrucións e coordinar o resto de elementos
- Memoria: Almacena los datos, as instrucións e os resultados. Clasificación:
 - Permanente/volátil: A información gardada na memoria permanente (por exemplo un disco duro) mantense una vez que se apaga o ordenador, na volátil non (por exemplo na RAM)
 - Principal/secundaria: a principal é na que se executan os procesos (RAM), na secundaria gardamos datos e programas (Discos). A primeira é volátil, e a segunda é permanente. A memoria principal é máis pequena, máis rápida e máis cara que a secundaria.
- Dispositivos de entrada/saída: Para proporcionar os datos e instrucións e recibir os resultados
- Bus de datos: Para compartir a información entre os compoñentes anteriores

1.3 Clasificación de software

Pódese clasificar o software en dous tipos:

- **Software de sistema.** Controla e xestiona o hardware facendo que funcione, ocultando ao persoal informático os procesos internos deste funcionamento. Exemplos deste tipo de software son:
 - Sistemas operativos
 - Controladores de dispositivos

- Ferramentas de diagnóstico
 - Servidores (correo, web, DNS,...)
 - Utilidades do sistema (compresión de información, rastreo de zoas defectuosas en soportes.)
 - **Software de aplicación.** Permite levar a cabo unha ou varias tarefas específicas en calquera campo de actividade dende que está instalado o software básico de sistema. Exemplos deste tipo de software son:
 - Aplicacións para control de sistemas e automatización industrial
 - Aplicacións ofimáticas
 - Software educativo
 - Software empresarial: contabilidade, nóminas, almacén,...
 - Bases de datos
 - Videoxogos
 - Software de comunicacións: navegadores web, clientes de correo,...
 - Software médico
 - Software de cálculo numérico
 - Software de deseño asistido por ordenador
 - **Software de programación** que é o conxunto de ferramentas que permiten ao programador desenvolver programas informáticos como por exemplo:
 - Editores de texto
 - Compiladores
 - Intérpretes
 - Enlazadores
 - Depuradores
 - Contornos de desenvolvemento integrado (IDE) que agrupa as anteriores ferramentas nun mesmo contorno para facilitar ao programador a tarefa de desenvolver programas informáticos e que teñen unha avanzada interface gráfica de usuario (GUI).
- Dentro do software de aplicación pódese facer unha división entre:
- Software horizontal ou xenérico que se poden utilizar en diversos contornos como por exemplo as aplicacións ofimáticas.
 - Software vertical ou a medida que só se poden utilizar nun contorno específico como por exemplo a contabilidade feita a medida para unha empresa en concreto.



Tarefa 1.2. Buscar en internet nomes comerciais de cada un do software de sistema e software de aplicación mencionados. Fai una táboa: Tipo de software, nome comercial, tipo de licencia.

1.4 Desenvolvemento de software

O proceso de desenvolvemento de software é moi diferente dependendo da complexidade do software. Por exemplo, para desenvolver un sistema operativo é necesario un equipo disciplinado, recursos, ferramentas, un proxecto a seguir e alguén que xestione todo. No extremo oposto, para

facen un programa que visualice se un número enteiro que se teclea é par ou impar, só é necesario un programador ou un afeccionado á programación e un algoritmo de resolución.

Fritz Bauer nunha reunión do Comité Científico da OTAN en 1969 propuxo a primeira definición de Enxeñaría de software como o establecemento e uso de principios de enxeñaría robustos, orientados a obter economicamente software que sexa fiable e funcione eficientemente sobre máquinas reais. Posteriormente, moitas personalidades destacadas do mundo do software matizaron esta definición que se normalizou na norma IEEE 1993 que define a Enxeñaría de software como a aplicación dun enfoque sistemático, disciplinado e medible ao desenvolvemento, funcionamento e mantemento do software.

Durante décadas os enxeñeiros e enxeñeiras de software foron desenvolvendo e mellorando paradigmas (métodos, ferramentas e procedementos para describir un modelo) que foran sistemáticos, predicibles e repetibles e así mellorar a produtividade e calidade do software.

A Enxeñaría de software é imprescindible en grandes proxectos de software, debe de ser aplicada en proxectos de tamaño medio e recoméndase aplicar algún dos seus procesos en proxectos pequenos.

Existen moitos modelos a seguir para desenvolver software. O máis clásico é o modelo en cascada. Destacan entre todos o modelo en espiral baseado en prototipos, a programación extrema como representativo dos métodos de programación áxil, e a Métrica 3 como modelo a aplicar en grandes proxectos relacionados coas institucións públicas.

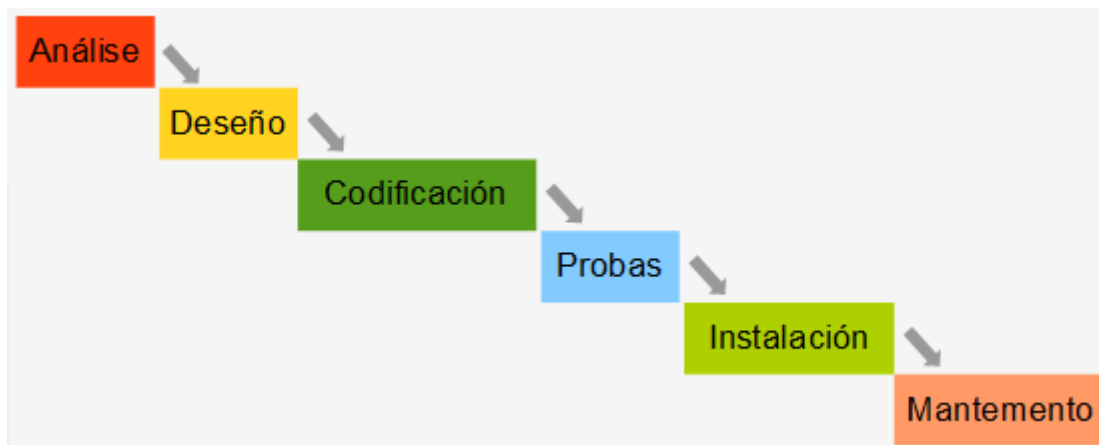
1.5 Paradigma de ciclo de vida clásico ou modelo en cascada

A Real academia da lingua define Paradigma como “Teoría ou conxunto de teorías cuxo núcleo central acéptase sen cuestionar e que subministra a base e o modelo para resolver problemas e avanzar no coñecemento”

Para a Enxeñaría de Software o paradigma é unha agrupación de métodos, ferramentas e procedementos co fin de describir un modelo. Cada metodoloxía de desenvolvemento de software ten mais ou menos o seu propio enfoque para o desenvolvemento de software.

O paradigma de ciclo de vida clásico do software, tamén chamado modelo en cascada consta das fases: análise, deseño, codificación, probas, instalación e mantemento e a documentación é un aspecto implícito en todas as fases. Algúns autores nomean estas fases con nomes lixeiramente diferentes, ou poden agrupar algunhas para crear unha nova fase ou nomear unha fase con máis detalle, pero en esencia as fases son as mesmas. Algunhas destas fases tamén se utilizan noutros modelos con lixeiras variantes.

As fases vanse realizando de forma secuencial utilizando a información obtida ao finalizar unha fase para empezar a fase seguinte. Deste xeito o cliente non pode ver o software funcionando ata as últimas fases e daría moito traballo corrixir un erro que se detecta nas últimas fases pero que afecta ás primeiras.



Análise

Nesta fase o analista captura, analiza e especifica os requisitos que debe cumprir o software. Debe obter a información do cliente ou dos usuarios do software mediante entrevistas planificadas e habilmente estruturadas nunha linguaxe comprensible para o usuario. O resultado desta captura de información depende basicamente da habilidade e experiencia do analista aínda que poida utilizar guías ou software específico.

Ao finalizar esta fase debe existir o documento de especificación de requisitos do software (ERS), no que estarán detallados os requisitos que ten que cumprir o software, debe valorarse o custo do proxecto e planificarse a duración do mesmo. Toda esta información ten que comunicarse ao cliente para a súa aceptación.

A linguaxe utilizada para describir os ERP (Enterprise Resource Planning, Planificación de Recursos Empresariais: son sistemas informáticos destinados a la administración de recursos en una organización) pode ser descritiva ou máis formal e rigorosa utilizando casos de usos na linguaxe de modelado UML. O estándar IEEE 830-1998 recolle unha norma de prácticas recomendadas para a especificación de requisitos de software.

Diseño

Nesta fase o deseñador deberá de descompoñer e organizar todo o sistema software en partes que podan elaborarse por separado para así aproveitar as vantaxes do desenvolvemento de software en equipo.

O resultado desta fase plásmase no documento de deseño de software (SDD= Software Design Description) que contén a estrutura global do sistema, a especificación do que debe facer cada unha das partes e a maneira de combinarse entre elas e é a guía que os programadores e probadores de software deberán ler, entender e seguir. Este documento incluírá o deseño lóxico de datos, o deseño da arquitectura e estrutura, o deseño dos procedementos, a organización do código fonte e a compilación, e o da interface entre o home e o software. Exemplos de diagramas que indican como se construírá o software poden ser: modelo E/R para os datos, diagramas UML de clases, diagramas UML de despregue e diagramas UML de secuencia.

Nesta fase debe tratarase a seguridade do proxecto mediante unha análise de riscos (recompilación de recursos que deben ser protexidos, identificación de actores e roles posibles, recompilación de requisitos legais e de negocio como encriptacións ou certificacións a cumprir, etcétera) e a relación de actividades que mitigan eses riscos.

Codificación

Esta fase tamén se chama fase de programación ou implementación¹. Nela o programador transforma o deseño lóxico da fase anterior a código na linguaxe de programación elixida, de tal forma que os programas resultantes cumpran os requisitos da análise e poidan ser executado nunha máquina.

Mentres dura esta fase, poden realizarse tarefas de depuración do código ou revisión inicial do mesmo para detectar erros sintácticos, semánticos e de lóxica.

Probas

Esta fase permite aplicar métodos ou técnicas ao código para determinar que tódalas sentencias foron probadas e funcionan correctamente.

As probas teñen que planificarse, deseñarse, executarse e avaliar os resultados. Considérase que unha proba é boa se detecta erros non detectados anteriormente. As probas realizadas inmediatamente despois da codificación poden ser:

- Probas unitarias cando permiten realizar tests a anacos pequenos de código cunha funcionalidade específica.
- Probas de integración cando permiten realizar tests a un grupo de anacos de código que superaron con éxito os correspondentes tests unitarios e que interactúan entre eles.

Outras probas sobre o sistema total poden ser:

- Probas de validación ou aceptación para comprobar que o sistema cumpre os requisitos do software especificados no ERS.
- Probas de recuperación para comprobar como reacciona o sistema fronte a un fallo xeral e como se recupera do mesmo.
- Probas de seguridade para comprobar que os datos están protexidos fronte a agresións externas.
- Probas de resistencia para comprobar como responde o sistema a intentos de bloqueo ou colapso.
- Probas de rendemento para someter ao sistema a demandas do usuario extremas e comprobar o tempo de resposta do sistema.

As probas deberán de ser realizadas en primeiro lugar polos creadores do software pero é recomendable que tamén sexan realizadas por especialistas que non participaron na creación e finalmente sexan realizadas por usuarios.

O software pode poñerse a disposición dos usuarios cando aínda no está acabado e entón noméase co nome comercial e un texto que indica o nivel de acabado. Ese texto pode ser:

¹ Segundo a Real Academia Española da lingua, implementar é poñer en funcionamento, aplicar métodos, medidas, etc., para levar algo a cabo. En informática esta palabra aplícase en varios contextos como por exemplo os seguintes textos extraídos das páxinas oficiais:

- "Mozilla Firefox respecta na súa implementación as especificacións de W3C" para indicar que o código de Mozilla cumpre esas especificacións.
- "Microsoft Visual Studio permite crear proxectos de implementación e instalación que permiten distribuír unha aplicación finalizada para a súa instalación noutros equipos" para indicar que permite planear, instalar e manter unha aplicación finalizada.

- Versión **Alfa**. Versión inestable, á que aínda se lle poden engadir novas características e aínda **non saíu ao exterior da empresa que o desenvolve ou so para usuarios coñecidos** (testers)
- Versión **Beta**. Versión inestable á que non se lle van a engadir novas características pero que pode ter erros. **Sae xa ao público, pero indicando que é unha versión aínda de proba.**
- Versión **RC (Release Candidate)** é case a versión final pero aínda poden aparecer erros aínda que serán xeralmente pequenos erros, xa que pasou por unha versión Beta.
- Versión **RTM (Release To Manufacturing)**. Versión estable para comercializar. *Windows empregaba este termo para indicar o Sistema Operativo que ía preinstalado no hardware, sen necesidade de instalalo ao comprar o equipo.*



Tarefa 1.3. Buscar en internet aplicacións software dispoñibles para o usuario en versión alfa, beta, RC ou RTM.

Instalación

Esta fase tamén se denomina despregue ou implantación e consiste en transferir o software do sistema ao ordenador destino e configuralo para que poida ser utilizados polo usuario final. Esta fase pode consistir nunha sinxela copia de arquivos ou ser máis complexo como por exemplo: copia de programas e de datos que están comprimidos a localizacións específicas do disco duro, creación de accesos directos no escritorio, creación de bases de datos en localizacións específicas, etcétera.

Existen ferramentas software que permiten automatizar este proceso que se chaman instaladores. No caso dunha instalación simple, pode ocorrer que o instalador xere uns arquivos que permitan ao usuario final facer de forma automática unha instalación guiada e sinxela; noutro caso a instalación ten que ser feita por persoal especializado.

O software pode pasar a produción despois de resolver o proceso de instalación, é dicir, pode ser utilizado e explotado polo cliente.

Mantemento

Esta fase permite mellorar e optimizar o software que está en produción. O mantemento permitirá realizar cambios no código para corrixir erros encontrados, para facer o código máis perfecto (optimizar rendemento e eficacia, reestruturar código, perfeccionar documentación, etcétera), para que evolucione (agregar, modificar ou eliminar funcionalidades segundo necesidades do mercado, etcétera), ou para que se adapte (cambios no hardware utilizado, cambios no software do sistema xestor de bases de datos, cambios no sistema de comunicacións, etcétera).

Ao redor do 2/3 partes do tempo invertido en Enxeñería de software está dedicado a tarefas de mantemento e ás veces estas tarefas son tantas e tan complexas que é menos custoso volver a deseñar o código.

As versións de software resultantes do mantemento noméanse de diferente maneira dependendo do fabricante. Por exemplo: Debian 7.6, NetBeans IDE 6.5, NetBeans IDE 7.0.1, NetBeans 8.0, Java SE 8u20 (versión 8 update 20). Algúns fabricantes subministran aplicación que son parches que melloran o software instalado como por exemplo Service Pack 1 (SP1) para Windows Server 2008 R2, ou Service Pack 2 (SP2) para Windows Server 2008 R2.



Tarefa 1.4. Buscar en internet as últimas versións de Debian, Wordpress, Java, Windows.

Documentación

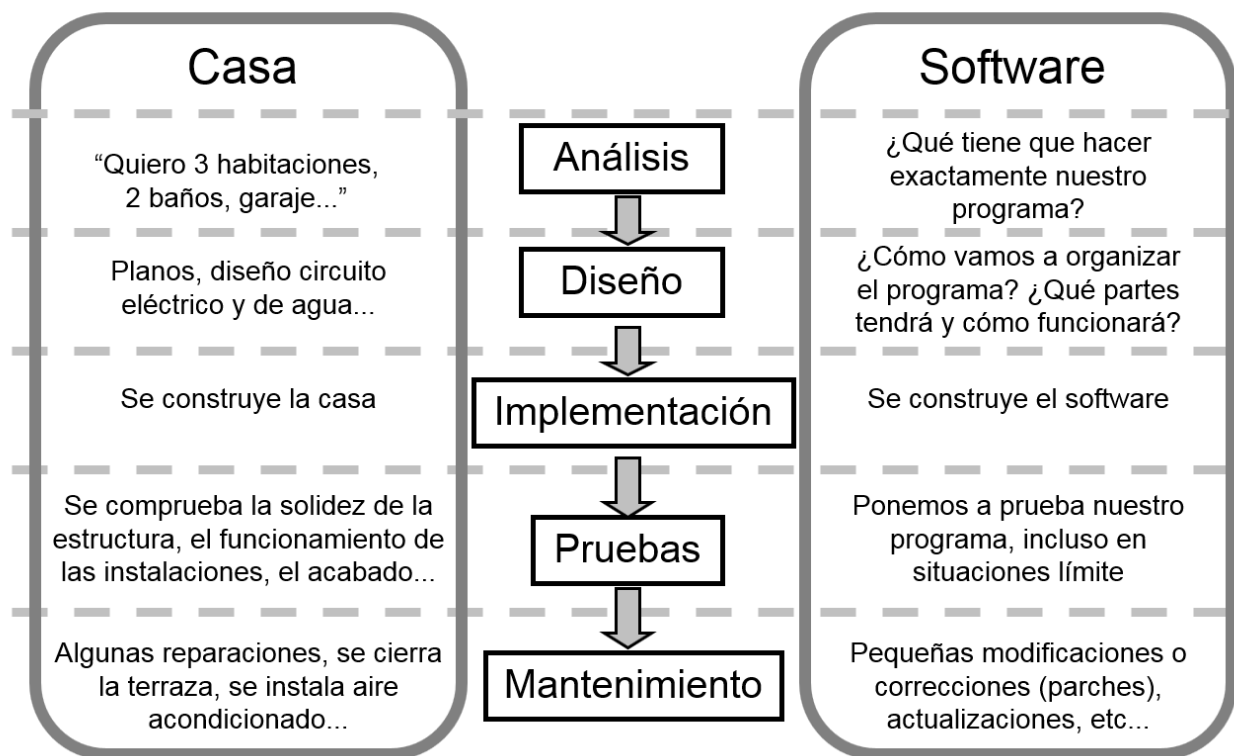
A creación de documentación está asociada a tódalas fases anteriores e en especial ás fases de codificación, probas e instalación. Para estas últimas fases pódese distinguir entre:

- Documentación interna. É a que aparece no código dos programas para que os programadores encargados de mantelo poidan ter información extra sen moito esforzo e para que de forma automática poida extraerse fóra do código esa información nun formato lexible, como por exemplo HTML, PDF, CHM, etcétera.

Exemplo de código Java con comentarios Javadoc que aportan información extra e permiten que algunhas aplicacións xeren automaticamente unha páxina web coa información que extraen dos comentarios:

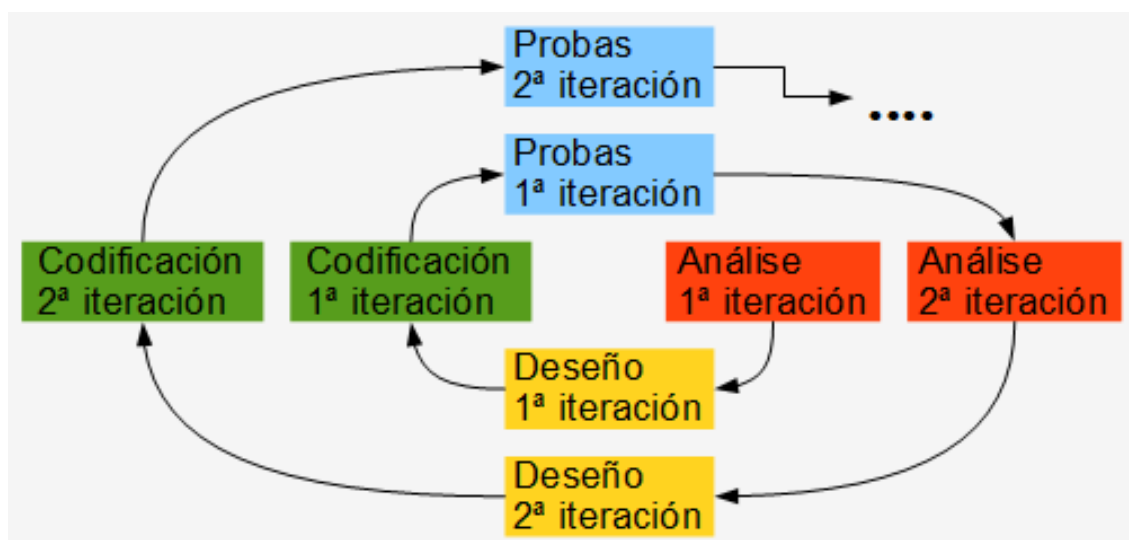
```
/** Este método calcula el número de reintentos de una
    determinada opción.
    @param opcion Cadena de texto con la opción a parsear
    para extraer el número de reintentos. La sintaxis es
    la siguiente: "reintentos:XXX", donde XXX será una
    cadena de texto que se interpretará como un
    número (valor retornado).
    @param similitud Flag que si vale true (valor por defecto)
    hará que el método no distinga las letras mayúsculas
    a la hora de intentar parsear la sintaxis, de modo
    que "reintentos:XXX" sea igual de válido que
    "ReinteNTos:XXX".
    @return Devuelve el número de reintentos o -1 en caso
    de error. */
int CalcularReintentos (const char *opcion, bool similitud = true);
```

- Documentación externa que é a que se adxunta aos programas e pode estar dirixida:
 - Aos programadores. Formada por exemplo por: código fonte, explicación de algoritmos complicados, especificación de datos e formatos de entrada/saída, listado de arquivos que utiliza ou xera a aplicación, linguaxe de programación, descrición de condicións ou valores por defecto que utiliza, diagramas de deseño.
 - Aos usuario. Formada por exemplo por: requisitos do sistema (tipo de ordenador no que funciona, sistema operativo que require, recursos hardware necesarios, etcétera), detalles sobre a instalación, explicacións para utilizar o software de forma óptima, descrición de posibles erros de funcionamento ou instalación e a forma de corrixilos.
- Autodocumentación que é documentación á que se accede durante a execución dos programas e pode conter: índice de contidos, guías sobre o manexo do programa, asistentes, axuda contextual, autores dos programas, versións dos mesmos, direccións de contacto, enlaces de referencia.



1.6 Modelo en espiral

Este modelo baséase na creación dun prototipo do proxecto que se vai perfeccionando en sucesivas iteracións a medida que se engaden novos requisitos, pasando en cada iteración polo proceso de análise, deseño, codificación e probas descritos no modelo en cascada. Ao final de cada iteración o equipo que desenvolve o software e o cliente analizaran o prototipo conseguido e acordarán se inician unha nova iteración. Sempre se traballa sobre un prototipo polo que no momento que se decida non realizar novas iteracións e acabar o produto, haberá que refinar o prototipo para conseguir a versión final acabada, estable e robusta.



1.7 Programación eXtrema²

A programación eXtrema ou *eXtreme Programming* é un método de desenvolvemento áxil de software baseado en iteracións sobre as fases de planificación, deseño, codificación e probas.

Planificación

Cada reunión de planificación é coordinada polo xestor do proxecto e nela:

- O cliente indica mediante frases curtas as prestacións que debe ter o software sen utilizar ningún tipo de tecnicismos nin ferramenta especial.
- Os desenvolvedores de software converterán cada prestación en tarefas que duren como máximo tres días ideais de programación de tal xeito que a prestación completa non requira máis de 3 semanas. De non conseguir estas cifras, revisaránse as prestación e as tarefas de acordo co desexo do cliente.
- Entre todos decídese o número de prestacións que formarán parte de cada iteración que se denomina velocidade do proxecto.
- Ao final de cada iteración farase unha reunión de planificación para que o cliente valore o resultado; se non o acepta, haberá que engadir as prestacións non aceptadas á seguinte iteración e o cliente deberá de reorganizar as prestacións que faltan para que se respecte a velocidade do proxecto.

É importante a mobilidade das persoas, é dicir, que en cada iteración os desenvolvedores traballen sobre partes distintas do proxecto, de forma que cada dúas ou tres iteracións, os desenvolvedores traballaran en todas as partes do sistema.

Deseño

A diferenza do modelo en cascada, nesta fase utilízase unha tarxeta manual tipo CRC (*class, responsibilities, collaboration*) por cada obxecto do sistema, na que aparece o nome da clase, nome da superclase, nome das subclases, responsabilidades da clase, e obxectos cos que colabora. As tarxetas vanse colocando riba dunha superficie formando unha estrutura que reflicta as dependencias entre elas. As tarxetas vanse completando e recolocando de forma manual a medida que avanza o proxecto. Os desenvolvedores reuniranse periodicamente e terán unha visión do conxunto e de detalle mediante as tarxetas.

² AYCART PÉREZ, David. GIBERT GINESTA, Marc HERNÁNDEZ MATÍAS, Martín, MAS HERNÁNDEZ, Jordi. *Ingeniería de software en entornos de SL*. Universitat Oberta de Catalunya.

Tamén pódese consultar máis información no seguinte enderezo
(https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema)

Nombre de la clase.	
Responsabilidades	Colaboradores

Codificación e probas

Unha diferenza desta fase con relación á fase de codificación do modelo en cascada é que os desenvolvedores teñen que acordar uns estándares de codificación (nomes de variables, sangrías e aliñamentos, etcétera), e cumprilos xa que todos van a traballar sobre todo o proxecto.

Outra diferenza é que se aconsella crear os test unitarios antes que o propio código a probar xa que entón se ten unha idea máis clara do que se debe codificar.

Unha última diferenza é que se aconsella que os programadores desenvolvan o seu traballo por parellas (*pair programming*) xa que está demostrado que dous programadores traballando conxuntamente fronte ao mesmo monitor pasándose o teclado cada certo tempo, fano ao mesmo ritmo que cada un polo seu lado pero o resultado final é de moita máis calidade xa que mentres un está concentrado no método que está codificando, o outro pensa en como ese método afecta ao resto de obxectos e as dúbidas e propostas que xorden reducen considerablemente o número de erros e os problemas de integración posteriores.

1.8 Métrica v.3

Métrica versión 3 é unha metodoloxía de planificación, desenvolvemento e mantemento de sistemas de información promovido pola *Secretaría de Estado de Administraciones Públicas* do *Ministerio de Hacienda y Administraciones Públicas* e que cubre o desenvolvemento estruturado e o orientado a obxectos .

Esta metodoloxía ten como referencia o Modelo de Ciclo de Vida de Desenvolvemento proposto na norma ISO 12.207 "*Information technology- Software live cycle processes*".

Consta de tres procesos principais: planificación, desenvolvemento e mantemento. Cada proceso divídese en actividades non necesariamente de execución secuencial e cada actividade en tarefas. No portal de administración electrónica (<http://administracionelectronica.gob.es/>) pódese acceder aos documentos pdf no que está detallada toda a metodoloxía e o persoal informático que intervéñen en cada actividade.

Planificación

O proceso de planificación de sistemas de información (PSI) ten como obxectivo a obtención dun marco de referencia para o desenvolvemento de sistemas de información que respondan a obxectivos estratéxicos da organización e é fundamental a participación da alta dirección da organización. Consta das actividades:

- Descrición da situación actual.
- Un conxunto de modelos coa arquitectura da información.

- Unha proposta de proxectos a desenvolver nos próximos anos e a prioridade de cada un.
- Unha proposta de calendario para a execución dos proxectos.
- A avaliación dos recursos necesarios para desenvolver os proxectos do próximo ano.
- Un plan de seguimento e cumprimento de todo o proposto.

Desenvolvemento

Cada proxecto descrito na planificación ten que pasar polo proceso de desenvolvemento de sistemas de información que consta das actividades:

- Estudio de viabilidade do sistema (EVS) no que se analizan os aspectos económicos, técnicos, legais e operativos do proceso e se decide continuar co proceso ou abandonalo. No primeiro caso haberá que describir a solución encontrada: descrición, custo, beneficio, riscos, planificación da solución. A solución pode ser desenvolver software a medida, utilizar software estándar de mercado, solución manual ou unha combinación delas.
- Análise do sistema de información (ASI) para obter a especificación de requisitos software que conterá as funcións que proporcionará o sistema e as restricións ás que estará sometido, para analizar os casos de usos, as clases e interaccións entre elas, para especificar a interface de usuario, e para elaborar o plan de probas.
- Deseño do sistema de información (DSI) no que se fai o deseño de comportamento do sistema para cada caso de uso, o deseño da interface de usuario, o deseño de clase, o deseño físico de datos (se é necesario tamén se fai o deseño da migración e carga inicial de datos), a especificación técnica do plan de probas e o establecemento dos requisitos de implantación (implantación do sistema, formación de usuarios finais, infraestruturas, etcétera).
- Construción do sistema de información (CSI) no que se prepara a base de datos física, se prepara o entorno de construción, xérase o código, execútanse as probas unitarias, as de integración e as de sistema, elabóranse os manuais de usuario, defínese a formación dos usuarios finais e constrúense os compoñentes e procedementos da migración e carga inicial de datos.
- Implantación e aceptación do sistema (IAS) que ten como obxectivo a entrega e aceptación do sistema total e a realización de todas as actividades necesarias para o paso a produción. Para iso séguense os pasos: formar ao equipo de implantación, formar aos usuarios finais, realizar a instalación, facer a migración e carga inicial de datos, facer as probas de implantación (comprobar que o sistema funcione no entorno de operación), facer as probas de aceptación do sistema (comprobar que o sistema cumpre os requisitos iniciais do sistema), establecer o nivel de mantemento e servizo para cando o produto estea en produción. O último paso é o paso a produción para o que se analizarán os compoñentes necesarios para incorporar o sistema ao entorno de produción, de acordo ás características e condicións do entorno no que se fixeron as probas e se realiza a instalación dos compoñentes necesarios valorando a necesidade de facer unha nova carga de datos, unha inicialización ou unha restauración, fíxase a data de activación do sistema e a eliminación do antigo.

Mantemento

O obxectivo deste proceso é a obtención dunha nova versión do sistema de información desenvolvido con Métrica 3, a partir das peticións de mantemento que os usuarios realizan con motivo de problemas detectados no sistema ou pola necesidade de mellora do mesmo.

1.9 Metodoloxías áxiles : Scrum

O áxil defínese como a habilidade de responder de forma versátil aos cambios para maximizar los beneficios. Responde con rapidez e da un bo resultado.

As metodoloxías áxiles son unha forma de traballar en equipo, para obter resultados rápidos, con moita interacción entre os membros do equipo de desenvolvemento de software e o cliente.

A continuación desenvolvemos en detalle a metodoloxía Scrum, unha das máis empregadas na actualidade.

¿Que é Scrum?

Scrum é unha forma de traballar, non un proceso, no que se aplican de maneira regular un conxunto de procesos para traballar en equipo, e obter o mellor resultado posible dun proxecto. Aplícase xeralmente a desenvolvemento de software, pero pode ser aplicado para o desenvolvemento de calquera produto que implique traballo intelectual e relación co cliente. Traballos en cadea non requiren Scrum.

Dicimos que non é un proceso, di “que hai que facer” e non “como hai que facelo”. A metodoloxía é fácil de entender, o importante é implantala correctamente xa que a interacción entre os participantes é fundamental.

Ideas fundamentais de Scrum son:

- Adoptar unha idea total da realización do produto, en lugar da planificación e execución completa do produto.
- Enfocarse máis nas zonas de solapamento, en lugar de realizar unha tras outra nun ciclo de cascada. Entregas progresivas do produto a modo de prototipos.
- Pon por encima aos individuos e a súa interacción por encima de procesos e ferramentas, colaboración co cliente por encima da negociación contractual.
- Eficaz resposta ante cambios, xa sexan cambios de requirimentos ou incidencias.
- Outras metodoloxías céntranse na documentación, para facer o proceso o máis independente das persoas involucradas (cada un ten totalmente definido o que ten que facer e como facelo). Scrum céntrase no contrario, as persoas, a comunicación, a interacción.

Historia

Este modelo foi identificado e definido por Ikujiro Nonaka e Hirotaka Takeuchi a principios dos 80, ao analizar como desenvolvían os novos produtos as principais empresas de manufactura tecnolóxica.

No seu estudo, Nonaka e Takeuchi compararon a nova forma de traballo en equipo, co avance en formación de melé (Scrum en inglés) dos xogadores de Rugby, por mor do cal quedou acuñado o termo “Scrum” para referirse a ela.

¿Como se usa?

Coa metodoloxía Scrum o cliente entusiásmase e comprométese co proxecto dado que o ve crecer parte por parte. Así mesmo, permítelle en calquera momento realiar o software cos obxectivos de negocio da súa empresa, xa que pode introducir cambios funcionais ou de prioridade no inicio de cada nova iteración sen ningún problema.

Esta forma de traballo promove a motivación e compromiso do equipo que forma parte do proxecto,

polo que os profesionais atopan un ámbito propicio para desenvolver as súas capacidades.

Os principios da metodoloxía Scrum son:

- Concentración
- Priorización
- Autoorganización
- Ritmo

Beneficios de Scrum

1. O cliente pode empezar a utilizar os resultados máis importantes do proxecto antes de que estea finalizado por completo.
2. O cliente establece as súas expectativas indicando o valor que lle achega cada requisito
3. Redución de riscos
4. Predicións de tempos
5. Maior produtividade

¿Que é un Sprint?

Sprint é o nome que vai recibir cada un dos ciclos ou iteracións que imos ter dentro de dentro dun proxecto Scrum.

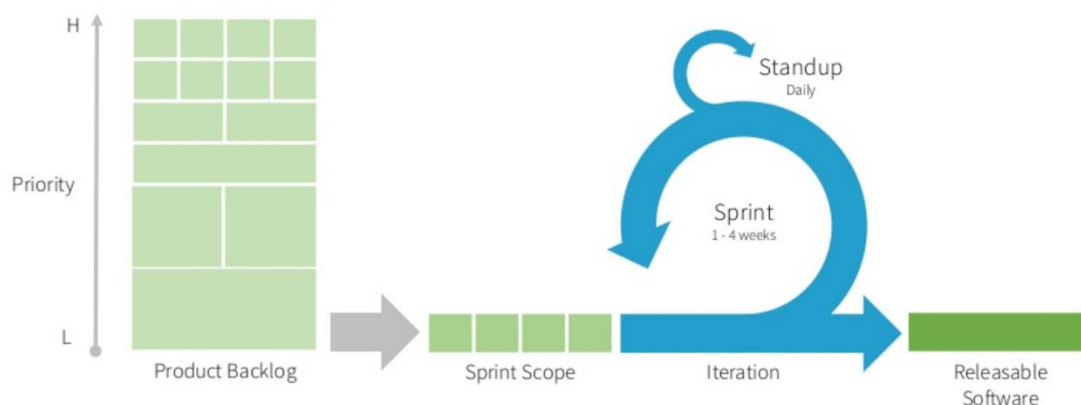
Van permitir ter un ritmo de traballo cun tempo prefixado, sendo a duración habitual dun Sprint vai entre unha e catro semanas, aínda que o que a metodoloxía di é que debería estar entre dúas semanas e un máximo de dous meses.

En cada Sprint ou cada ciclo de traballo o que imos conseguir é o que se denomina un entregable ou incremento do produto, que achegue valor ao cliente.

A idea é que cando temos un proxecto bastante longo, por exemplo, un proxecto de 12 meses, imos poder dividir ese proxecto en doce Sprints dun mes cada un. En cada un deses Sprints imos ir conseguindo un produto, que sempre, e isto é moi importante, sexa un produto que estea a funcionar.

Imos velo de forma máis clara nesta imaxe:

Traditional Scrum

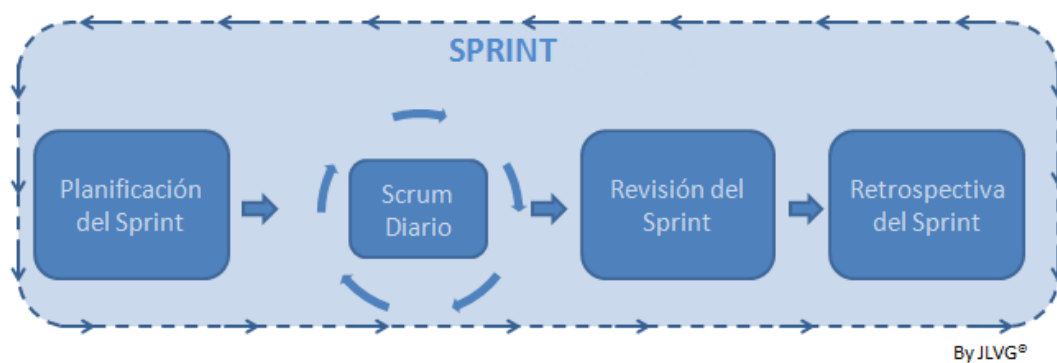


Na mesma temos unha pila ou product backlog, que serían todos os requisitos que nos pide o cliente, é dicir, o ano completo de traballo. A idea é ir seleccionando eses requisitos nos que temos a pila dividida, e ímolos a ir facendo en diferentes Sprints, e realizamos todos os pasos que conforman un Sprint, é dicir, a toma de requisitos, deseño, implementación, probas e despregamento no prazo establecido, e así imos ter sempre un software que sexa válido, un software funcionando.

Cando falamos de Sprint, falamos da parte máis importante de Scrum, por iso imos ver as etapas e como se desenvolven en profundidade.

Fases de un Sprint

Cando falamos de Sprint en Scrum técnico, engloba todo o proceso, é dicir, desde que decidimos que imos facer para ese Sprint, ata que estudamos como traballamos nese Sprint.



Cando estamos nun proxecto Scrum para cada Sprint imos ter unha serie de reunións:

Reunión de Planificación do Sprint. É a primeira reunión do Sprint, e nela imos decidir o que imos facer e como o imos a facer, o número de tarefas ou de historias de usuario que imos realizar no Sprint.

Reunións de Scrum diario, que van ser pequenas reunións cos membros do equipo.

Revisión do Sprint, na que imos aceptar ou denegar o Sprint.

Reunión de retrospectiva, onde imos ver como traballou o equipo e que problemas tivo durante o desenvolvemento e como o podemos corrixir.

O Spring engloba todo lo anterior, desde que comeza o mesmo hasta que es aceptado ou denegado, e o equipo se pregunta como traballou.

¿Qué es el Planning Póker?

O Planning Póker é un proceso de estimación de duración ou esforzo nas tarefas dun proxecto por parte de todo o equipo implicado. Para levar a cabo esta técnica, cada participante conta cunha baralla de cartas con números (baseados en Fibonacci: 1,2,3,5,8,13, ou similar) representando o grao de complexidade (esforzo) necesario para completar unha tarefa.

Comeza o cliente explicando un obxectivo. De forma individual, cada membro do equipo pon sobre a mesa a carta que representa a puntuación que el considera necesaria para levar a cabo dita tarefa do proxecto en cuestión. De haber puntuacións moi dispares, cada participante explica os motivos da súa decisión. E se fose necesario, pode repetirse a votación ata chegar a un consenso. Así, o resultado é unha estimación consensuada e validada por todo o equipo.

Ten moitas vantaxes: todos os membros do equipo expresan a súa opinión sen sentirse condicionados polo resto, ése máis consciente do esforzo que require unha tarefa, o que mellora a implicación. Ao sentirse partícipes, o grao de compromiso co proxecto tamén aumenta.

1.- Reunión de Planificación del Sprint

- Lévese a cabo ao comezo do ciclo
- Seleccionar que traballo se fará (tarefas do Product backlog)
- O Product Owner presenta as funcionalidades (user stories) e as súas prioridades (valoradas en story points). O equipo acorda con el as actividades para desenvolver nese Sprint. Ten que haber consenso (somos capaces de facer todas esas actividades no período que dura este sprint?)
- Esas actividades seleccionadas almacénanse no Sprint Backlog e conforman o obxectivo (Goal) do sprint actual. Ata aquí a primeira parte da reunión.
- Agora o equipo planifica as actividades, stories, e pode ser necesario volver a reestimar as actividades que se poden chegar a realizar neste sprint. Cada unha destas actividades se “despezan” en tarefas que poden ser desenvoltas por cada membro técnico do equipo. É cada membro do equipo o que toma a tarefa ou tarefas que vai realizar.
- Esta reunión ten unha duración máxima de oito horas.
- A partir deste intre, todo o equipo ponse a traballar nas tarefas asignadas.

Queda claro que o feedback de pártela cliente, co product owner, é unha parte fundamental desta metodoloxía. O seguimento constante impide que se chegue a unha entrega final de produto e o cliente non estea satisfeito, como pode ocorrer cunha metodoloxía non áxil.

2.- Daily Scrum

É unha reunión diaria coas seguintes características:

- A reunión comeza puntualmente á súa hora.
- Todos os asistentes deben manterse de pé.
- A reunión debe ocorrer na mesma localización e á mesma hora todos os días.
- Tempo límite da reunión 15 minutos.
- Durante a reunión, cada membro do equipo contesta a tres preguntas:
 1. Que fixeches desde onte?

2. Que é o que estás a planear facer hoxe?
3. Tiveches algún problema que che impedira alcanzar o teu obxectivo? Esta última pregunta pode modificar o traballo dese día dalgún membro do equipo, se isto é necesario para que outro membro poida realizar as súas tarefas.

3.- Reunión de Revisión del Sprint

É unha reunión para revisar o traballo que foi completado e non completado.

Debe estar presente o Propietario do Produto. A relación co cliente é máis intensa que en metodoloxías tradicionais.

Presentar o traballo ao Propietario do Produto, mediante unha demo para a súa aprobación. Aínda que teña algún erro emendable, o produto pode ser aceptado.

O traballo incompleto non pode ser demostrado.

A falta de éxito dun sprint vén dada porque o produto presentado non corresponde ás necesidades do cliente ou ben que non deu tempo a resolvelo completamente. Haberá que telo en conta para que non ocorra no seguinte sprint.

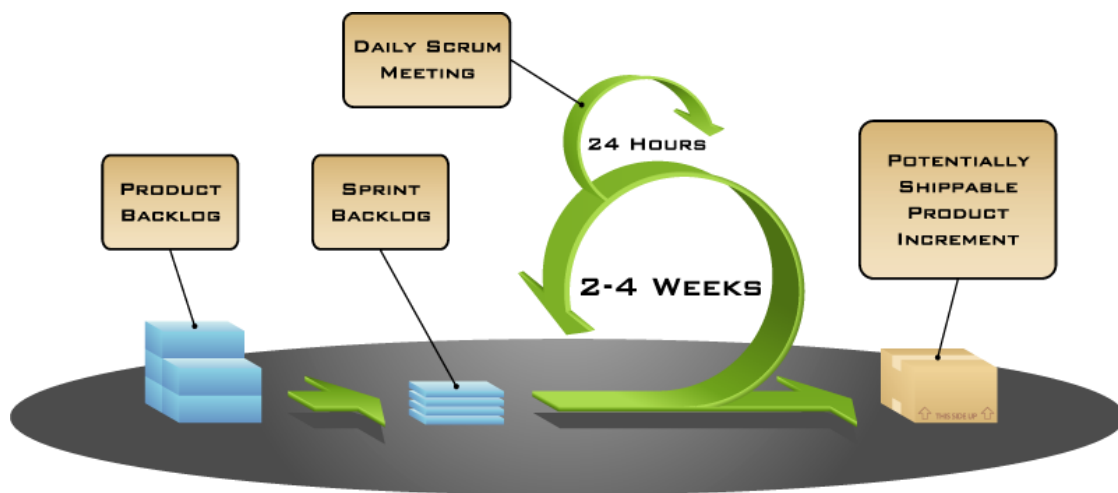
Catro horas como límite.

4.- Retrospectiva del Sprint

Despois de cada sprint, leva a cabo unha retrospectiva do sprint, na cal todos os membros do equipo deixan as súas impresións sobre o sprint recentemente superado. O propósito da retrospectiva é realizar unha mellora continua do proceso. As tres preguntas desta reunión son:

1. En que melloramos desde o noso último sprint? É dicir, que debemos seguir facendo.
2. En que Podemos mellorar para os próximos sprints?
3. Que deberíamos parar de facer?

Esta reunión ten un tempo fixo de catro horas.

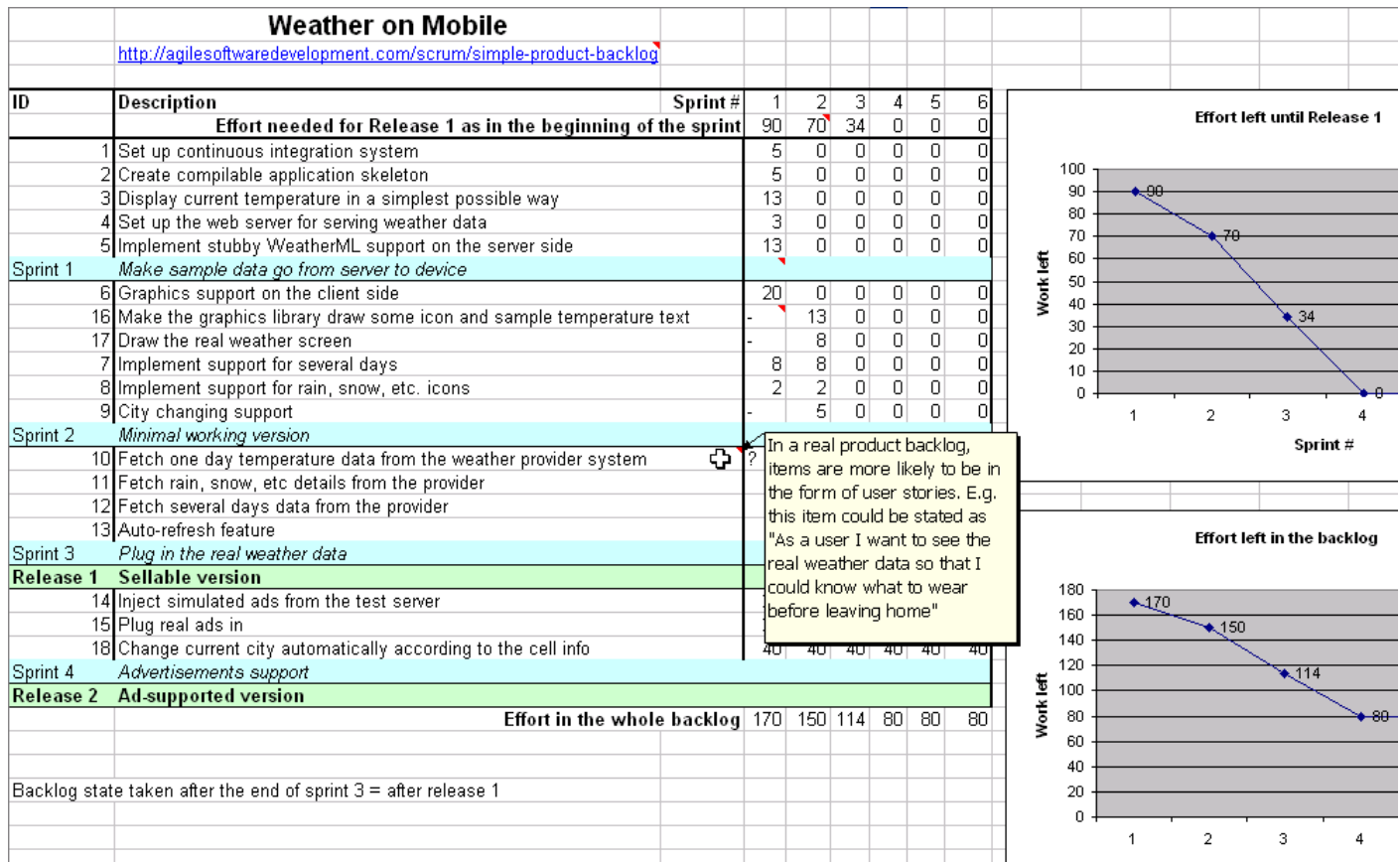


COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Documentos del Scrum

Product backlog

O product backlog é un documento de alto nivel para todo o proxecto. Contén descrições xenéricas de todos os requirimentos, funcionalidades desexables, etc., priorizadas segundo o seu valor para o negocio (business value). É aberto e calquera pode modificalo, aínda que o Product Owner é o seu responsable. É frecuente que ao final dun sprint sufra algunha modificación.



Sprint backlog

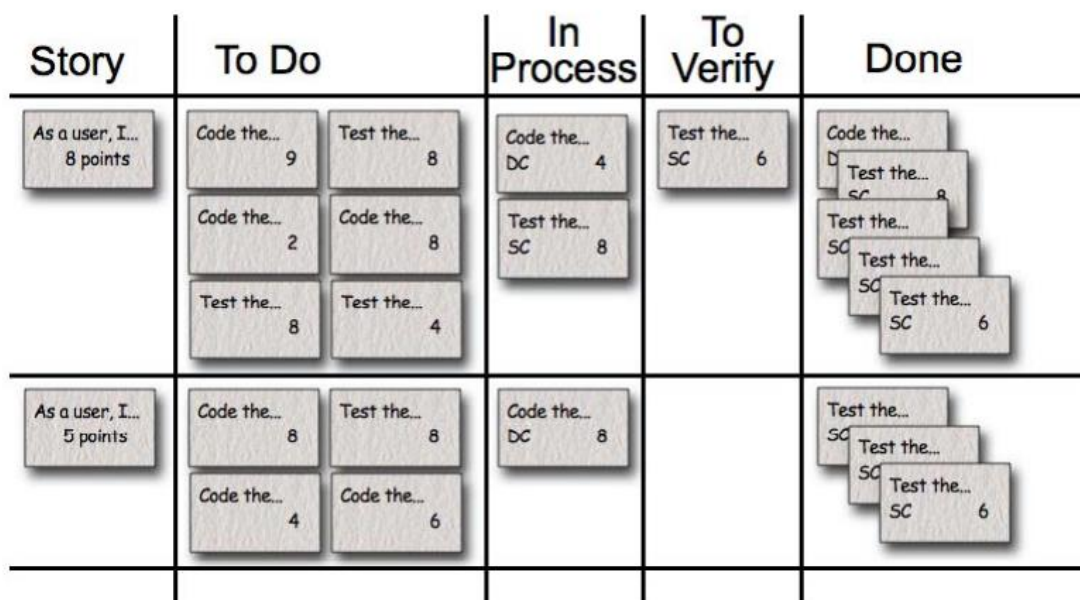
O sprint backlog é un documento detallado onde se describe como o equipo vai implementar os requisitos durante o seguinte sprint. Consiste en tomar as actividades de maior prioridade do Product Backlog. As tarefas ordénanse por prioridade, coas súas horas estimadas de dedicación, pero ningunha tarefa de duración superior a 16 horas. Se unha tarefa é maior de 16 horas, deberá ser rota en maior detalle. As tarefas no sprint backlog nunca son asignadas, son tomadas polos membros do equipo do modo que lles pareza oportuno.

Pódese engadir a este documento a porcentaxe de tarefa realizada, para ver de forma fácil e visual o progreso do sprint.

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

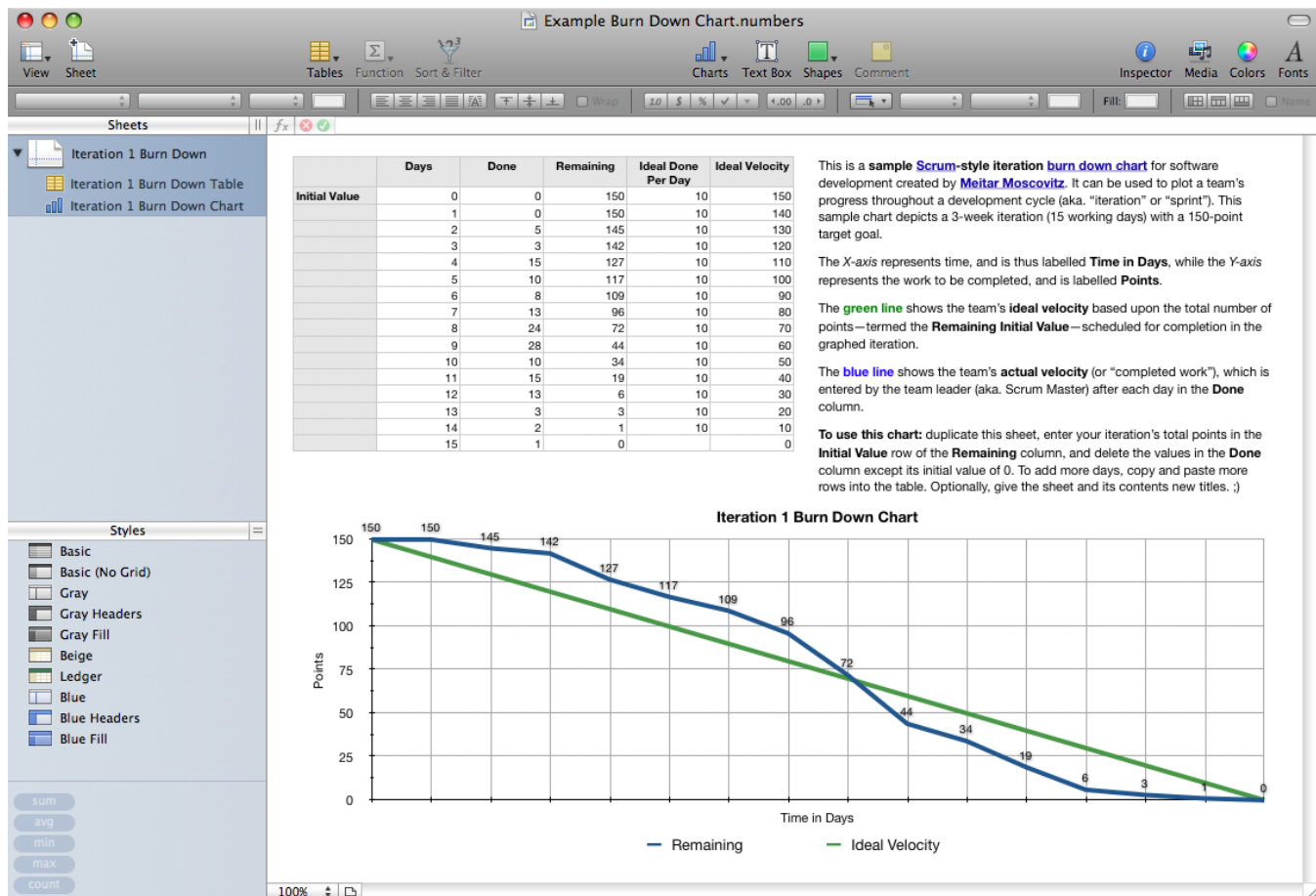
Outra forma de mostrar as tarefas de sprint é mediante una *pizarra de tarefas*:

Pizarra tareas



Burn down

A burn down chart é unha gráfica mostrada publicamente que mide a cantidade de requisitos no Backlog do proxecto pendentes ao comezo de cada Sprint. Debuxando unha liña que conecte os puntos de todos os Sprints completados, poderemos ver o progreso do proxecto. O normal é que esta liña sexa descendente, ata chegar ao eixo horizontal, momento no cal o proxecto se ha terminado.



Na figura superior dun Burn Down Chart, a liña azul representa a cantidade de requisitos pendentes e a liña verde representa a “velocidade ideal”, baseada en sprints previos. A velocidade se correspondería á cantidade de requirimentos (user stories) do sprint, divididas polo número de días do mesmo.

Roles

ScrumMaster

O ScrumMaster é o responsable de xestionar o proxecto, facilitar a comunicación entre os compoñentes e que se sigan as pautas da metodoloxía. Tamén ten que resolver todas incidencias que se vaian presentando.

O ScrumMaster ten que explicar e convencer ao equipo do uso dos procesos de Scrum, cando en moitos casos non ten autoridade xerárquica sobre eles.

As incidencias que ten que resolver poden estar relacionadas co cliente, ou con terceiras partes involucradas no produto, por exemplo, provedores.

As incidencias que ten que resolver poden estar relacionadas co cliente, ou con terceiras partes involucradas no produto, por exemplo, provedores.

Product Owner

O Product Owner representa a voz do cliente. Sabe exactamente que é o que o cliente quere. Nun mundo perfecto, sería o mesmo cliente, pero en xeral adoita ser un intermediario. O Product Owner ten que coñecer en detalle todos os requisitos que se teñen que desenvolver, coa prioridade de cada

un deles e as súas dependencias.

Debe ser tamén capaz de contestar a todo tipo de cuestións relacionadas co produto e é o responsable do mantemento do Product Backlog e a descrición das actividades.

Outra responsabilidade é decidir, negociado co equipo, as actividades de cada Sprint.

Ao final de cada sprint, na Reunión de Retrospectiva, o Product Owner aproba ou rexeita as características desenvolvidas. E tamén pode dar suxerencias para seguintes Sprints.

Finalmente, tamén é responsable de planificar as distintas entregas de produto, as distintas versións ou prototipos. El sabe como ten que ser o produto.

Ten que haber moita interacción co equipo e retroalimentación mutua.

Team

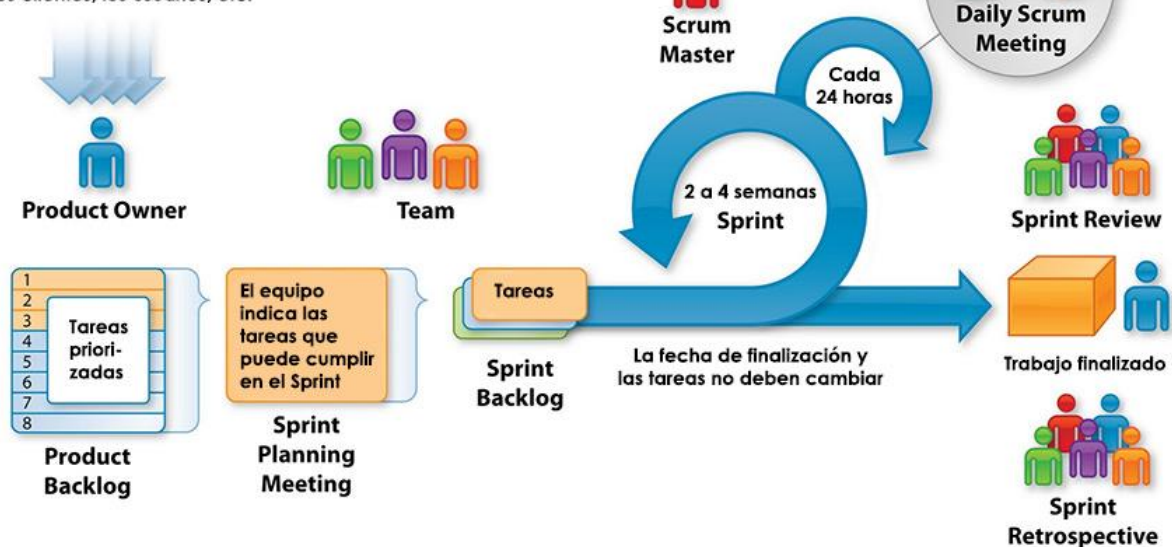
Son as persoas que desenvolven o equipo, son responsables de deseñar, desenvolver e probar o produto que están a implementar.

O Product Owner alimenta con información ao equipo e asegúrase de que o equipo segue o camiño correcto e coa calidade adecuada.

O ScrumMaster lidera ao equipo para que este siga a metodoloxía Scrum.

The Agile: Scrum Framework at a glance

Información de los ejecutivos,
el quipo, los implicados,
los clientes, los usuarios, etc.



Últimas Consideracións

É válido Scrum para todo o mundo?

Definitivamente non. Algunhas persoas están demasiado afeitas á súa forma de traballar, e pode estar ben. Se o que estás a facer está funcionando ben pode que non haxa ningunha razón para cambiar o proceso. Non obstante, se observa que non se cumpren os requisitos, o produto final non é útil para o cliente, é posible que deba pensar máis sobre o uso de Scrum.

Cando non empregar Scrum?

Scrum non se debe usar en equipos pequenos, de ata tres persoas. Neste caso, a experiencia demostra que o marco non é tan útil por algunhas razóns. Os membros do equipo probablemente falan entre eles todo o tempo, facendo que as reunións diarias non teñan sentido. Ademais, dado que o equipo é bastante pequeno, as melloras son fáciles de detectar e adoitan implementarse en canto se identifican. Isto non significa, con todo, que non se deben empregar boas prácticas de desenvolvemento áxil. Só significa que non é necesario todo o marco Scrum.

Outra situación na que Scrum pode non ser necesario é cando o alcance do produto é ben coñecido e seguramente non cambiará. Ademais, o equipo é competente na tecnoloxía empregada, polo que non poden xurdir sorpresas por incertezas. Como se pode observar, este escenario é bastante raro. Non obstante, nesta situación específica, non son necesarios comentarios constantes, mellora continua do proceso nin cambios na priorización porque todo está ben determinado.

É Scrum perfecto, unha bala de prata?

Absolutamente non. Como se dixo antes, Scrum é moi difícil de poñer en práctica debido á interacción das persoas que poden xurdir. Ademais, é posible que as organizacións non estean abertas a todo o necesario para seguir Scrum.

Ademais, Scrum non resolve o problema das estimacións a longo prazo. Aínda é moi difícil estimar un proxecto longo con precisión, que aínda é un problema para os xestores que necesitan vender un produto.

Como comezar a empregar Scrum?

O mellor consello é facelo paso a paso, en pequenos proxectos preferentemente con membros acostumados a Scrum. Quizais comezar con Sprint, Daily Meetings e despois engadir comentarios. Probablemente estes mostrarán problemas no seu proxecto, como a comunicación e demasiados erros con funcións que xa se supoñía que estaban listas. As reunións diarias probablemente non serán tan curtas como se supoñía, polo que isto é algo que hai que solucionar.

Máis tarde, introducir unha retrospectiva sobre o proceso a mellorar. Despois de que o equipo estea acostumado para seguir todas as cerimonia de Scrum, probablemente o software en si sexa mellor. Non obstante, notará que haberá espazo para melloras, como probas automatizadas, mellor codificación e menos erros.

Isto non será doado: o cambio de mentalidade é enorme e hai xente que simplemente non se adapta a iso. Así, o importante é non frustrarse e tentar esforzarse en implementar Scrum se se pensa que paga a pena.

Ademais, hai que ter en conta que non adoptar todo o marco Scrum non é malo. O que é bo é o que funciona para ti. Quizais na túa empresa a company ownership funcione mellor para unha persoa que para un equipo. Ou quizais Sprints teña que variar os seus marcos de tempo. O importante é

empregar as ferramentas proporcionadas por Scrum para mellorar o seu proceso de desenvolvemento de software e facelo áxil.

Son os procesos en Cascada inútiles?

Absolutamente non. Non esqueza que os procesos en cascada, principalmente os relacionados con CMM, existen desde hai moito tempo. Aínda que presentan problemas, moi comentados aquí, moitas empresas conseguiron un gran éxito con elas, e bastantes aínda o fan. Lembre tamén que antes deles non había nada que organizar o desenvolvemento de software, polo que foron un gran avance porque os procesos semellantes ao CMM proporcionaron metodoloxías para o desenvolvemento de software por primeira vez.

Estas metodoloxías permitiron pensar e discutir os requisitos antes de implementar o software, en lugar do caos absoluto que había antes dese tempo. Ademais, e na miña opinión, a maior vantaxe: permitíase o tempo para a arquitectura e deseño de software, o que mellorou moito a calidade do software. Ademais, o deseño e a arquitectura permitiron o desenvolvemento e uso de patróns de deseño e boas prácticas que, de novo, contribuíron enormemente á calidade do software.

Ademais diso, desenvolvéronse estratexias de proba, xunto con probas automáticas e unitarias, que contribuíron de novo á mellora do software.

Por iso, a mensaxe aquí é: non despreza as metodoloxías de software da vella escola porque nos proporcionaron boas ideas para un bo desenvolvemento de software. Ademais: a arquitectura, probas automatizadas, patróns de deseño, etc. son aínda moi aplicables coas metodoloxías Agile.

Vexa as metodoloxías áxiles como un paso na dirección correcta para o desenvolvemento de software. Finalmente, non nos enganemos, chegará un momento no que comezaremos a ver os problemas das metodoloxías áxiles e espero que xurda algo mellor. E despois, despois diso, xurdirán problemas e o ciclo continuará para sempre.



Tarefa 1.5. En equipos de 4, supoñede que sodes unha consultora a que o concello lle contrata un proxecto para o impulso do turismo da súa vila. Empregando a metodoloxía Scrum, asignádevos roles e facede unha planificación do proxecto especificando cada requirimento e as súas tarefas asociadas. Un deses requirimentos será o desenvolvemento dun “flyer” de difusión da vila e durará un so sprint de 15 días. Será ese o único sprint que faredes realmente, do resto so teremos a súa planificación. Teredes que entregar un vídeo coa evolución do proxecto cos elementos propios desta metodoloxía.

Axuda para a tarefa:

- Un membro do equipo será o Product Owner, outro o Scrum Master e os outro dous formarán o equipo (Team). Esta distribución pode trocar a medida que avancedes no proxecto.
- . O Product Owner debe falar con profesor para definir o alcance do proxecto e o que se pretende facer.
- So desenvolveredes realmente as tarefas dos flyer (como por exemplo a especificación de requirimentos, o deseño, a impresión, a difusión...). Deberedes facelo en dúas semanas, coa estrutura e reunións propias dun sprint.
- Ao final do proxecto faredes unha presentación aos vossos compañeiros.
- A entregar:
 - Product Backlog (xls)
 - Sprint backlog (xls)
 - Video. Tedes que describir a planificación global do proxecto, con todos os seus sprints, e incluíredes reunións do sprint real (do flyer).
- Dentro das tarefas a realizar estarían nese spring: definir os contidos do flyer, obter recursos como imaxes, o

deseño, solicitar presupostos para a súa impresión, a difusión, obter axudas ou patrocinios, etc.

1.10 Licencias de software

A licencia de software é o contrato entre o autor do software e o usuario final que o vai a empregar. Nas cláusulas dese contrato establécense:

- Condicións de instalación e uso: tanto o custo como quen o pode usar, en que equipos, etc.
- Posibilidade de modificación, cesión e distribución do software.
- Prazos e territorio onde se aplica a licencia, xa que cada país ou rexión pode ter leis distintas.
- Responsabilidade por erros no software
- Atribución (dicir quen é o autor)

En xeral, o software de uso público denomínase: software libre, de código aberto, copyleft, freeware, dominio público (aínda que teñen distintos matices) e o software de pago denomínase: con copyright, software propietario, código pechado.

GPL

La GPL (General Public License) tamén coñecida como GNU General Public License é una licencia de código aberto na que os usuarios poden usar, estudar, compartir e modificar o software, sempre que os produtos derivados sigan tendo licencia GPL.

O software baixo esta licencia ten que poñer a disposición dos usuarios, non só o código executable senón que tamén o código fonte (para poder modificalo).



Tarefa 1.6. Buscar en internet que é Creative Commons e os tipos de licencias que a compoñen (inclúe na resposta as distintas iconas que representan as distintas licencias). Finalmente, buscar páxinas de recursos gratuítos para incluír nos nosos programas: imaxes, música, etc. e comentar os requirimentos para o seu uso.

2. Linguaxes de programación e ferramentas de desenvolvemento

2.1 Clasificación das linguaxes informáticas

Unha linguaxe informática é unha linguaxe que permite comunicarse co ordenador e está formada por un conxunto de símbolos e palabras que seguen unhas normas de sintaxe.

Non existe unha clasificación das linguaxes informáticas adoptada pola maioría dos autores se non que hai grandes diferenzas entre eles. Unha clasificación posible é: linguaxes de marcas, especificación, consulta, transformación e programación.

Linguaxes de marcas

Permiten colocar distintivos ou sinais no texto que serán interpretadas por aplicacións ou procesos para distintos. Exemplos de linguaxes de marcas:

- A linguaxe XML (*eXtensible Markup Language*) que é unha metalinguaxe extensible pensada para a transmisión de información estruturada e que pode ser validada.

Exemplo de código XML editado en NetBeans:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Exemplo básico de XML
-->
<alumnos>
  <alumno>
    <nome>Pepe</nome>
    <apelidos>Ruiz Arias</apelidos>
  </alumno>
  <alumno>
    <nome>María Dolores</nome>
    <apelidos>González Paz</apelidos>
  </alumno>
</alumnos>
```

- As linguaxes HTML (*Hiper Text Markup Language*) ou XHTML (*eXtensible Hiper Text Markup Language*) =XML+HTML que serven ambas para publicar hipertexto na Word Wide Web. As marcas modifican a aparencia do contido, por exemplo debuxan unha táboa, un formulario, etc.

Exemplo de código XHTML editado en Notepad++:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Exemplo moi básico de XHTML con estilo externo -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemplo básico de xhtml</title>
    <meta content="text/html; charset=utf-8"
          http-equiv="Content-Type" />
    <link rel="stylesheet" href="exemplo_css.css"
          type="text/css" />
  </head>
  <body>
    <h1>Cabeceira principal</h1>
    <p class="principal">Este párrafo contén texto e un enlace a
      <a href="http://www.realacademiagalega.org/">Real Academia Galega</a>
    </p>
  </body></html>
```


Linguaxes de especificación

Describen algo de forma precisa. Por exemplo CSS (*Cascading Style Sheets*) é unha linguaxe formal que especifica a presentación ou estilo dun documento HTML, XML ou XHTML. Exemplo de código CSS que se podería aplicar ao exemplo XHTML anterior e editado en Notepad++:

```
body{
    font-family: "MS Sans Serif", Geneva, sans-serif;
    font-size: 15px;
    color: Black;
    border:black 2px double;
    padding: 40px;
    margin: 20px;}

h1 {
    font: 40px "Times New Roman", Times, serif;
    font-weight: bolder;
    word-spacing: 25px;}

p.principal{
    text-align: center;
    font: 10px "MS Serif", "New York", serif;}
```

Linguaxes de consultas

Permiten sacar ou manipular información de un grupo de información, xeralmente este grupos de información son bases de datos. Por exemplo, a linguaxe de consultas SQL (*Standard Query Language*) permite buscar e manipular información en bases de datos relacionais e a linguaxe XQuery permite buscar e manipular información en bases de datos XML nativas. Exemplo de script SQL:

```
CREATE DATABASE `empresa`;

USE `empresa`;

CREATE TABLE `centros` (
  `cen_num` int(11) NOT NULL default '0',
  `cen_nom` char(30) default NULL,
  `cen_dir` char(30) default NULL,
  UNIQUE KEY `numcen` (`cen_num`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `centros` VALUES
  (10,'SEDE CENTRAL','C/ ALCALA, 820-MADRID'),
  (20,'RELACION CON CLIENTES','C/ ATOCHA, 405-MADRID');

CREATE TABLE `deptos` (
  `dep_num` int(11) NOT NULL default '0',
  `dep_cen` int(11) NOT NULL default '0',
  `dep_dire` int(11) NOT NULL default '0',
  `dep_tipodir` char(1) default NULL,
  `dep_presu` decimal(9,2) default NULL,
  `dep_depen` int(11) default NULL,
  `dep_nom` char(20) default NULL,
  UNIQUE KEY `numdep` (`dep_num`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `deptos` VALUES
  (122,10,350,'F','60000.00',120,'PROCESO DE DATOS'),
  (121,10,110,'P','200000.00',120,'PERSONAL'),
  (120,10,150,'P','30000.00',100,'ORGANIZACION'),
  (112,20,270,'F','90000.00',110,'SECTOR SERVICIOS'),
  (111,20,400,'P','111000.00',110,'SECTOR INDUSTRIAL'),
  (200,20,600,'F','80000.00',100,'TRANSPORTES'),
  (100,10,260,'P','120000.00',NULL,'DIRECCION GENERAL');
```

Exemplo de expresión FLOWR (*for, let, order by, where, return*) en XQuery (<https://es.wikipedia.org/wiki/XQuery>):

```
for $libro in doc("libros.xml")/bib/libro
let $editorial := $libro/editorial
where $editorial="MCGRAW/HILL" or contains($editorial, "Toxosoutos")
return $libro
```

Linguaxes de transformación

Actúan sobre unha información inicial par obter outra nova. Por exemplo, a linguaxe XSLT (*eXtensible Stylesheet Language Transformations*: https://es.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations) permite describir as transformacións que se van realizar sobre un documento XML para obter outro arquivo. Exemplo de transformación XSL sinxela editada en NetBeans, que cando actúa sobre o exemplo XML anterior obtén unha páxina HTML cunha lista dos alumnos:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo sinxelo de transformación XSL -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="alumnos">
        <html>
            <head>
                <title>fundamentos.xml</title>
            </head>
            <body>
                <h1>Alumnos</h1>
                <ul>
                    <xsl:for-each select="alumno">
                        <li>
                            <xsl:value-of select="apelidos"/>,
                            <xsl:value-of select="nome"/>
                        </li>
                    </xsl:for-each>
                </ul>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Linguaxes de programación

Permiten comunicarse cos dispositivos hardware e así poder realizar un determinado proceso e para iso poden manexar estruturas de datos almacenados en memoria interna ou externa, e utilizar estruturas de control. Dispoñen dun léxico, e teñen que cumprir regras sintácticas e semánticas.

O léxico é o conxunto de símbolos que se poden usar e poden ser: identificadores (nomes de variables, tipos de datos, nomes de métodos, etcétera), constantes, variables, operadores, instrucións e comentarios.

A regras de sintaxe especifican a secuencia de símbolos que forman unha frase ben escrita nesa linguaxe, é dicir, para que non teña faltas de ortografía.

As regras de semántica definen como teñen que ser as construcións sintácticas e as expresións e tipos de datos utilizadas.

Exemplo de código Java sinxelo editado en NetBeans:

```
package parimpar;

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        int n;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Teclee un número enteiro");
        n = teclado.nextInt();
        if (n%2==0){
            System.out.println(n + " é par");
        }
        else{
            System.out.println(n+ " é impar");
        }
    }
}
```

2.2 Clasificación das linguaxes de programación

As linguaxes de programación poden clasificarse segundo o afastadas ou próximas que estean do hardware, por xeracións, polo paradigma da programación, pola forma de traducirse a linguaxe máquina e de executarse e na arquitectura cliente servidor.

Clasificación segundo a distancia ao hardware

Poden clasificarse en linguaxes de baixo e alto nivel.

Linguaxes de baixo nivel

Están baseados directamente nos circuítos electrónicos da máquina polo que un programa escrito para unha máquina non poderá ser utilizada noutra diferente. Poden ser linguaxe máquina ou linguaxe ensambladora.

A linguaxe máquina é código binario (ceros e uns) ou hexadecimal (números de 0 a 9 e letras de A a F) que actúa directamente sobre o hardware. É a única linguaxe que non necesita tradución xa que o procesador recoñece as instrucións directamente.

0	0	1	0	1	1	1	1	1	→	95
1	0	0	0	0	0	1	1	1	→	7
2	1	0	0	1	1	0	1	0	→	sumar
3	0	1	1	0	0	1	1	0	→	102

A codificación en linguaxe ensambladora é mnemotécnica, é dicir, utiliza etiquetas para describir certas operacións. É necesario traducir o código ensamblador a linguaxe máquina para que o procesador recoñeza as instrucións.

Exemplo:

```
.model small
.stack
.data
Cadenal DB 'Hola Mundo.$'
.code

programa:
mov ax, @data mov ds, ax
mov dx, offset Cadenal
mov ah, 9 int 21h mov ah,4ch int 21h
end programa
```

Como vantaxe dicir que é moi rápido, é a propia linguaxe do ordenador, pero como inconvenientes, é moi propenso a erros, complicado de programar, e emprégase só para programar tarefas moi concretas de certos microprocesadores.



Tarefa 1.7. Buscar en internet un anaco de código escrito en linguaxe ensambladora.

Linguaxes de alto nivel

Tentan acercarse á linguaxe humana e sepáranse do coñecemento interno da máquina e por iso necesitan traducirse a linguaxe máquina. Esta tradución fai que a execución sexa máis lenta que nas linguaxes de baixo nivel pero ao non depender do procesador poden executarse en diferentes ordenadores.

Clasificación por xeracións

Poden clasificarse en 5 xeracións.

- **Primeira xeración** (1GL) formada polas linguaxes de programación utilizadas nos primeiros ordenadores: linguaxe máquina e ensambladora.
- **Segunda xeración** (2GL) formada pola linguaxe macroensambladora que é a linguaxe ensambladora combinada con instrucións de control e de manexo de datos máis complexas. Estas linguaxes son específicas para unha familia de procesadores e o hardware relacionado con ela. Aínda se sigue utilizando para programar os núcleos (*kernel*) dos sistemas operativos e os controladores de algúns dispositivos (*device drivers*).
- **Terceira xeración** (3GL) formada pola maior parte das linguaxes de alto nivel actuais. O código é independente da máquina e a linguaxe de programación é parecida á linguaxe humana. Por exemplo, Java, C, C++, PHP, JavaScript, e Visual Basic.
- **Cuarta xeración** (4GL) formada por linguaxes e contornos deseñados para unha tarefa ou propósito moi específico como acceso a bases de datos, xeración de informes, xeración de interfaces de usuario, etcétera. Por exemplo, SQL, Informix 4GL, e Progress 4GL.
- **Quinta xeración** (5GL) formada por linguaxes nas que o programador establece o problema a resolver e as condicións a cumprir. Úsanse en intelixencia artificial, sistemas baseados

no coñecemento, sistemas expertos, mecanismos de inferencia ou procesamento da linguaxe natural. Por exemplo, Prolog, Smalltalk e Lisp.

Clasificación polo paradigma da programación

Un paradigma de programación é unha metodoloxía ou filosofía de programación a seguir cun núcleo central incuestionable, é dicir, as linguaxes que utilizan o mesmo paradigma de programación utilizarán os mesmos conceptos básicos para programar. A evolución das metodoloxías de programación e a das linguaxes van parellas ao longo do tempo. Poden clasificarse en dous grandes grupos: o grupo que segue o paradigma imperativo e o que segue o paradigma declarativo.

Paradigma imperativo

O código está formado por unha serie de pasos ou instrucións para realizar unha tarefa organizando ou cambiando valores en memoria. As instrucións execútanse de forma secuencial, é dicir, hasta que non se executa unha non se pasa a executar a seguinte. Por exemplo, Java, C, C++, PHP, JavaScript, e Visual Basic.

Dentro das linguaxes imperativas distínguese entre que as que seguen a metodoloxía estruturada e as que seguen a metodoloxía orientada a obxectos.

A finais dos anos 60 naceu a metodoloxía estruturada que permitía tres tipos de estruturas no código: a secuencial, a alternativa (baseada nunha decisión) e a repetitiva (bucles). Os programas están formados por datos e esas estruturas.

A metodoloxía estruturada evolucionou engadindo o módulo como compoñente básico dando lugar á programación estruturada e modular. Os programas estaban formados por módulos ou procesos por un lado e datos por outro. Os módulos necesitaban duns datos de entrada e obtiñan outros de saída que á súa vez podían ser utilizados por outros módulos. Por exemplo, C é unha linguaxe estruturada e modular.

Nos anos 80 apareceu a metodoloxía orientada a obxectos que considera o obxecto como elemento básico. Esta metodoloxía popularizouse a principios dos 90 e actualmente é a máis utilizada. Por exemplo, C++ e Java son linguaxes orientadas a obxectos. Cada obxecto segue o patrón especificado nunha clase. Os programas conteñen obxectos que se relacionan ou colaboran con outros obxectos da súa mesma clase ou doutras clases. A clase está composta de atributos (datos) e métodos (procesos) e pode ter as propiedades:

- Herdanza. Poden declararse clases filla que herdán as propiedades e métodos da clase nai. Os métodos herdados poden sobrecargarse, é dicir, dentro da mesma clase pode aparecer definido con distinto comportamento o mesmo método con diferente firma (número de parámetros e tipo dos mesmos).
- Polimorfismo. Propiedade que permite que un método se comporte de diferente maneira dependendo do obxecto sobre o que se aplica.
- Encapsulamento. Permite ocultar detalles internos dunha clase.

Paradigma declarativo.

O código indica que é o que se quere obter e non como se ten que obter, é dicir, é un conxunto de condicións, proposicións, afirmacións, restricións, ecuacións ou transformacións que describen o problema e detallan a súa solución. O programador ten que pensar na lóxica do algoritmo e non na secuencia de ordes para que se leve a cabo. Por exemplo, Lisp e Prolog son linguaxes declarativas.

Clasificación pola forma de traducirse a linguaxe máquina e executarse

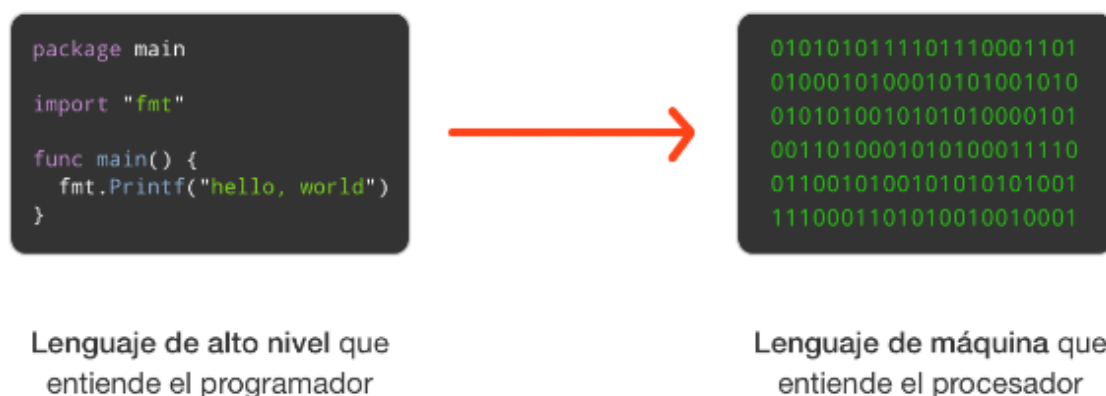
Chámase código fonte ao código escrito nunha linguaxe de programación simbólica mediante unha ferramenta de edición. Este código gárdase nun arquivo coñecido como arquivo fonte e terá que traducirse a linguaxe máquina para poder executarse. A tradución pode facerse mediante compiladores e/ou intérpretes.

Pode considerarse que hai tres grupos de linguaxes de programación tendo en conta a forma de traducirse a linguaxe máquina e de executarse: Linguaxes compiladas, interpretadas, e de máquina virtual ou execución administrada. Algunhas linguaxes como por exemplo Prolog poden ser consideradas como compiladas ou interpretadas xa que dispoñen de compiladores ou intérpretes.

Linguaxe compilada

Estas linguaxes dispoñen dun compilador ou programa que traduce o código fonte a código máquina creando o arquivo executable en tempo de compilación. En tempo de execución pode lanzarse o arquivo executable na plataforma para a que serve o compilador. Algunhas características deste tipo de linguaxes son:

- Existe un arquivo executable para a máquina real.
- O proceso de tradución faise en tempo de compilación e unha soa vez.
- A execución é moi rápida xa que o executable está en linguaxe máquina.
- O executable só funciona na plataforma para a que foi creado.
- O usuario que executa non ten que coñecer o código fonte, só ten o código executable que non é manipulable facilmente para obter o código fonte, e como consecuencia o programador ten o código fonte máis protexido.
- O executable non se xerará se existen erros léxicos, sintácticos ou semánticos.
- Interromper a execución pode ser difícil para o sistema operativo e pode afectar á plataforma.
- A modificación do código fonte implica volver a xerar o arquivo executable.



Linguaxe interpretada

Estas linguaxes precisan dun interprete ou programa en memoria para que en tempo de execución se vaia traducindo o código fonte a linguaxe máquina. Algunhas características deste tipo de linguaxes son:

- Non existe un arquivo executable.
- O proceso de tradución faise cada vez que se executa.
- A execución é lenta xa que ao proceso de execución ten que sumarse o de tradución.

- O arquivo pode executarse en diferentes plataformas sempre que haxa intérprete para elas.
- O usuario utiliza o código fonte para executar o programa.
- Os erros de tipo léxicos, sintácticos ou semánticos poden aparecer na execución.
- Interromper a execución só afecta normalmente ao intérprete e non á plataforma.
- Pode executarse en calquera plataforma sempre que haxa intérprete para ela.
- A modificación do código fonte non require ningunha operación extra antes de executar o programa.

Linguaxe de máquina virtual ou de execución administrada

Estas linguaxes precisan inicialmente dun compilador que en tempo de programación traduce o código fonte a un código intermedio multiplataforma, e a continuación necesita doutro software que traduza ese código intermedio a código máquina. No caso destes linguaxes como Java ocorre que:

- En tempo de compilación o compilador Java xera o código intermedio chamado *bytecode* Java independente da plataforma.
- En tempo de execución, necesita dunha máquina virtual Java (JVM) que interprete o bytecode. Esta máquina virtual é un software que simula unha máquina real co seu propio sistema operativo e que fai de intermediaria coa máquina real, polo que as máquinas virtuais son diferentes para Linux, Windows, Mac e Solaris.

No caso de linguaxes de execución administrada como as da plataforma .NET³ de Microsoft ocorre que:

- En tempo de compilación, o compilador xera o código CIL (*Common Intermediate Language*) independente da plataforma.
- Para executar é necesario ter no equipo cliente a versión de .NET Framework (e por tanto de [CRL](#)) adecuada e entón faise a última fase da compilación, na que o CRL traducirá o código intermedio a código máquina mediante un compilador JIT ([Just In Time](#)).

Algunhas características deste tipo das linguaxes de máquina virtual ou de execución administrada son:

- Existe un código intermedio que se executa nun software específico pero non é un executable.
- O proceso de tradución inicial faise antes da execución e o de tradución final faise cada vez que se executa.
- A execución é máis lenta que nas linguaxes compiladas pero máis rápida que nas interpretadas.
- O arquivo pode executarse en diferentes plataformas sempre que exista o software específico.
- O usuario non ten o código fonte, só o código intermedio que non é manipulable facilmente para obter o código fonte polo que o programador ten o código fonte máis protexido.

³ Esta plataforma é un conxunto de tecnoloxías de software que ten un contorno de traballo (.NET Framework) e varias linguaxes de programación como por exemplo VisualBasic.NET, C# ou C++. O contorno de traballo, incluído no sistema operativo Windows, inclúe un contorno de execución administrado chamado CRL (*Common Runtime Language*) que executa código e proporciona servizos que facilitan o proceso de desenvolvemento.

- Os erros de tipo léxico, sintáctico ou semántico detéctanse na fase de compilación, e se existen non se xerará o código intermedio.
- Interromper a execución só afecta normalmente ao intérprete e non á plataforma.
- A modificación do código fonte implica volver a repetir o proceso de compilación.



Clasificación na arquitectura cliente-servidor

A arquitectura cliente-servidor consiste basicamente nun programa cliente que realiza peticións a un servidor. Esta arquitectura é máis útil cando o cliente e o servidor están comunicados mediante unha rede, aínda que tamén se pode aplicar cando están na mesma máquina. Nesta arquitectura diferénciase entre linguaxes que se executan:

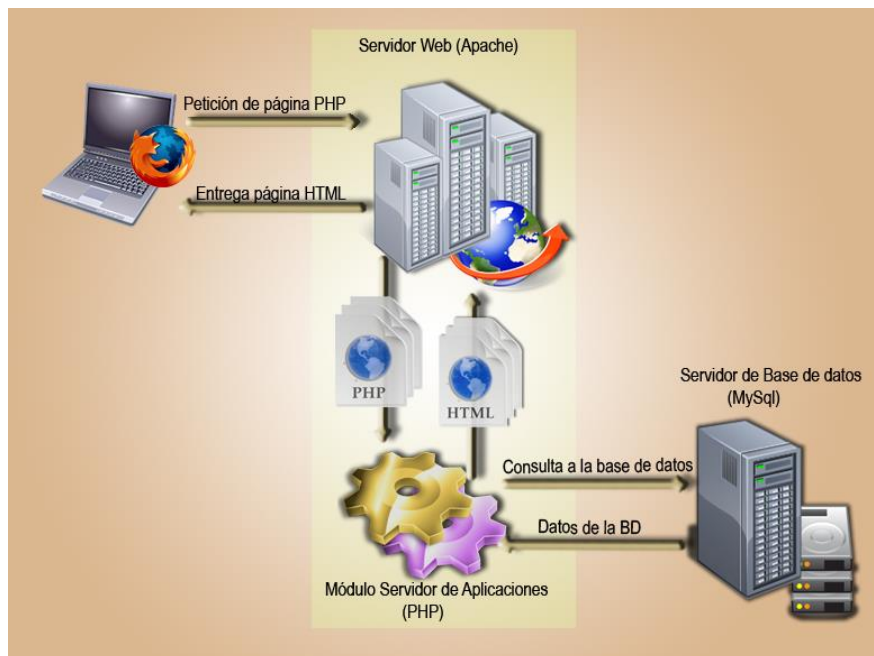
- do lado do servidor como PHP ou Java en formato de servlets / JSP
- do lado do cliente como JavaScript.

Para explicar isto pódese supoñer un navegador cliente e un servidor web con intérprete de PHP e a situación mostrada nos seguintes pasos:

- Un usuario dende un navegador cliente pide a un servidor Web unha páxina HTML que ten incrustado código PHP e código JavaScript.
- O servidor Web procesa a petición, interpreta o código PHP, execútao colocando o resultado da execución no sitio onde estaba o código PHP e devolve ao cliente unha páxina web con código HTML e JavaScript. O servidor Web debería de executar as ordes relativas á manexo dunha base de datos nun servidor de base de datos se o código PHP tivera ese tipo de instrucións.
- O navegador cliente interpreta o código JavaScript e mostra a páxina ao usuario.

O usuario pode interactuar coa páxina ocorrendo que cada vez que fai peticións ao servidor recárgase a páxina completamente coa resposta do servidor.

O usuario final nunca poderá ver o código PHP ou Java xa que o servidor web o transformou a HTML. Porén, si pode ver o código JavaScript, xa que este se executa no ordenador do usuario, non no servidor.



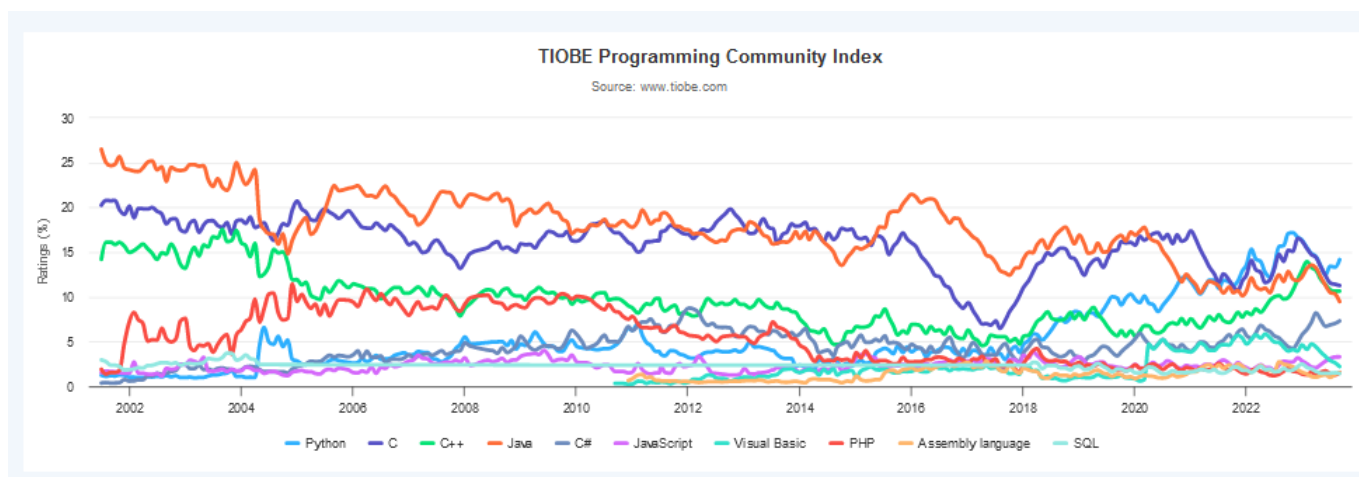
Existe unha técnica denominada **AJAX** (*Asynchronous JavaScript And XML*), que permite que JavaScript procese algún evento iniciado polo usuario facendo unha petición ao servidor que este responde co resultado en XML. JavaScript procesa o resultado XML actualizando seccións da páxina sen ter que recargala totalmente e logrando así unha interacción asíncrona entre servidor e cliente.

 **Tarefa 1.8. Visita a páxina de Instagram (ou outra) e comproba se ten código Javascript.**

2.3 Linguaxes máis difundidas

Para saber cales son as linguaxes máis difundidas poden consultarse os seguintes índices ou clasificacións:

- Índice Tiobe (<https://www.tiobe.com/tiobe-index/>) que basea a clasificación das linguaxes de programación no número de enxeñeiros cualificados en cada linguaxe en todo o mundo, cursos de linguaxes ofertados, provedores de terceiros, buscas realizadas en Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube y Baidu unha serie de premisas como por exemplo que a linguaxe exista como linguaxe de programación en Wikipedia e que teña polo menos 10000 visitas en Google.



- Índice PYPL ou *PopularitY of Programming Language index* (<http://pypl.github.io/PYPL.html>) que basea a clasificación das linguaxes de programación no análise da frecuencia de busca de tutoriais ou guías das linguaxes de programación en Google utilizando Google Trends.
- A [clasificación Redmonk](#) que non basea a clasificación nos buscadores senón nos proxectos albergados no repositorio GitHub e nas preguntas da web de StackOverflow orientada a programadores. Inclúe linguaxes informáticos e non só linguaxes de programación.

As linguaxes comúns a todos estes índices en setembro 2023 nos 10 primeiros postos son: Phtyton, C, C++, Java, C#, JavaScript, Visual basic, PHP, ASM y SQL.

Sep 2023	Sep 2022	Change	Programming Language	Ratings	Change
1	1		 Python	14.16%	-1.58%
2	2		 C	11.27%	-2.70%
3	4	▲	 C++	10.65%	+0.90%
4	3	▼	 Java	9.49%	-2.23%
5	5		 C#	7.31%	+2.42%
6	7	▲	 JavaScript	3.30%	+0.48%
7	6	▼	 Visual Basic	2.22%	-2.18%
8	10	▲	 PHP	1.55%	-0.13%
9	8	▼	 Assembly language	1.53%	-0.96%
10	9	▼	 SQL	1.44%	-0.57%
11	15	▲	 Fortran	1.28%	+0.26%
12	12		 Go	1.19%	+0.03%
13	14	▲	 MATLAB	1.19%	+0.13%
14	22	▲	 Scratch	1.08%	+0.51%
15	13	▼	 Delphi/Object Pascal	1.02%	-0.07%
16	16		 Swift	1.00%	+0.02%
17	26	▲	 Rust	0.97%	+0.47%
18	18		 R	0.97%	+0.02%
19	20	▲	 Ruby	0.95%	+0.30%
20	34	▲	 Kotlin	0.90%	+0.59%

C

C é unha linguaxe creada en 1972 nos laboratorios Bell para codificar o sistema operativo UNIX. Está catalogada como unha linguaxe de alto nivel, estruturada e modular, pero con moitas características de baixo nivel como por exemplo utilizar punteiros para facer referencia a unha posición física da memoria RAM. Estas características posibilitan traballar moi cerca da máquina

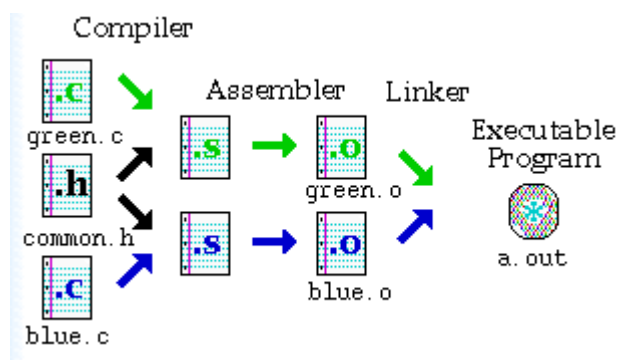
pero os programas son máis complicados e propensos a ter máis erros. Influíu no deseño das linguaxes C++, C#, Java, PHP, JavaScript entre outros. Exemplo de código C:

```
/* Programa C que suma os números enteiros do 1 ó 20
 */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
    int i;                /* contador de números enteiros */
    long int suma;        /* sumador dos números enteiros */
    int final;  suma = 0;
    for (i = 1; i <=20; i++) {
        suma = suma + i;
    }
    printf("Programa C\nA suma total e:  %ld\n", suma);
    printf("\nTeclee calquera numero para finalizar...");
    scanf("%d",&final);
    fflush(stdin);
    return (EXIT_SUCCESS);
}
```

C++

Deseñada en 1980 por Bjarne Stroustrup para estender C con mecanismos que permitan a POO (programación orientada a obxectos). Exemplo de código C++ editado en NetBeans:

```
/*
 * Programa C++ que suma os números enteiros do 1 ó 20
 */
#include <iostream>
int main(int argc, char**argv) {
    int i;                /* contador de números enteiros */
    long int suma;        /* sumador dos números enteiros */
    suma = 0;
    for (i = 1; i <=20; i++) {
        suma = suma + i;
    }
    std::cout <<"A suma total e "<< suma<< std::endl;
    return 0;
}
```



Java

É unha linguaxe de programación orientada a obxectos desenvolvida por Sun Microsystems en 1995. Toma moita da súa sintaxe de C e C++ pero cun modelo de obxectos máis simple e elimina as ferramentas de baixo nivel que se utilizaban en C. Exemplo de código Java editado en NetBeans:

```
package sumaenteiros;
/*
 * File: Main.java
 * Author: profesor
 * Date: 14/08/2020 12:25:00
 */
```

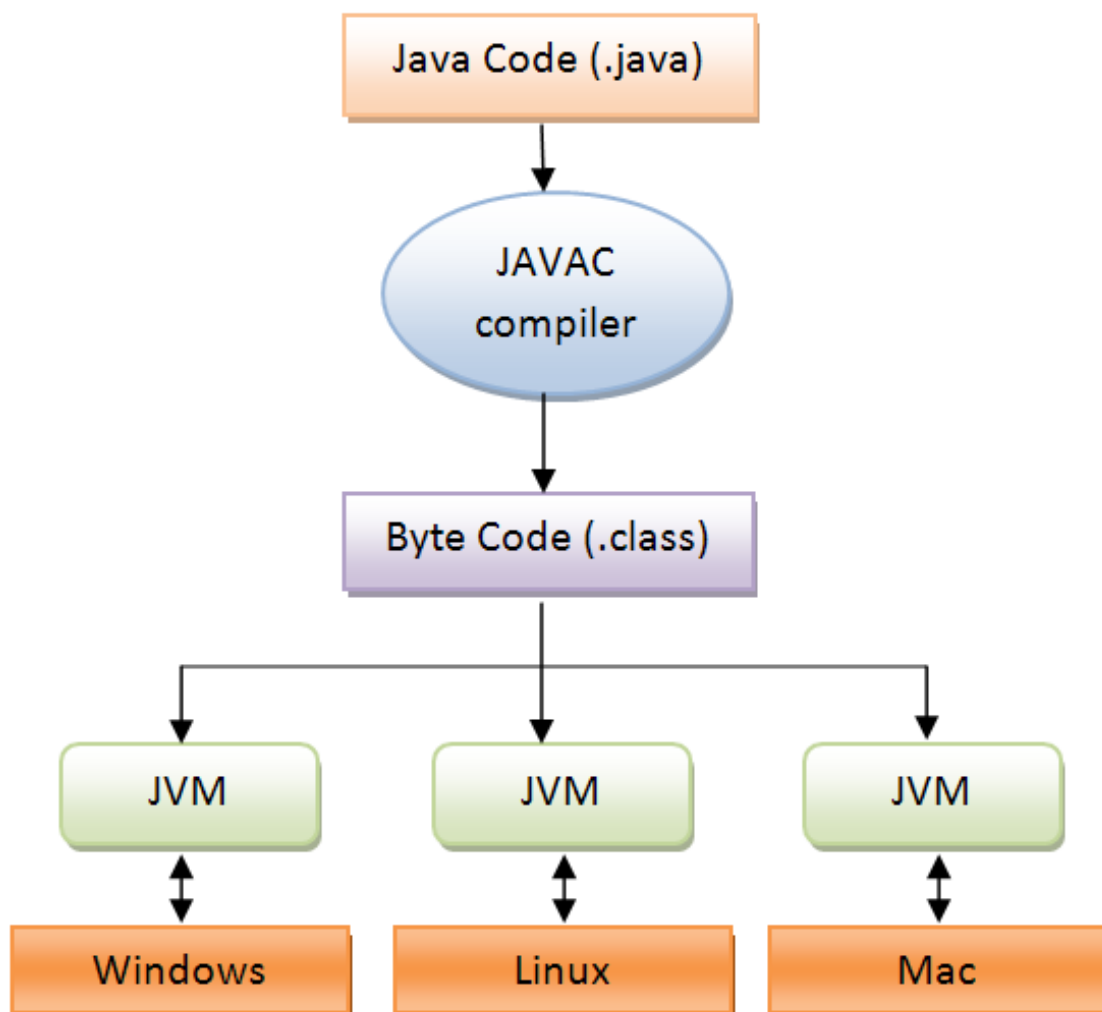
```

/* Obxectivo: visualizar a suma dos 20 primeiros números naturais
*/
public class Main {

    public static void main(String[] args) {
        int suma=0;        /* sumador dos números enteiros */
        for (int i=1;i<=20;i++)
        {
            suma=suma+i;
        }
        System.out.println("Exemplo Java");
        System.out.println("Suma dos 20 primeiros números naturais = " + suma);
    }
}

```

Non segue un modelo nin compilado nin interpretado. Ten unha primeira compilación que xera un código intermedio, case executable. Logo un intérprete (á máquina virtual Java) executa liña a liña ese código. Consigue así independencia da plataforma (cada plataforma terá que desenvolver a súa máquina virtual, pero o código fonte valerá para todas elas).



Como se ve na imaxe anterior o arquivo final a executar será o `.class`, este é o bytecode que interpreta a máquina virtual. En realidade, as aplicacións Java distribúense en arquivos de tipo `.jar`

que son arquivos comprimidos que inclúen no soamente os bytecode se non o resto de arquivos que compoñen a aplicación: imaxes, audios, etc.

É a linguaxe máis empregada actualmente, en moita variedade de dispositivos distintos. A idea da máquina virtual permite que un programa escrito en Java poida ser executado en distintos elementos hardware moi diferentes.

C#

É unha linguaxe de programación orientada a obxectos desenvolvido por Microsoft no ano 2000 como parte da plataforma .NET. Visual C# proporciona un editor de código avanzado, deseñadores de interface de usuario, e numerosas ferramentas para facilitar o desenvolvemento de aplicacións en C# e .NET Framework. Exemplo de código C# creado con Visual Studio 2013 e editado con Notepad++:

```
using System;
namespace exemplo_csharp_consola{
    class Program    {
        static void Main(string[] args)    {
            Console.WriteLine("Hola Mundo");
        }
    }
}
```

PHP

PHP (*Hypertext PreProcessor*) é unha linguaxe interpretada do lado do servidor que se pode usar para crear calquera tipo de programa pero que onde ten máis popularidade é na creación de páxinas web dinámicas. O código PHP soe estar incrustado en código HTML e precisa dun cliente que fai unha petición a un servidor e dun servidor web que o executa a petición. Foi deseñada por Rasmus Lerdorf en 1995 e actualmente segue sendo desenvolvido polo grupo PHP (<http://php.net/>). Exemplo de código PHP incrustado nunha páxina HTML editado con Notepad++:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8">
        <title>Exemplo de php</title>

    </head>
    <body>
        <?php
            $suma=0;
            for ($i=1;$i<=20;$i=$i+1)
            {
                $suma=$suma+$i;
            }
            echo "<p>A suma total é: ".$suma."</p>";
        ?>
    </body>
</html>
```

Python

É unha linguaxe de programación interpretado que permite a programación orientada a obxectos, cunha sintaxe moi limpa que favorece que o código sexa lexible. Foi deseñado por Guido Van

Rossum en 1991 e actualmente é administrado pola *Python Software Foundation* (<http://www.python.org/>). Posúe unha licenza de código aberto denominada *Python Software Foundation License 1* compatible coa licenza pública xeral de GNU a partir da versión 2.1.1. Exemplo de código Python editado en Notepad++:

```
# suma os enteiros do 1 ao 20

def sumarEnteiros(n):
    sum=0
    for i in range(1,n+1):
        sum=sum+i

    return sum

A=sumarEnteiros(20)

print "Suma total de enteiros do 1 ao 20 : " + str(A)
```

JavaScript

É unha linguaxe de programación dialecto do estándar ECMAScript. Defínese como orientado a obxectos, baseado en prototipos, debilmente tipado (declaración de tipos) e dinámico. Utilízase normalmente no lado do cliente (*client-side*) e está implementado como parte dun navegador web aínda que tamén existe un JavaScript do lado do servidor (*Server-side JavaScript* ou SSJS). Exemplo de código JavaScript editado en Notepad++:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo de JavaScript</title>
    <meta charset= "utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Números naturais impares ata 9</h1>
    <script type="text/javascript">
      <!--
      var i;
      for(i=1;i<=10;i+=2)
        document.write(i+" ");
      // -->
    </script>
  </body>
</html>
```

2.4 Proceso de xeración de código

A xeración de código consta dos procesos de edición, compilación, e enlace. Cando o código estea finalizado, poderase executar para producir resultados.

Edición

Esta fase consiste en escribir o algoritmo de resolución nunha linguaxe de programación mediante un editor de texto ou unha ferramenta de edición incluída nun contorno de desenvolvemento. O código resultante chámase código fonte e o arquivo correspondente chámase arquivo fonte.

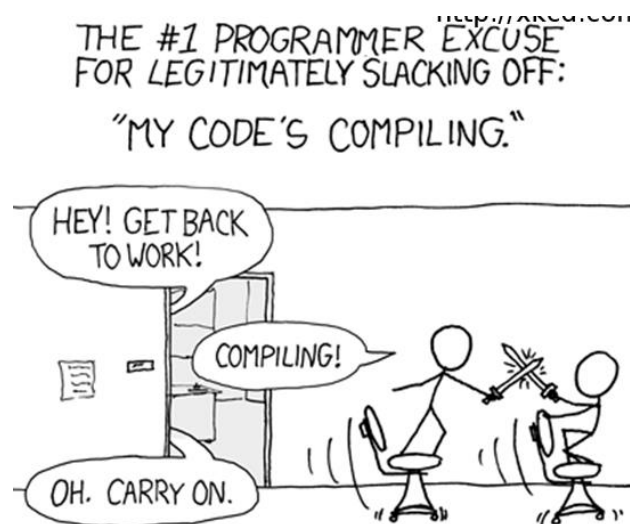
Aínda que o arquivo ten extensión .java, .c, .py, etc. dependendo da linguaxe, sempre é texto sen formato (txt).

Editores moi empregados (aparte dos contornos de desenvolvemento) poden ser DevC++ ou Code::Blocks para C, Pycharm para Python, ou cun carácter máis xeral: Sublime Text. Estes editores facilitan a edición completando código, resaltando con diferentes cores as distintas estruturas da linguaxe e algúns deles permiten facer a compilación e execución desde o mesmo editor.

Compilación

Consiste en analizar e sintetizar o código fonte mediante un compilador, para obter, se non se atopan erros, o código obxecto ou un código intermedio multiplataforma. As persoas non entenden ese código e non se pode executar directamente.

Esta fase non se aplicará ás linguaxes interpretadas aínda que estas poden ter ferramentas que permitan facer un análise léxico e sintáctico antes de pasar a executarse.



Análise

As análises realizadas son:

- Análise léxico no que se comproba que os símbolos utilizados sexan correctos, incluídos os espazos en branco.
- Análise sintáctico no que se comproba que as agrupacións de símbolos cumpran as regras da linguaxe.
- Análise semántico no que se fan o resto de comprobacións, como por exemplo, que as variables utilizadas estean declaradas, ou a coherencia entre o tipo de dato dunha variable e o valor almacenado nela, ou a comparación do número e tipo de parámetros entre a definición e unha chamada a un método.

Síntese

A síntese permite:

- A xeración de código intermedio independente da máquina. Algunhas linguaxes compiladas como C pasan antes por unha fase de preprocesamento na que se levan a cabo operacións como substituír as constantes polo valor ou incluír arquivos de cabeceira. Outras

linguaxes como Java xeran *bytecode Java* que xa poderá ser executado nunha JVM e outras como C# xera o código CIL que será executado no contorno CLR.

- A tradución do código intermedio anterior a código máquina para obter o código obxecto. Esta tradución leva consigo tamén unha optimización do código. Este novo código aínda non está listo para executarse directamente.

Enlace

Esta fase consiste en enlazar mediante un programa enlazador o arquivo obxecto obtido na compilación con módulos obxectos externos para obter, se non se atopan erros, o arquivo executable.

O arquivo obxecto obtido na compilación pode ter referencias a códigos obxecto externos que forman parte de bibliotecas⁴ externas estáticas ou dinámicas:

- Se a biblioteca é estática, o enlazador engade os códigos obxecto das bibliotecas ao arquivo obxecto, polo que o arquivo executable resultante aumenta de tamaño con relación ao arquivo obxecto, pero non necesita nada máis que o sistema operativo para executarse.
- Se a biblioteca é dinámica (*Dinamic Link Library* - DLL en Windows, *Shared objects* en Linux), o enlazador engade só referencias á biblioteca, polo que o arquivo executable resultante apenas aumenta de tamaño con relación ao arquivo obxecto, pero a biblioteca dinámica ten que estar accesible cando o arquivo executable se execute.

Execución

A execución necesita de ferramentas diferentes dependendo de se a linguaxe é interpretada, compilada ou de máquina virtual ou execución administrada.

Se a linguaxe é interpretada, será necesario o arquivo fonte e o intérprete para que este baia traducindo cada instrución do arquivo fonte a linguaxe máquina (análise e síntese) e executándoa. Por Exemplo: PHP.

Se a linguaxe é compilada será necesario o arquivo executable, e nalgúns casos tamén se necesitan bibliotecas dinámicas. Por exemplo: C.

Se a linguaxe precisa de máquina virtual, será necesario ter o código intermedio e a máquina virtual para que esta vaia traducindo o código intermedio a linguaxe máquina e executándoo. Por exemplo: Java ou os programas para a plataforma Android. Estes últimos utilizan Java (adaptado) e a máquina virtual Dalvik (DVM) ou a máquina ART (*Android Runtime*) que teñen unha estrutura distinta a JVM, para interpretar o código intermedio.

Contornos de Desenvolvemento Integrado IDE

Os contornos de desenvolvemento integrados (Integrated Development Environment) son aplicacións que facilitan as tarefas descritas previamente. Soportan moitas linguaxes de programación distintas, ben de forma nativa, ben mediante plugins. Inclúen as seguintes funcións:

- Editor de código fonte con axudas como resaltar en cores distintas estruturas de código, completando texto automaticamente, detectando erros sintácticos, etc.

⁴ As bibliotecas (*library*) son coleccións de códigos obxecto que tratan un tema común. As linguaxes teñen como mínimo a biblioteca estándar. Poden existir outras librerías como por exemplo unha para gráficos. Os programadores dispoñen de ferramentas para crear novas bibliotecas.

- Depuradores de código (debugger) permitindo executar paso a paso o código escrito, permitindo observar o estado dos datos e variables do programa en cada momento.
- Utilidades para a xeración de interfaces gráficos de forma sinxela e visual.
- Control de versións do programa desenvolvido.
- Ferramentas adicionais para facilitar outras tarefas: migracións, probas, etc.

Os máis populares dos que soportan múltiples linguaxes son Netbeans, Eclipse e Microsoft Visual Studio.



Tarefa 1.9. Elaborar unha presentación na que se describa brevemente a diferenza entre linguaxes compilados, interpretados e semicompilados (baseándose en un programa sinxelo que sume dous números, en C, Python e Java respectivamente) con capturas de pantalla mostrando o software implicado nos tres casos tanto e vendo o proceso completo para a edición, compilación se é o caso, execución, nos tres casos, sobre os exemplos propostos.

Axuda para a tarefa

- Empregar unha máquina virtual con Windows e usar Notepad++ como editor para as tres linguaxes.
- Pedir ao profesor os exemplos de código en Java, C e Python, así como o software necesario.
- Para compilar e executar o exemplo en Java, usar *jdk*. Primeiro descomprimir jdk no disco duro, cambiar a variable PATH engadindo o cartafol bin do jdk. Finalmente desde unha ventá de comandos: `javac exemplo.java` para obter `exemplo.class` (bytecode). Logo, para executalo: `java exemplo` (sen `.class`). Xa non precisaríamos `exemplo.java`.
- Para compilar o exemplo en C, instalar MinGW sobre `c:\` (non en Program Files). Unha vez rematada a instalación, seleccionar os paquetes MinGW Base > Compiler Suite e instalalos. Engadir MinGW/bin á variable PATH. Rematado isto, compilamos co comando `gcc exemplo.c -o exemplo.exe`. Para executar simplemente necesitamos o arquivo final obtido: `exemplo.exe`.
- Para executar o script en Python, instalamos Python (na instalación xa temos un check para que el mesmo actualice a variable PATH). Executamos o código mediante o intérprete: `python exemplo.py`
- Na presentación debe quedar claro en que casos é necesario o código fonte para a execución, ou en que casos hai un executable, etc. e tamén a vantaxe do bytecode para acadar a independencia da plataforma (filosofía WORA: *Write once, run anywhere*)

2.5 Frameworks

Un framework (Plataforma, entorno, marco de traballo de desenvolvemento rápido de aplicacións) ao programador, en base á que podemos desenvolver proxectos sen partir desde cero. etc.

Trátase dunha plataforma software onde están definidos programas soporte, bibliotecas, linguaxe interpretado, que axuda a desenvolver e unir os diferentes módulos ou partes dun proxecto.

Co uso do framework podemos pasar máis tempo analizando os requirimentos do sistema e as especificacións técnicas da nosa aplicación, xa que a tarefa laboriosa dos detalles de programación queda resolta.

- Vantaxes de empregar un framework:
 - **Desenvolvemento rápido** de software.
 - **Reutilización** de partes de código para outras aplicacións.
 - **Deseño** uniforme del software.
 - **Portabilidade** de aplicacións dun computador a outro, xa que os bytecodes que se xeran a partir da linguaxe fonte poderán ser executados sobre calquera máquina virtual.
- Inconvenientes:
 - Gran dependencia do código respecto ao framework utilizado (sen cambiamos de framework, haberá que reescribir gran parte da aplicación).
 - A instalación e implementación do framework no noso equipo consume recursos do sistema.



Tarefa 1.11. Busca en Infojobs os frameworks máis demandados en Java, JavaScript e PHP e fai unha pequena descrición de cada un deles.