1. O software e os proxectos de desenvolvemento de software

1.1 Introdución

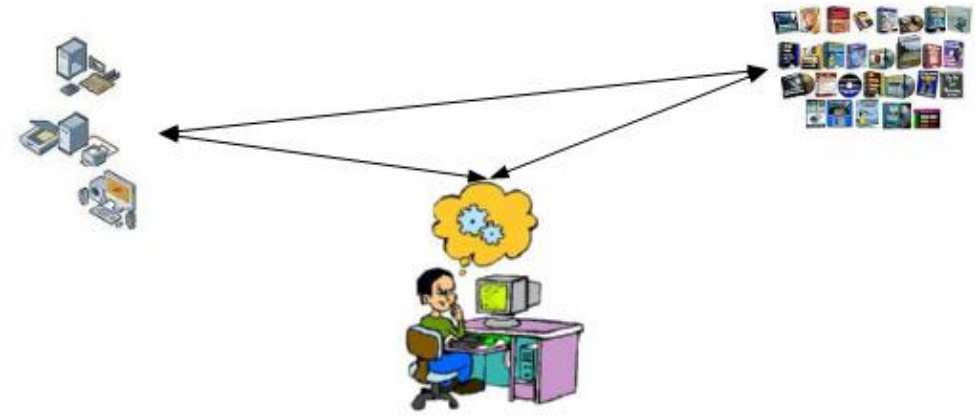
Nesta parte da unidade didáctica preténdense os seguintes obxectivos:

- Identificar algoritmo, software, versións, software de aplicación, software de sistema, hardware e recoñecer a relación entre eles.
- Identificar e caracterizar as fases de desenvolvemento de software no modelo en cascada, en espiral, nas metodoloxías áxiles e en Métrica 3.

1.2 O 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:

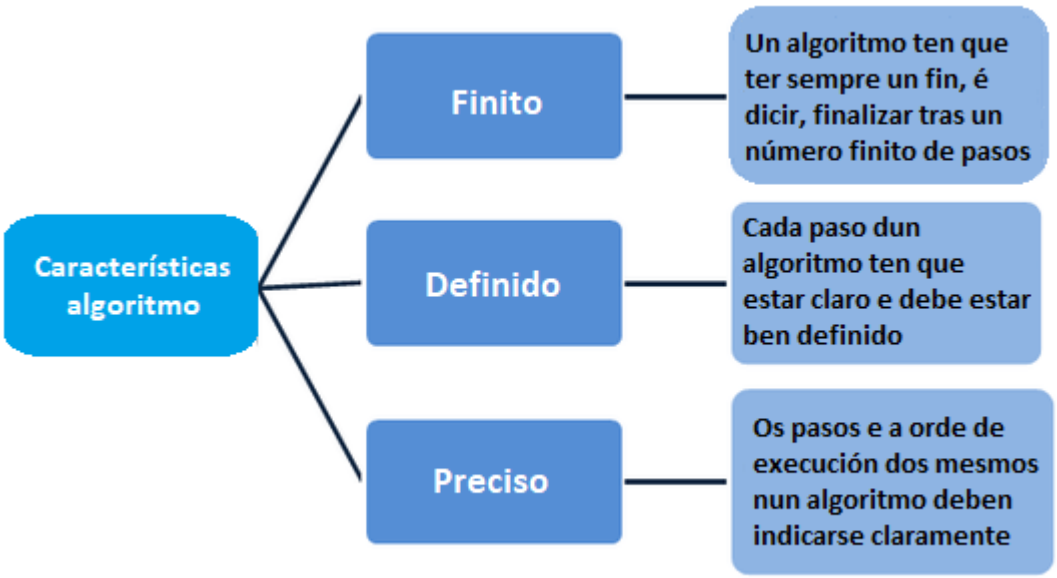
- **Soporte físico** ou hardware formado por todos os compoñentes electrónicos tanxibles involucrados no tratamento. Segundo a Real Academia Galega é máis aconsellable dicir soporte físico que hardware e defíneo como a maquinaria ou elementos que compoñen un ordenador.
- **Soporte lóxico** ou **software** 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.



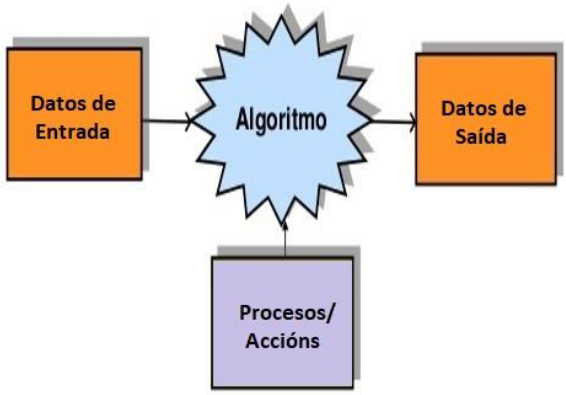
O concepto de programa informático está moi ligado ao concepto de **algoritmo**. Un algoritmo é un conxunto de instrucións ou regras ben definidas ordenadas e finitas que permite resolver un problema.

As características fundamentais que debe cumprir todo algoritmo son:

- **Un algoritmo debe ser preciso** e indicar a orde de realización de cada paso.
- **Un algoritmo debe estar definido**: Se seguimos o algoritmo para a mesma entrada varias veces os resultados obtidos deben ser os mesmos.
- **Un algoritmo debe ser finito**: O algoritmo debe contar cun número finito de pasos.



Na definición dun algoritmo debemos distinguir tres partes: a entrada de datos, o proceso dos mesmos e a devolución de resultados.



Na nosa vida diaria facemos uso de algoritmos continuamente para unha multitude de cousas, desde conducir a operacións matemáticas de uso común. Gran parte da nosa aprendizaxe realizámola mediante a memorización dos algoritmos que nos permiten resolver tipos de problema concretos (sumar, restar, multiplicar, dividir, sacar raíces, elaborar unha tortilla de patacas, etc.). Se nos fixamos ben, en todos estes casos aprendemos unha serie de pasos que aplicamos mecanicamente para chegar ao resultado.

Algoritmo que permita obter un refresco dunha máquina automática expendedora de bebidas embotelladas.

1. Inicio
2. Localizar a bebida desexada e mirar o seu custo
3. Introducir no sitio correspondente unha cantidade maior ou igual ao custo, preferiblemente igual
4. Pulsar o botón correspondente coa bebida
5. Si a máquina non ten produto
    a) No panel visualizarase produto esgotado
    b) Retornará o importe introducido

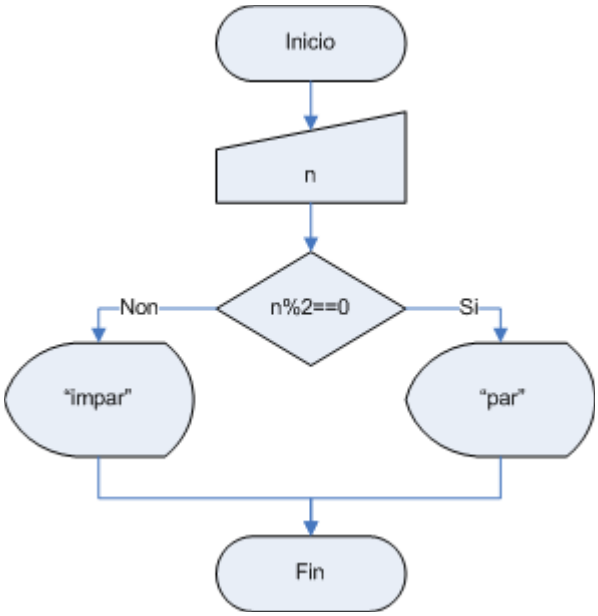
   SiNon

   Si a cantidade introducida é menor que o custo
    a) No panel visualizarase importe insuficiente
    b) Retornará o importe introducido

   SiNon
    a) Na bandexa de saída sairá o produto seleccionado

   Si a cantidade é maior que o custo
    a) Retornarase a diferenza de custo na bandexa de cambio
6. Fin

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 **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) {
        short n;
        Scanner teclado = new Scanner(System.in);
        System.out.printf("Teclea un número enteiro entre %d e %d:",
      Short.MIN_VALUE, Short.MAX_VALUE);
        n = teclado.nextShort();
        if(n%2==0){
            System.out.printf("%d é par\n",n);
        }
        else{
            System.out.printf("%d é impar\n",n);
        }
    }
}
```

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 periféri-

cos 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 do 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:

```
run:
Teclee o número enteiro entre -32768 e 32767:11
11 é impar
BUILD SUCCESSFUL (total time: 7 seconds)
```

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.

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, rastrexo de zonas defectuosas en soportes,...)
- **Software de aplicación**. Permite levar a cabo unha ou varias tarefas especificas 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 - Integrated Development Environment) 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 - Graphical User Interface).

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 pode utilizar nun contorno específico como por exemplo a contabilidade feita a medida para unha empresa en concreto.

1.4 Desenvolvemento de software

A finais dos anos 1950 e principios dos anos 1960, a potencia computacional das máquinas era bastante limitada. Por este motivo, os **programas** que se desenvolvían eran bastante **simples** tendo en conta o punto de vista actual. O desenvolvemento de software era, nese momento, unha tarefa artesanal e o seu mantemento era moi custoso. **Programar** era unha **arte** para a que se nacía. Os usuarios das computadoras da década de 1950 eran na súa maioría científicos e grandes empresas que con frecuencia tiñan que escribir o seu propio software. **Non** existía unha **metodoloxía** ou camiño a seguir para o desenvolvemento de software.

Na década de 1960, o mundo da informática experimentou o que se denominou unha **crise de software** cando os desenvolvedores de software non puideron construír o software que se lles pedía. A demanda de software aumentaba pero a capacidade do software e os desenvolvedores era limitada. Os desenvolvedores non puideron manterse ao día coa complexidade dos proxectos nos que se lles pediu que traballaran. A isto axudou o feito de que a potencia das máquinas aumentou de forma considerable.

Entre as características do software daquel tempo estaban:

- Custes por riba do presupostado
- Retrasos nas entregas
- Prestacións inadecuadas
- Mantemento case imposible
- Modificacións prohibitivas
- Falta de fiabilidade

É posible que agora teñamos máis solucións que nos anos 60, pero os problemas indicados seguen a ser problemas comúns do software.

A crise do software pode definirse como a dificultade de escribir programas libres de defectos, facilmente comprensibles e que sexan verificables. As conferencias da OTAN de 1968 e 1969 sobre desenvolvemento de software chegaron á conclusión de que era necesario dar un enfoque de **enxeñaría** no desenvolvemento de software.

A norma IEEE 1993 define a Enxeñaría de software como a aplicación dun enfoque sistemático, disciplinado e medible ao desenvolvemento, funcionamento e mantemento do software.

A **enxeñaría do software** ten como obxectivos:

- Desenvolver software de calidade
- Aumentar a produtividade
- Diminuír o tempo de desenvolvemento
- Desenvolver software económico

A enxeñaría do software é algo máis que programar. Comeza bastante antes de escribir liñas de código e continúa despois de que se ten completado a primeira versión do produto.

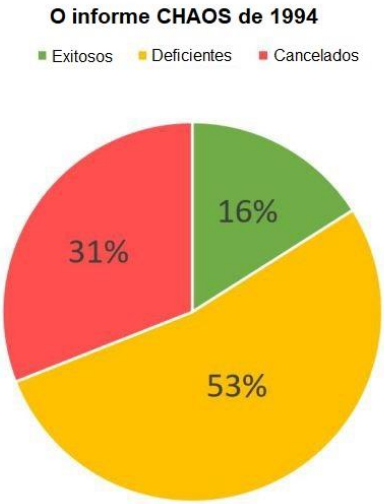
O proceso, a administración, a planificación, o seguimento e o control rigoroso do proxecto é esencial.

A crise do software é máis unha **enfermidade crónica** que unha crise puntual.

Existen varios estudos ou estatísticas sobre o estado dos proxectos de software, sendo o máis utilizado o informe de Standish Group, chamado **Chaos Report**. Este informe é a estatística de referencia máis citada en enxeñaría do software.

Entre estes informes Chaos, o máis famoso é o publicado en 1994, e que de modo resumido informaba de que:

- O 31% dos proxectos se cancelaron.
- O 53% tiñan deficiencias.
- O 16% foron un éxito.
- De media os proxectos tiñan un 189% de sobrecustes.
- De media os proxectos tardaban en realizarse un 222% sobre o tempo orixinal estimado.



No ano 2015 o informe CHAOS analizou 50.000 proxectos en todo o mundo, desde pequenas melloras ata implementacións masivas de reenxeñaría de sistemas.

Os resultados amosan que aínda queda traballo por facer para lograr resultados satisfactorios nos proxectos de software.

EVOLUCIÓN DOS INFORMES CHAOS

	1994	2000	2006	2012	2015
Exitosos	16%	28%	35%	27%	29%
Deficientes	53%	49%	46%	56%	52%
Fracasos	31%	23%	19%	17%	19%

O informe mide o éxito dos proxectos só en base a se remataron en **tempo**, **presuposto** e cumpriron cos **requisitos**, deixando fora aspectos como a calidade, o risco e a satisfacción do cliente.

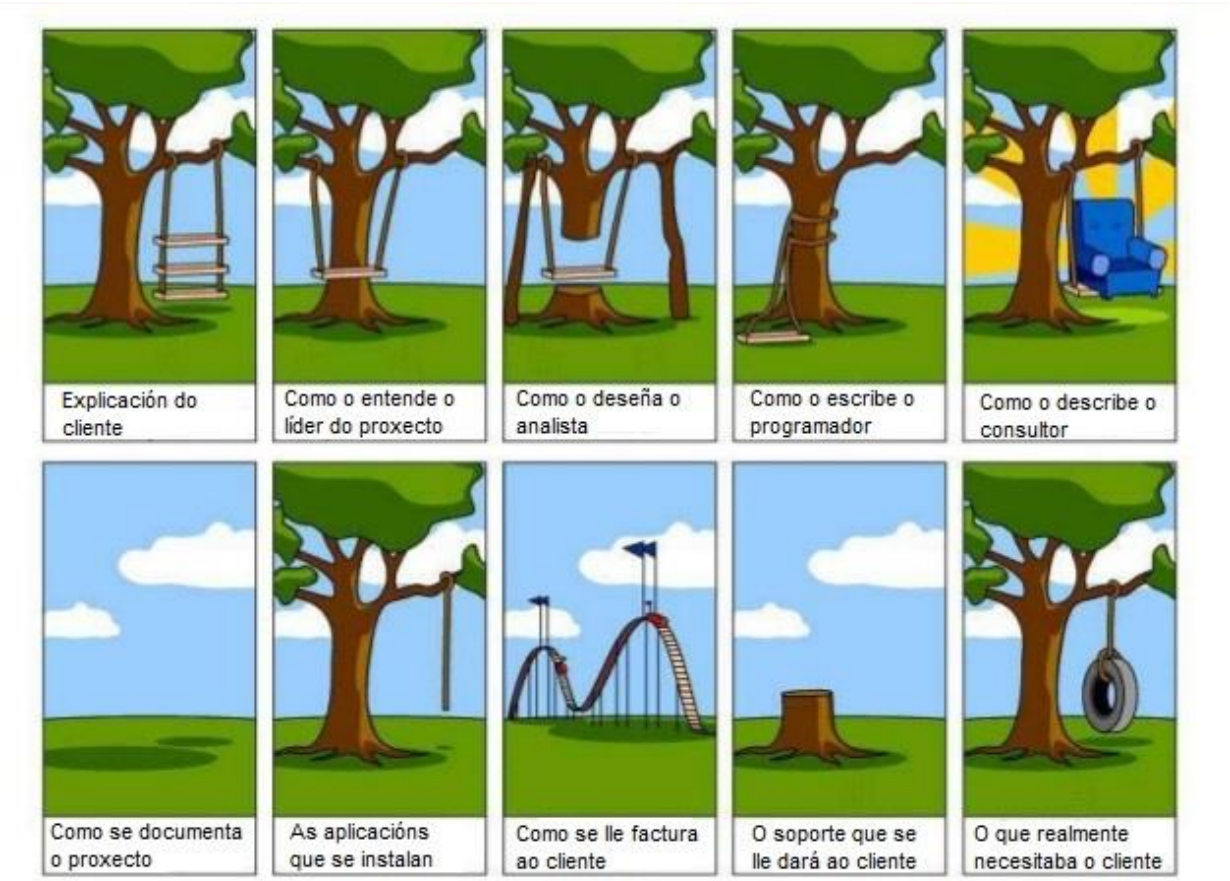
Buscando algún patrón para determinar como maximizar o éxito dos proxectos e minimizar o fracaso, dentro do informe CHAOS amósanse estas porcentaxes segmentadas polo tamaño dos proxectos. O resultado é claro. Un proxecto **pequeno** ten máis probabilidades de éxito.

INFORME CHAOS SOBRE O TAMAÑO DOS PROXECTOS

	ÉXITO	DEFICIENTE	FRACASO
Enorme	2%	7%	17%
Grande	6%	17%	24%
Medio	9%	26%	31%
Moderado	21%	32%	17%
Pequeno	62%	16%	11%
TOTAL	100%	100%	100%

O informe mostra resultados de proxectos realizados entre os anos 2011 e 2015

Na seguinte imaxe trata de mostrarse a gran problemática do desenvolvemento de software para un proxecto de construción dunha randeeira.



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 facer 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.

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.

O **ciclo de vida do software** é unha sucesión de **etapas** polas que pasa o software no seu desenvolvemento, desde que se concibe a idea ata que o software deixa de utilizarse (obsolescencia).

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, os métodos de programación áxil, e a Métrica 3 como modelo a aplicar en grandes proxectos relacionados coas institucións públicas.

1.5 Modelos de ciclo de vida

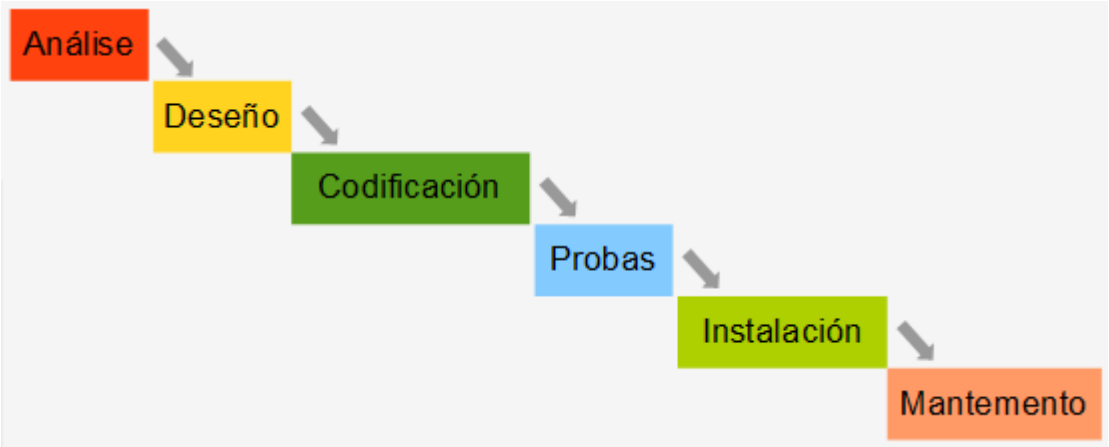
Paradigma de ciclo de vida clásico ou modelo en cascada

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.

Este modelo que tamén se denomina ciclo de vida **predictivo**, foi o primeiro método amplamente utilizado na industria do software. Aínda se utiliza a día de hoxe cando os requi-

sitos e características dun software poden ser claramente definidos durante a fase conceptual.



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 ERS 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.

A primeira fase do modelo en cascada inclúe tamén unha análise da definición dos requisitos.

Deseño

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) 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 incluirá o deseño lóxico de da-

tos, 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 construirá o software poden ser: modelo E/R para os datos, diagramas UML de clases, diagramas UML de despregamento e diagramas UML de secuencia.

Nesta fase debe tratarse 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 implementación[1]. 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 todas as sentenzas 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.

[1] Segundo a Real Academia Galega da lingua, implementar é aplicar ou poñer en funcionamento unha medida, un plan ou un programa.. 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.

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 documento de 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 non está acabado e entón noméase co nome comercial e un texto que indica o nivel de acabado. Ese texto pode ser:

- **Versión Alfa**. Versión inestable, á que aínda se lle poden engadir novas características.
- **Versión Beta**. Versión inestable á que non se lle van a engadir novas características pero que pode ter erros.
- **Versión RC** (Release Candidate) é case a versión final pero aínda poden aparecer erros.
- **Versión RTM** (Release To Manufacturing). Versión estable para comercializar.

Instalación

Esta fase tamén se denomina despregamento 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 complexa 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 os erros atopados, 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 aplicacións 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) tamén para Windows Server 2008 R2.

Documentación

A creación de documentación está asociada a todas as 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.

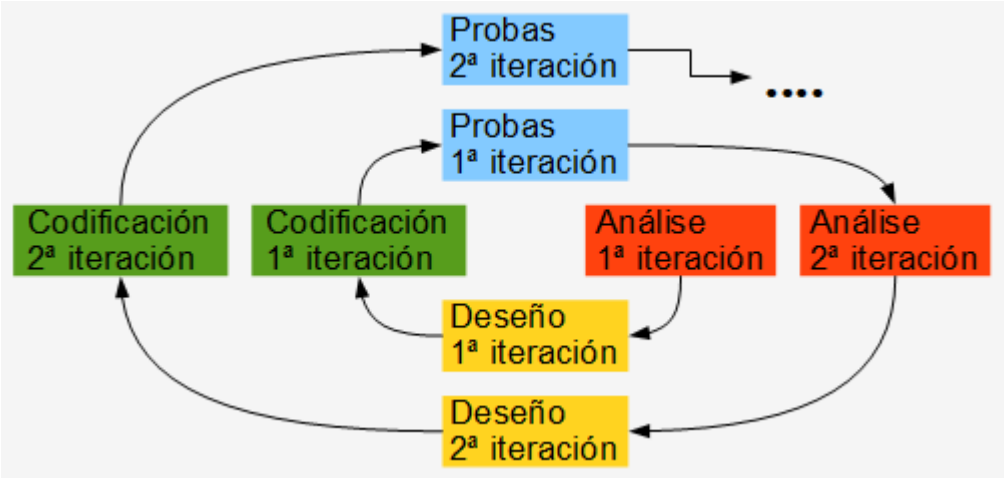
  Exemplo de código Java con comentarios Javadoc que aportan información extra e permiten que algunhas aplicacións xeren automaticamente unha páxina web coa información que extraen dos comentarios:

```
/** 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 anexa 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 **usuarios**. 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.

Modelo en espiral

Este modelo, tamén coñecido como modelo **incremental**, 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 analizarán o prototipo conseguido e acordarán se inician unha nova iteración. Sempre se traballa sobre un prototipo polo que no momento en 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.



Modelo adaptativo ou áxil

Os ciclos de vida adaptativos, tamén coñecidos como métodos orientados ao cambio ou métodos áxiles, responden a niveis altos de cambio e á participación continua dos interesados.

Xeralmente óptase polos métodos áxiles en contornos que cambian rapidamente, cando o alcance é confuso ou cando a aportación de valor é moi cambiante e con equipos altamente involucrados. Hai que ter en conta que, nun contorno cada vez máis dinámico e competitivo, as empresas sen capacidade de reacción e adaptación non teñen futuro. As metodoloxías áxiles propoñen un proceso de co-creación no que se involucra ao cliente no proceso de desenvolvemento do produto, gañando en flexibilidade e inmediatez.

Estas metodoloxías teñen a súa orixe no **Manifesto Axil**, promovido por 17 críticos dos modelos de desenvolvemento de software tradicionais. As catro grandes ideas do manifesto áxil son:

- **Individuos e interaccións** por riba de procesos e ferramentas.
- **Software funcional** por riba de documentación extensiva.
- **Colaboración co cliente** por riba de negociación contractual.
- **Resposta ante o cambio** por riba de seguir un plan.

Dependendo da natureza do proxecto e do grado de madurez da empresa no uso de Agile, pode utilizarse unha metodoloxía ou outra, sendo as máis comúns: Extreme Programming, Scrum e Kanban.

Programación eXtrema

A programación eXtrema ou eXtreme Programming é un método de desenvolvemento áxil de software baseado en iteracións ou fases sobre un ciclo completo de planificación, deseño, codificación e probas. É idónea para proxectos con requisitos imprecisos e moi cambiantes, ao poñer énfase na **realimentación continua** entre o **cliente** e o **equipo de desenvolvemento**. Grazas a que existe un fluxo de comunicación constante, os desenvolvedores poden responder rapidamente aos cambios aínda en fases tardías do ciclo de vida do desenvolvemento.

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 tecnicismo 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, revisaríanse 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 teñan traballado en todas as partes do sistema.

Deseño

Á diferenza do modelo en cascada, nesta fase utilízase unha tarxeta manual tipo CRC (class, responsabilities, 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.

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 desa maneira tense 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, van 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.

Scrum

Este método está especialmente deseñado para proxectos de alto nivel de incerteza, carga laboral e prazos reducidos. O secreto do seu éxito está nos **sprints**: intervalos de tempo fixos nos que se desenvolven **miniproxectos** dentro do proxecto principal.

En Scrum realízanse entregas parciais e regulares do produto final, priorizadas polo beneficio que aportan ao receptor do proxecto.

Planificación do sprint

Cada sprint ten un obxectivo particular.

Na primeira reunión do equipo definiranse aspectos como a funcionalidade, obxectivos, riscos do sprint, prazos de entrega, entre outros.

Desenvolvemento

Cando o traballo do sprint está en curso, os encargados deben garantir que non se xeren cambios de última hora que podan afectar aos obxectivos do mesmo. Ademais, asegúrase o cumprimento dos prazos establecidos para o seu remate.

Revisión do sprint

Ao finalizar o desenvolvemento do miniproxecto, é posible analizar e avaliar os resultados. En caso de ser necesario, todo o equipo colaborará para saber os aspectos que necesitan ser cambiados. Nesta fase foméntase a colaboración e realimentación entre todos.

Realimentación

Os resultados poden entregarse para recibir unha realimentación non só por parte dos profesionais dentro do proxecto, senón tamén das persoas que utilizarán directamente o que se desexa lograr, é dicir, os clientes potenciais. As leccións aprendidas durante esta etapa permitirán que o seguinte sprint poda ser máis efectivo e áxil.

A metodoloxía Scrum non se utiliza en todos os casos. Emprégase cando a empresa ten recursos, madurez e experiencia, unha estrutura organizacional áxil e innovadora, entre outros factores. Contar cun profesional que asegure estes principios será o primeiro paso.

Vídeo explicativo sobre Scrum: https://www.youtube.com/watch?v=P25JP0u6UKw

Máis información sobre Scrum e exames de acreditación:

https://scrummanager.net/files/scrum_master.pdf

https://www.scrummanager.net/oks/

Kanban

Kanban é unha palabra xaponesa que significa algo así como “tarxetas visuais” (kan significa visual e ban tarxeta). Esta técnica creouse en Toyota a finais da década de 1940 e utilizouse inicialmente para controlar o avance do traballo, no contexto dunha liña de produción.

Kanban non é unha técnica específica de desenvolvemento de software, o seu obxectivo é xestionar de maneira xeral como se van completando as tarefas, pero nos últimos anos tense utilizada na xestión de proxectos de desenvolvemento de software, a miúdo con Scrum.

A metodoloxía Kanban baséase no traballo en equipo e no fluxo de tarefas permanente, potenciando a visualización para aforrar tempo na planificación.

As principais regras de Kanban son as tres seguintes:

- Visualizar o traballo e as fases do ciclo de produción ou fluxo de traballo
- Determinar o límite do traballo en curso
- Medir o tempo en completar unha tarefa

Ao igual que Scrum, Kanban baséase no desenvolvemento incremental, dividindo o traballo en partes. Unha das principais aportacións é que utiliza técnicas visuais para ver a situación de cada tarefa.

O traballo divídese en partes. Normalmente, cada unha destas partes escríbese nun post-it e pégase nun muro ou taboleiro. Os post-it soen ter información variada, pero, ademais da descrición, deberían ter unha estimación da duración da tarefas.

O muro ten tantas columnas como estados polos que pode pasar unha tarefa (exemplo, en espera de ser desenvolvida, en análise, en deseño, etc.

O obxectivo desta visualización é que quede claro o traballo a realizar, en que está traballando cada persoa, que todo o equipo teña algo que facer e ter claras as prioridades das tarefas. As fases do ciclo de produción ou fluxo de traballo decídense segundo o caso, non hai nada acoutado.

Quizais unha das principais ideas de Kanban é que o traballo en curso debería estar limitado, é dicir, que o número máximo de tarefas que se poden realizar en cada fase debe ser algo coñecido.

En Kanban debe definirse cantas tarefas como máximo poden realizarse en cada fase do ciclo de traballo (exemplo, como máximo 4 tarefas en desenvolvemento, como máximo 1 en probas, etc). A este número de tarefas chámaselle límite do “work in progress”. A isto engádeselle outra idea razoable como que para comezar cunha nova tarefa algunha outra tarefa previa debe ter finalizado.

Simulación Kanban: http://www.kanbanboardgame.com/

Vídeos:

Comparativa entre modelos de ciclos de vida:

https://www.youtube.com/watch?v=EwmI5NDKLBo

Kanban:

https://www.youtube.com/watch?v=H4e5gKWJ04g

Scrum e Kanban:

https://www.youtube.com/watch?v=8qa6io8wHQA

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 12207 "*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 (https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html) pódese acceder aos documentos pdf nos que está detallada toda a metodoloxía e o persoal informático que intervén 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.