Universidad del País Vasco
Euskal Herriko Unibertsitatea

Metodología de la Programación

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Escuela de Ingeniería de Bilbao (UPV/EHU)

1^{er} curso

Curso académico 2022-2023

Tema 1: Introducción

JOSÉ GAINTZARAIN IBARMIA

Departamento de Lenguajes y Sistemas Informáticos

Última actualización: 06 - 01 - 2023

Índice general

Índice de figuras

Índice de tablas

Tema 1

Introducción

1.1.

Presentación

En la asignatura "Metodología de la Programación" se aborda el estudio de métodos y técnicas formales que sirven para los siguientes cometidos:

(a) Expresar y describir, de manera precisa, las propiedades de los programas.

(b) Comprobar la corrección de los programas.

(c) Diseñar programas correctos.

Para todo ello, se utilizarán herramientas basadas en la lógica matemática. Las propiedades de los programas se representarán por medio de fórmulas lógicas [1] escritas en el lenguaje de la lógica de primer orden (lógica de predicados). Para comprobar la corrección de los programas y para diseñar programas correctos, se hará uso de un cálculo que está basado también en la noción de consecuencia lógica [2] y en la noción de sustitución [3]. Nos referiremos a dicho cálculo como el "cálculo de Hoare" o "sistema formal de Hoare" [4].

[1] Una fórmula lógica es una expresión cuyo valor es o cierto o falso. Por ejemplo, $x > y$, donde x e y son dos variables cuyo valor es un número entero. Esa fórmula será cierta o falsa dependiendo de los valores concretos de x e y. Si x es 10 e y es 7, entonces la fórmula es cierta. En cambio, si x es 12 e y es 20, entonces la fórmula es falsa.

[2] Una fórmula lógica γ es consecuencia lógica de otra fórmula lógica β si siempre que β es cierta γ es también cierta. Habitualmente se escribe de la siguiente forma: $\beta \rightarrow \gamma$. Por ejemplo, $(x > 10) \rightarrow (x > 5)$, donde x es una variable cuyo valor es un número entero. Si x es mayor que 10, necesariamente x ha de ser mayor que 5.

[3] La sustitución de una variable x por un valor v en una fórmula lógica γ consiste en reemplazar todas las apariciones de x por v en la fórmula lógica γ. Habitualmente se escribe de la siguiente forma: γ_x^v. Por ejemplo, $(x > y)_y^{z+8}$ representa la fórmula lógica $x > (z + 8)$, mientras que $((x > 10) \rightarrow (x > 5))_x^{w+1}$ representa la fórmula lógica $((w + 1) > 10) \rightarrow ((w + 1) > 5)$.

[4] El sistema formal de Hoare —definido por C. A. R. Hoare— está formado por axiomas y reglas de inferencia. Los axiomas son propiedades que se asume que son ciertas de partida. Las reglas de inferencia sirven para inferir u obtener nuevas propiedades a partir de propiedades que se conocen de antemano.

1.2.

Motivación

1.2.1 Métodos formales y métodos no formales

La informática (o ciencia de la computación) es la rama de la ciencia que estudia el tratamiento automático de la información.

El estudio del tratamiento automático de la información incluye la programación.

La programación es el área de la informática que se encarga de la elaboración de programas en los que se recogen las instrucciones precisas para llevar a cabo un cálculo por medio de un ordenador. Por tanto, las instrucciones que conforman un programa sirven para llevar a cabo un tratamiento automático de información.

Elaborar un programa conlleva, en primer lugar, entender bien el problema que se pretende resolver mediante el programa. En segundo lugar, idear una estrategia que sirva para resolver el problema. Y, finalmente, plasmar la estrategia ideada en un programa, utilizando para ello un lenguaje de programación concreto.

Una opción es abordar la programación desde un enfoque artesanal. Este enfoque para la elaboración de programas estará fudamentalmente basado en el planteamiento de "prueba y error". Este planteamiento consiste en ir construyendo el programa sin más herramientas que las habilidades artesanales del programador. Además, es habitual que la comunicación entre el programador y el cliente sea imprecisa y defectuosa, lo cual propicia que haya muchas ambigüedades y malentendidos con respecto a lo que se desea construir. Si al probar el programa construido se detecta algún error o algún resultado inadecuado o alguna característica que no coincide con lo que esperaba el cliente, se revisará el programa para intentar subsanar el problema. Este proceso de prueba y error puede dar lugar a una sucesión de correcciones y modificaciones. Lo peor es que al final no se podrá garantizar que el programa sea correcto. Solo se podrá afirmar que no se han detectado más errores o resultados inadecuados, aunque puede que los haya.

En la asignatura Metodología de la Programación se propone un enfoque formal basado en el conocimiento matemático. Las técnicas y los métodos presentados en esta asignatura hacen posible elaborar programas con garantía de corrección y, además, sin hacer uso del planteamiento de "prueba y error", puesto que las técnicas y métodos mencionados guían todo el proceso desde el principio hasta el final.

De todas formas, conviene aclarar que las técnicas y los métodos presentados en esta asignatura no sirven para construir programas de manera automática a partir de cero y sin intervención de ninguna persona. Se ha probado matemáticamente que es imposible tener un sistema informático de propósito general que, dada una tarea cualquiera que se quiere informatizar, sea capaz de obtener automáticamente (sin la intervención humana) un programa que realice la tarea que se quería informatizar. Pero esto ocurre también en otras áreas de la ingeniería. Construir un puente no es automatizable, la intervención humana es imprescindible. Pero el conocimiento sobre materiales y otros aspectos hace posible construir puentes con garantía de idoneidad y calidad.

1.2.2 Dos métodos para construir programas

En MP **se presentarán dos métodos para obtener programas**:

(i) Construcción de un programa a partir de cero. Se utilizarán el lenguaje de la lógica de primer orden y el cálculo de Hoare.

(ii) Construcción de un programa por transformación. Dado un programa P que realiza el cálculo que se desea, el programa P es transformado para obtener otro programa P' que es equivalente pero más eficiente. Se utilizarán el lenguaje de la lógica de primer orden y el método de desplegado/plegado (método de Burstall).

1.3.

Fases en el desarrollo de programas

1.3.1 Secuencia de fases

A la hora de desarrollar un sistema informático que resuelva un problema, las etapas o fases básicas son las siguientes:

- **Análisis** del problema: quien se encargue de realizar el análisis, estudiará el problema para establecer las características de los datos de entrada y las características del resultado que se desea obtener. Al finalizar esta fase, se tendrá la **especificación** del problema. Dicha especificación expresará, de manera concisa, **qué** se desea hacer.
- **Diseño** de la solución: quien se encargue de realizar el diseño, estudiará la especificación del problema —preparada en la fase de análisis— y elaborará una estrategia para resolver el problema. Dicha estrategia ha de ser formalizada mediante un **algoritmo**. Mediante el algoritmo se ha de precisar **cómo** resolver el problema que se ha expresado mediante la especificación.
- **Implementación** del algoritmo: quien se encargue de implementar el algoritmo, trasladará a un **programa ejecutable**, la estrategia recogida en el algoritmo elaborado en la fase de diseño. Por tanto, se ha de trasladar a un lenguaje de programación concreto, el **cómo** expresado mediante el algoritmo.
- **Mantenimiento**: siempre que surja el requerimiento de modificar alguna funcionalidad del sistema o de añadir alguna nueva funcionalidad, se volverá a alguna de las tres fases anteriores, dependiendo del cambio a realizar.

En la figura 1.3.1 de la página 20 se muestra el esquema de las fases del desarrollo de programas.

1.3.2 Ejemplo para las fases en el desarrollo de programas

A modo de ejemplo, se ha considerado la tarea de calcular en la variable s la suma de los componentes del vector $A(1..n)$:

1.3.2.1 Análisis y especificación

Tras realizar el análisis del problema, la especificación se puede formular de la siguiente manera:

$$\varphi \equiv n \geq 1$$
$$\psi \equiv n \geq 1 \wedge s = \sum_{k=1}^{n} A(k)$$

donde φ es la precondición, es decir, lo que ha de cumplir el dato de entrada $A(1..n)$ y ψ es la postcondición, es decir, lo que ha de cumplir el resultado. Dicho de otra forma, φ es lo que se ha de cumplir al principio del programa y ψ es lo que se ha de cumplir al final del programa.

1.3.2.2 Diseño y algoritmo

En la fase de diseño, a partir de la especificación, se ha de elaborar una estrategia para resolver el problema. Una posible estrategia, descrita informalmente, podría ser la siguiente:

> Utilizar la variable i como índice para recorrer las posiciones del vector. El vector se recorrerá de izquierda a derecha. La variable i irá tomando los valores que van desde 1 hasta $n + 1$. Cuando i tome un valor nuevo, en s se tendrá la suma de los elementos que van desde la posición 1 hasta la posición $i - 1$.

Dicha estrategia se formaliza mediante la siguiente fórmula lógica:

$$INV_1 \equiv (n \geq 1) \wedge (1 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k)$$

Esta fórmula representa el algoritmo y en Metodología de la Programación lo llamaremos *invariante* porque se cumple cada vez que i toma un valor nuevo. En general, el invariante se abreviará como INV. En este caso se ha puesto INV_1 porque a continuación se definirán dos algoritmos más. Los subíndices servirán para diferenciarlos.

Es importante tener claro que se pueden plantear distintas estrategias y, por tanto, distintos algoritmos. Por ejemplo, una segunda estrategia (o algoritmo) podría ser la siguiente:

> Utilizar la variable i como índice para recorrer las posiciones del vector. El vector se recorrerá de izquierda a derecha. La variable i irá tomando los valores que van desde 0 hasta n. Cuando i tome un valor nuevo, en s se tendrá la suma de los elementos que van desde la posición 1 hasta la posición i.

La fórmula lógica correspondiente sería la siguiente:

$$INV_2 \equiv (n \geq 1) \wedge (0 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)$$

Una tercera estrategia (o algoritmo) podría ser la siguiente:

> Utilizar la variable i como índice para recorrer las posiciones del vector. El vector se recorrerá de derecha a izquierda. La variable i irá tomando los valores que van desde n hasta 1. Cuando i tome un valor nuevo, en s se tendrá la suma de los elementos que van desde la posición n hasta la posición i.

A continuación se muestra la fórmula lógica correspondiente:

$$INV_3 \equiv (n \geq 1) \wedge (1 \leq i \leq n) \wedge s = \sum_{k=i}^{n} A(k)$$

1.3.2.3 Implementación y programa

En la fase de implementación se obtendrá un *programa ejecutable* a partir del algoritmo diseñado en la fase previa. El programador deberá elegir un lenguaje. A partir de un mismo algoritmo se pueden obtener distintos programas, aunque todos ellos serán equivalentes.

Si consideramos el primer algoritmo INV_1, podríamos construir el siguiente programa en ADA:

```
i : = 1;
s : = 0;
while i ≠ n + 1 loop
    s : = s + A(i);
    i : = i + 1;
end loop;
```

Pero también se podría obtener el siguiente programa en ADA:

```
s : = 0;
i : = 1;
while i ≤ n loop
    i : = i + 1;
    s : = s + A(i - 1);
end loop;
```

Y también se podría obtener el siguiente programa recursivo en un lenguaje similar a ADA:

```
f (A(1..n): array of Integer; return s: Integer)
    s : = f_aux(A(1..n), 0, 1);

f_aux(A(1..n): array of Integer; w: Integer; i: Integer; return r: Integer)
    if i = n + 1 then r : = w;
    else r : = f_aux(A(1..n), w + A(i), i + 1);
    end if;
```

Este programa recursivo consta de dos funciones (f y f_aux) y el parámetro w es utilizado para ir acumulando la suma. En la asignación s : = f_aux(A(1..n), 0, 1); de la función f, la suma es inicializada a 0 y el índice a 1. En cambio, en la asignación r : = f_aux(A(1..n), w + A(i), i + 1); de la función f_aux, la suma es actualizada sumando el elemento A(i) y el índice es actualizado incrementando su valor en 1.

Si consideramos el segundo algoritmo INV_2, podríamos construir el siguiente programa en ADA:

```
i : = 0;
s : = 0;
while i ≠ n loop
    s : = s + A(i + 1);
    i : = i + 1;
end loop;
```

Por su parte, si consideramos el tercer algoritmo INV_3, podríamos construir el siguiente programa en ADA:

```
i : = n;
s : = A(n);
while i ≠ 1 loop
    s : = s + A(i - 1);
    i : = i - 1;
end loop;
```

1.3.2.4 Mantenimiento

En cuanto al mantenimiento, los motivos pueden ser de distinta naturaleza y, consecuentemente, afectarán a una u otra fase dependiendo de la naturaleza del motivo. Por ejemplo, si, además de la suma, se quiere calcular el producto, habrá que volver a la fase de análisis para modificar la especificación. Ello acarreará también modificaciones en el algoritmo (o invariante) y, finalmente, se retocará el programa. En cambio, si lo único que se quiere es modificar el texto del mensaje mediante el cual se presenta el resultado s por pantalla, eso no afecta ni a la especificación ni al algoritmo. Solo afecta al programa: el programador ha de cambiar, en el programa, el texto del mensaje.

1.3.3 Distintas maneras de formalizar la especificación y el algoritmo

A la hora de formalizar la especificación y el algoritmo correspondiente a un programa que se quiere construir, se pueden utilizar distintos formalismos. En Metodología de la Programación se utilizará el formalismo presentado en el ejemplo del apartado anterior.

1.3.4 Diferencia entre algoritmo y programa

De ese ejemplo trabajado en el apartado 1.3.2 (página 15), ha de quedar claro que el algoritmo y el programa son dos cosas diferentes. Si consideramos el proceso de construcción de un puente, las fases serían las mismas. En concreto, el algoritmo se correspondenría con los planos del puente a construir, mientras que el programa se correspondería con el propio puente construido. De la misma forma que tenemos muy clara la diferencia entre los planos del puente y el propio puente, también hemos de tener muy clara la diferencia entre algoritmo y programa correspondiente al algoritmo.

A partir de un algoritmo se pueden construir distintos programas que serán equivalentes entre ellos pero pueden estar escritos incluso en distintos lenguajes de programación.

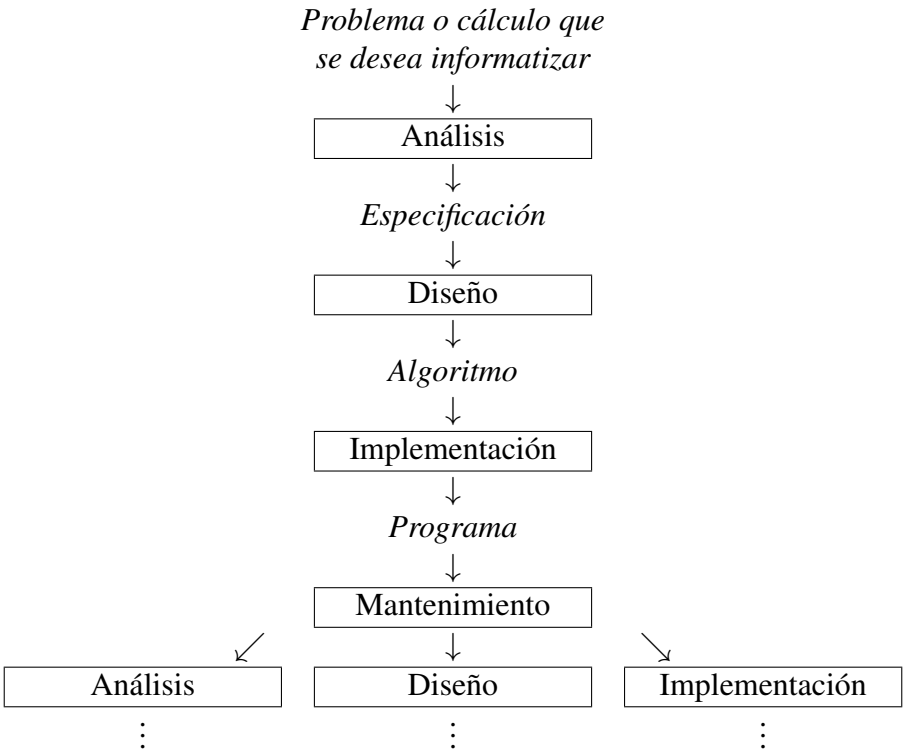

Figura 1.3.1. Fases del desarrollo de programas.

1.4.

Temario y puntuación de cada tema

1.4.1 Temario

- **Tema 2**: Especificación y documentación formal de programas.

  La especificación y documentación formal de programas consiste en expresar, mediante fórmulas de la lógica de primer orden, las propiedades que se cumplen en distintos puntos del programa. Por ejemplo, en el caso de un programa iterativo P, habrá que dar la precondición φ, la postcondición ψ, el invariante INV, la expresión cota E y las aserciones intermedias.

  Si consideramos el siguiente programa escrito en ADA:

```
i : = 1;
s : = 0;
while i ≠ n + 1 loop
    s : = s + A(i);
    i : = i + 1;
end loop;
```

  Tras documentarlo, tendríamos lo que se muestra en la figura 1.4.1 de la página 22. Las fórmulas φ_1, φ_2, φ_3, φ_4 y φ_5 son las aserciones intermedias. El invariante INV sería la expresión INV_1 que se ha formulado previamente:

$$INV \equiv INV_1 \equiv (n \geq 1) \wedge (1 \leq i \leq n+1) \wedge s = \sum_{k=1}^{i-1} A(k)$$

  En cuanto a E, es la expresión cota, que indica cuántas vueltas quedan por dar como máximo.

  En resumen, en el Tema 2 nos darán un programa P y tendremos que formular φ, ψ, INV, E y las aserciones intermedias $\varphi_1, \varphi_2, \ldots$.

$\{\varphi \equiv n \geq 1\}$
i : = 1;
$\{\varphi_1 \equiv n \geq 1 \wedge i = 1\}$
s : = 0;
$\{\varphi_2 \equiv n \geq 1 \wedge i = 1 \wedge s = 0\}$
while $\{INV\}$ $\{E = n + 1 - i\}$ i $\neq$ n + 1 loop
$\{\varphi_3 \equiv (n \geq 1) \wedge (1 \leq i \leq n) \wedge s = \sum_{k=1}^{i-1} A(k)$
s : = s + A(i);
$\{\varphi_4 \equiv (n \geq 1) \wedge (1 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)$
i : = i + 1;
$\{\varphi_5 \equiv (n \geq 1) \wedge (2 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k)$
end loop;
$\{\psi \equiv n \geq 1 \wedge s = \sum_{k=1}^{n} A(k)\}$

$$INV \equiv (n \geq 1) \wedge (1 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k)$$

Figura 1.4.1. Programa documentado (Apartado 1.4.1, página 21).

- **Tema 3**: Verificación formal de programas.

  La verificación consiste en comprobar si un programa hace lo que se espera que haga. Por tanto, en este tema, dados un programa P y las fórmulas φ, ψ, INV y E, se comprobará si P es correcto con respecto a las fórmulas φ, ψ, INV y E. Dicho de otra forma, las fórmulas φ, ψ, INV y E describen lo que se espera que el programa haga y, consecuentemente, se ha de comprobar si P hace lo que las fórmulas φ, ψ, INV y E dicen que P debería hacer.

  Para realizar dicha comprobación se utilizará el cálculo de Hoare que consta, por un lado, de un conjunto de axiomas y, por otro lado, de un conjunto de reglas de inferencia.

  De momento, lo importante es quedarse con la idea de que es posible comprobar matemáticamente si un programa es correcto o no. A continuación se muestra un ejemplo muy sencillo. En dicho ejemplo, se utilizan propiedades matemáticas muy conocidas como la conmutatividad de la suma, la conmutatividad del producto y la distributividad del producto con respecto a la suma.

  Supongamos que se le ha pedido a un programador que construya un programa que deje en la variable entera r el resultado $x * (y + z)$, donde x, y y z son enteros, y que el programador ha construido el siguiente programa:

```
r1 : = y * x;
r2 : = z * x;
r : = r2 + r1;
```

  Para comprobar, de manera matemática, que ese programa calcula $x * (y + z)$, podemos desarrollar el siguiente razonamiento:

r = r2 + r1	por la asignación r : = r2 + r1
r = r1 + r2	por conmutatividad de +
r = (y * x) + (z * x)	por la asignación r1 : = y * x; y la asignación r2 : = z * x;
r = (x * y) + (x * z)	por conmutatividad de *
r = x * (y + z)	por distributividad de * con respecto a +

  De esta manera podemos asegurar que al final en la variable r tendremos el resultado de la expresión $x * (y + z)$.

- **Tema 4**: Derivación formal de programas.

  En el tema 4, dadas las fórmulas φ, ψ, INV y E, el objetivo es construir un programa P que sea correcto con respecto a las fórmulas φ, ψ, INV y E. La construcción de P estará basada en el cálculo de Hoare.

- **Tema 5**: Especificación algebraica (o ecuacional) de Tipos Abstractos de Datos.

  En los temas anteriores se utilizan solo tipos de datos básicos (enteros y booleanos principalmente), pero todo lo presentado en los temas anteriores sigue siendo válido con datos de otros tipos más complejos. En el tema 5 se dice que para definir un nuevo tipo de datos se han de indicar los valores que pertenecen al tipo, las operaciones que se pueden realizar con esos valores y las propiedades de las operaciones. Para ello, se utilizarán la recursividad y la inducción.

- **Tema 6**: Transformación de programas recursivos.

  En el apartado 1.2.2, se ha dicho que vamos a trabajar don enfoques distintos a la hora de construir programas: (i) partiendo de cero y (ii) partiendo de otro programa que hace lo mismo y transformándolo.

  Las técnicas de transformación se aplican a programas recursivos. Existen distintas técnicas. En algunas de esas técnicas se obtiene otro programa recursivo pero que cumple alguna propiedad interesante. En otras técnicas, se obtiene un programa iterativo. En todos los casos, el objetivo es obtener un programa más eficiente.

  En este tema se presentará la técnica de desplegado/plegado, que sirve para transformar programas recursivos en iterativos. Este método también se conoce como el método de Burstall (su autor).

1.4.2 Puntuación por temas

- **Tema 2**: Especificación y documentación formal de programas.

  Habrá dos ejercicios independientes y en total valdrán **2 puntos**.

- **Tema 3**: Verificación formal de programas.

  El parcial constará de un único ejercicio y valdrá **2 puntos**.

- **Tema 4**: Derivación formal de programas.

  También en este caso, el parcial constará de un único ejercicio pero valdrá **1,5 puntos**.

- **Tema 5**: Especificación algebraica (o ecuacional) de Tipos Abstractos de Datos.

  En el tema 5, el parcial lo conformarán cinco ejercicios independientes y en total valdrán **2,5 puntos**.

- **Tema 6**: Transformación de programas recursivos.

  Finalmente, el parcial del tema 6 tendrá un único ejercicio que valdrá **2 puntos**.

1.5.

Explicación gráfica

En la figura 1.5.1 de la página 26 se resume, de manera gráfica, la problemática que surge al desarrollar programas. De ahí se deduce la importancia de hacer uso de métodos formales o matemáticos que garanticen el correcto desarrollo de los programas.

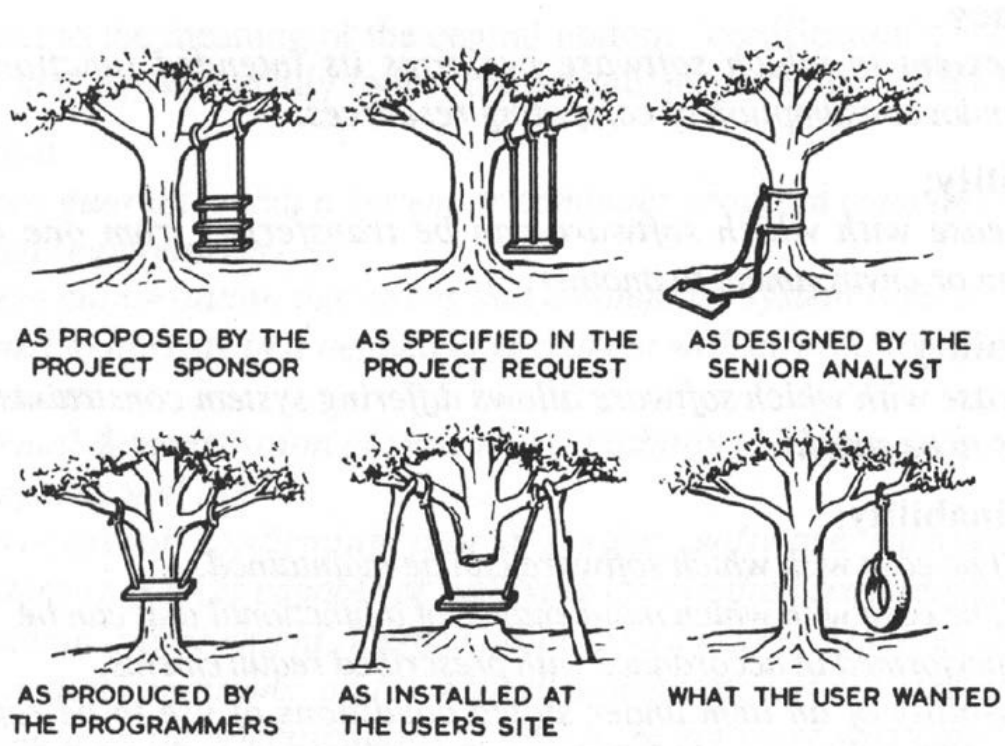

Figura 1.5.1. The problematics of software development. "Specification and Transformation of Programs". Helmut A. Partsch, Springer-Verlag, 1990

1.6.

Símbolos griegos

Es habitual utilizar símbolos del alfabeto griego para representar fórmulas de la lógica matemática y también para denominar funciones. Debido a ello, en la tabla 1.6.1 de la página 28, se muestran los símbolos griegos que se pudieran utilizar.

Alfabeto griego moderno

	Mayúscula	minúscula	Nombre (en castellano)
1	A, A	$\alpha, \mathsf{\alpha}$	alfa
2	B, B	$\beta, \mathsf{\beta}$	beta
3	$\Gamma, \mathsf{\Gamma}$	$\gamma, \mathsf{\gamma}$	gamma
4	$\Delta, \mathsf{\Delta}$	$\delta, \mathsf{\delta}$	delta
5	E, E	$\epsilon, \mathsf{\epsilon}, \varepsilon, \mathsf{\varepsilon}$	épsilon
6	Z, Z	$\zeta, \mathsf{\zeta}$	dseta
7	H, H	$\eta, \mathsf{\eta}$	eta
8	$\Theta, \mathsf{\Theta}$	$\theta, \mathsf{\theta}, \vartheta$	ceta
9	I, I	$\iota, \mathsf{\iota}$	iota
10	K, K	$\kappa, \mathsf{\kappa}, \varkappa, \mathsf{\varkappa}$	kappa
11	$\Lambda, \mathsf{\Lambda}$	$\lambda, \mathsf{\lambda}$	lambda
12	M, M	$\mu, \mathsf{\mu}$	mu
13	N, N	$\nu, \mathsf{\nu}$	nu
14	$\Xi, \mathsf{\Xi}$	$\xi, \mathsf{\xi}$	ksi
15	O, O	o, o	ómicron
16	$\Pi, \mathsf{\Pi}$	$\pi, \mathsf{\pi}, \varpi$	pi
17	P, P	$\rho, \mathsf{\rho}, \varrho, \mathsf{\varrho}$	ro
18	$\Sigma, \mathsf{\Sigma}$	$\sigma, \mathsf{\sigma}, \varsigma$	sigma
19	T, T	$\tau, \mathsf{\tau}$	tau
20	Υ, Y	$\upsilon, \mathsf{\upsilon}$	ípsilon
21	$\Phi, \mathsf{\Phi}$	$\phi, \mathsf{\phi}, \varphi, \mathsf{\varphi}$	fi
22	X, X	$\chi, \mathsf{\chi}$	ji
23	$\Psi, \mathsf{\Psi}$	$\psi, \mathsf{\psi}$	psi
24	$\Omega, \mathsf{\Omega}$	$\omega, \mathsf{\omega}$	omega

Letras griegas no pertenecientes al alfabeto griego moderno

	Mayúscula	minúscula	Nombre (en castellano)
25	F	F	digamma
26	Q	$\mathsf{\varrho}$	qoppa
27	$\mathsf{\Koppa}$	$\mathsf{\koppa}$	koppa
28	$\mathsf{\Sampi}$	$\mathsf{\sampi}$	sampi
29	$\mathsf{\Stigma}$	$\mathsf{\stigma}$	estigma

Tabla 1.6.1. Símbolos griegos.