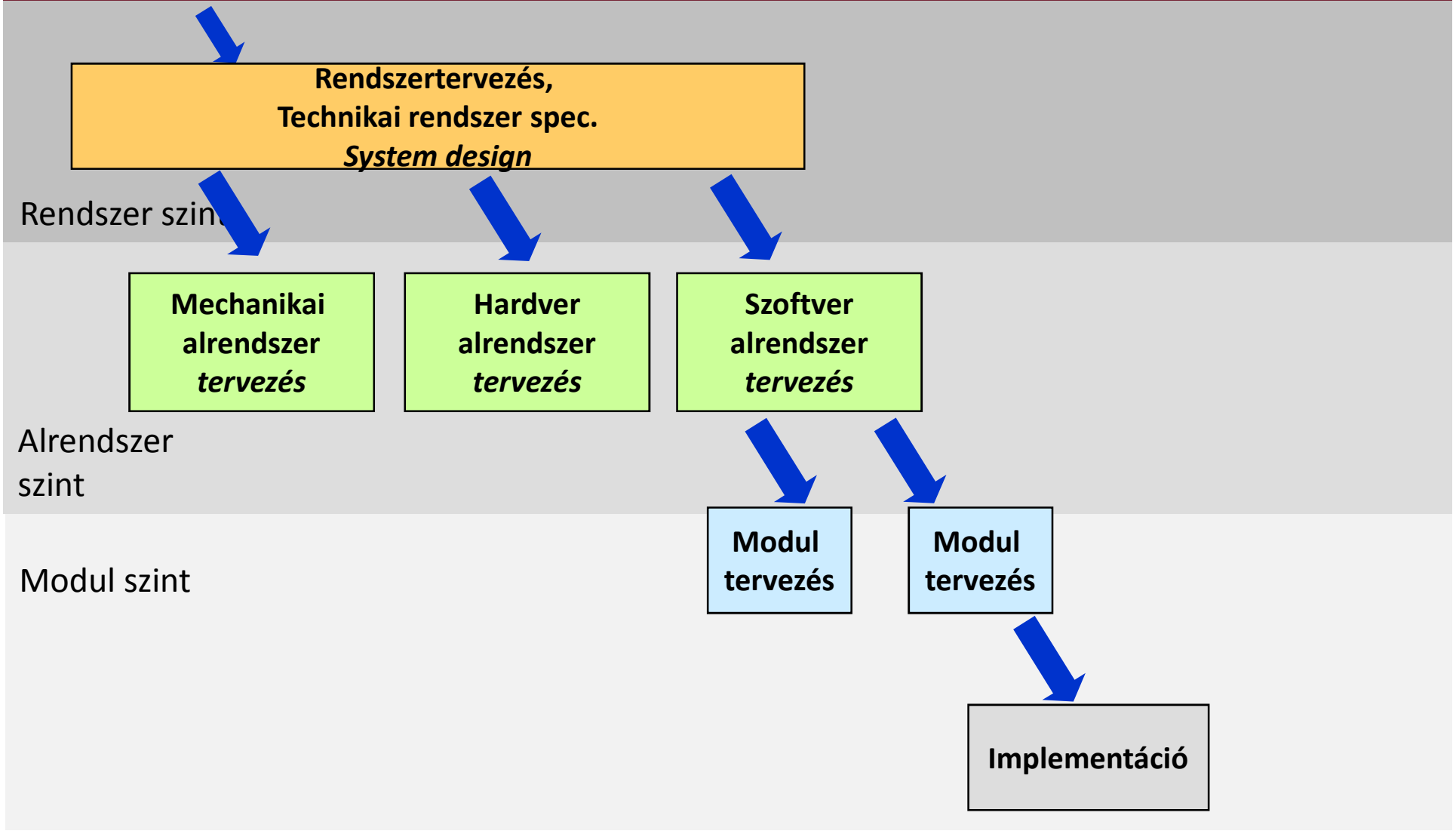
Szoftver tervezési ág

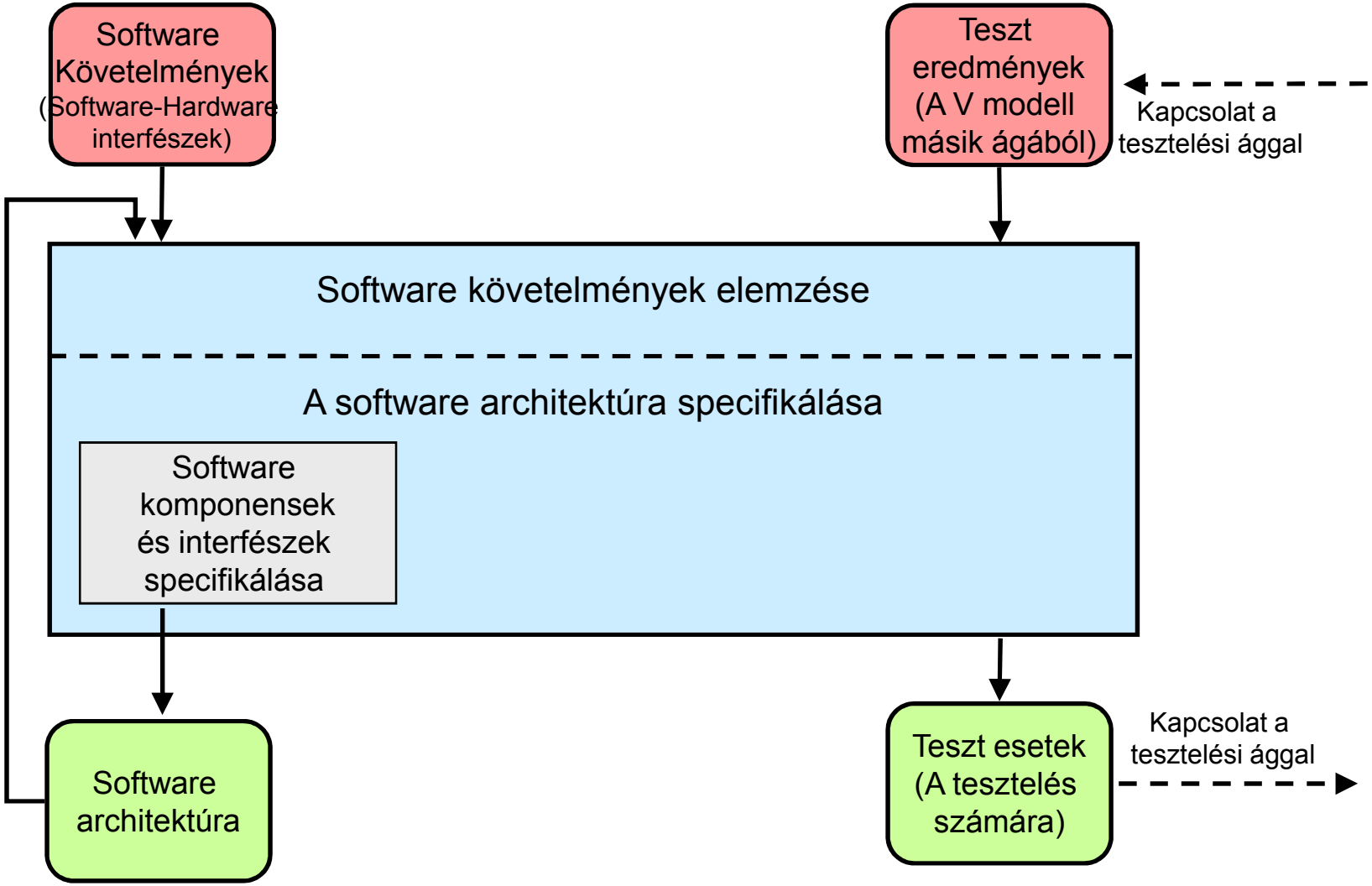
VIMIMA11 Rendszertervezés és –integráció

Scherer Balázs

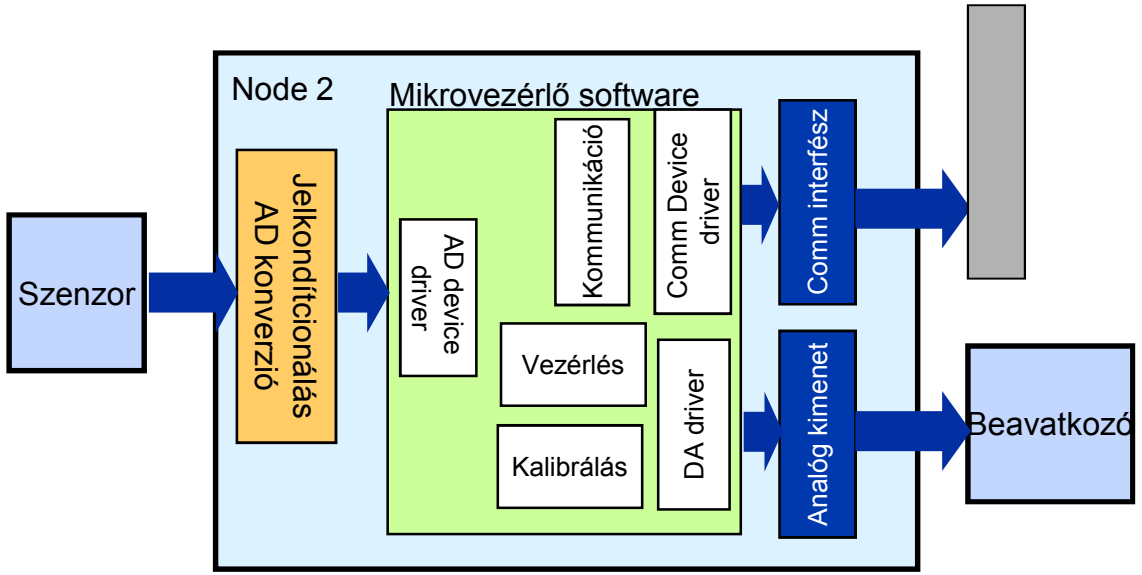
Szoftver irány

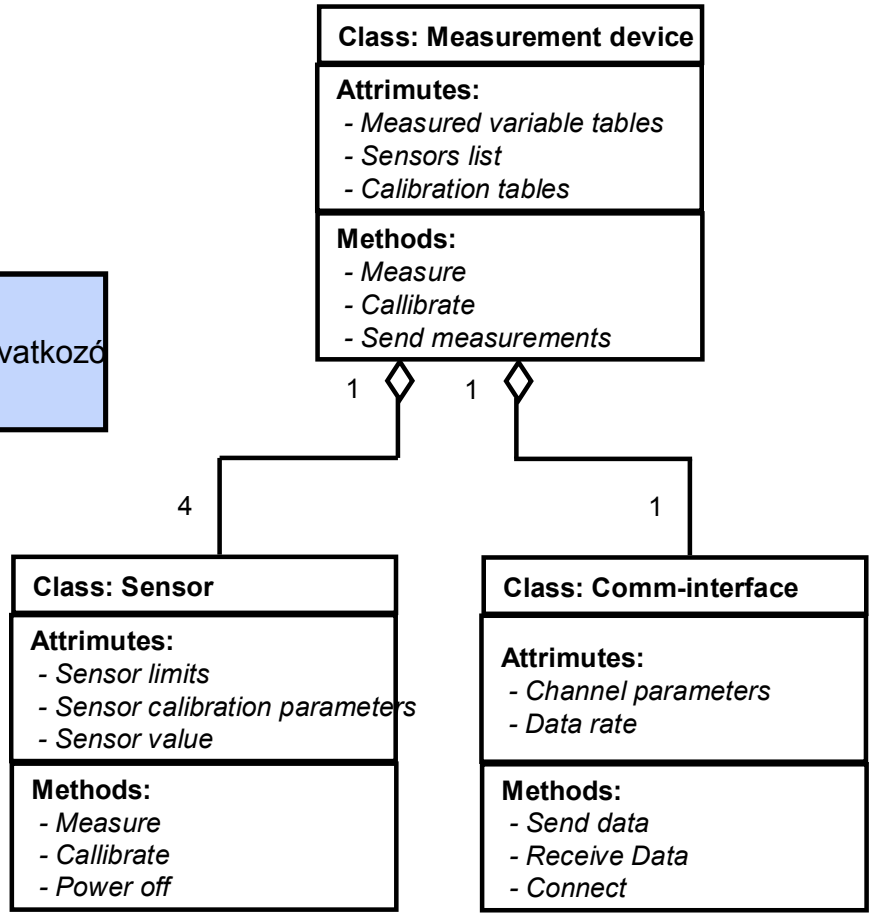


Szoftver architektúra

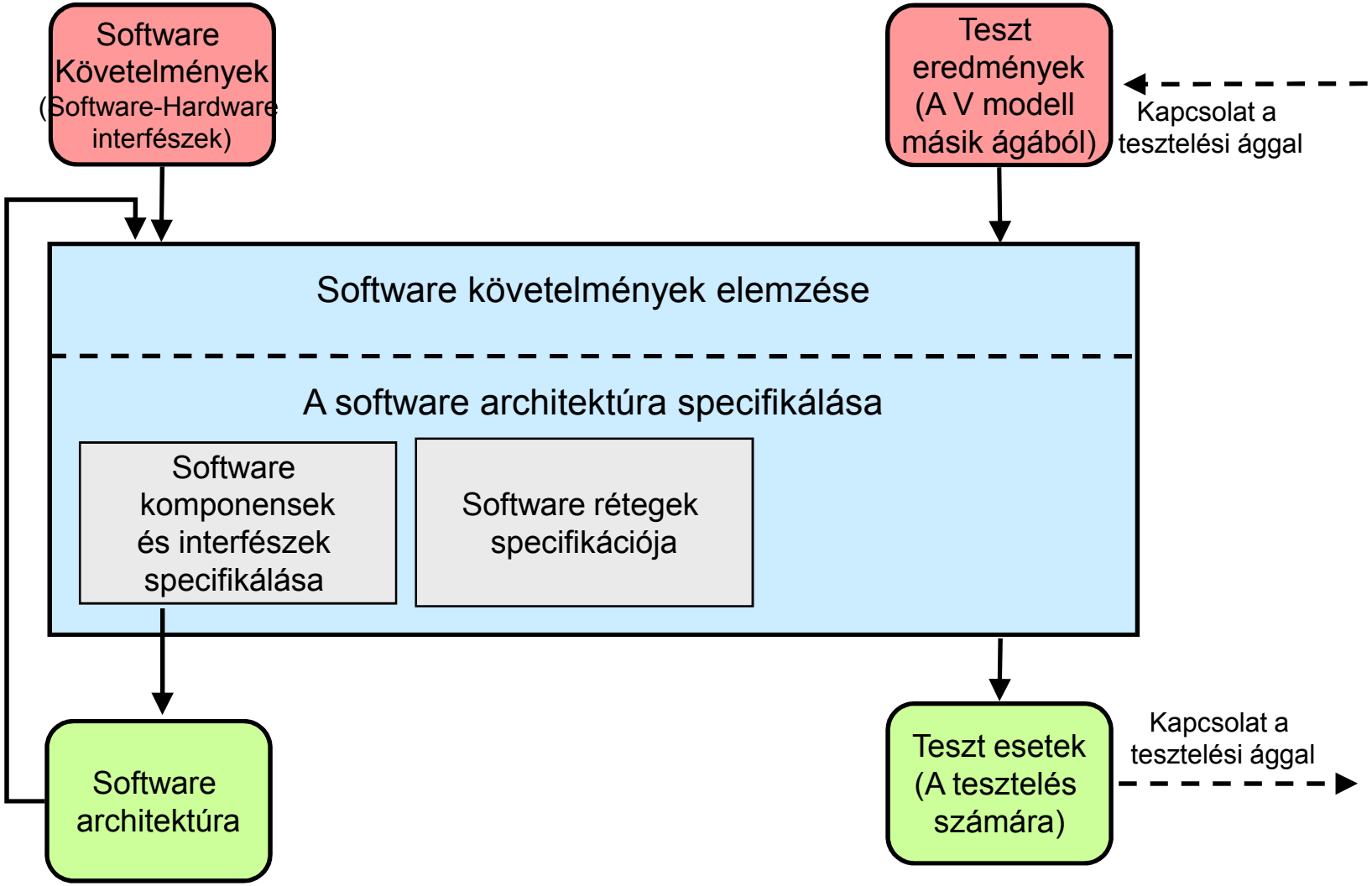


Szoftver komponensek és interfészeinek meghatározása

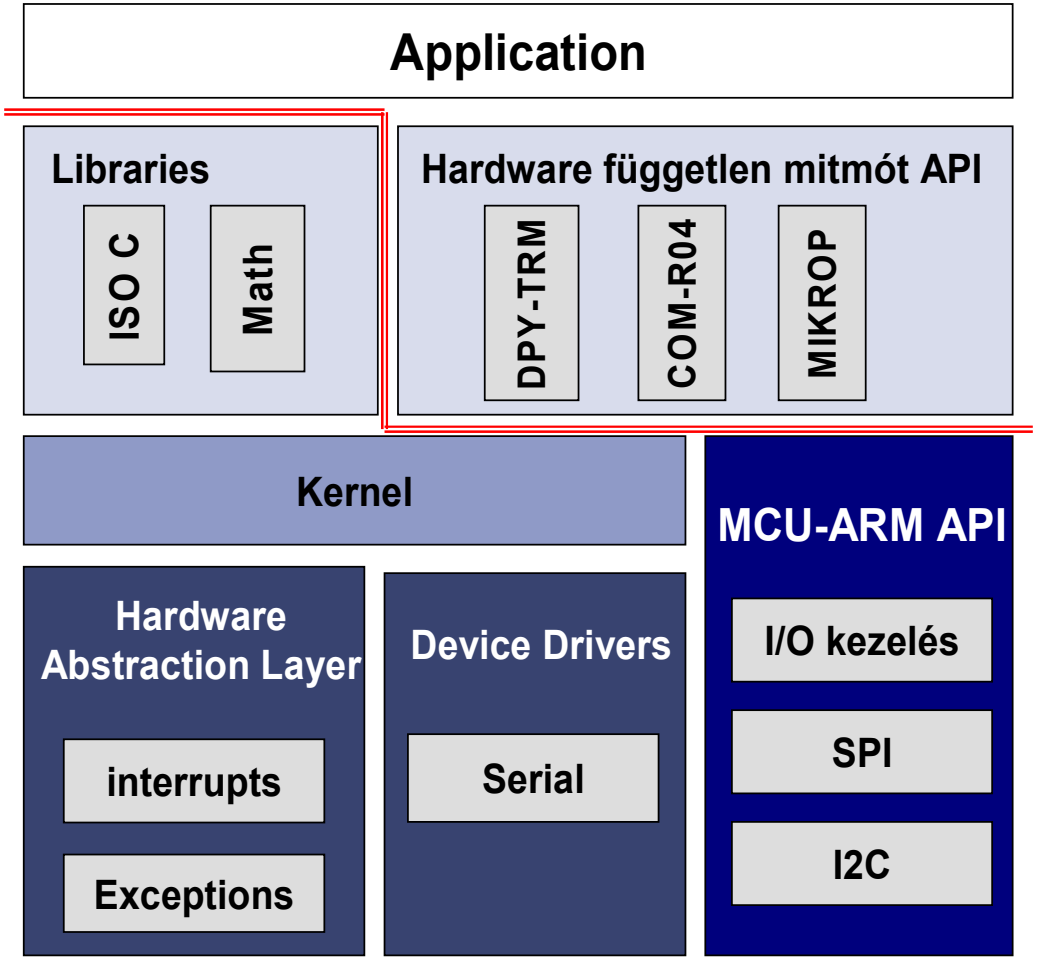




Szoftver architektúra



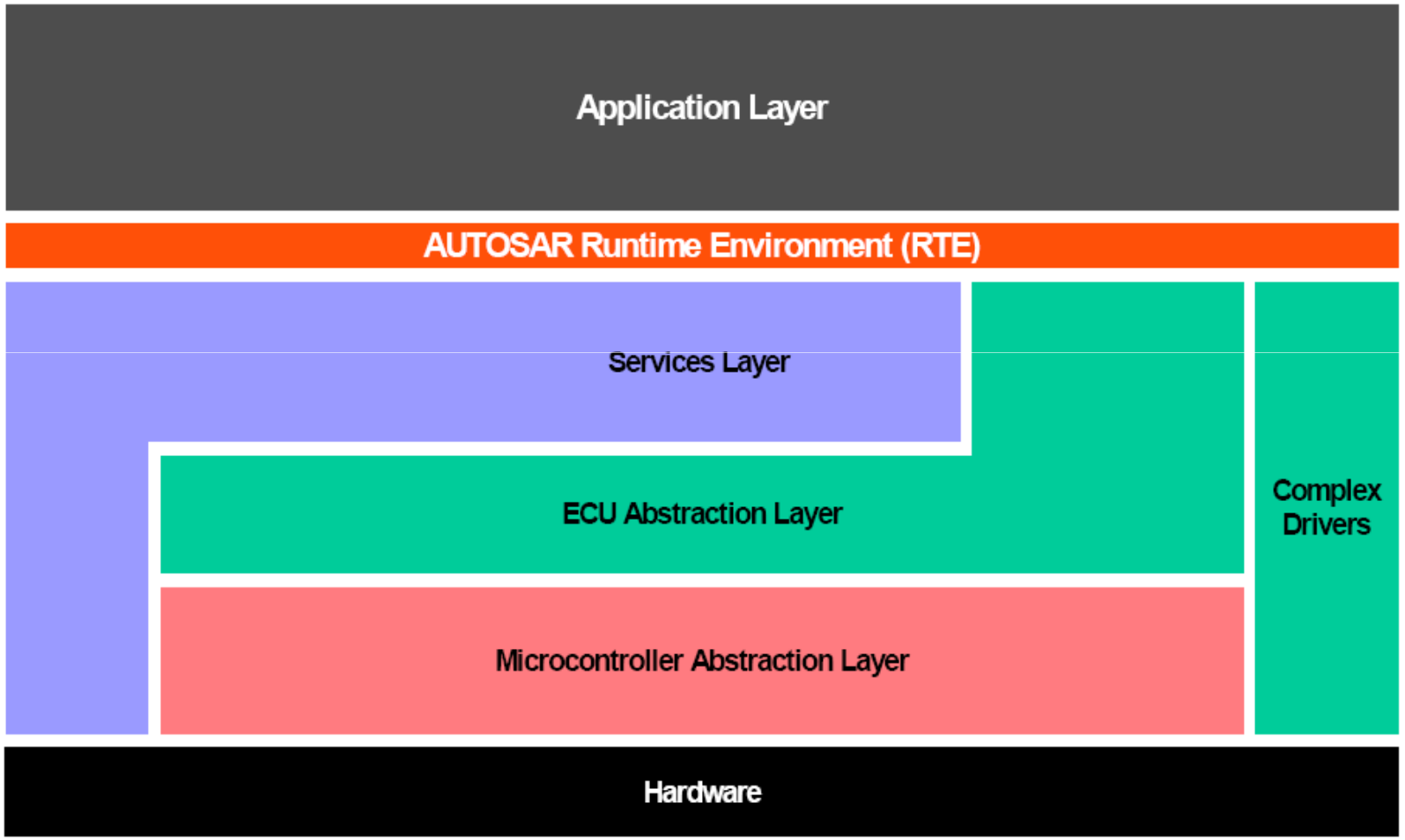
Egy általános beágyazott rendszer SW architektúrája



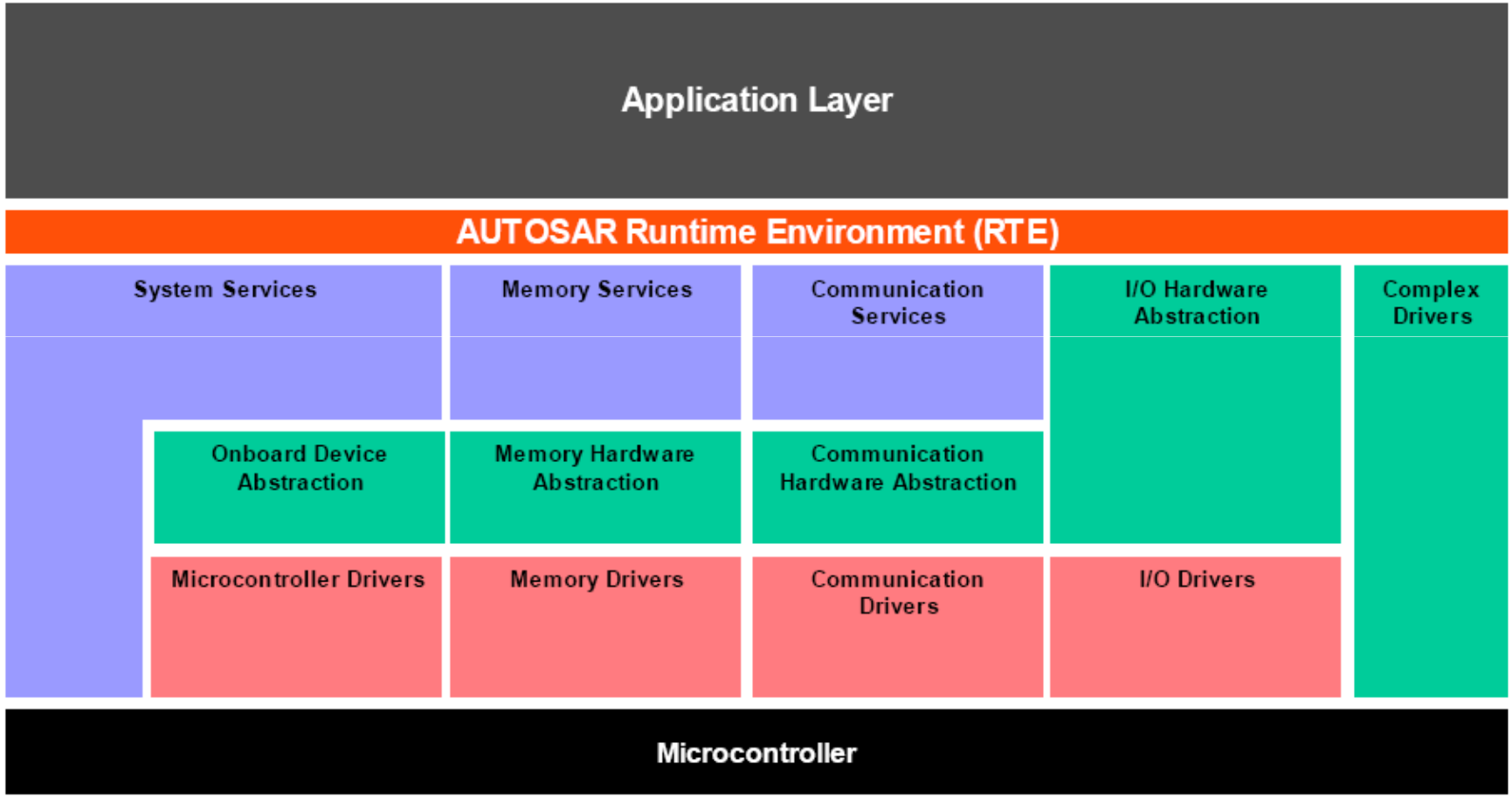
Felhasználói project

Előre fordított eCos distribúció

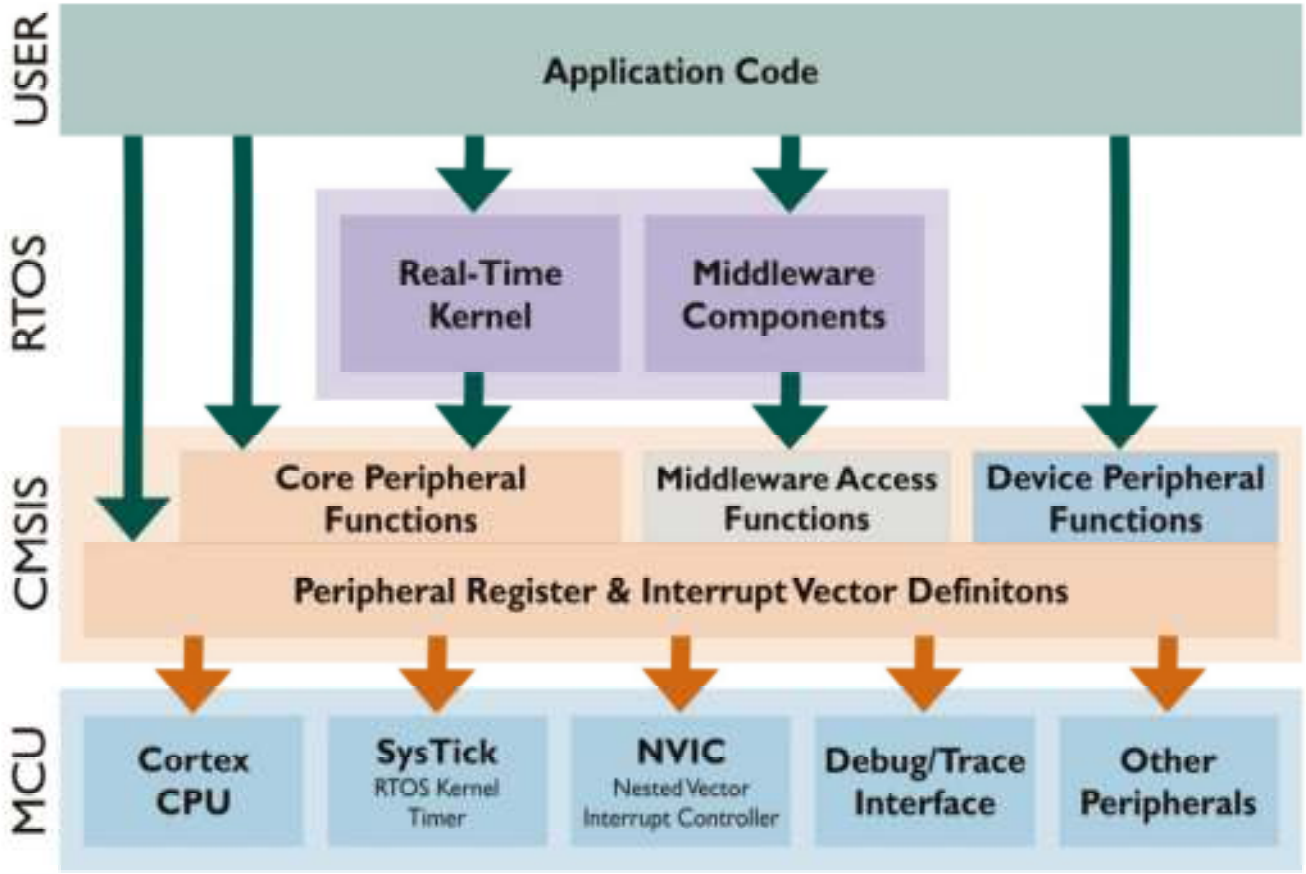
Réteges szemléletű szoftver architektúra példa AUTOSAR



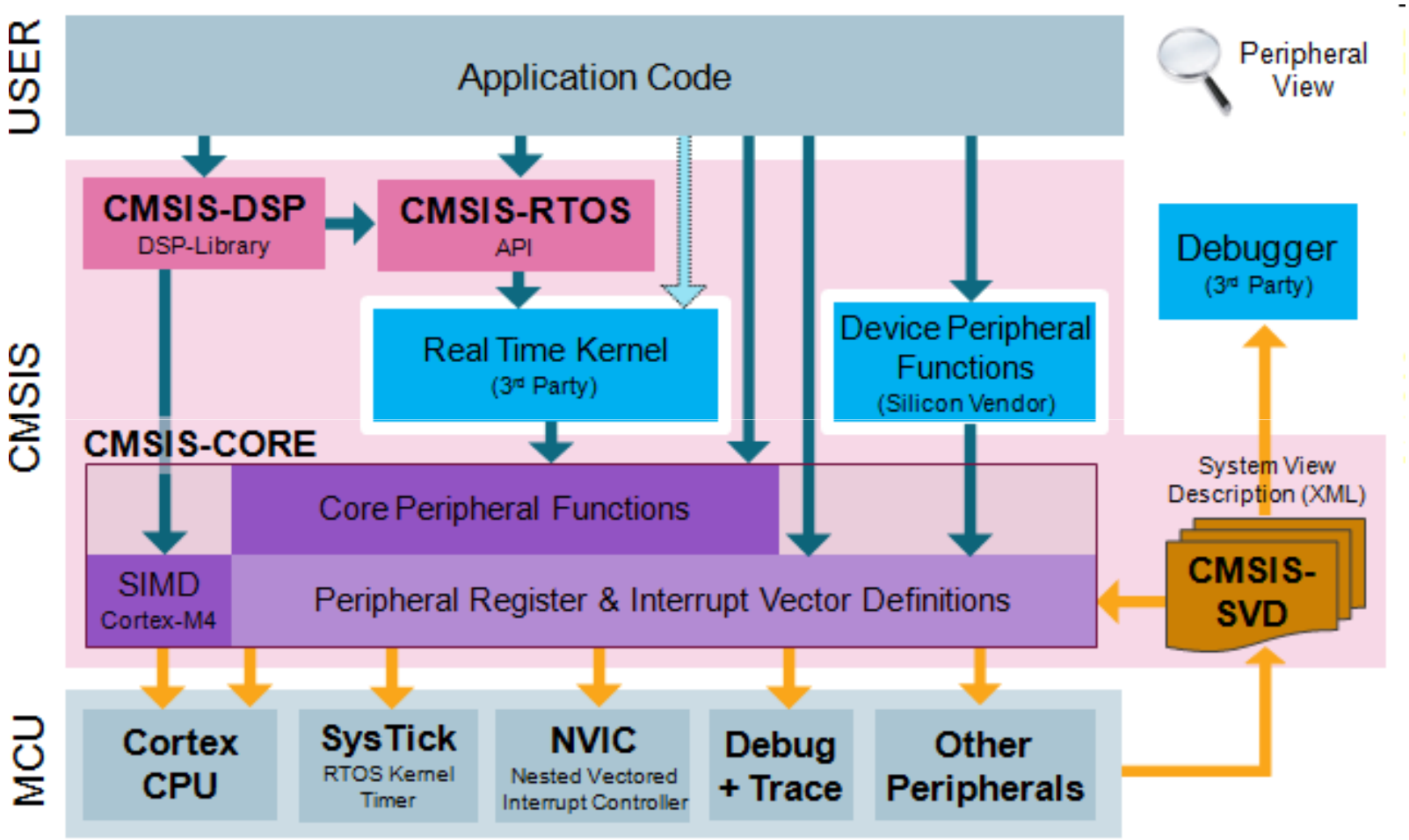
Réteges szemléletű szoftver architektúra példa AUTOSAR



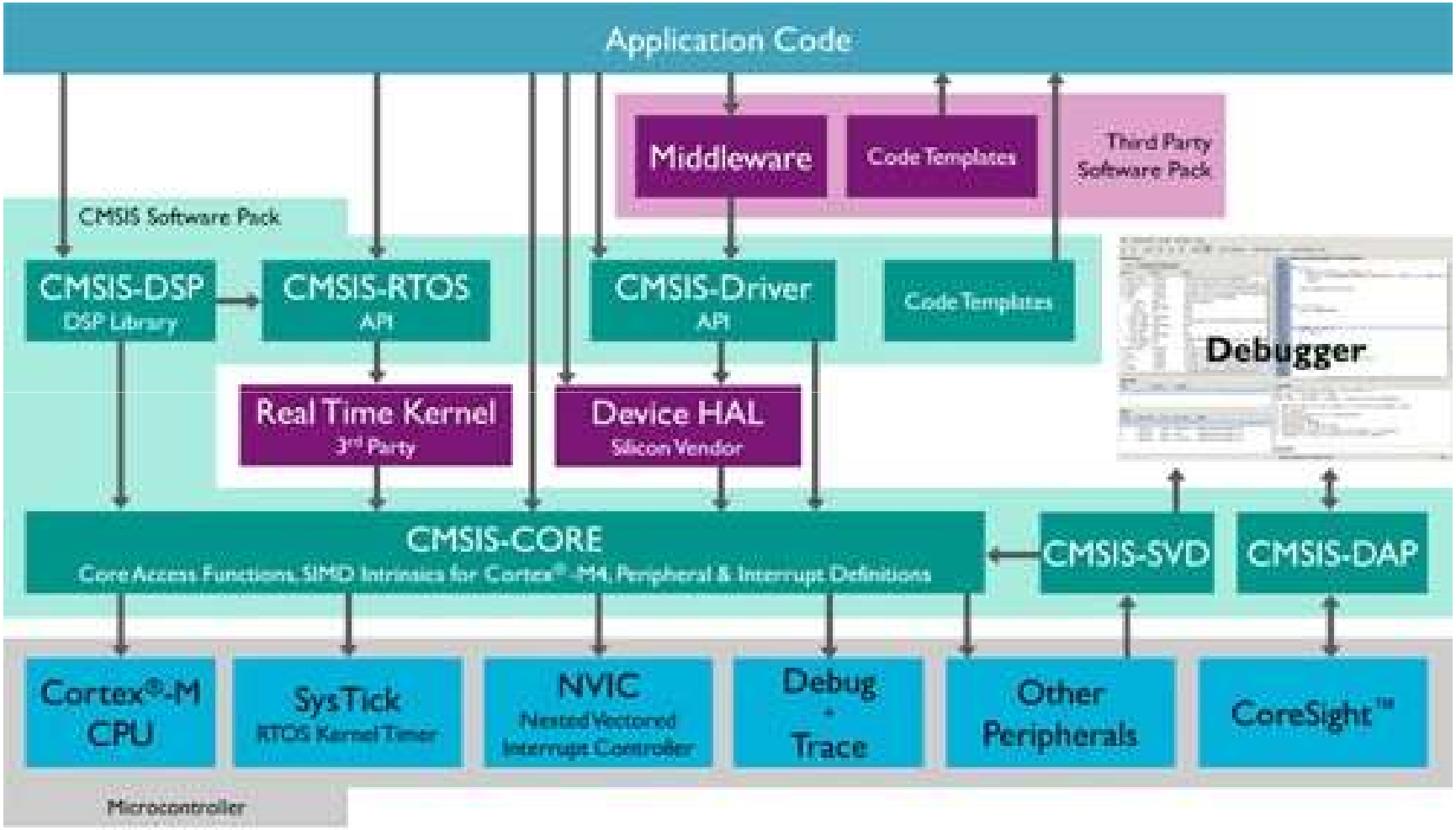
CMSIS szerkezete (v1.3)



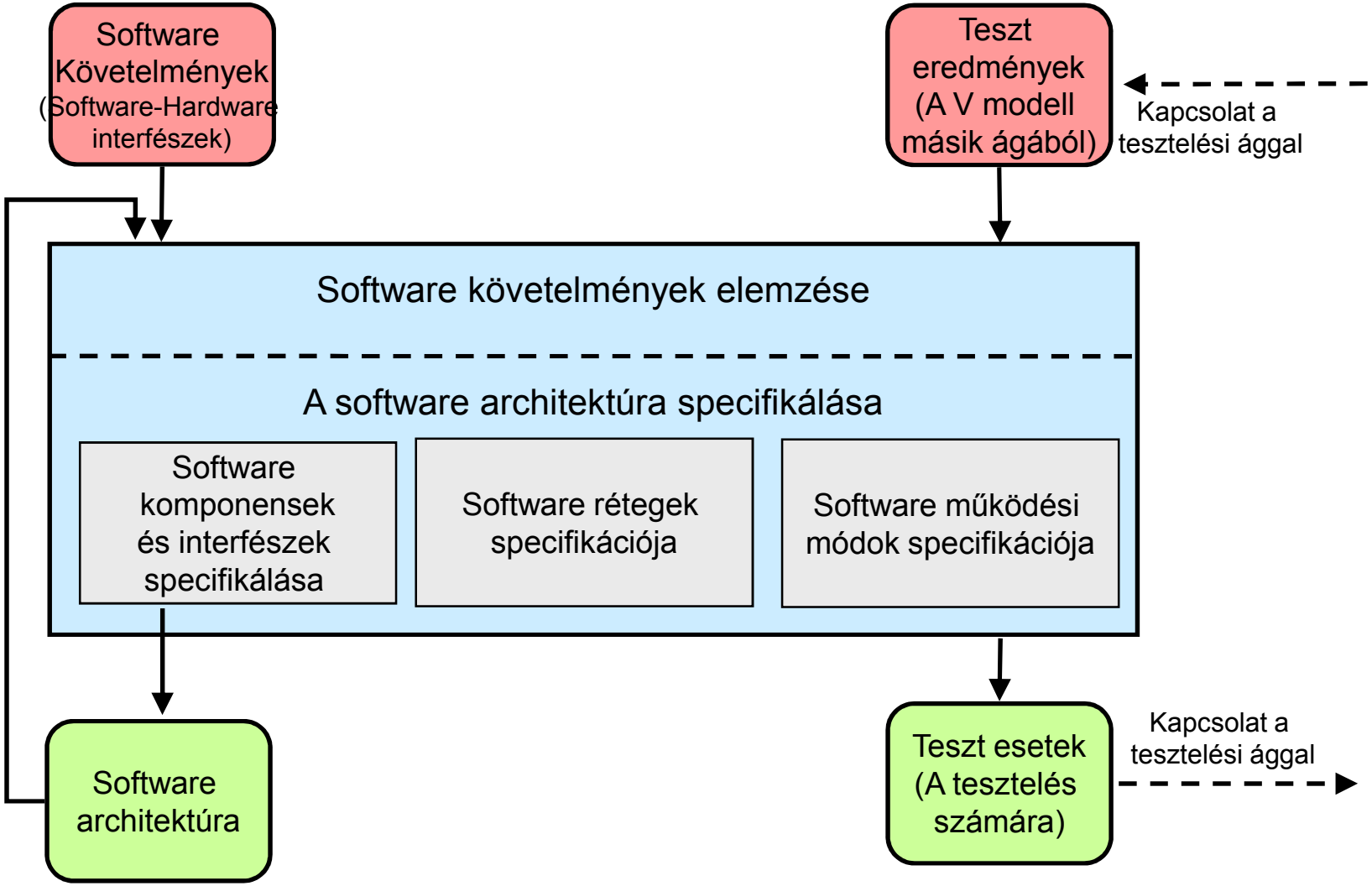
CMSIS szerkezete (v3)



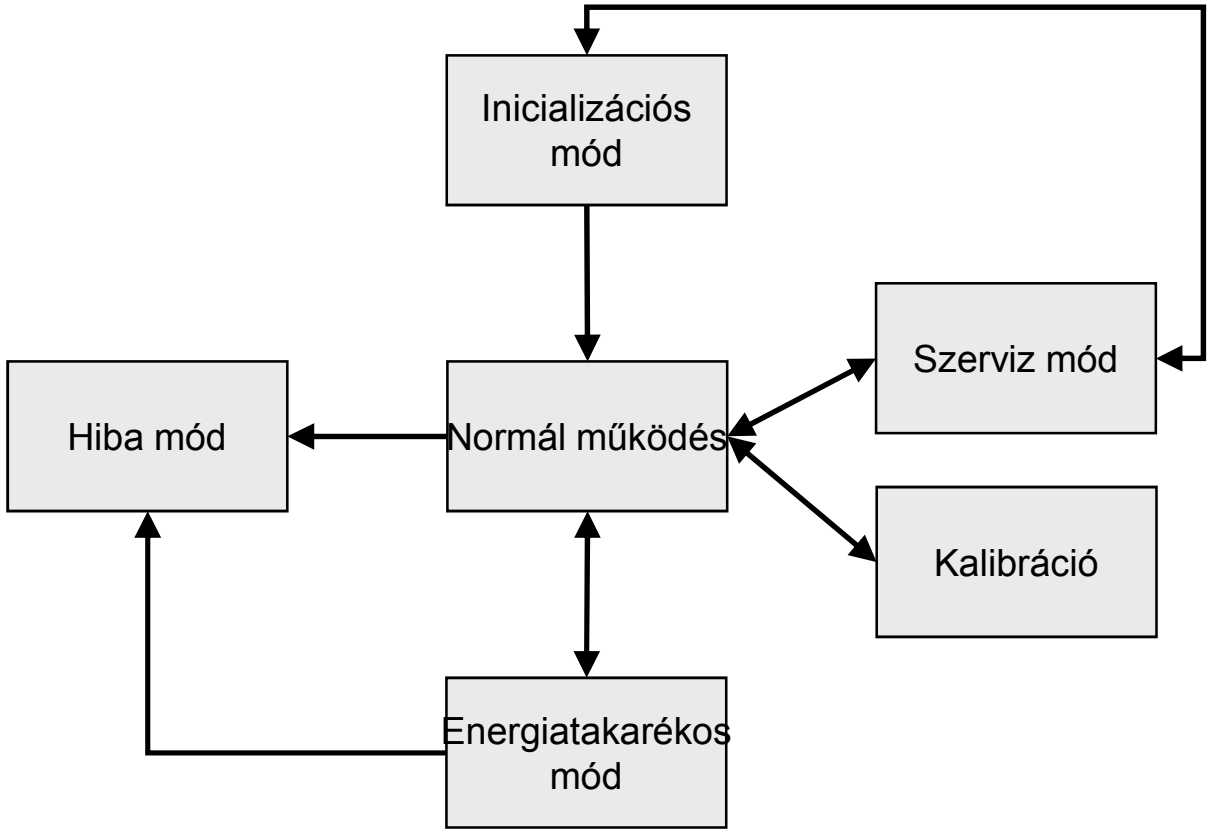
CMSIS szerkezete (v4)



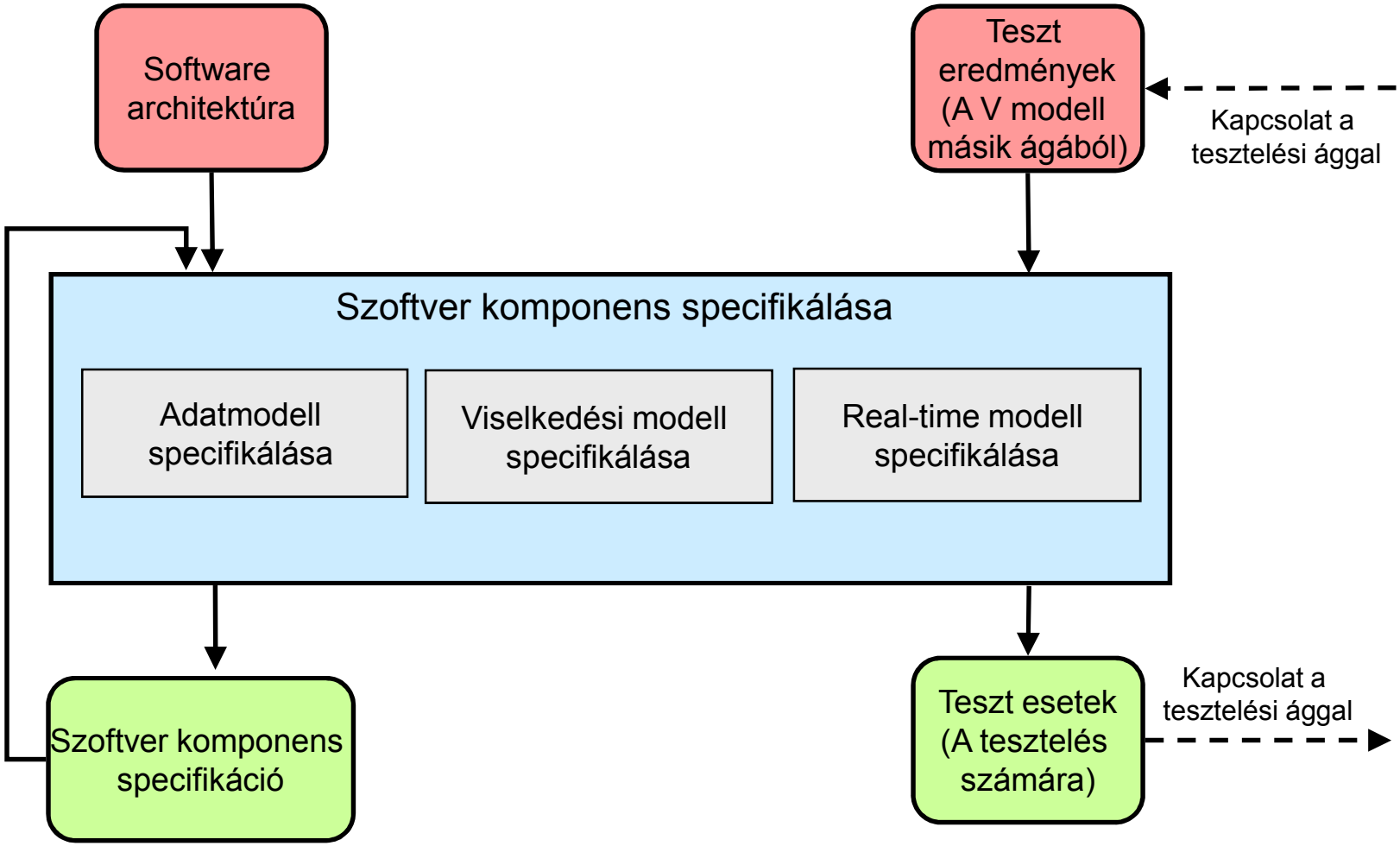
Szoftver architektúra



Szoftver működési módok

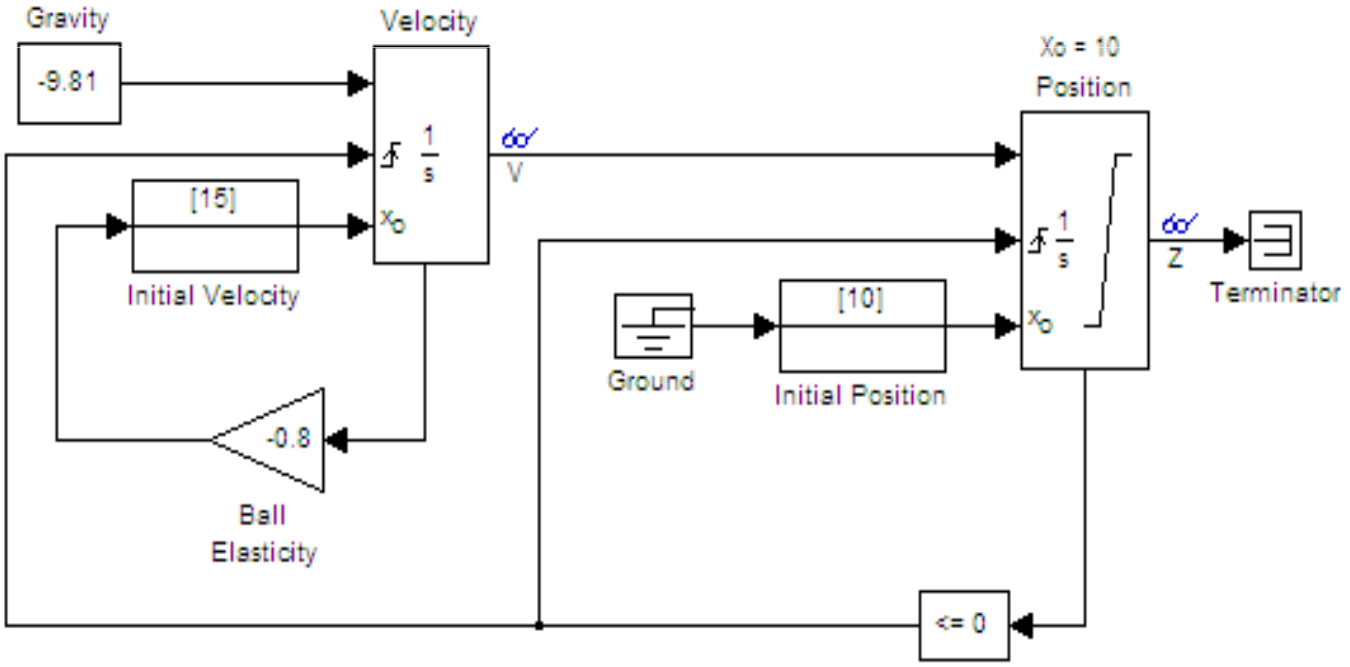


Szoftver komponens tervezés



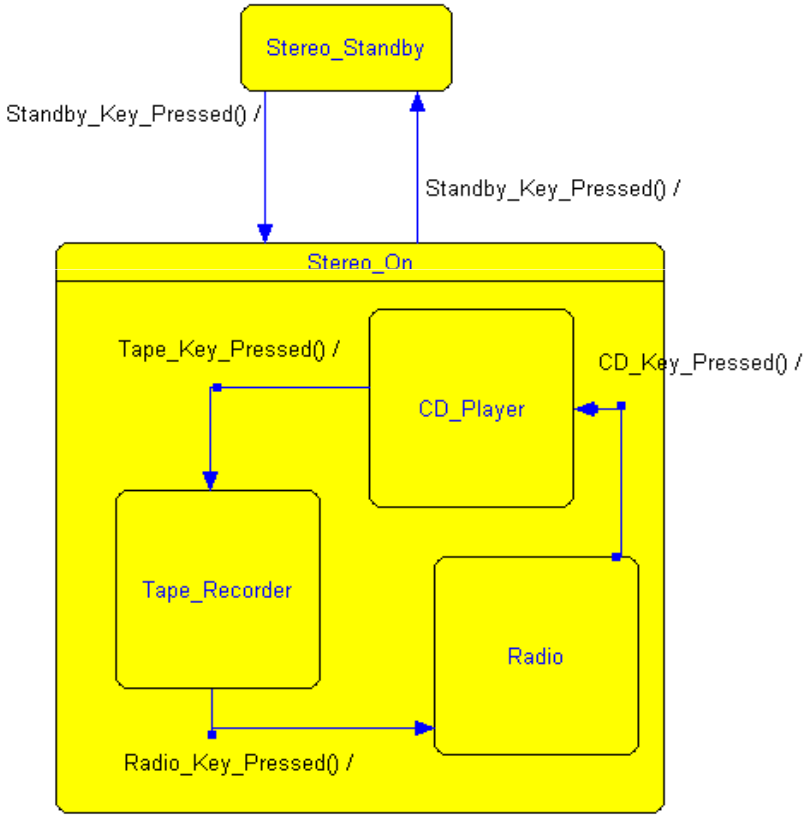
Adatmodell, adatfolyam meghatározása

- Sokszor domain specifikus nyelvvel:
  - Simulink
  - *ASCET*



Viselkedési modell

- Legtöbb esetben valamilyen állapotgépes leírás

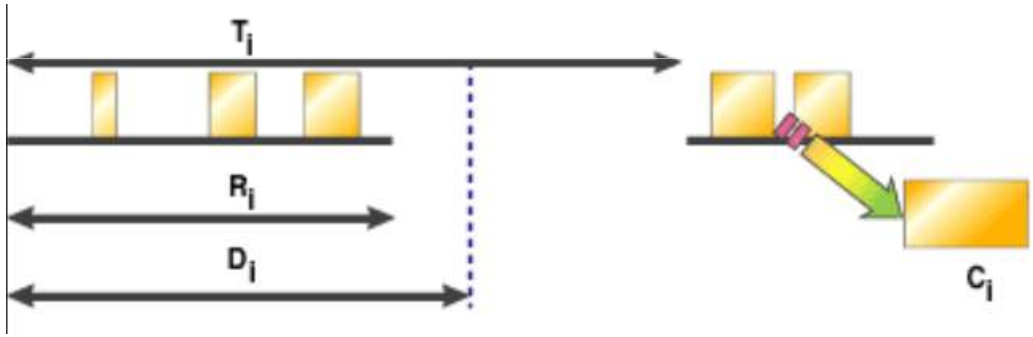


Real-Time modell specifikálása

- Tipikusan fixen ütemezett taszkok: 2,5ms, 5ms, 10ms …

Real-Time modell specifikálása

- Tipikusan fixen ütemezett taszkok: 2,5ms, 5ms, 10ms …
- **DMA:** Deadline Monotonic analysis



$$R_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Table 1 DMA example

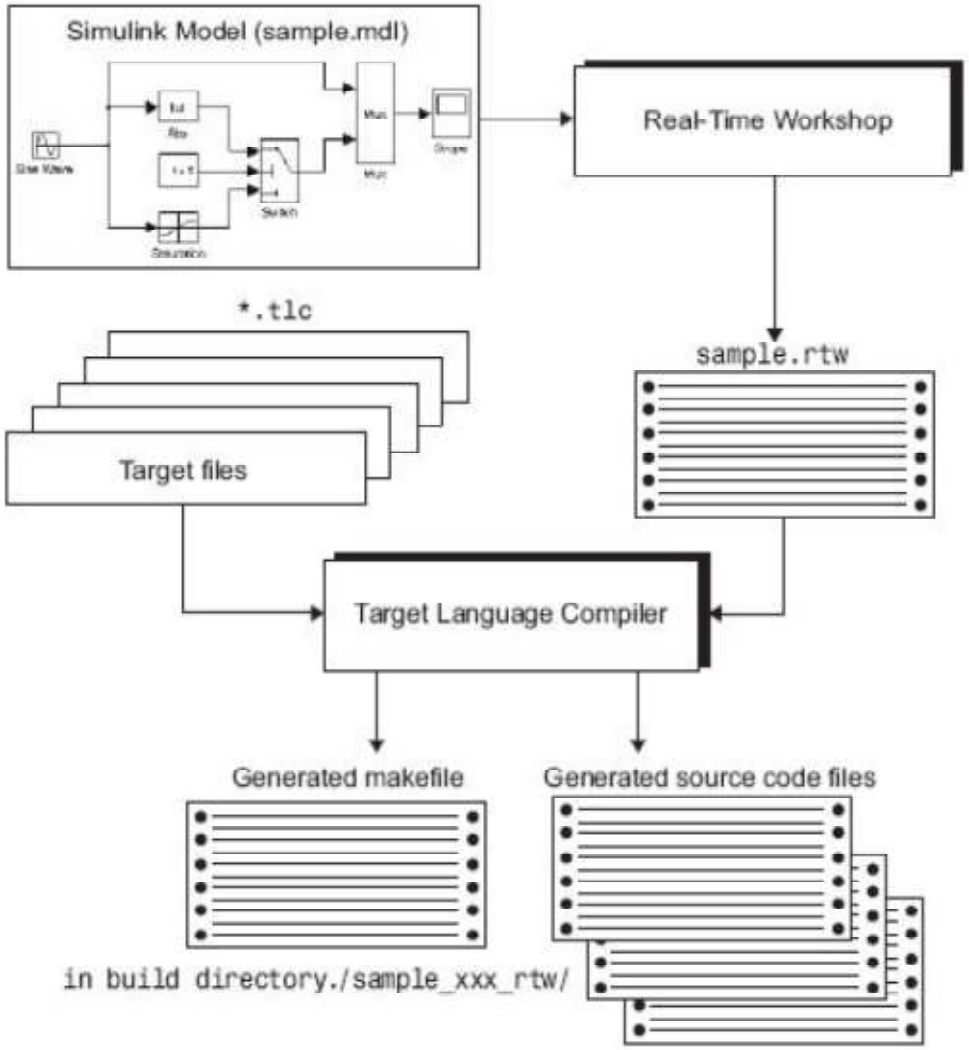
Task	T	C	D
1	250ms	5ms	10ms
2	10ms	2ms	10ms
3	330ms	25ms	50ms
4	1000ms	29ms	1000ms

Table 2 Calculation of worst-case Task 3 response time

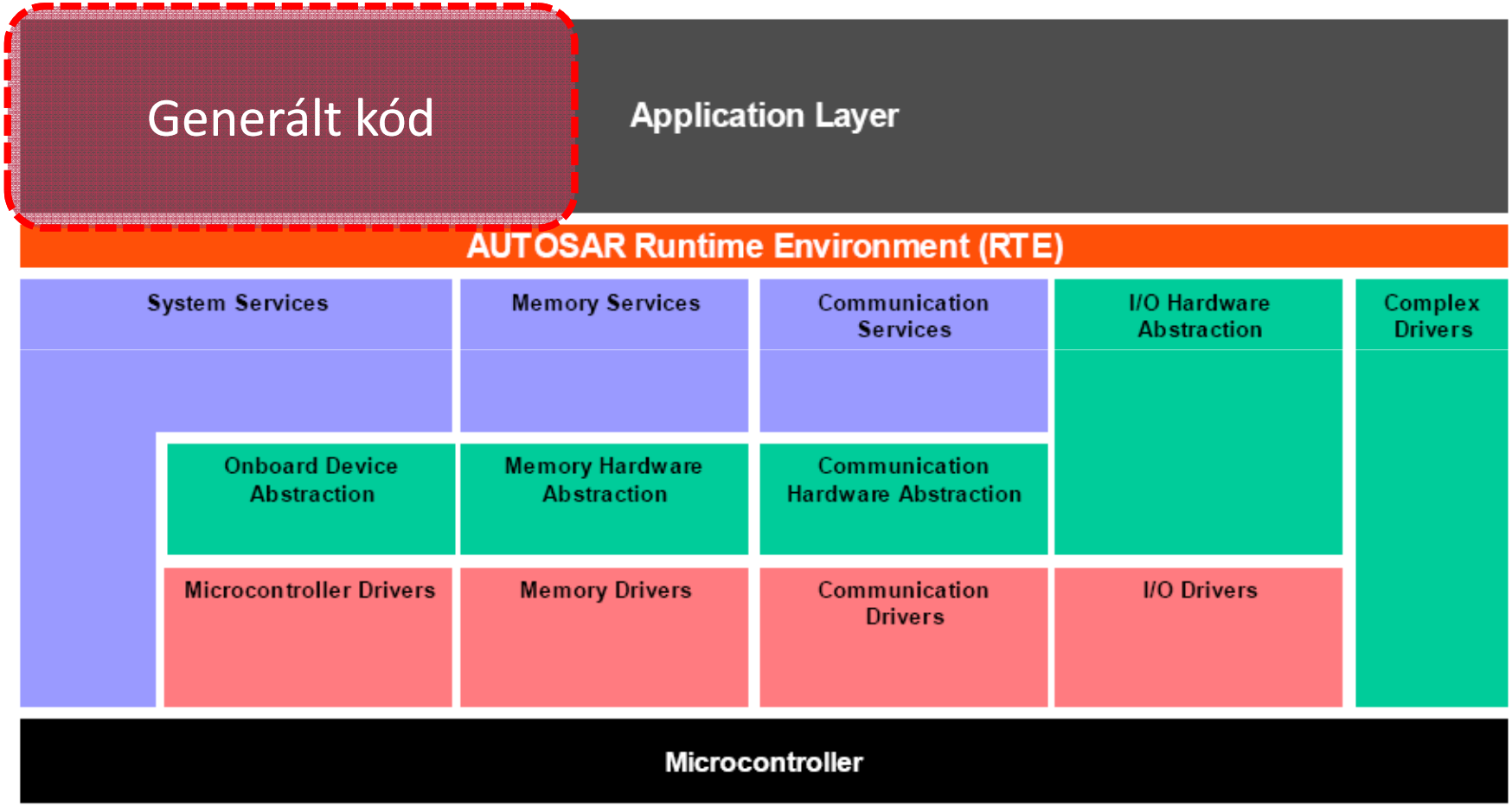
Step	R^n	I	R^{n+1}
1	0	0	25
2	25	5+3x2=11	36
3	36	5+4x2=13	38
4	38	5+4x2=13	38

Modell alapú kódgenerálás

Simulink Real-Time Workshop



Kódgenerálás alkalmazási köre, helye



Implementálás

Néhány implementációhoz kötődő megfontolás

- Hardware limitációk által okozott problémák:
  - Fixpontos, vagy lebegőpontos számábrázolás
  - Online számolás, vagy lookup table alkalmazása
  - Lebegőpontos számábrázolásból fakadó hibák

- Hardware specialitások figyelembe vétele
  - Speciális hardware-hez kötődő technológiák alkalmazása: DMA és annak specialitásai
  - Cache és annak viselkedése
  - Tightly-coupled memory
  - Belső, vagy külső RAM-ba helyezett változó
  - Speciális üzemállapotok: pl.: Sleep és az abban való viselkedés

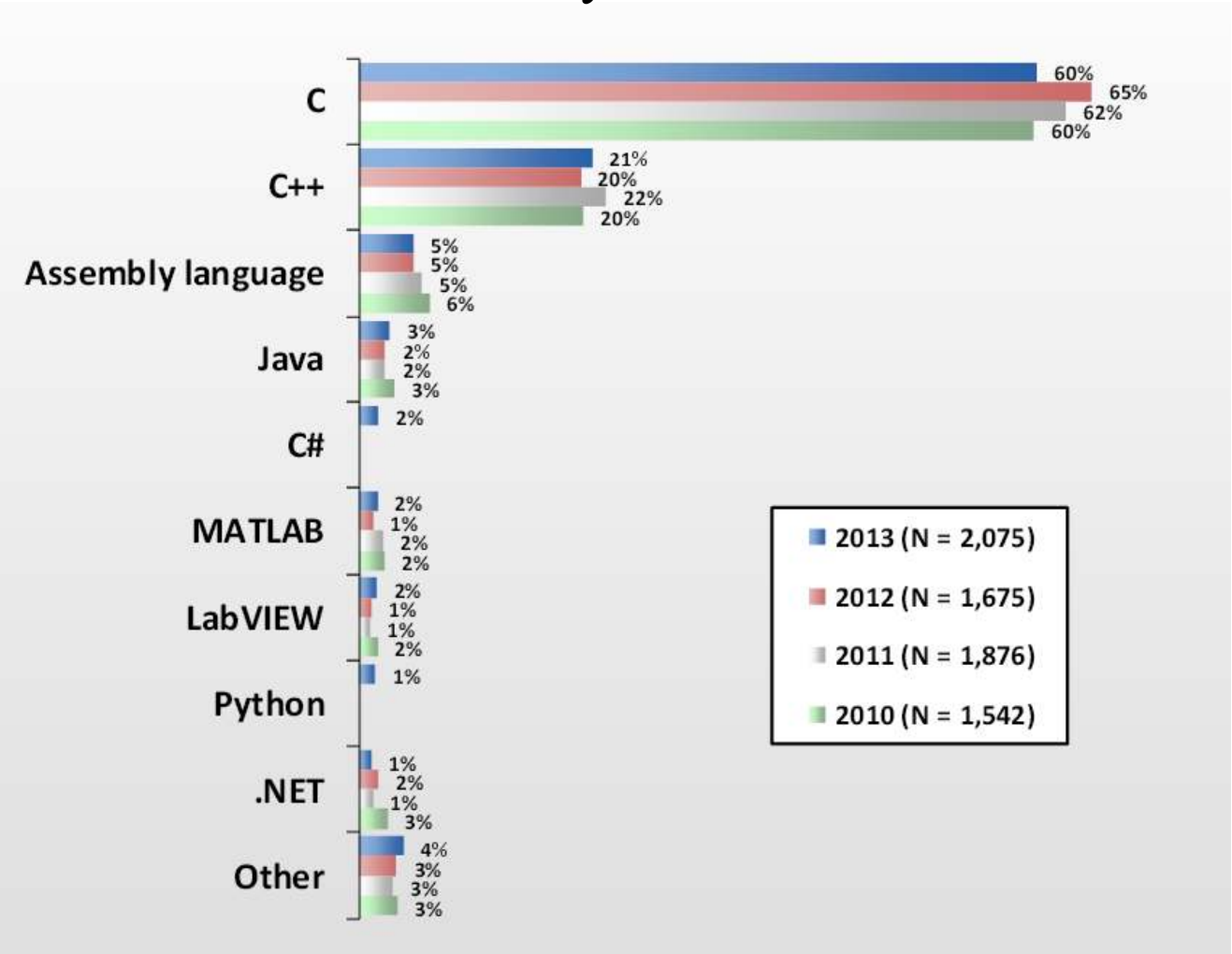
Szoftver implementálás általános szabályai

- Szinte SIL/ASIL szint függetlenül mindig kötelezőek

	Topics	ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

Implementálási nyelv

- Az Embedded Market Study statisztikái



MISRA-C nyelvi készlet korlátozás

(Motor Industry Software Reliability Association)

- MISRA-C 1998: Az első MISRA C szabályverziót 1998-ban hozták létre célja az volt, hogy javítsa az UK (United Kingdom) autóiparában használt szoftverek minőségét.
- A MISRA-C 1998 sikere nem csak az autóiparra terjedt ki, hanem széles körben alkalmazásra került a repülőgépiparban valamit az egészségügyi iparban is.
- A MISRA-C 2004: Elfogadásra került az USA-ban (SAE J2632) és Japánban is.
  - Javítja a MISRA-1998 hibáit, és pontosítja azt.
  - 121 Mandatory és 20 Advisory szabályt tartalmaz a C nyel leszűkítésére
- MISRA-C 2012: 2013 közepén megjelent új verzió a C99-es szabványára épül

MISRA-C célja

- Cél a C szabados programozói nyelv leszűkítése úgy, hogy a lehető legmegbízhatóbb nyelvi alrendszert kapjuk.
  - Szintaktikailag helyes, szemantikailag rossz megoldásokra felhívni a figyelmet.
  - Tiltani a nem egyértelmű változó típus használatot.
  - Szabályozni a precendencia zárójelezéseket.
  - Tiltani a nem strukturált programozást eredményező szerkezeteket.

Triviális szabályok

- Kódsort soha nem szabad kikommentezni
  - A kikommentezett sorok gondot okozhatnak az egymásba ágyazott kommentek, illetve a kódérthetőség miatt is.
- Ciklusváltozót nem lehet a ciklus belsejében módosítani.

```
flag = 1;
for (i = 0; (i<5) && (flag == 1); i++)
{
        flag = 0; /* Engedélyezett a ciklus gyors befejezése */
        i = i + 3; /* Nem Engedélyezett ciklusváltozó módosítás */
}
```

- A *goto* használata tiltott!
- A *continue* használata tiltott!
- Bitmanipuláló műveletet nem szabad *signed,* vagy *floating* típusokon végrehajtani (>>, <<, ~, &, ^)

Elgondolkodtató szabályok

- Kiválasztott compiler egész osztásokra való reakcióját ellenőrizni és dokumentálni kell.

  Egy ISO kompatibilis fordító két ponton is mutathat eltérő viselkedést negatív egész osztás esetében:

  - (-5/3) lehet -1 ahol a maradék -2 illetve
  - (-5/3) lehet -2 ahol a maradék +1

- Változók összeadásánál szorzásánál kicsúszás az értéktartományból:

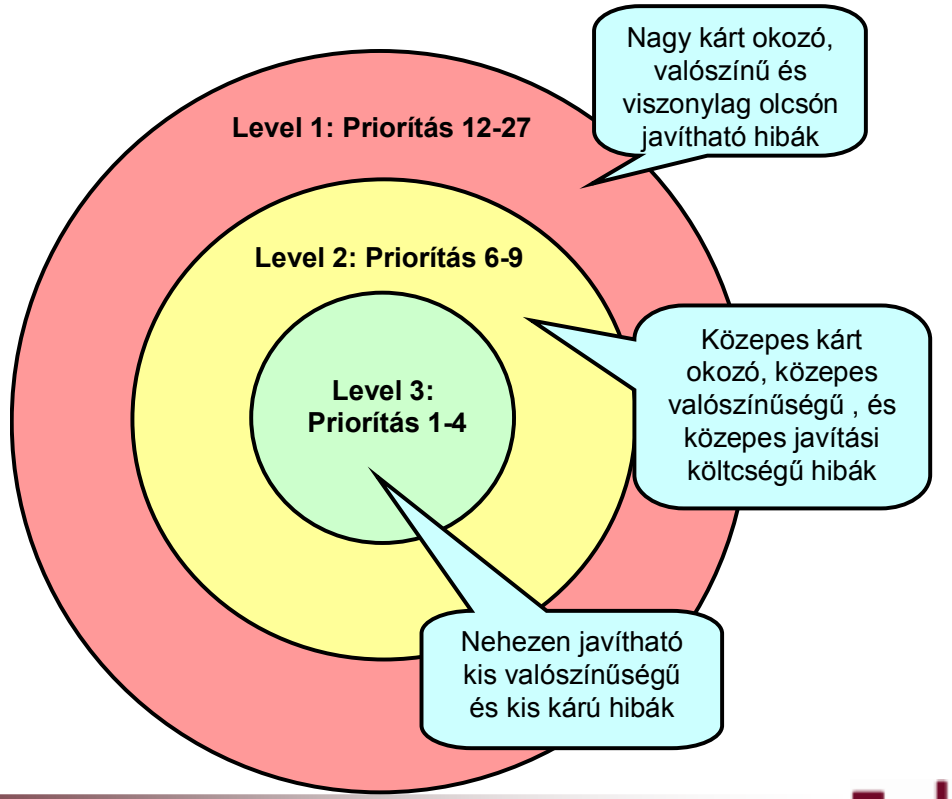
```
uint16_t u16a = 40000;
uint16_t u16b = 30000;
uint32_t u32x;

U32x = u16a + u16b; /* u32x = 70000 vagy 4464? */
```

Az összeadás aritmetikáját nem az eredmény tárolási típusa, hanem a compiler belső aritmetikája szabályozza (belső int típus), ezért, ha a belső aritmetika 16 bites akkor az összeadásban túlcsordulás következhet be. Ilyen jellegű hibalehetőségből van még jó pár darab.

SEI CERT Coding Standards

- MISRA-hoz hasonló szabályozás kevésbé beágyazott környezetre
- C, C++, JAVA, Perl, Android
- Hibák kategorizálása: Severity, Likehood, Remediation cost



Style guides

- Cél a kód egységes kinézete és szervezése
  - Nem egységes kód nehezíti a csapatmunkát és az átláthatóságot
- Nincs egységes nemzetközi szabvány
- Tipikusan cégekre jellemző szabályozások
  - C és header file-ok szerkezete
  - Változók elnevezésének stílusa
  - Vezérlési szerkezetek stílusa
  - Kommentezés módja

Tipikus C file szerkezet

1. Komment a file nevéről, rendeltetéséről, szerzőjéről, verziójáról, és verzió history-árol. (Egyes verziókövető rendszerek automatikusan generálják, módosítják a file verzióra vonatkozó kommenteket. Ilyenkor ezeket a fejlesztőnek általában tilos kézzel módosítani.)
2. Header file include-ok
3. Definiciók a következő sorrendben: Típus definíciók, Define-ok, Konstans makrók függvény makrók,
4. Globális változók: extern, nem static, static global sorrendben
5. függvények: általában hívási sorrendben, ha egy mód van rá.

Tipikus H file szerkezet

1. Komment a file nevéről rendeltetéséről, szerzőjéről, verziójáról, és verzió history-árol. A file neve sohasem lehet normál C inklúdnévvel egyező pl.: „math.h”.

2. A következő file kezdési szerkezet kötelező:

```
#ifndef EXAMPLE_H
#define EXAMPLE_H
...     /* body of example.h file */
#endif /* EXAMPLE_H *
```

3, Fejléc file-okban ne definiáljunk változókat ha egy mód van rá.

Elnevezési szabályok

- Egyik leghíresebb az ún. *Hungarian Notatinon*
  - Simonyi Károly vezetett be a Microsoftnál
- Az elnevezési szisztéma a magyar elnevezési logikából indul ki, ahol a vezetéknév megelőzi az „adott” keresztnevet
- Ezt próbálja a változókra is bevezetni, ahol először a típus, vagy alkalmazási kör szerepel és utána csak az „adott” név
- Két alsítulsa van a System és az Application
  - *System* esetében a változó típusa
  - Az *Application* esetében az alkalmazás típusa van belekódolva a változó nevébe.
- A jelölésrendszer sokszor kiegészül a láthatóságot jelölő előtéttel is

Példák *Hungarian notation*-ra

bBusy: boolean
cApples: count of items
dwLightYears: double word (system)
fBusy: boolean (flag)
nSize: integer (system) vagy count (application)
iSize: integer (system) vagy index (application)

g_nWheels: member of a global namespace, integer
m_nWheels: member of a structure/class, integer
s_wheels: static member of a class
_wheels: local variable

Vezérlési szerkezetek stílusa

brace placement	styles

`while (x == y) {`
`  something();`
`  somethingelse();`
`}`
	K&R and variants:
	1TBS, Stroustrup, Linux kernel, BSD KNF

`while (x == y)`
`{`
`  something();`
`  somethingelse();`
`}`
	Allman

`while (x == y)`
`  {`
`  something();`
`  somethingelse();`
`  }`
	GNU

`while (x == y) {`
`  something();`
`  somethingelse();`
`  }`
	Ratliff

`while (x == y) {`
`  something();`
`  somethingelse(); }`
	Lisp

Automatikus formázó tool-ok

- Artistic Style: ingyenesen leölethető formázó tool

```
--style=allman / --style=bsd / --style=break / -A1
```

Allman style uses broken brackets.

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

```
--style=pico / -A11
```

```
int Foo(bool isBar)
{   if (isBar)
    {   bar();
        return 1; }
    else
        return 0; }
```

Általános dokumentációs problémák

1. Megírjuk a kódot
2. Még talán kommentezzük is
3. Nagy erőszakra megírjuk a dokumentációt

4. Módosítjuk a kódot
5. Talán a kommentet is
6. A doksit biztos nem

Inkonzisztencia:

kód – komment – dokumentáció között

Automatikus dokumentáció generálás: Doxygen

- 1997-ben jelent meg az első verzió v0.1
- A komment – dokumentáció közötti inkonzisztenciát igyekszik feloldani
- Többfajta kommentezési stílust képes kezelni
- C stilusú JAVA doc szerű kommentek

```
/**
... text ...
*/
```

- C stílusú ! kommentek

```
/*!
... text ...
*/
```

Doxygen kommentek példa

```
/*! \fn void UART1_Init(unsigned long baud_rate, void (*handler)(void));
 *  \brief An inicialisation function to redirect STDIO to UART1
 *  \param baud_rate: Baudrate in bit/sec
 *  \param handler: Callback function for receiving UART characters with IT
 *  \return nothing
*/
```

```
/** @fn void UART1_Init(unsigned long baud_rate, void (*handler)(void));
 *  @brief An inicialisation function to redirect STDIO to UART1
 *  @param baud_rate: Baudrate in bit/sec
 *  @param handler: Callback function for receiving UART characters with IT
 *  @return nothing
*/
```

Csoportosítás

- Alapvetően file szintű dokumentálás, ahhoz, hogy logiaki egység szintű dokumentációt csináljunk szükséges a csoportok megalkotása

- Szinte az összes firmware library ezt alkalmazza

```
/** @addtogroup Az én csoportom
@{
*/

.....
/**
@}
*/
```

Csoportosítás, struktúra, hivatkozás

```
/** @brief  I2C Init structure definition  */
typedef struct
{
  uint32_t I2C_ClockSpeed;        /*!< Specifies the clock frequency */
  uint16_t I2C_Mode;              /*!< Specifies the I2C mode.
This parameter can be a value of @ref I2C_mode */
} I2C_InitTypeDef;
```

```
/** @defgroup I2C_mode

@{

*/
#define I2C_Mode_MASTER                 1
#define I2C_Mode_SLAVE                  0
/**
@}
*/
```