



# Лекция 11

## Фаза на проектиране

DAAD Project  
“Joint Course on Software Engineering”

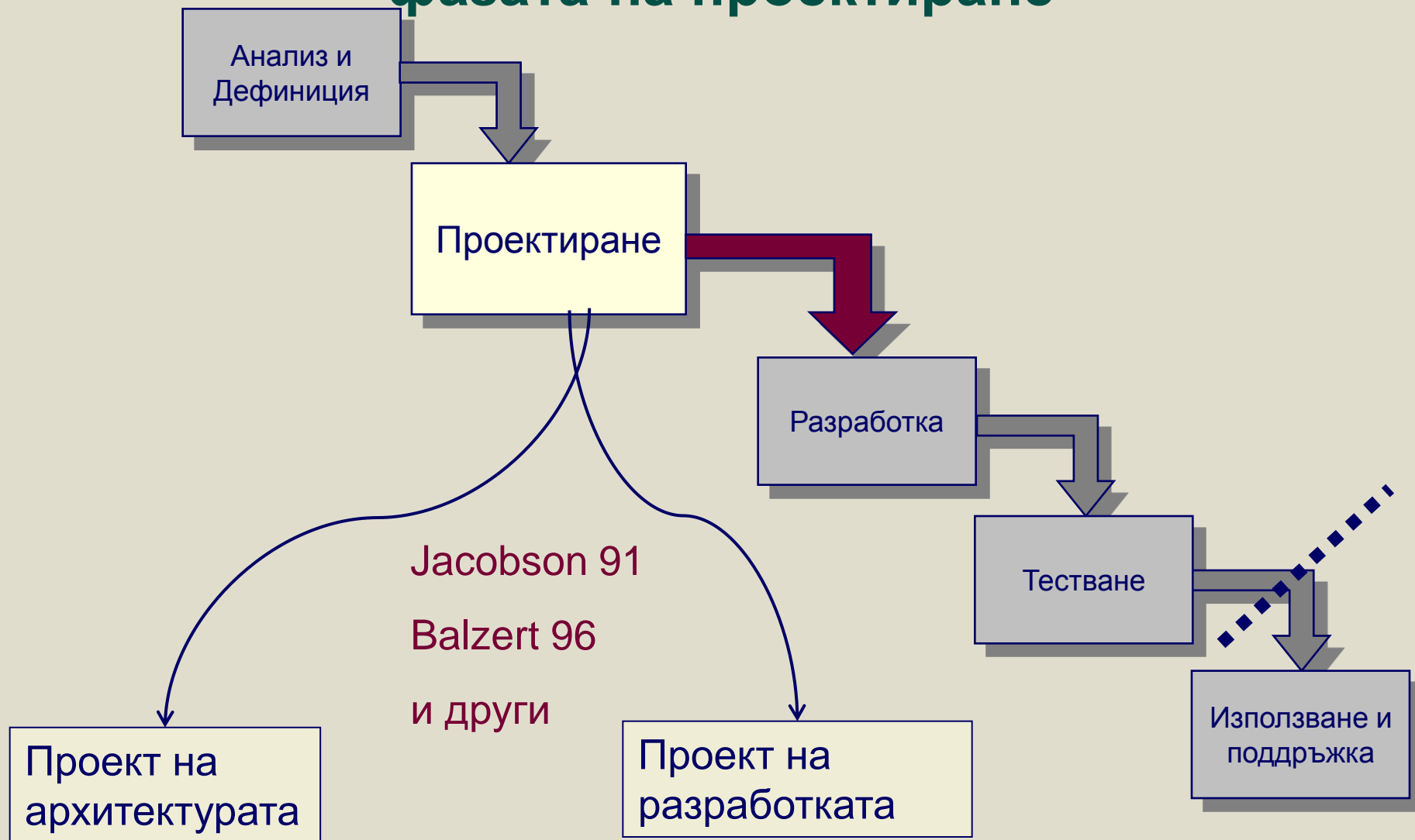
Humboldt University Berlin, University of Novi Sad, University of Plovdiv,  
University of Skopje, University of Belgrade, University of Niš, University of Kragujevac

Parts of this topic use material from the textbook  
H. Balzert, “Software-Technik”, Vol. 1, 2nd ed., Spektrum Akademischer Verlag, 2001

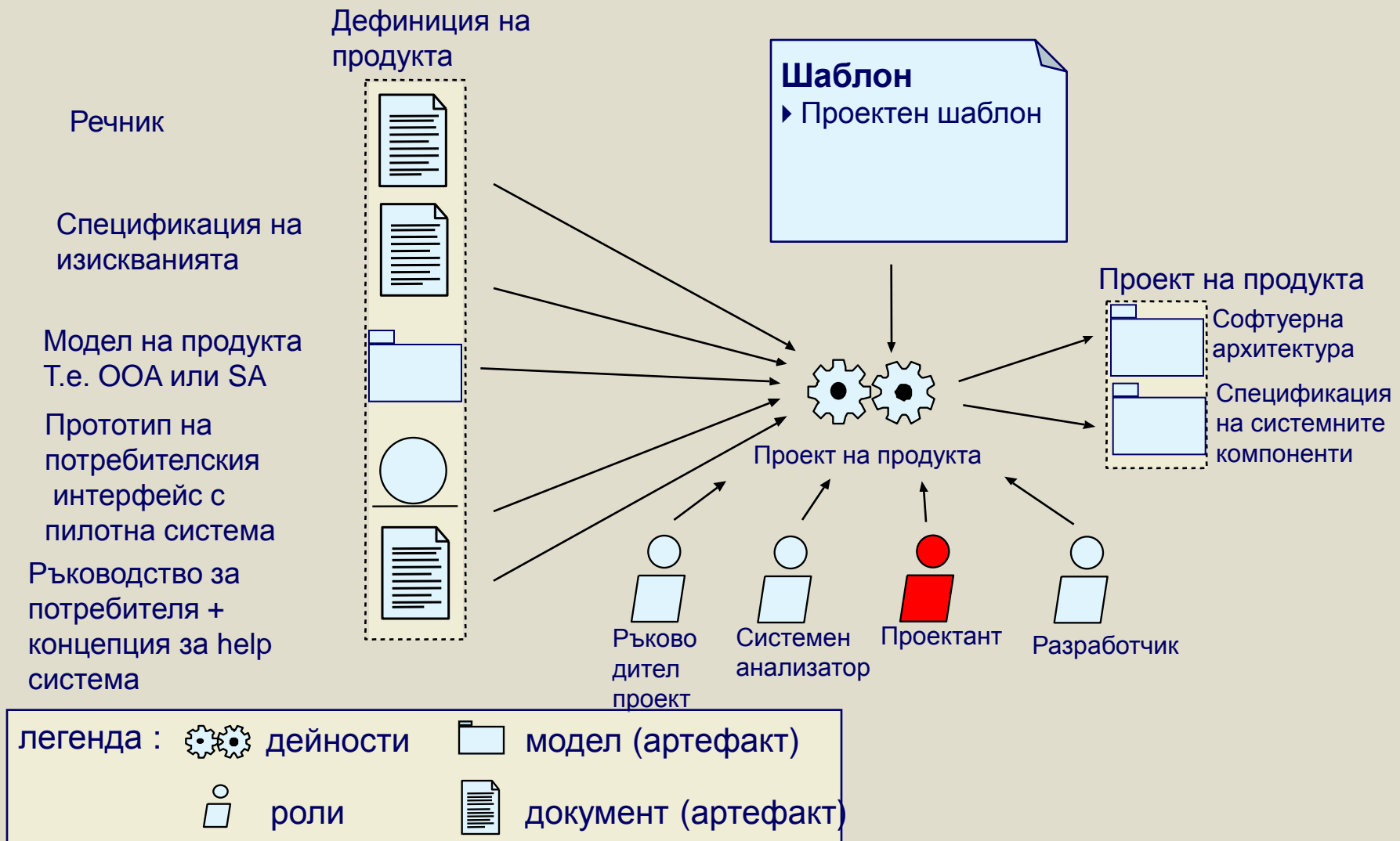
# 15. Фаза на проектиране

- a) Документи във фазата на проектиране
- b) Критерии за качество
- c) Методи за проектиране
- d) Влияещи фактори върху софтуерната архитектура

# Класическия водопаден модел: разделяне на фазата на проектиране

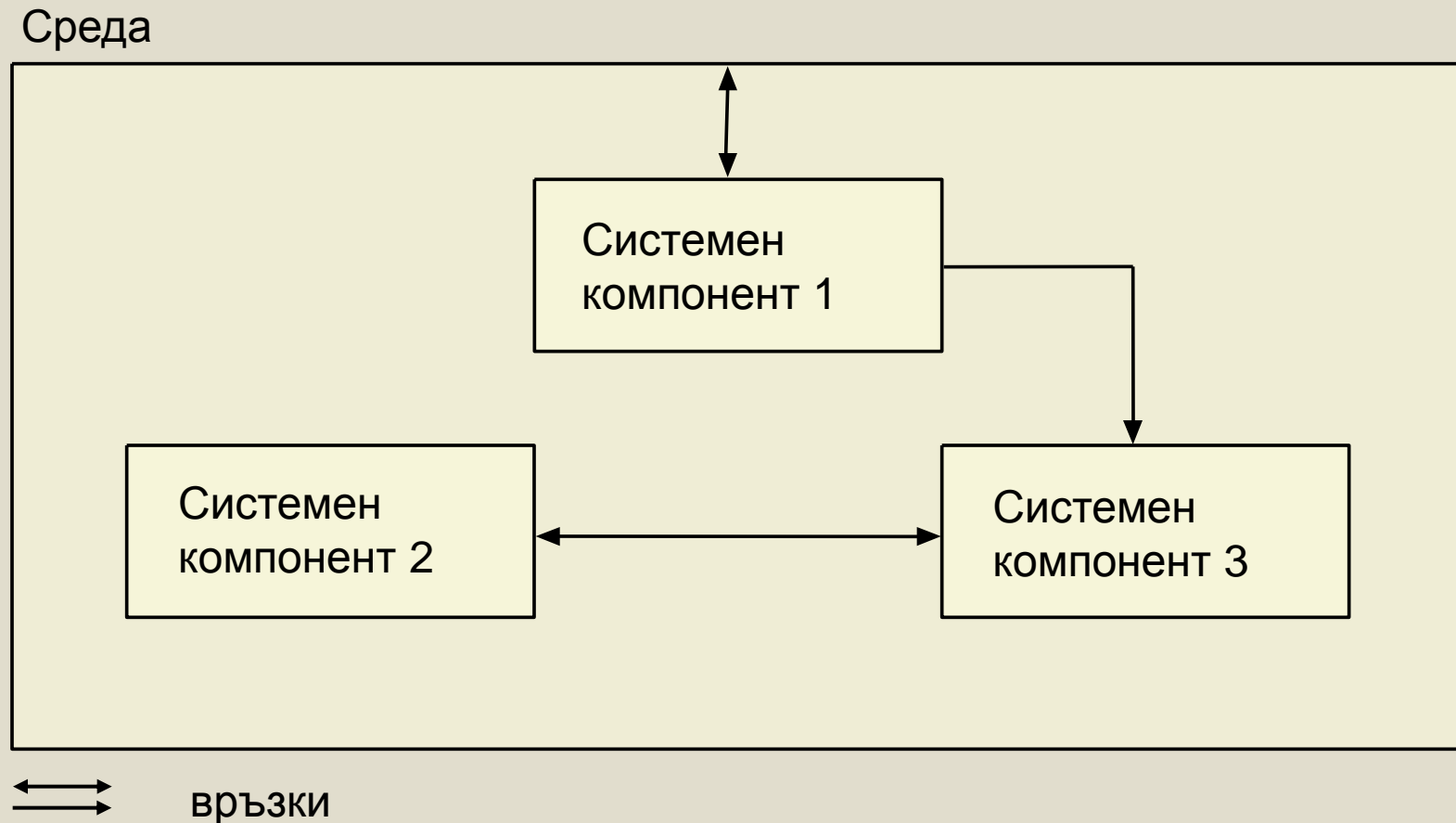


# Процеса на проектиране в контекста на софтуерната разработка



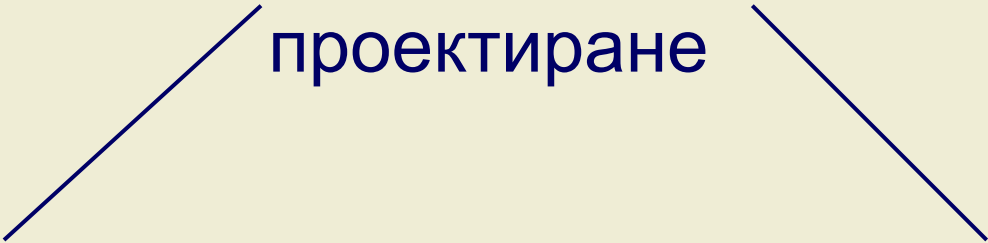
# Софтуерна архитектура

- Състои се от системни компоненти и връзки



# SW архитектура и спецификацията на компонентите (1)

## Документи във фазата на проектиране



```
graph TD; A[Документи във фазата на проектиране] --> B[SW архитектура]; A --> C[Спецификация на компонентите]
```

SW архитектура

Спецификация на компонентите

= КОМПОНЕНТИ +  
ВРЪЗКИ

M. Shaw, D. Garland:  
Software Architecture,  
Prentice Hall, 1996

Компонентите  
като черни кутии

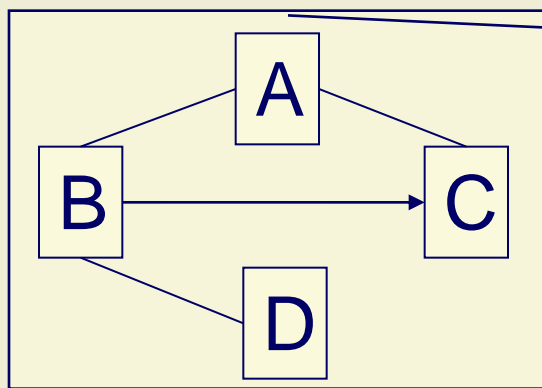
- Външно поведение
- Вече е прилагано във фазата на A&D :  
компонентите като подсистеми

# SW архитектура и спецификация на компонентите (2)

## Документи във фазата на проектиране

SW архитектура

Спецификация на компонентите



А: Интерфейс

Синтаксис

Семантики

Тип на компонентите:

- атомарни: функция, модул, клас
- подсистеми и други

Тип на връзките:

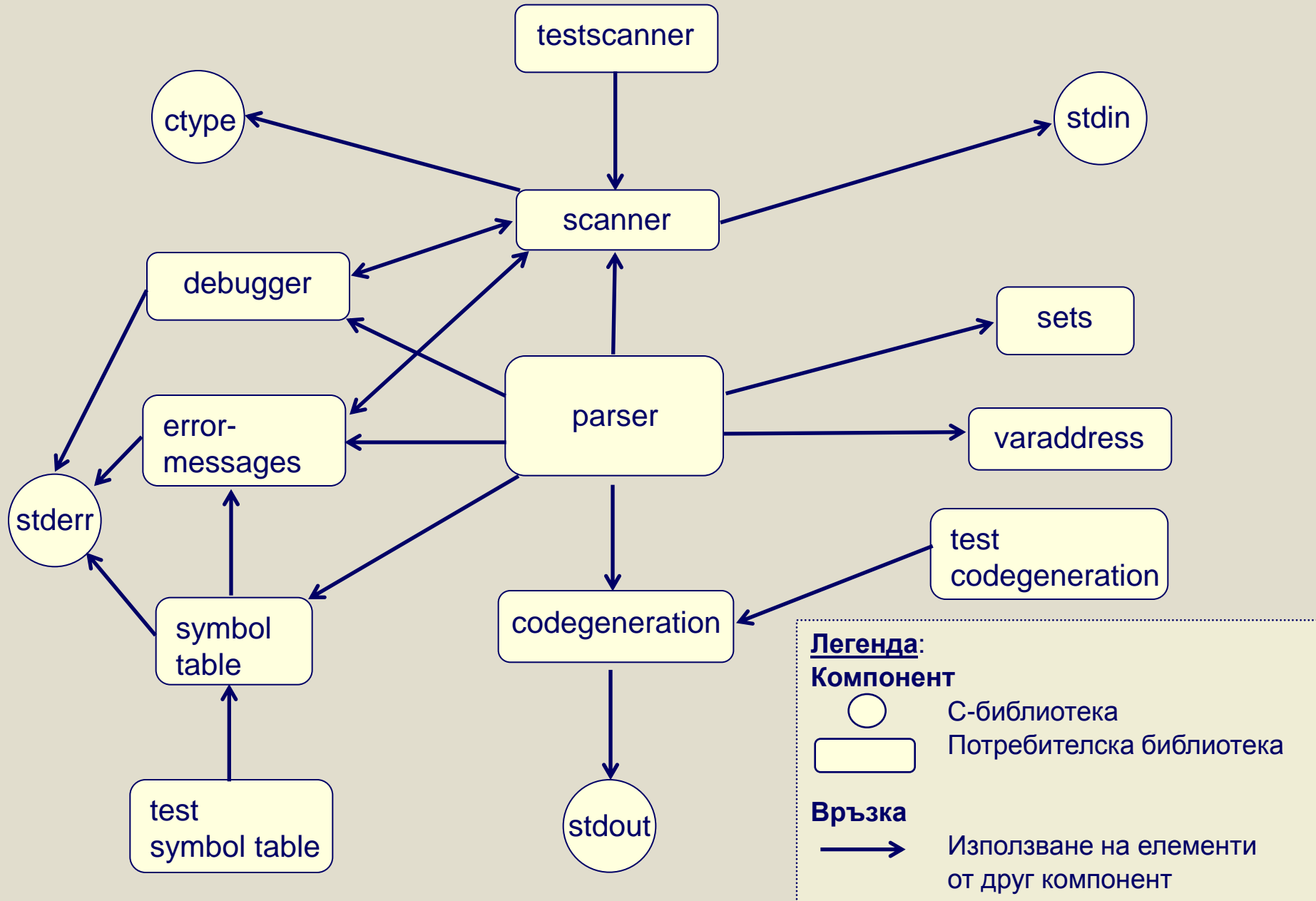
- извикване, асоциация, агрегация, и

Функция (Процедура):

сигнатура, поведение (неформална, формална спецификация)

Класове: Клас диаграми, езикова и формална спецификация

# Софтуерна архитектура на еднопасов компилатор на Pascal





# Компонентна спецификация на scanner:

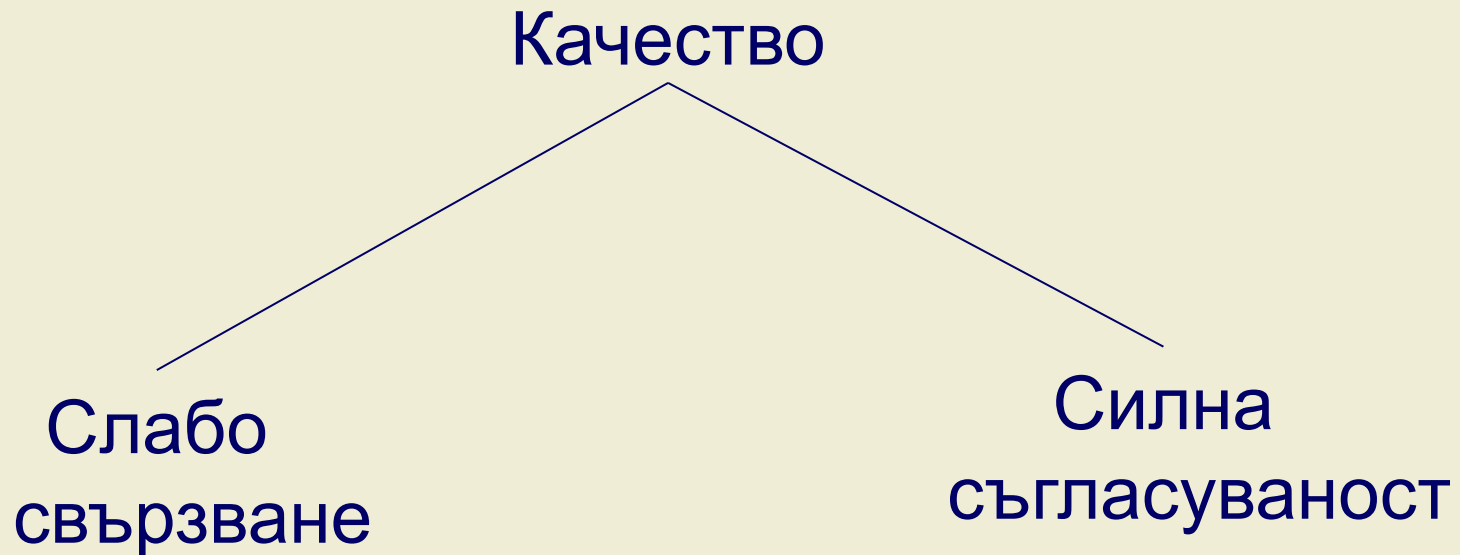
## клас диаграма с езиково описание

Scanner	<u>Атрибути</u> Представяне на текущ символ <ul style="list-style-type: none"><li>▶ sy символ клас</li><li>▶ line брой линии на текущия символ</li><li>▶ pos брой колони на текущия символ</li><li>▶ intval стойност integer, ако текущия символ е цяло число</li><li>▶ floatval стойност float, ако текущия символ е реално число</li><li>▶ idval идентификатор представен като string</li></ul>
sy: SymbolClass line: int pos: int intval: int floatval: float idval: string  initscanner() getsy() is_sy(sk) int_val() float_val() get_line() get_pos()	<u>Операции</u> <ul style="list-style-type: none"><li>▶ initscanner() връща първия текущ символ</li><li>▶ getsy() връща следващия текущ символ</li><li>▶ is_sy(sk) тества дали текущия символ принадлежи на символ клас <b>sk</b></li><li>▶ get_pos() връща текущата позиция на колоната</li></ul>

# 15. Фаза на проектиране

- a) Документи във фазата на проектиране
- b) Критерии за качество
- c) Методи за проектиране
- d) Влияещи фактори върху софтуерната архитектура

# Какво е „добра“ софтуерна архитектура?



Важно: съдържа произволни типове компоненти (функции, класове, подсистеми, и др.)

# Свързване

## Интерфейсни връзки *между компоненти*

### ► *Цел:*

- Интерфейсите колкото е възможно по-прости (обмяна на малко информация)

### ► *Резултат:*

- по-лесни за разбиране
- по-лесни за модифициране (за промяна на компоненти)

# Съгласуваност

Логическа връзка между елементи (данни, операции) във *вътрешността на компонента*

## ► *Цел:*

- Всички елементи на компонента решават обща задача

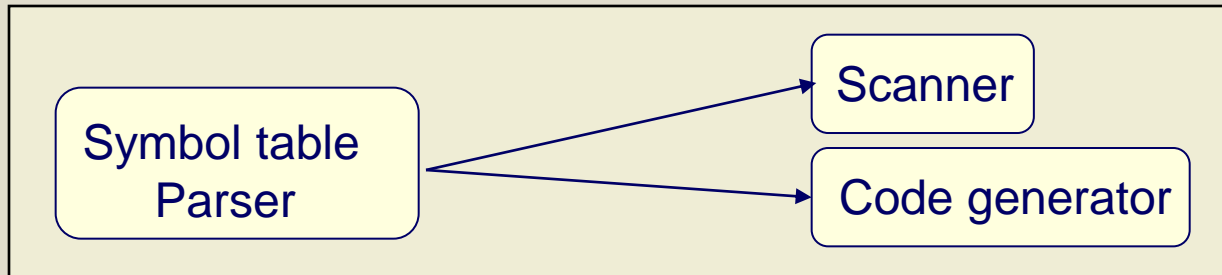
## ► *Резултат:*

- разработване на компоненти за решаване на конкретен проблем
- по-лесни за разбиране

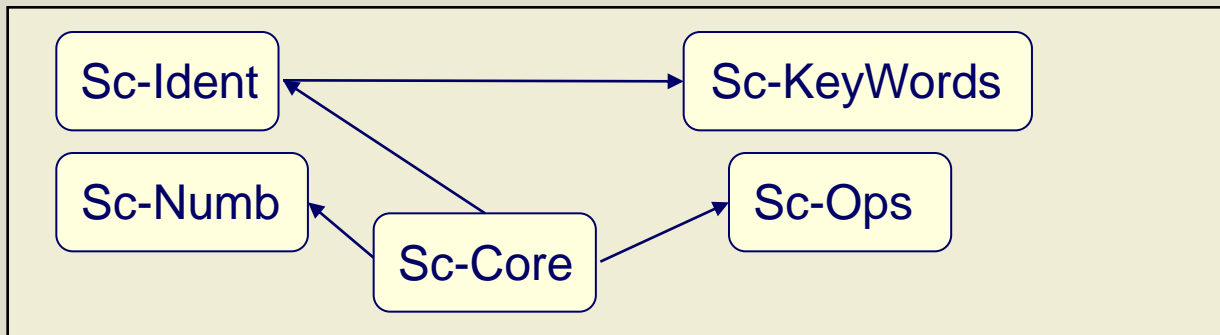
# Съгласуваност: проблеми

## ► *проблеми:*

- Няколко задачи решени в един компонент, например компилатор:



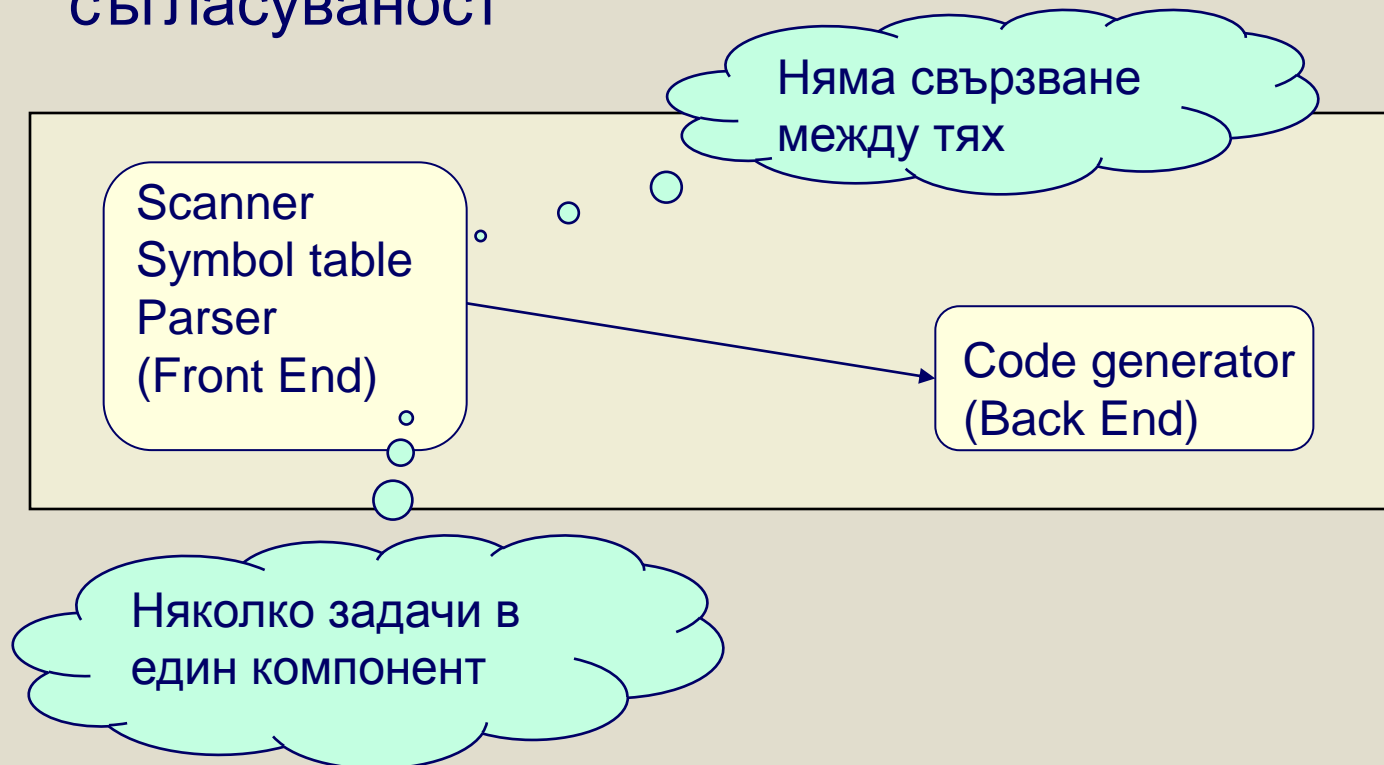
- Една задача разпределена в няколко (или доста) компоненти



→ Прекалено много интерфейсни връзки

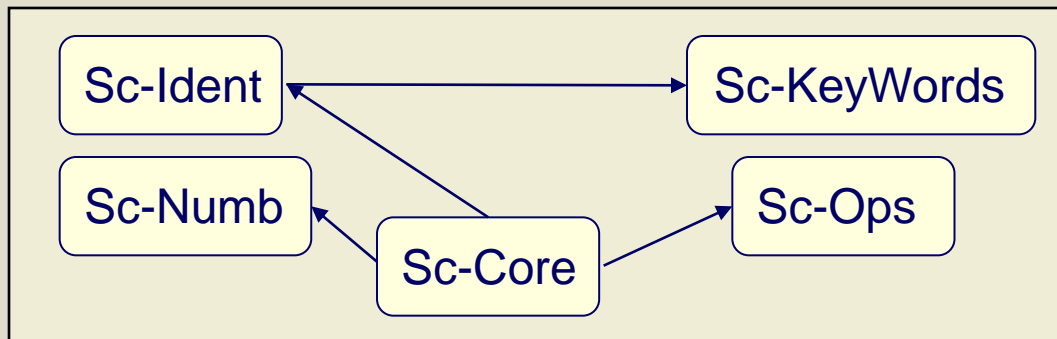
# Свързването в сравнение със съгласуваността (1)

- **Проблем:** тези две изисквания (слабо свързване, силна съгласуваност) са в съперничество
  - По-малко компоненти: по-малко свързване, по-малка съгласуваност



# Свързването в сравнение със съгласуваността (2)

- **Проблем:** тези две изисквания (слабо свързване, силна съгласуваност) са в съперничество :
- Твърде много компоненти: много връзки, голяма съгласуваност
- Скенера се разделя на няколко части (под-скенери)



Висока  
съгласуваност  
между под-  
задачите

Много нови ,под-интерфейси' в  
компонента ,Скенер' които  
трябва да бъдат разбрани



# Изборен списък „Как да открием под-системите“ (1)

Пример: свързване и съгласуваност за *под-системите* в  
ОО свят

(Balzert, vol.1, 1. edition, 1996, p. 391)

Под-системата притежават ли *силна съгласуваност*?

Силна съгласуваност има когато под-системата

- a. съдържа област от въпроси, които могат да бъдат разгледани и разбрани самостоятелно(т.е., за себе си),
- b. има добре дефиниран интерфейс към средата,
- c. не е направена само за известен брой класове, но също позволява системата да се види в по-високо ниво на абстракция,
- d. има изразително име което съответства на имената на класовете .

# Изборен списък „Как да открием под-системите“ (2)

## 2. Има ли под-системата *слабо свързване*?

Следното трябва да е изпълнено за интерфейсите между две под-системи:

- a. За всеки под-клас всички супер-класове трябва да се съдържат в същата под-система.
- b. За всеки клас всички агрегирани класове трябва да бъдат в същата под-система.
- c. Интерфейса трябва да съдържа колкото е възможно по-малко асоциации и пътища на съобщения.

# Изборен списък „Как да открием под-системите“ (3)

## 3. *bottom-up* или *top-down* подход?

bottom-up:

- a. Започва се от съществуващи класове, проучва се кои класове са логически свързани.
- b. Проверка на характеристиките за съгласуваност и свързване на потенциална под-система.

top-down:

- a. Първо се разделя цялата система на под-системи и ако е необходимо разделяме подсистемите по-нататък.
- b. Проверка на характеристиките за съгласуваност и свързване на потенциална под-система

## 4. Какъв размер трябва да има под-системата?

Критерии за разумен размер са:

- 10 до 15 класа
- една A4 страница.

# 15. Фаза на проектиране

- a) Документи във фазата на проектиране
- b) Критерии за качество
- c) Методи за проектиране
- d) Влияещи фактори върху софтуерната архитектура

# Преход между фазите: Анализ и дефиниция → проектиране

OOA модел



OOD модел

*разширен  
модифициран  
оптимизиран  
адаптиран към средата*

Същите механизми за описание: Клас диаграми

Структурен анализ



Структурно проектиране

*Структурата се променя*

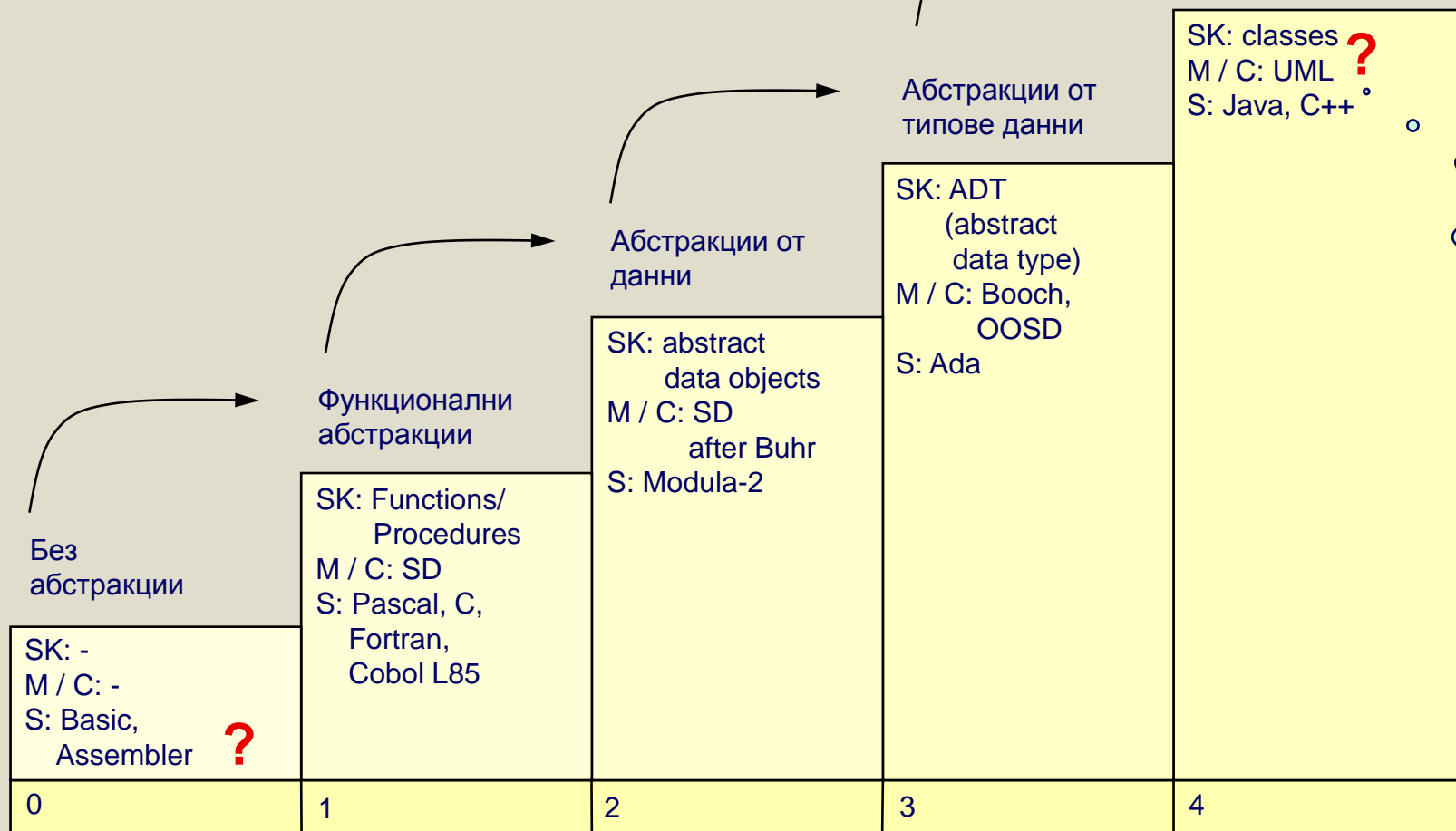
Промяна: DFD → Структурна схема

(Методология за преход от анализ към проектиране)

# Развитие на методите за проектиране

Проблем ?

Обектно-ориентирано проектиране ?



ООП

Легенда: SK = Системен компонент  
M / C = Метод / понятие  
S = Програмен език

време

# Видове абстракции

„Клас абстракция“ =

Абстрактни данни  
+ наследяване  
+ полиморфизъм

Обектно-ориентирано проектиране

Абстрактни данни

Абстрактни данни

Функционални абстракции

Без абстракции

SK: -  
M / C: -  
S: Basic, Assembler

SK: Functions/  
Procedures  
M / C: SD  
S: Pascal, C,  
Fortran,  
Cobol L85,  
Basic,  
Assembler

SK: abstract  
data objects  
M / C: SD  
after Buhr  
S: Modula-2

SK: ADT  
(abstract  
data typ)  
M / C: Booch,  
OOSD  
S: Ada

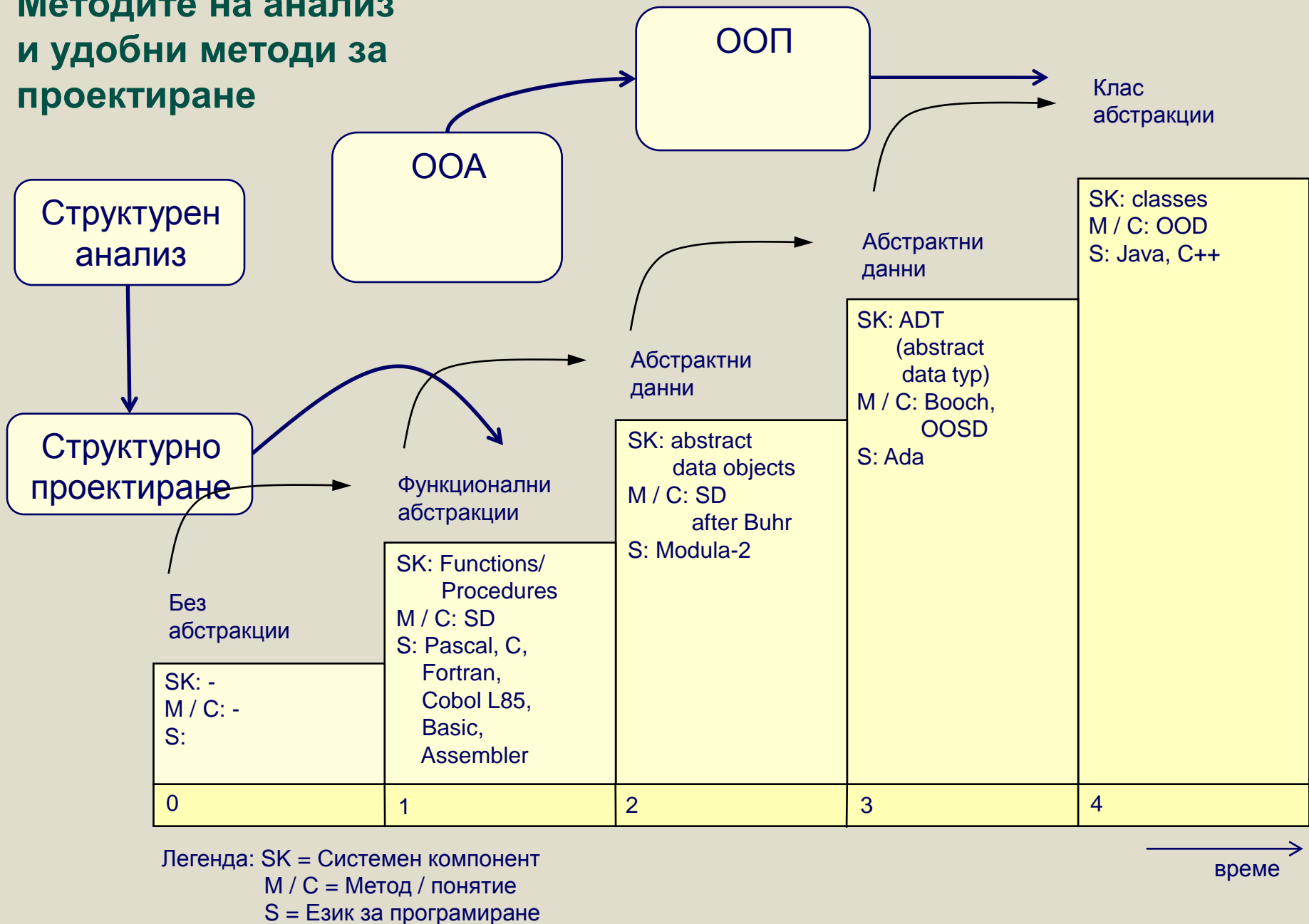
SK: classes  
M / C: UML  
S: Java, C++

0	1	2	3	4
---	---	---	---	---

Легенда: SK = Системен компонент  
M / C = Метод / понятие  
S = Програмен език

време

# Методите на анализ и удобни методи за проектиране

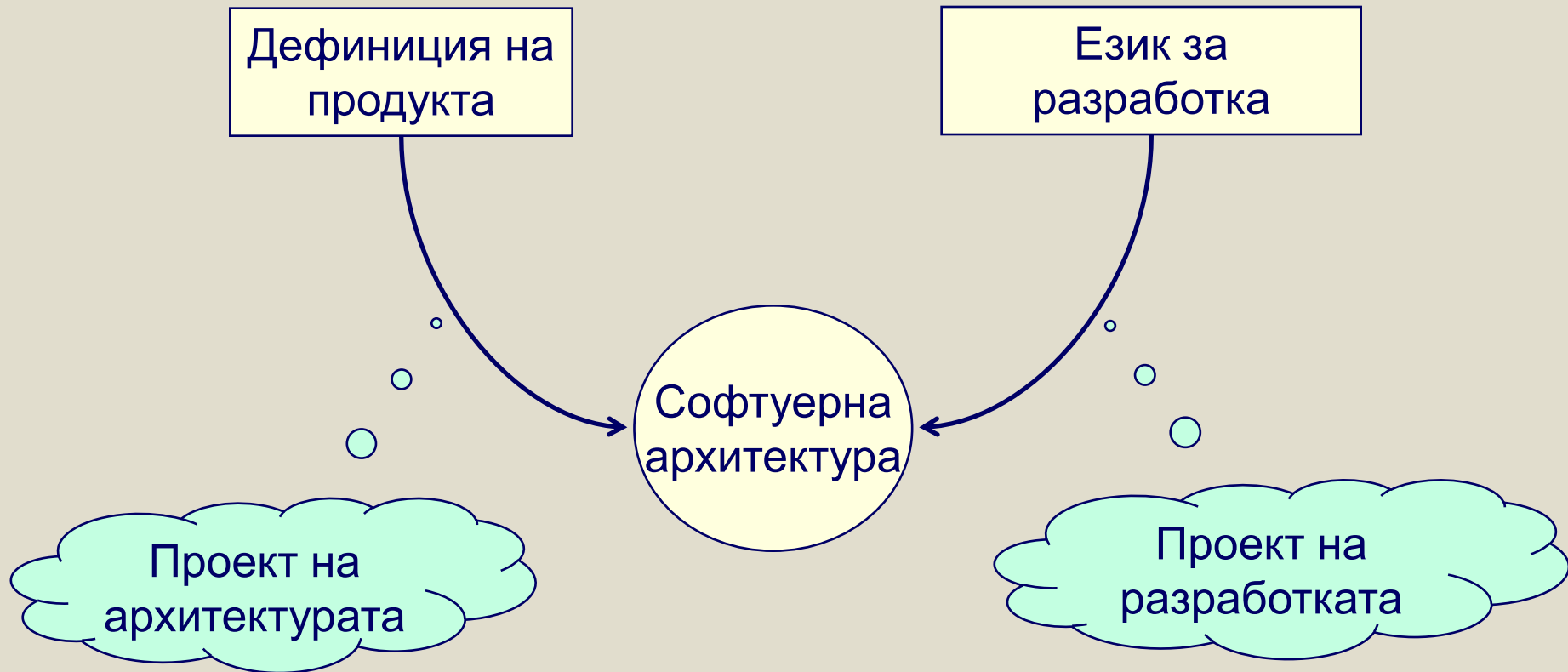


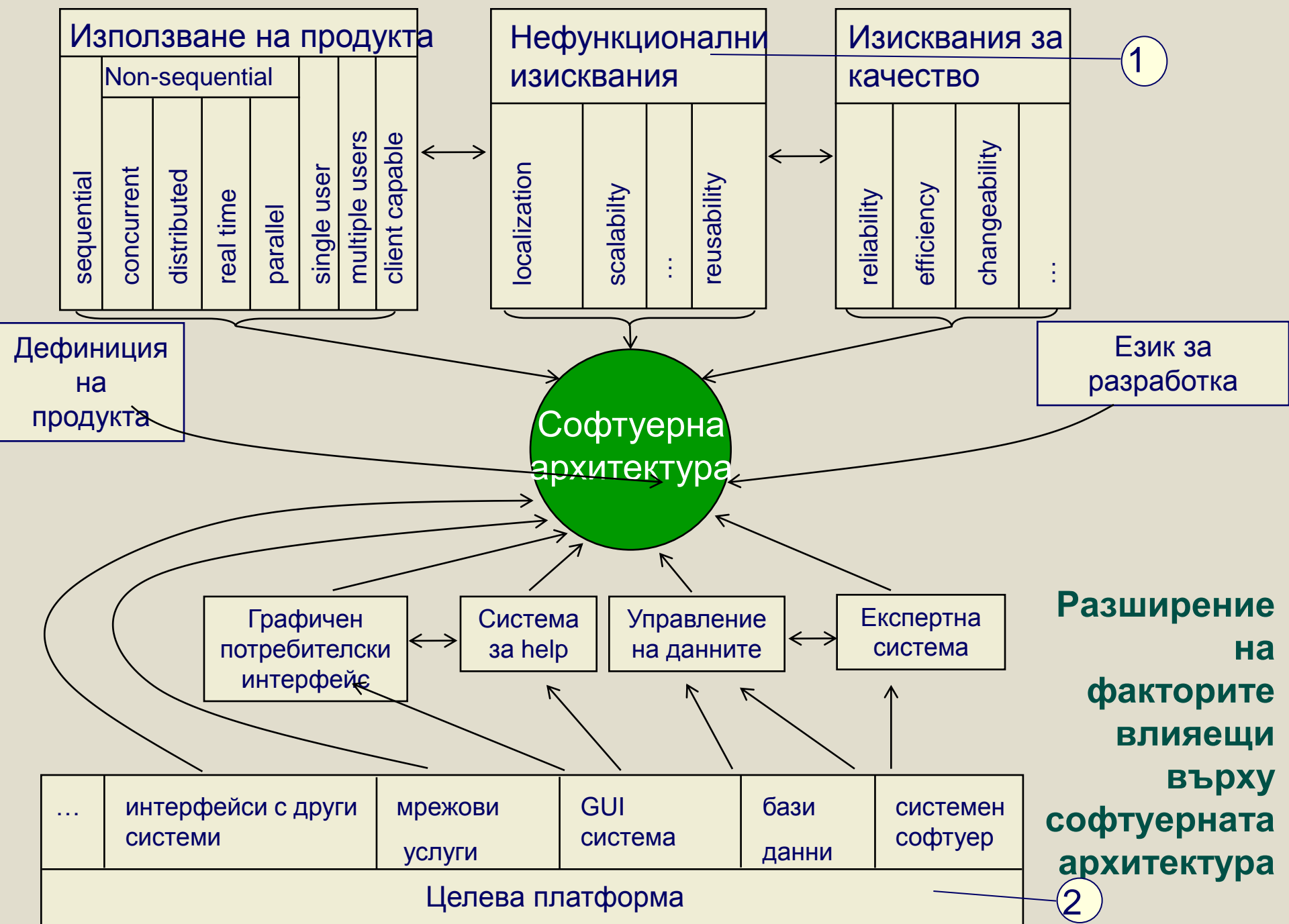


# 15. Фаза на проектиране

- a) Документи във фазата на проектиране
- b) Критерии за качество
- c) Методи за проектиране
- d) Влияещи фактори върху софтуерната архитектура

# Основни влияещи фактори върху софтуерната архитектура





# Влияещи фактори: използване на продукта

Използване на продукта					
последователен	непоследователен				един потребител
	едновременно	разпределен	real time	паралелен	
едновременно от повече потребители					удобен за клиента

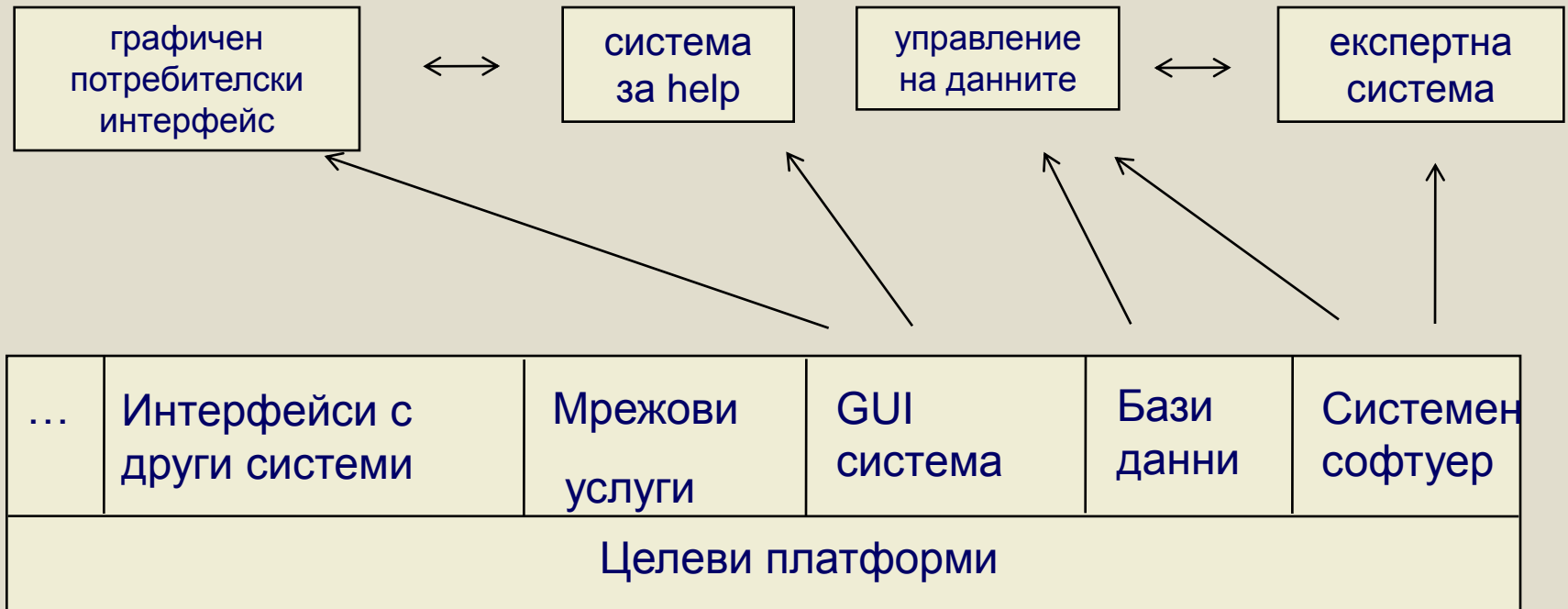
# Влияещи фактори: нефункционални изисквания

нефункционални изисквания			
интернационален	машабируем	...	повторно използване

# Влияещи фактори: изисквания за качество

ИЗИСКВАНИЯ ЗА КАЧЕСТВО			
надеждност	продуктивност	променливост	...

# Влияещи фактори: целеви платформи



# Влияещи фактори: нефункционални изисквания

## ► Интернационалност:

- Различни национални езици
- Специфични за страната функционалност (като ДДС)

## ► Мащабируемост:

- Инсталация за различни платформи
- Различни GUI системи

Заменяеми  
компоненти

## ► Наличност на библиотеки за повторно използване

- Проверка дали проекти и проектни шаблони са

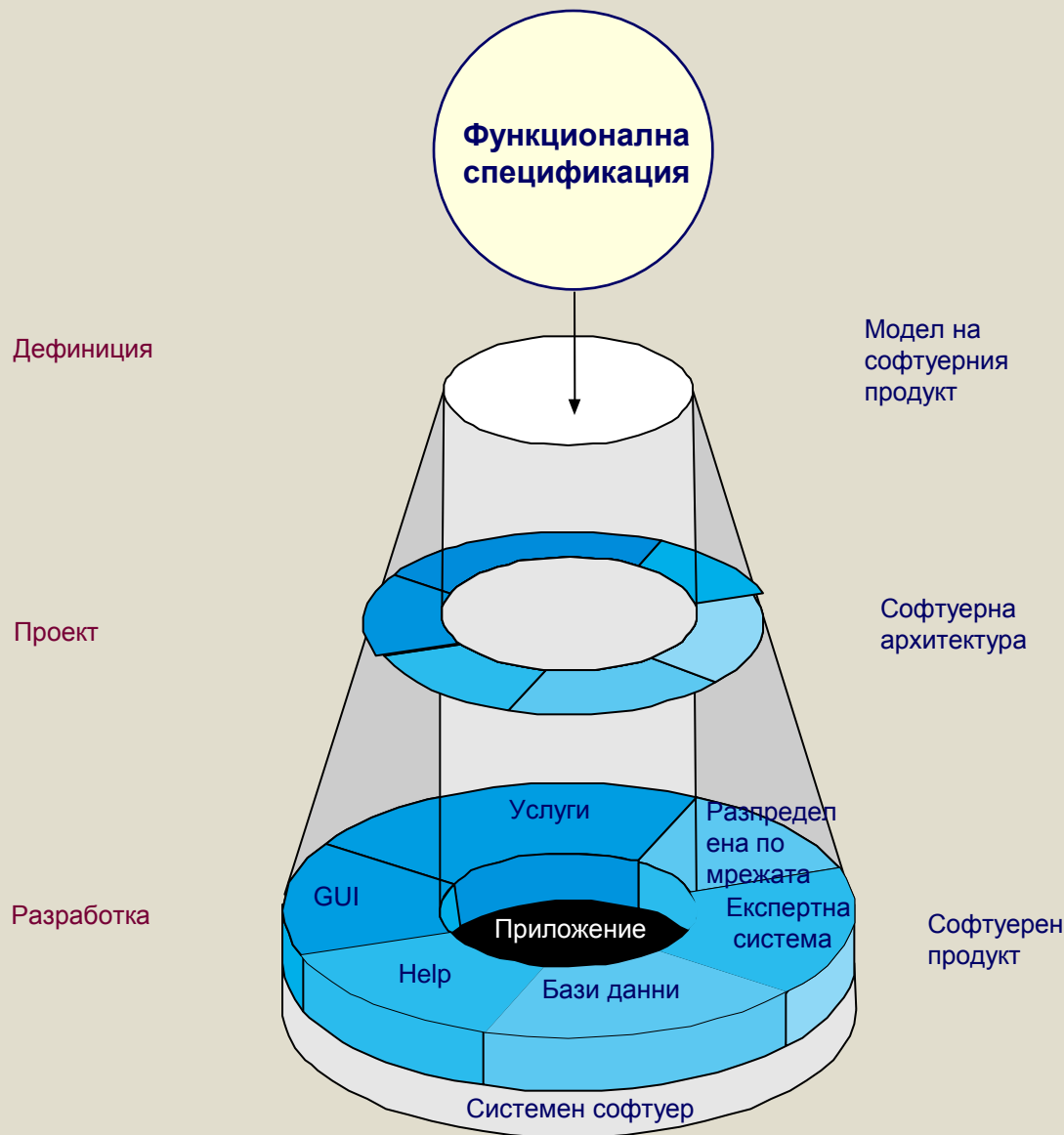
налични за избраната архитектура



## 2 Влияещи фактори: целеви платформи

- ▶ Основно софтуерният продукт трябва да бъде интегриран в платформената архитектура на целевата система.
- ▶ Те описват многообразие от интерфейси, услуги и продукти с голяма вътрешна зависимост и за различни системи.
- ▶ Платформените услуги помагат да се изолира софтуерната архитектура на новия софтуерен продукт от основната функционалност на системата
- ▶ Така софтуерните продукти могат да бъдат преносими и независими от производителя
- ▶ Като допълнение се спестяват усилия, поради използването на платформени услуги, а тези услуги не е нужно да бъдат част от софтуерния продукт.

# От функционалната спецификация до крайното приложение



Софтуерен продукт = разработване на функционална спецификация  
+ разработване на допълнителни услуги  
+ интеграция на съществуващи "помощни системи"  
+ интеграция в системна среда