


# Софтуерни метрики и формални модели на софтуерни системи



# Agenda/Съдържание

---

- ▶ Софтуерни метрики
- ▶ Понятие за качество на софтуерните системи
  - ▶ Измерване на качеството
- ▶ Формални модели

# Софтуерни метрики

---

- ▶ Показател за това доколко дадена софтуерна система или процес притежава някакво свойство.
- ▶ Какво обикновено се измерва за софтуера?
  - ▶ Функционалност?
  - ▶ Нещо друго?

# Нефункционални изисквания

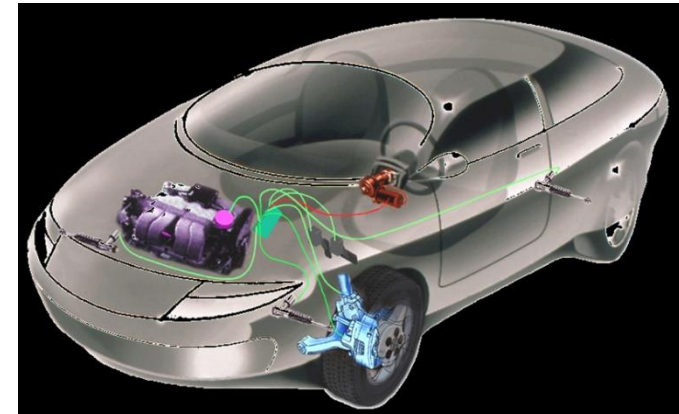
---

- ▶ Функционалните изисквания определят **какво** трябва да прави софтуерната система
- ▶ Нefункционалните изисквания определят **как** софтуерната система да работи
  - ▶ На практика качествата поставят ограничения върху начина по който системата ще се изпълнява

# Нефункционални изисквания

---

- ▶ Доста често ги наричаме и качествени изисквания
- ▶ Други имена:
  - ▶ Нефункционални свойства/атрибути (Properties/Attributes)
  - ▶ Ограничения (constraints)
  - ▶ “-ilities”
    - ▶ Performance
    - ▶ Reliability
    - ▶ Availability
    - ▶ Modifiability
    - ▶ Usability
    - ▶ Testability
    - ▶ Survivability
    - ▶ Maintainability
    - ▶ Accessibility
    - ▶ Etc.



Как се измерва качеството на софтуерните  
системи?

## Мерни единици за наличност (availability)

---

- ▶ Наличността на системата се дефинира като вероятността тя да бъде в изправност, когато има нужда от нея
- ▶ Може да се представи като:

$$\alpha = \frac{\Delta t_f}{\Delta t_f + \Delta t_c},$$

- ▶ Където  $\Delta t_f$  е средното време между отказите, а  $\Delta t_c$  е средното време за отстраняване на повредата

# Интерпретация на наличността

---

- ▶ Наличността обикновено се изразява като процент от времето, през което системата е достъпна за предоставяне на услуги (напр. 99,95%).
- ▶ Това обаче не отчита два фактора:
  - ▶ Броят на потребителите, засегнати от прекъсването на услугата. (Напр. загуба на достъп нощно време може да не е от значение за редица системи, за разлика от загубата на услуга по време на периоди на пикова употреба).
  - ▶ Продължителността на прекъсването. (Напр. няколко кратки прекъсвания е по-малко вероятно да бъдат разрушителни от 1 дълго прекъсване.)



# Availability vs. Reliability

---

- ▶ Наличност и надеждност са различни понятия.
- ▶ Дадена система може да има висока степен на наличност и в същото време да е ненадеждна и обратно
  - ▶ Ако системата отказва редовно - за по 1 милисекунда на всеки час, то тя има наличност от над 99,9999 процента, но е много ненадеждна.
  - ▶ Система, която "никога" не отказва, но е недостъпна за две седмици всеки август (например за профилактика), има висока надеждност, но само ~96 процента наличност

# Мерни единици за производителност (Performance)

---

- ▶ Количествено, резултатите в сценариите за производителност могат да се характеризират чрез различни параметри, напр.:
  - ▶ Латентност (latency) – времето между пристигането на заявката и обработката ѝ;
  - ▶ Времева граница (Навременност) (deadline) – максимално време за наблюдаема реакция на системата;
  - ▶ Отклонение (jitter) – вариация в латентността;
  - ▶ Пропускливост (throughput) – броя заявки, които системата може да обработи за определен интервал от време;
  - ▶ Брой на необработените заявки (поради претовареност).

## Мерни единици за сигурност

---

- ▶ Време/усилия/ресурси, необходими за заобикаляне на сигурността с вероятност за успех;
- ▶ Вероятност за разкриване на атаката
- ▶ Вероятност за разкриване на самоличността на извършителите
- ▶ Процент на работоспособност (напр. при DoS)
- ▶ Обхват на пораженията

## Мерни единици за сигурност

---

- ▶ A list of popular software vulnerabilities and their measures, together with scoring of vulnerabilities are used, both represented by industry standards
- ▶ System security is given by:

$$Sec = \sum_n (P_n \times W_n),$$

where  $W_n$  is the weight of the n-th software weakness, which is defined by the vulnerabilities that activate it and  $P_n$  represents the risk of the corresponding weakness.

# Testability

---

- ▶ Някои от най-популярните метрики се основават на различни обектно ориентирани характеристики
  - ▶ Дълбочина на дървото на наследяване (Depth of Inheritance Tree – DIT) – размерът на най-дългия път в дървото на наследяване на класа
  - ▶ Number of children – броят класове, които наследяват даден клас
  - ▶ Number of methods – броят на методите, които предоставя даден клас
  - ▶ Брой полета

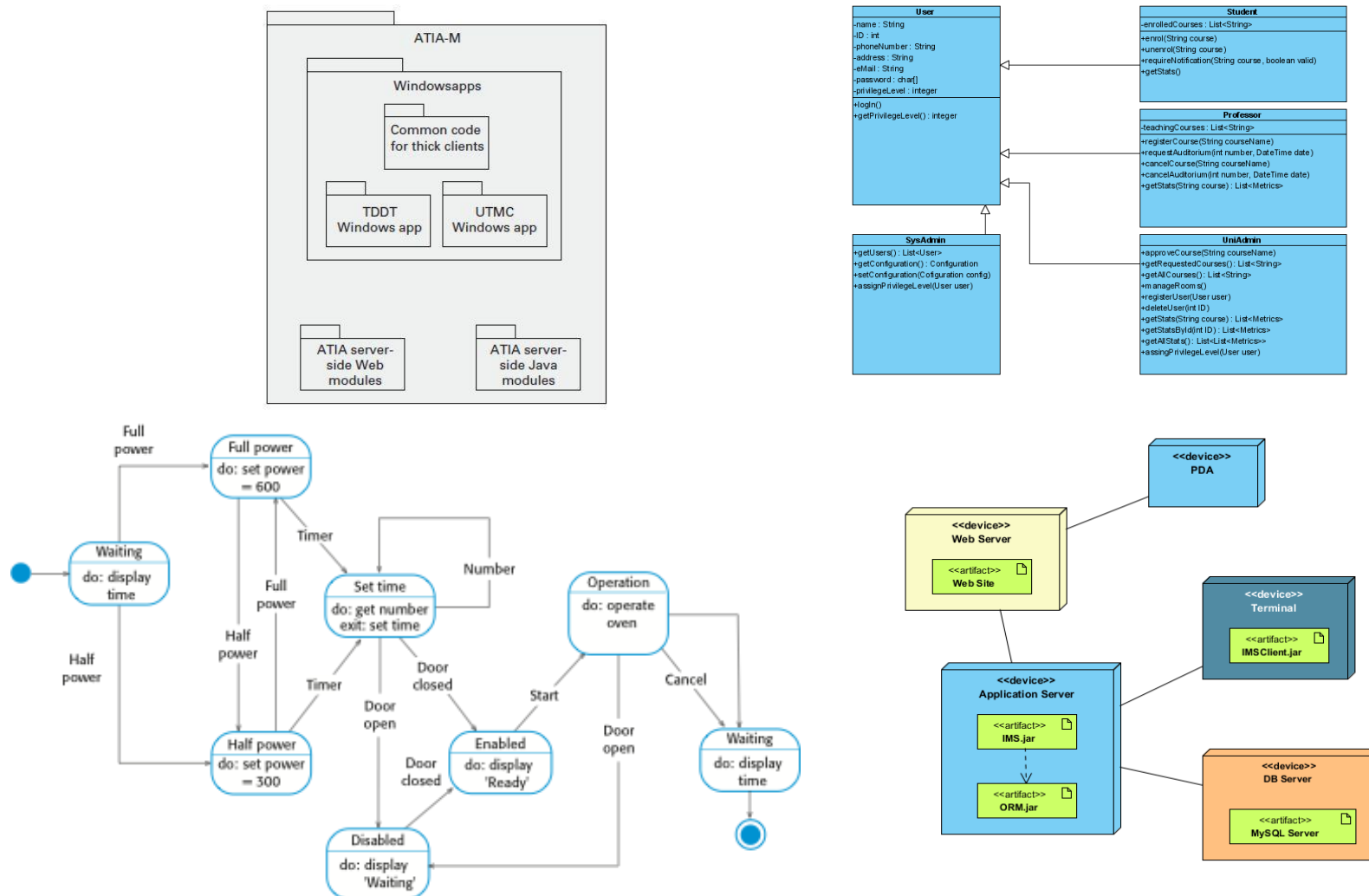
# Мерни единици за възможност за промяна (modifyability)

---

- ▶ Стойност
  - ▶ Например – брой променени елементи, усилие, пари
- ▶ Доколко промяната се отразява на останалата функционалност и свойства
- ▶ Различни метрики за комплексност
  - ▶ Cyclomatic complexity
  - ▶ Halstead complexity

# Формални модели в софтуерното инженерство

# Какво разбираме под модел на софтуерна система?





# Модели на софтуерните системи

---

- ▶ Достатъчен ли е UML?
- ▶ Може ли да се моделира семантиката на софтуера с UML?
- ▶ Може ли да се моделират нефункционални характеристики с UML?

# Формални методи

---

- ▶ Формалното моделиране представлява част от по-общ набор от технологии, познати като “формални методи”
- ▶ Базират се на математическо представяне и анализ на софтуера
- ▶ Формалните методи в софтуера включват
  - ▶ Формално моделиране (спецификация)
  - ▶ Анализ и доказателство на спецификацията
  - ▶ Разработка, базирана на трансформации (Transformational development)
  - ▶ Програмна верификация

# Необходимост от формални методи в софтуерното инженерство

---

- ▶ Формален анализ на софтуерни системи
- ▶ Динамични архитектури
- ▶ Генерация на код
  
- ▶ Всички те обаче са неприложими без съответните инструменти (tool support)

# Формални модели на софтуерните системи

---

- ▶ CSP (communicating sequential processes)
- ▶  $\pi$  – calculus
- ▶ Z schema
- ▶ Мрежи на Петри
- ▶ И т.н.

# Различни ADLs

---

- ▶ MetaH
- ▶ Rapide
- ▶ UniCon
- ▶ Darwin
- ▶ Wright
  - ▶ Базира се на CSP за формално описание
- ▶ xADL
- ▶ ACME
- ▶ AADL

# Примерен модел на архитектурата с EOA Wright

---

```
System Example
  Component CompA
    Port provide [provide protocol]
    Spec [ComponentA specification]
  Component CompB
    Port request [request protocol]
    Spec [ComponentB specification]
  Connector A B connector
    Role CompA [ComponentA protocol]
    Role CompB [ComponentB protocol]
    Glue [glue protocol]
  Instances
    a: CompA;
    b: CompB;
    ab: A B connector;
  Attachments
    a.provide as ab.CompA;
    b.request as ab.CompB;
end Example
```

# Клиент-сървър с Wright

---

```
System SimpleExample
  component Server =
    port provide [provide protocol]
    spec [Server specification]
  component Client =
    port request [request protocol]
    spec [Client specification]
  connector C-S-connector =
    role client [client protocol]
    role server [server protocol]
    glue [glue protocol]
Instances
  s: Server
  c: Client
  cs: C-S-connector
Attachments
  s.provide as cs.server;
  c.request as cs.client
end SimpleExample.
```

# Модел на connector с Wright

---

```
connector C-S-connector =  
  role Client = (request!x → result?y → Client) □ §  
  role Server = (invoke?x → return!y → Server) □ §  
  glue = (Client.request?x → Service.invoke!x → Service.return?y → Client.result!y → glue)  
  □ §
```

- ▶ С въпросителен знак се маркират входни данни
- ▶ С удивителен знак се маркират изходни данни



# Примерно описание на СМ – общо поле за данни

---

---

```
connector Shared Data1 =  
  role User1 = set→User1  $\sqcap$  get→User1  $\sqcap$  §  
  role User2 = set→User2  $\sqcap$  get→User2  $\sqcap$  §  
  glue = User1.set→glue  $\sqcap$  User2.set→glue  
         $\sqcap$  User1.get→glue  $\sqcap$  User2.get→glue  $\sqcap$  §
```

## Примерно описание на СМ – общо поле за данни (2)

В този модел съществува компонент, който инициализира полето

---

```
connector Shared Data2 =  
  role Initializer =  
    let A = set → A ⊓ get → A ⊓ §  
    in set → A  
  role User = set → User ⊓ get → User ⊓ §  
  glue = let Continue = Initializer.set → Continue  
          [ User.set → Continue  
            [ Initializer.get → Continue  
              [ User.get → Continue ] §  
            ]  
          in Initializer.set → Continue ] §
```

## Примерно описание на СМ – общо поле за данни (3)

---

Този модел е подобен на предишния, но не е задължително полето да се инициализира от компонента `Initializer`

```
connector Shared Data3 =  
  role Initializer =  
    let A = set → A  $\sqcap$  get → A  $\sqcap$  §  
    in set → A  
  role User = set → User  $\sqcap$  get → User  $\sqcap$  §  
  glue = let Continue = Initializer.set → Continue  
           $\sqcap$  User.set → Continue  
           $\sqcap$  Initializer.get → Continue  
           $\sqcap$  User.get → Continue  $\sqcap$  §  
  in Initializer.set → Continue  
     $\sqcap$  User.set → Continue  $\sqcap$  §
```

# Fault-tolerant pool от сървъри – описание с Wright

---

## Component Client

**Port** Primary =  $\S \sqcap ((\overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{Primary}) \sqcap \text{down} \rightarrow (\S \sqcap \text{up} \rightarrow \text{Primary}))$

**Port** Secondary =  $\S \sqcap \overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{Secondary}$

**Computation** = UsePrimary

**where** UsePrimary =  $\overline{\text{internalCompute}} \rightarrow (\text{TryPrimary} \sqcap \S)$

TryPrimary =  $\overline{\text{primary.request}} \rightarrow \text{primary.reply} \rightarrow \text{UsePrimary}$

$\sqcap \text{primary.down} \rightarrow \text{TrySecondary}$

UseSecondary =  $\overline{\text{internalCompute}} \rightarrow (\text{TrySecondary} \sqcap \S)$

TrySecondary =  $\overline{\text{secondary.request}} \rightarrow \text{secondary.reply} \rightarrow \text{UseSecondary}$

$\sqcap \text{primary.up} \rightarrow \text{TryPrimary}$

# Z

---

- ▶ Z-нотация (notation)
- ▶ Разработена в края на 80-те години
- ▶ Първоначално не е била предназначена за описание на софтуерни системи
- ▶ Комбинира формално и неформално описание и има възможност за графичен модел

# Пример

---

- ▶ Система, която документира имена на хора и дати
- ▶ Birthday book
- ▶ [NAME;DATE]

*BirthdayBook*

*known* :  $\mathbb{P}$  NAME

*birthday* : NAME  $\rightarrow$  DATE

*known* = dom *birthday*

# Birthday book

---

*AddBirthday*

$\Delta \text{BirthdayBook}$

$\text{name?} : \text{NAME}$

$\text{date?} : \text{DATE}$

$\text{name?} \notin \text{known}$

$\text{birthday}' = \text{birthday} \cup \{\text{name?} \mapsto \text{date?}\}$

*FindBirthday*

$\Xi \text{BirthdayBook}$

$\text{name?} : \text{NAME}$

$\text{date!} : \text{DATE}$

$\text{name?} \in \text{known}$

$\text{date!} = \text{birthday}(\text{name?})$

# Birthday book

*Remind*

$\exists \text{BirthdayBook}$

$\text{today?} : \text{DATE}$

$\text{cards!} : \mathbb{P} \text{NAME}$

$\text{cards!} = \{ n : \text{known} \mid \text{birthday}(n) = \text{today?} \}$

*AlreadyKnown*

$\exists \text{BirthdayBook}$

$\text{name?} : \text{NAME}$

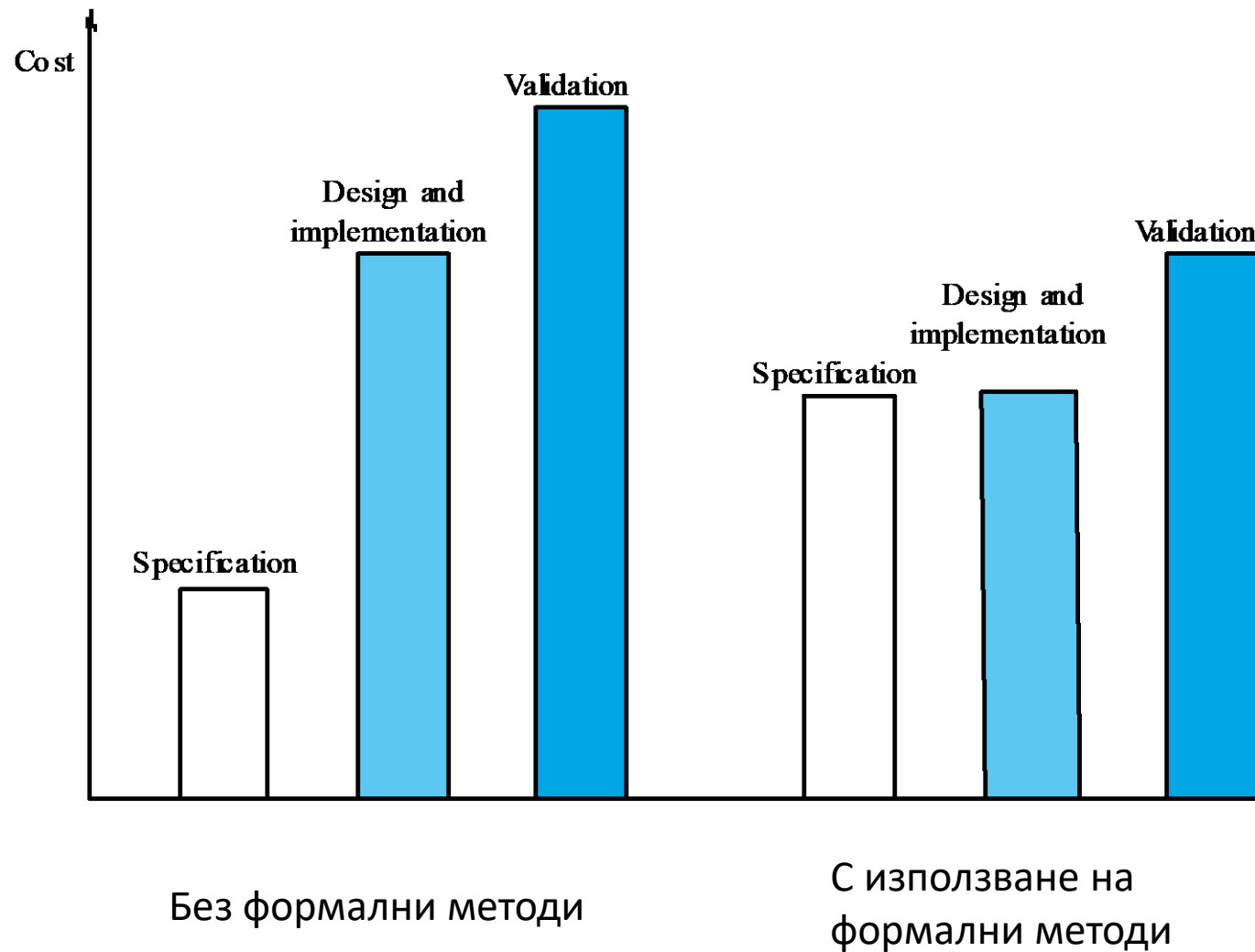
$\text{result!} : \text{REPORT}$

$\text{name?} \in \text{known}$

$\text{result!} = \text{already\_known}$



# Разходи за разработката



## Минуси на формалните методи

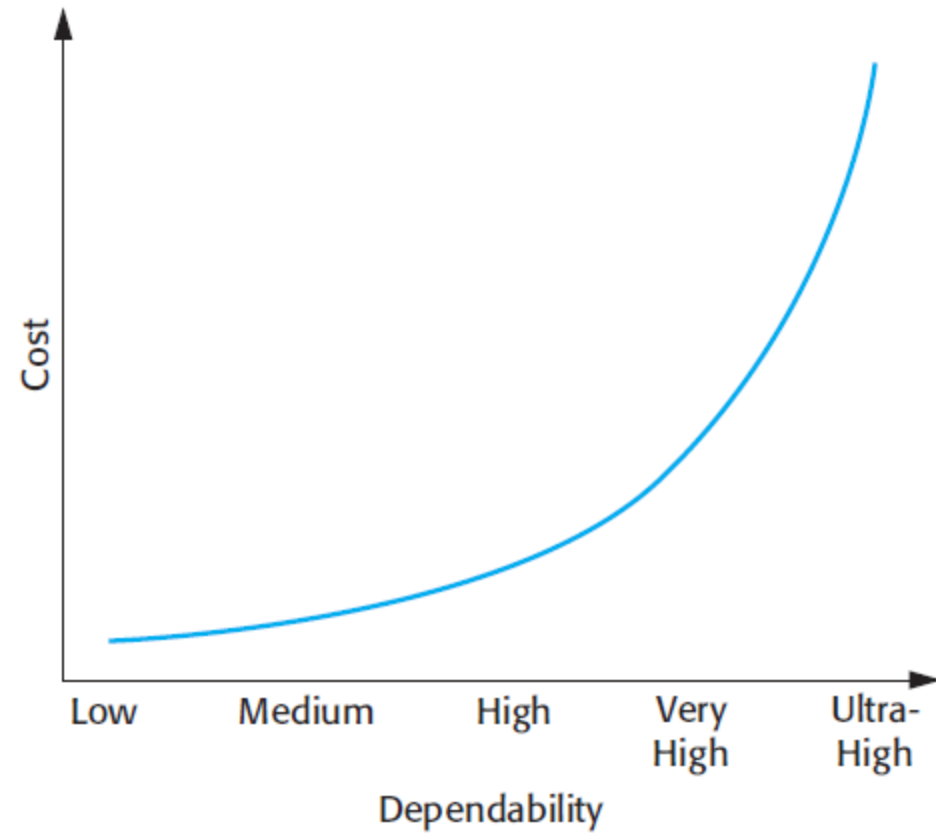
---

- ▶ Ефективността на неформалните технологии в СЕ в редица проблемни области, намалява необходимостта от формални методи
- ▶ Формалните методи не намаляват времето за разработка (time-to-market)
- ▶ Не са подходящи за описание на потребителски интерфейс и взаимодействие
- ▶ Проблеми с приложението в големи и развиващи се системи

# Формални модели на качествените изисквания

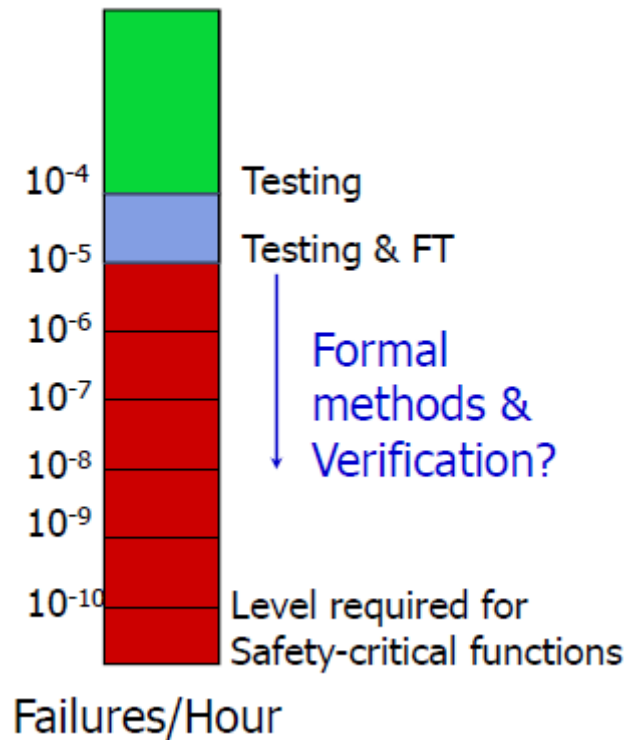
# Cost of dependability/reliability

---



# Attainable levels of SW reliability

- ▶ FAA (Federal Aviation Administration) & NASA safety=critical requirement is less than  $10^{-10}$  failures per 10 hrs of flight.



# The problem with testing

---

- ▶ Reliability of life-critical and real-time software is infeasible to be quantified by testing [Butler & Finelli 1993]
- ▶ Only small reliabilities (99,999%) are possible to be estimated in a obtainable period of time for testing
- ▶ For example assuring that a program has failure rate of about  $10^{-7}$  per hour may require thousands (and even more) years of testing
- ▶ There may not exist effective oracle to carry out statistical testing

# Софтуерна надеждност

---

- ▶ Леко се различава от традиционната математическа теория на надеждността
  - ▶ Отказите са детерминистични събития, поведението на потребителя не е
- ▶ Дефинира се като вероятност за отказ в рамките на даден времеви интервал
- ▶ На практика се използва като индикатор за това колко тестване на системата е достатъчно



# Как измерваме надеждността на софтуер

---

- ▶ Има различни явни метрики
  - ▶ Probability of failure (success)
  - ▶ Mean time to failure  $\mu$
  - ▶ Failure rate  $\lambda$ 
    - ▶  $\lambda = 1/\mu$
- ▶ Неявни метрики
  - ▶ Брой редове код
  - ▶ Test coverage
  - ▶ Други...



- 
- ▶ Измерването на надеждността на софтуер е трудно, поради сложната природа на софтуерните продукти, която все още остава неразбрана (недефинирана)
  - ▶ Дори метрики, които би трябвало да са очевидни, като размер на софтуера, все още нямат общоприета дефиниция
  - ▶ Отказите на софтуера, за разлика от тези на хардуера са детерминистично явление, поведението на потребителя не е детерминистично
  - ▶ Трудно е да се оцени надеждността на компонентно-базирана система, поради сложността на възможните взаимодействия между съставлящите я компоненти

# Методи за оценка на надеждността

---

- ▶ **Модели от тип черна кутия**
  - ▶ Основани са на статистическа обработка на данни, най-често от тестване на системите
  - ▶ Software Reliability Growth Models (SRGMs)
- ▶ **Модели от тип бяла кутия**
  - ▶ Базират се на данни за вътрешната организация на системата, напр. - архитектурата
  - ▶ Формализират процеса на отказите на компонентите в системата
  - ▶ Интегрира се поведението на отказите със модел на архитектуата



# Методи за оценка на надеждността

---

- ▶ Методите на бялата кутия не изключват тези на черната
- ▶ Съществува значителен проблем с гарантирането на много висока надеждност  $\sim (1-10^{-9})$  само чрез тестване



## Изчисляване чрез данни от тестването

---

- ▶ Доказано е, че е неподходящо надеждност на критични системи да се изчислява само чрез тестване [Butler & Finelli 1993]
- ▶ Само малки надеждности (99,999%) може да се изчислят на базата на данни, придобити от разумно дълъг период на тестване
- ▶ Например, за да сме сигурни че дадена система има надеждност от порядъка на  $(1-10^{-7})$ , трябва да тестваме хиляди години без прекъсване

# Модели на черната кутия

---

- ▶ Разработени са десетки такива модели
- ▶ Те се основават на данни от тестването на софтуер, като време между отказите, брой откази и т.н.
- ▶ Правят се редица опростяващи допускания, които влошават качеството на оценката за надеждност

# Модели на черната кутия

---

- ▶ Допуска се че отказите се подчиняват на някакво статистическо разпределение
  - ▶ Експоненциално – модел на Jelinski-Moranda
  - ▶ Поасново (Nonhomogeneous Poisson process – NHPP) – модели на Musa

# Базов модел за надеждност на Муса

---

- ▶ Честотата на отказите на системата е:

$$\lambda(\mu) = \lambda_0 \left[ 1 - \frac{\mu}{\nu_0} \right]$$

- ▶ където:

- ▶  $\lambda_0$  е началната честота на отказите в началото на тестването
- ▶  $\mu$  е средния (очакван) брой откази на системата в даден момент от време  $\tau$ :

$$\mu(\tau) = \nu_0 \left[ 1 - \exp \left[ - \frac{\lambda_0}{\nu_0} \tau \right] \right]$$

- ▶  $\nu_0$  е общия брой бъгове в системата

## Пример

Нека допуснем, че в даден софтуер има общо 100 бъга и началната честота на откази е 10 отказа/час. В настоящия момент на тестване са регистрирани 50 отказа.

Тогава честотата на отказите е:

$$\lambda(\mu) = \lambda_0 \left[ 1 - \frac{\mu}{\nu_0} \right] = 10 \left[ 1 - \frac{50}{100} \right] = 5 \text{ отказа/час}$$

Броят откази след 10 часа е:

$$\mu(\tau) = 100 \left( 1 - \exp \left[ -\frac{10}{100} (10) \right] \right) = 100 [1 - \exp(-1)] = 63$$

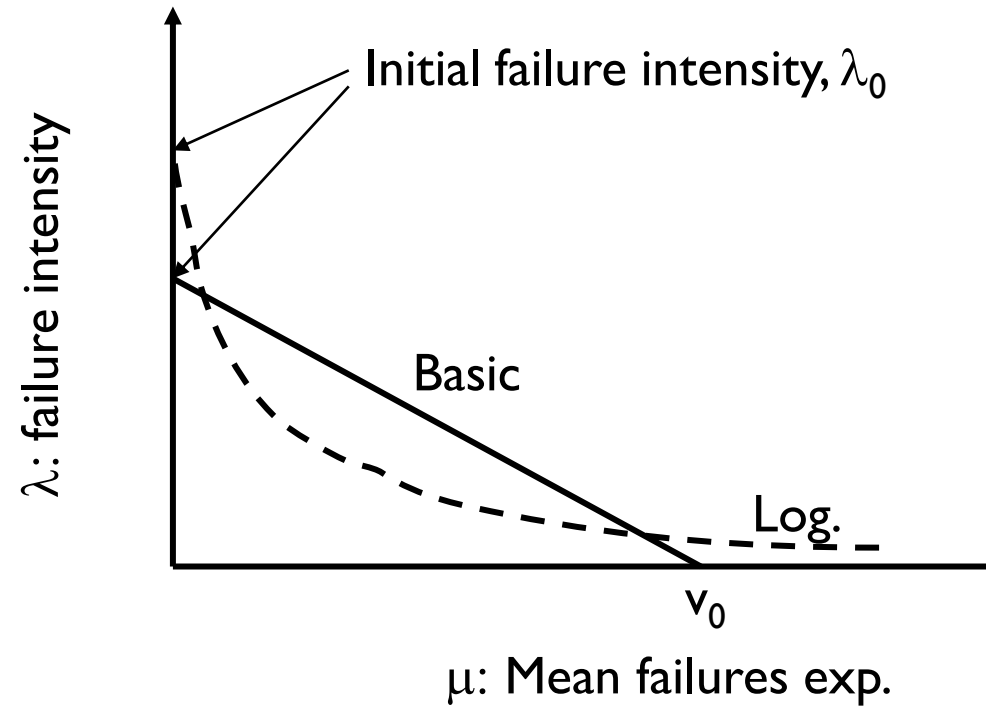
Съответно след 100 часа:

$$\mu(\tau) = 100 \left( 1 - \exp \left[ -\frac{10}{100} (100) \right] \right) = 100 [1 - \exp(-10)] = 100$$

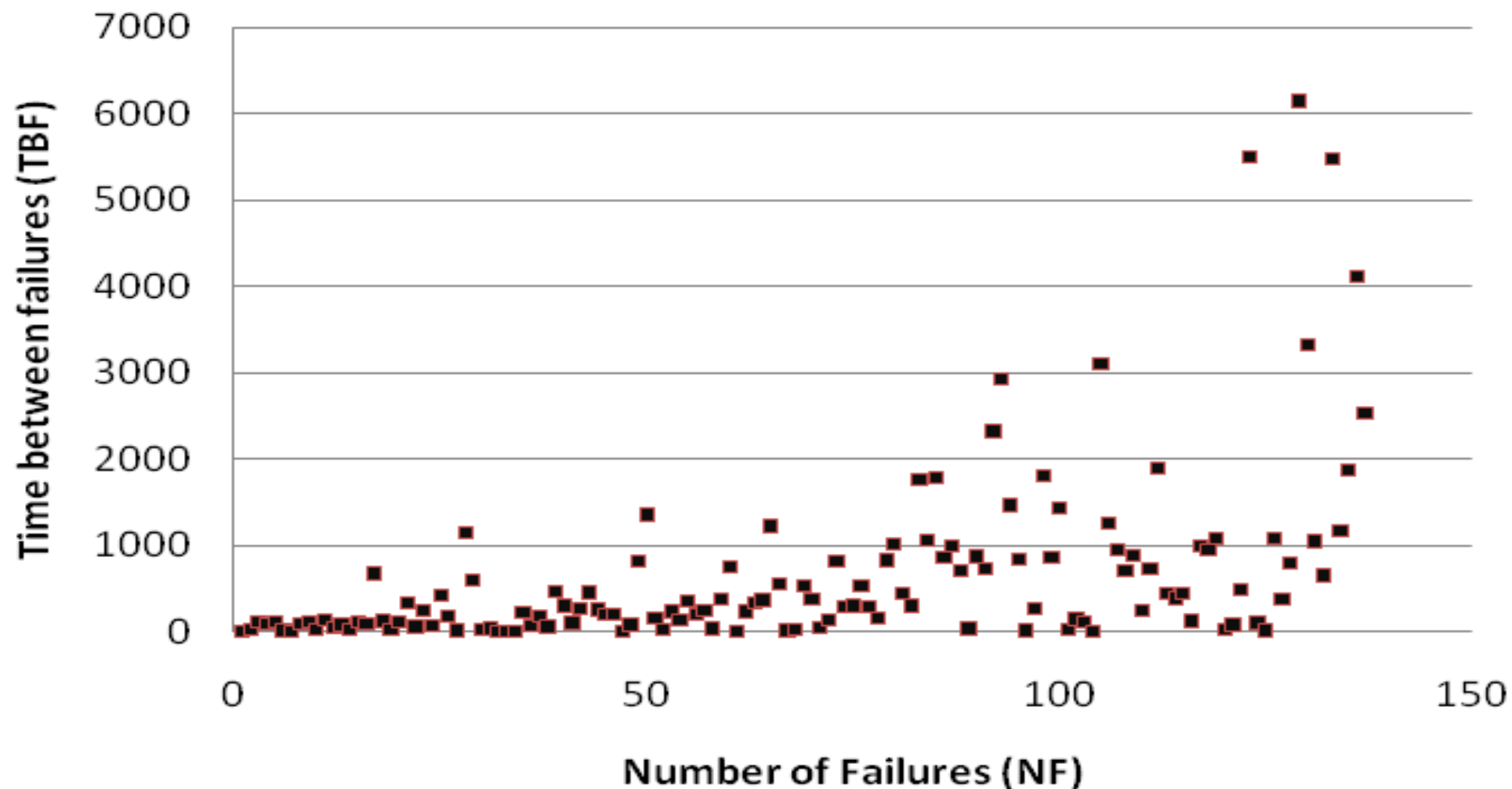


# Failure Intensity Reduction Concept

$\lambda$ : Failure intensity  
 $\lambda_0$ : Initial failure intensity  
at start of execution  
 $\mu$ : Average total number of  
failures at a given point  
in time  
 $v_0$ : Total number of failures  
over infinite time

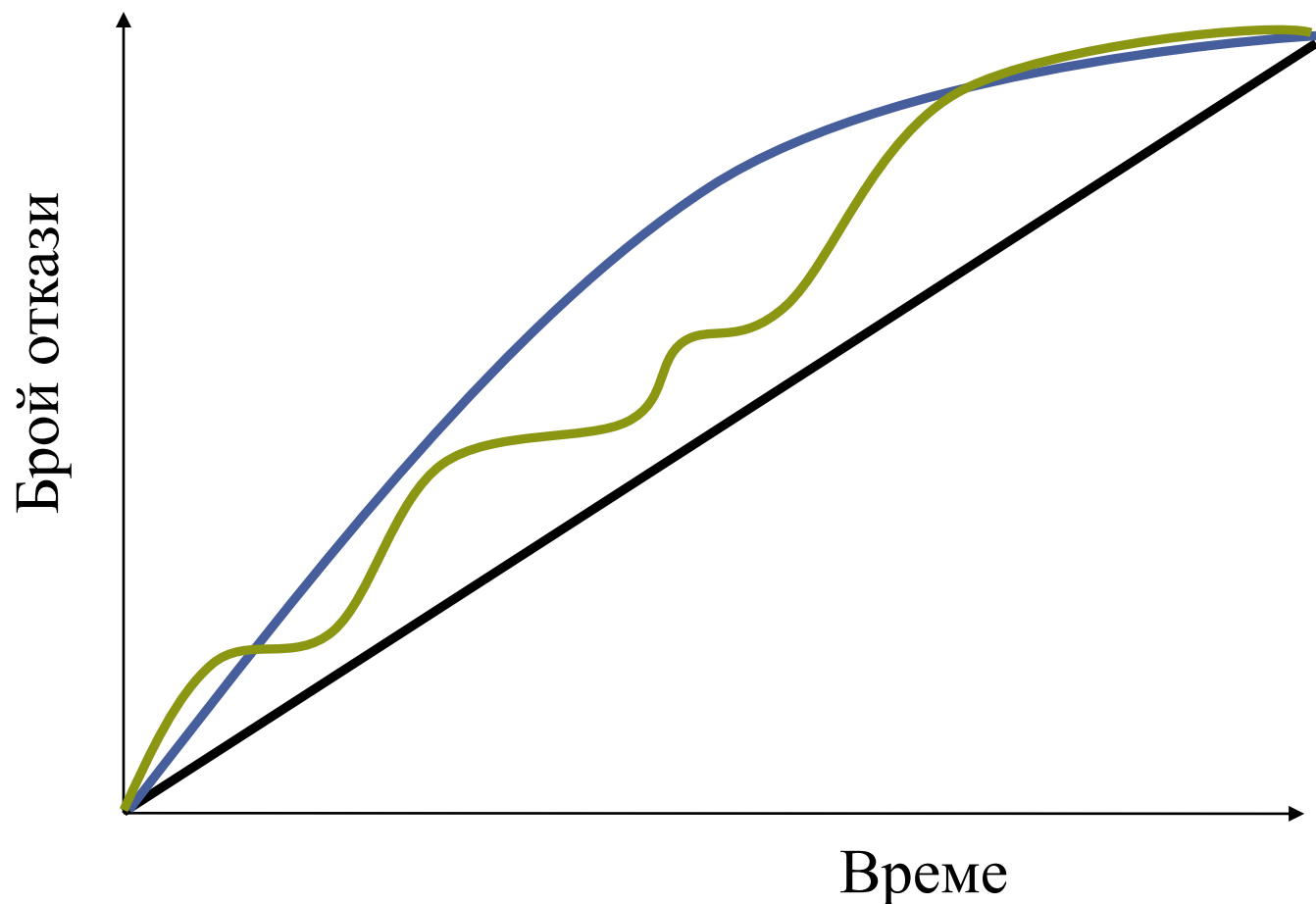


# Примерни данни за оценка на надеждност



## Характерни модели на поведението на грешките в системата

---



# Модели от тип „бяла кутия“

---

- ▶ Модели на състоянието (state-based models)
- ▶ Модели на пътя на изпълнение (path-based models)

[Goseva-Popstojanova & Trivedi, 2001]

# Модели на състоянието

---

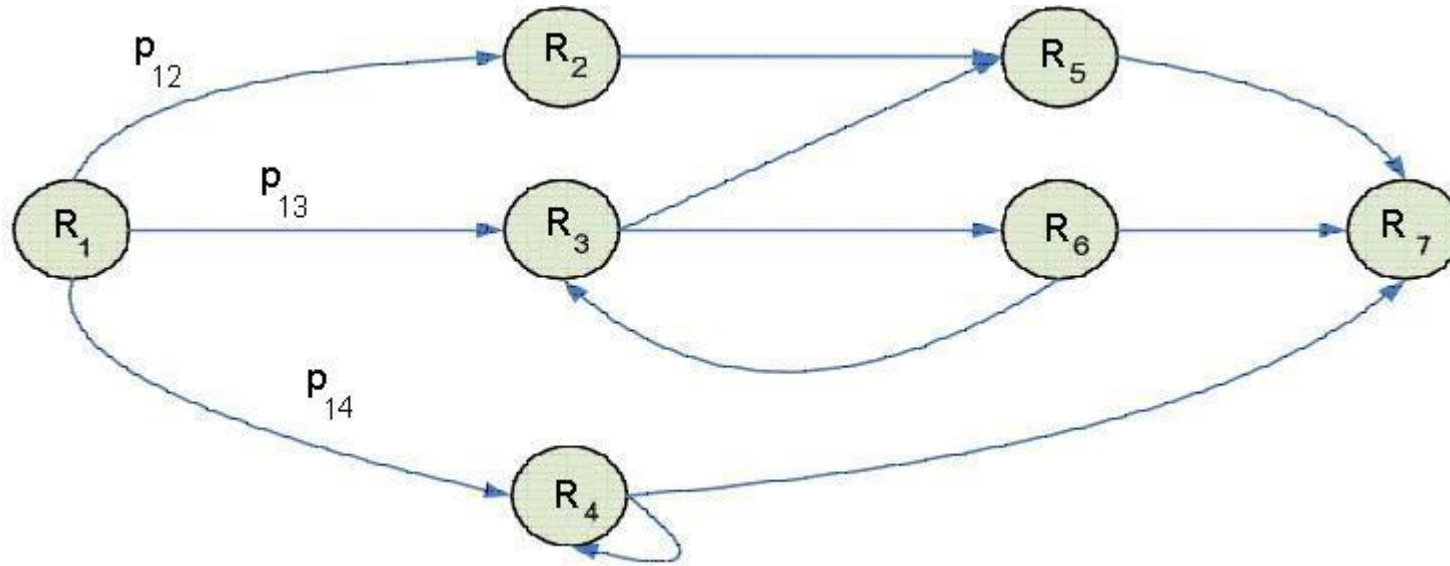
- ▶ Основни стъпки при моделирането
  - ▶ Идентификация на модулите в системата
  - ▶ Построяване на модел на архитектурата на системата
    - ▶ Характерно тук е че трябва да се определи т.нар. профил на употреба – т.е. кои компоненти в системата с каква вероятност/честота се използват
  - ▶ Определяне на поведението на отказите на всеки модул – намиране на конкретна стойност за надеждността
  - ▶ Комбиниране на модела на архитектурата с поведението на отказите

## Някои общи за всички модели допускания

---

- ▶ Данните за надеждността на компонентите са предварително известни
- ▶ Профилът на употреба е предварително известен
- ▶ Отказите на компонентите са независими събития
- ▶ Преходите (предаването на изпълнението) от един компонент към друг са независими събития
- ▶ Вероятността за отказ на даден компонент е константа и не се променя във времето

# Модели на състоянието



- ▶ Използва се верига на Марков
  - ▶ Състоянията моделират компонентите
  - ▶  $R_i$  – Надеждност на компонента  $i$
  - ▶  $P_{ij}$  – вероятност за преход от компонент  $i$  към компонент  $j$  по време на изпълненето на системата (това е профила на употреба)

# Модели на състоянието

---

- ▶ Модел на Рьоснер/Ченг [Cheung 1978, Reussner et al 2003]

- ▶ Съставя се матрица  $S$  със следните елементи

$$s_{ij} = p_{ij}R_i$$

- ▶ Добавят се ред и колона в  $S$ , които представят начално и крайно състояние – съответно с индекси  $I$  и  $J$
- ▶ Може да се докаже, че надеждността на системата е:



$$r_{system} = (I - S)^{-1}_{(I,J)}$$



# Модели на пътя на изпълнение

---

- ▶ Отново се използва верига на Марков за модел на архитектурата
- ▶ Изпълняват се  $N$  теста на системата, върху различни точно определени пътища на изпълнение в модела на системата
- ▶ Надеждността на всеки път на изпълнение е:

$$R_t = \prod_{\forall m \in M(t)} R_m$$

- ▶ Окончателно надеждността на системата е:

$$R_c = \frac{\sum_{\forall t \in T} R_t}{|T|}$$

- ▶ където  $T$  е множеството на тестовете на системата

## Модели на пътя на изпълнение

---

- ▶ Надеждността на системата намалява значително при наличие на циклично изпълнение на даден компонент
- ▶ Надеждността на системата е по-чувствителна от профила на употреба, отколкото от архитектурата на системата

## Допълнителни материали

---

- ▶ Goseva-Popstojanova, K., A. P. Mathur and K. S. Trivedi, Comparison of Architecture-Based Software Reliability Models, 12<sup>th</sup> IEEE International Symposium on Software Reliability Engineering (ISSRE 2001), Hong Kong, Nov. 2001, pp.22-31.
- ▶ Dimov A. (2013). Measurement of Software Quality Characteristics. Computer Science and Technologies. Vol. 1, 2013. 199-204.