Extrémne programovanie a iné agilné metodológie

Zimný semester 2007/08

Ing. František Gyárfáš, PhD.

Katedra aplikovanej informatiky

gyarfas@ii.fmph.uniba.sk

http://www.ii.fmph.uniba.sk/~gyarfas/

Čo je program?

cvicenie1_1.cpp

cvicenie1_1.bpf | cvicenie1_1.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, world";
}
```

8: 1 Modified Insert

cvicenie1_1.cpp

cvicenie1_1.bpf | cvicenie1_1.cpp

```
private double totalSalary() {
 double result = 0;
 for (int i = 0; i < people.length; i++){
    result += people[i].salary;
 }
 return result;
}
```

3: 41 Modified Overwrite

C:\Prednasky\2006-2007\Cplusplus200...

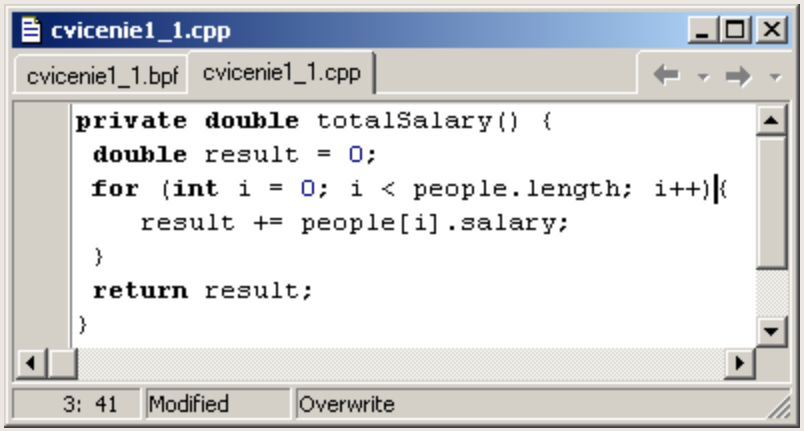
cvicenie8.bpf | cvicenie8.cpp | cvicenie8.h

```
class Double{
protected:
    double d;
    char *ret;
public:
    Double(double d0 = 0.0);
    Double(char *v);
    char *cislo(){ return ret;}
    double vrat(){return d;};
    virtual ~Double();
};
```

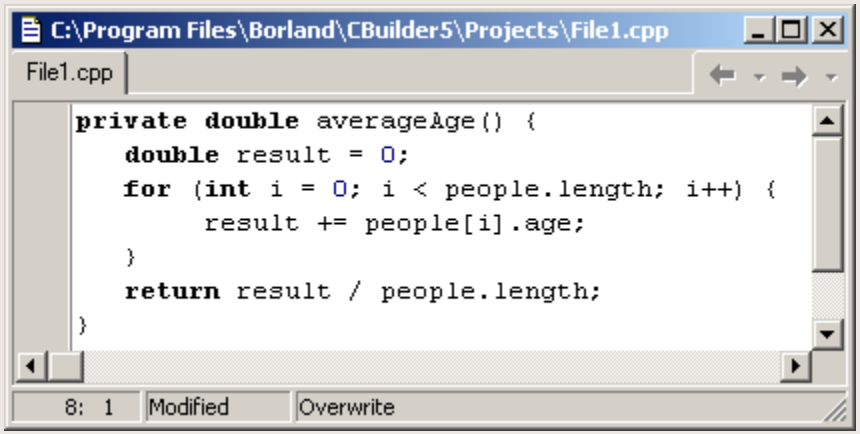
1: 1 Overwrite

Čo je programovanie?

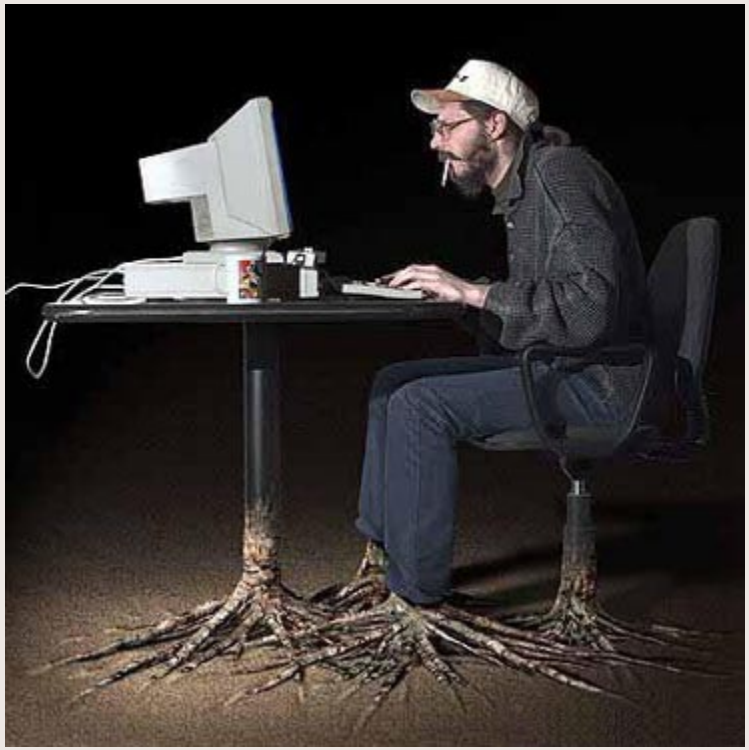
Písanie programov:



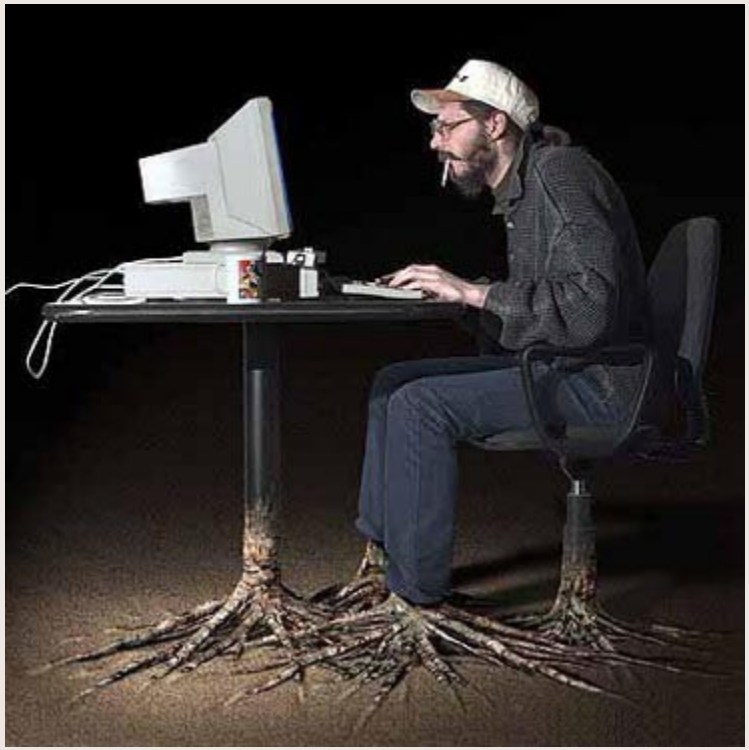
Zisťovanie, prečo program nerobí, čo má, nefunguje, správa sa čudne, padá a snaha o nápravu.



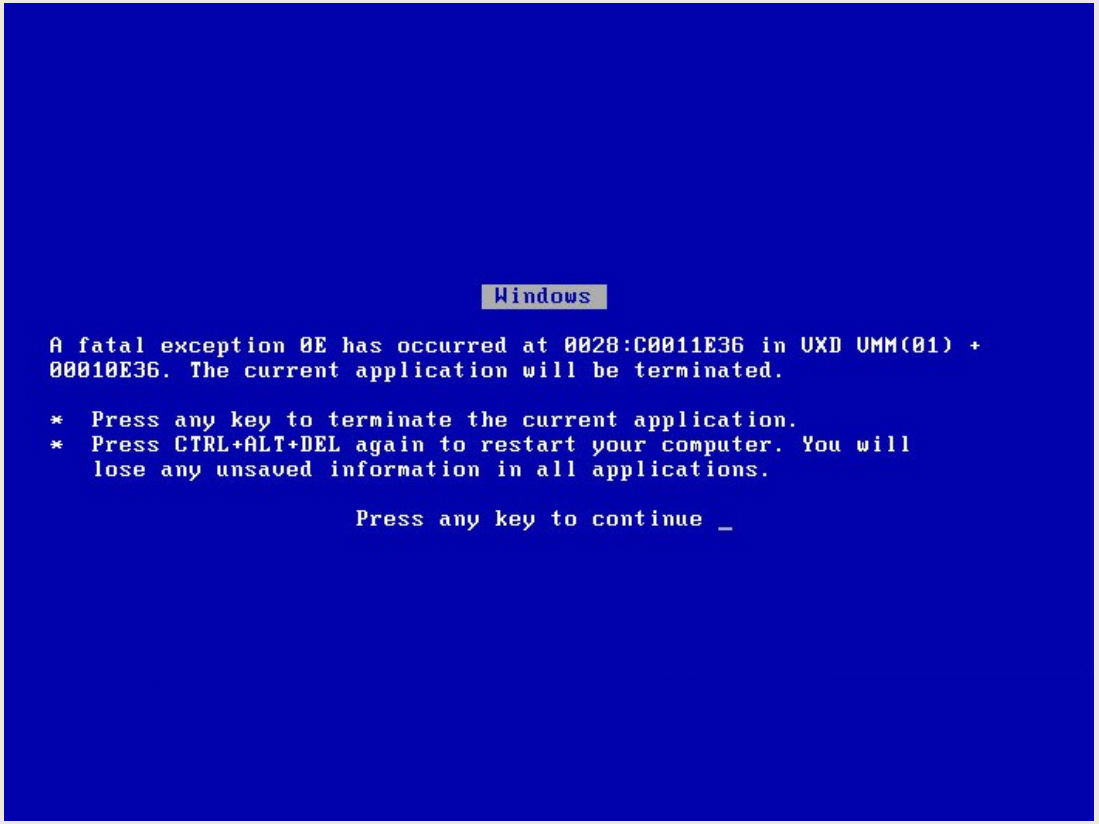
Kto je programátor?



Čo robí programátor?



Kedy program nefunguje?



Kedy program funguje?

?

Koľko ľudí treba na vytvorenie programu?

Ideálny počet: **1**

Realistický počet: **2 - 10**

Veľkorysý počet: **desiatky - tisíce**

Ak jeden programátor urobí množstvo práce M, koľko práce urobia dvaja programátori?

Ideálny stav:	**2 x M**	
Optimistický odhad:	**spočiatku 0.8 M**	**neskôr 1.4 M**
Skeptický odhad:	**spočiatku 0.8 M**	**neskôr 0.6 M**

Ako vyzerá program?

Ak ho vyrába jeden programátor:

Ako vyzerá program (2-10)?

Ak ho vyrábajú 2-10 programátori:

Ako vyzerá program (100>)?

Ak ho vyrábajú stovky programátorov:

Písanie programov nie je stavanie domov

- **Nejasná a neúplná špecifikácia zadania**
- **Postupné dopĺňanie zadania**
- **Inkrementálny a iteratívny vývoj programu**
- **Neúplnosť riešenie**
- **Chybovosť programu**
- **Dopĺňanie riešenia o ďalšie moduly, zmeny koncepcie, výzoru, dátových štruktúr**
- **Tvorba nových verzií**

Softvérové inžinierstvo

Softvérové inžinierstvo – zaoberá sa špecifikovaním, návrhom, vývojom a údržbou softvéru s využitím poznatkov informatiky, projektového manažmentu a ďalších oblastí.

Softvérové inžinierstvo – „*Je to zavedenie a používanie riadnych inžinierskych prístupov tak, aby sme dosiahli ekonomickú tvorbu softvéru, ktorý je spoľahlivý a pracuje účinne na dostupných výpočtových prostriedkoch*.“

Fritz Bauer (Konferencia NATO, 1968)

Ciele softvérového inžinierstva

Dva základné ciele:

- **zlepšovanie kvality** softvéru
- **zvyšovanie produktivity** softvérových inžinierov

Vonkajšia kvalita softvéru

Vonkajšia kvalita (z hľadiska používateľa):

- **Funkcionalita**: schopnosti, vlastnosti, bezpečnosť
- **Použiteľnosť**: používateľská prítulnosť, zrozumiteľnosť, pohodlnosť, estetika, jednotný štýl,
- **Spoľahlivosť**: frekvencia chýb, zotaviteľnosť po chybe, stredná doba výpadku
- **Výkon**: rýchlosť, efektívnosť, spotreba zdrojov, doba odozvy
- **Podpora**: rozšíriteľnosť, prispôsobiteľnosť, obtiažnosť údržby, prenositeľnosť, konfigurovateľnosť

Vnútorná kvalita softvéru

Vnútorná kvalita (z hľadiska vývoja programu):

- **Štruktúrovanosť -** počet tried, počet metód, globálnych premenných, hĺbka hierarchií, počet priateľských tried, modulárnosť, integrácia, atď.
- **Rozsah** kódu, počet riadkov, priemerný počet riadkov na funkciu, atď.
- **Kvalita** kódu, čitateľnosť, elegantnosť, spoľahlivosť, bezchybnosť, dokumentácia.

Ponaučenie z kvality

Kvalita softvéru nie je absolútna kategória. Má zmysel ju merať iba vo vzťahu k očakávaniam, ktoré na softvér kladie jeho používateľ.

Zákon skromnej spokojnosti

Aj zdanlivo zle napísaný, nie príliš funkčný, kazivý, pomalý a neelegantný program môže byť pozitívne hodnotený, ak zodpovedá potrebám a očakávaniam jeho používateľa.

Zákon skeptickej múdrosti

Aj geniálny, supermoderný, elegantný, absolútne bezchybný program môže byť odmietnutý, ak nerobí to, čo od neho používateľ očakáva.

Fitness for purpose – vhodnosť pre zamýšľaný účel.

Kto sa zúčastňuje tvorby softvéru?

- **Zákazník / Používateľ**

- **Analytik** – komunikuje so zadávateľom, formuluje jeho požiadavky, definuje jednotlivé funkčné bloky systému a ich informačné prepojenia

- **Návrhár** – vytvára špecifikáciu softvéru, architektúru, vyberá z viacerých alternatív, zvažuje budúcnosť systému, jeho udržovateľnosť a údržbu, ďalší vývoj a verzie.
- **Programátor** – implementuje navrhnuté riešenie, testuje jeho funkčnosť
- **Tester** – systematicky testuje správnosť programu a jeho funkcií, integráciu a doplnky
- **Údržbár** – udržuje program v chode, hľadá a odstraňuje chyby
- **Školitelia, konzultanti, operátori, technici, systémová podpora, atď.**

Kľúčové úvahy pri tvorbe softvéru

- **Nájsť spoločnú reč so zadávateľom.** Porozumieť jeho očakávaniam a vysvetliť mu spôsob riešenia. Bez vzájomného porozumenia a komunikácie je vytvorenie úspešného softvéru temer nemožné.
- **Vytvorenie vhodného týmu.** Primeraný počet, vhodná kompozícia kvalifikácií aj osobnostných typov, fungujúce rozdelenie rolí, nahraditeľnosť.
- **Vyber vhodného programového prostredia, operačného systému a vývojových nástrojov.** Prihliadanie na možnosti a predstavy zákazníka.
- **Vyvíjať či kúpiť?**
- **Ekonomické hľadiská –** termíny a cena.
- **Udržovateľnosť a údržba**.

Pasce pri špecifikácii požiadaviek

- **Nájsť spoločný jazyk**
- **Porozumieť, čo zadávateľ skutočne chce, aj keď to sám presne nevie**
- **Častá zmena požiadaviek**
- **Neúplnosť zadania**
- **Nedostatok znalostí z oblasti aplikácie**

Pasce pri tvorbe softvéru

- **Špecifikácia požiadaviek –** používateľ nevie, čo chce, ale bude vedieť, čo nechcel
- **Produktivita programátorskej práce**
- **Nedostatočné testovanie**
- **Spolupráca programátorov v tíme**
- **Integrácia**
- **Chybovosť**
- **Dokumentácia a jej špecifiká (množstvo, úplnosť, zastaralosť, udržovanie, konzistentnosť, štýl)**
- **Rôzne verzie**
- **Udržovanie, poradenstvo**

Softvérová kríza

Koncom šesťdesiatych rokov sa objavil vážny problém, ktorý dostal názov **softvérová kríza.** Išlo o stroskotanie vývoja veľkého množstva serióznych a drahých programových systémov.

Na ilustráciu sa zvykne uvádzať táto dobová štatistika - zo zakázok americkej vlády za mnoho miliónov dolárov:

- 40% zakázok bolo dodaných, ale nikdy úspešne nepoužitých
- 25% zakázok bolo zaplatených, ale nikdy nedodaných
- 15% zakázok bolo po drastických úpravách zahodených

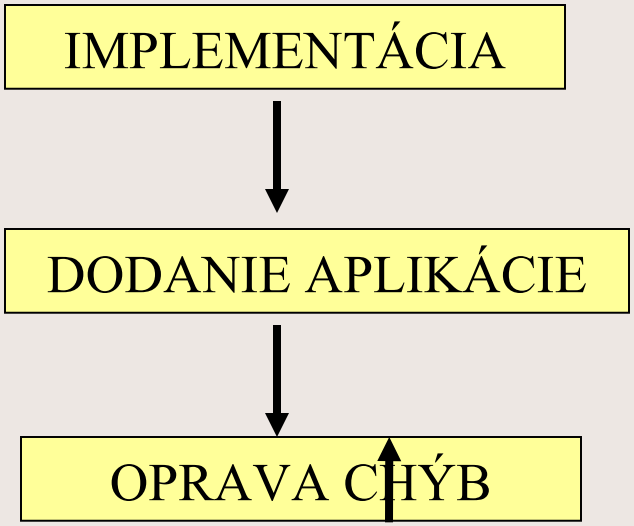
Stroskotania softvérových projektov

Najčastejšie príčiny zastavenia softvérových projektov podľa analýzy viac než 8000 aplikácií (Pfleeger, 1998)

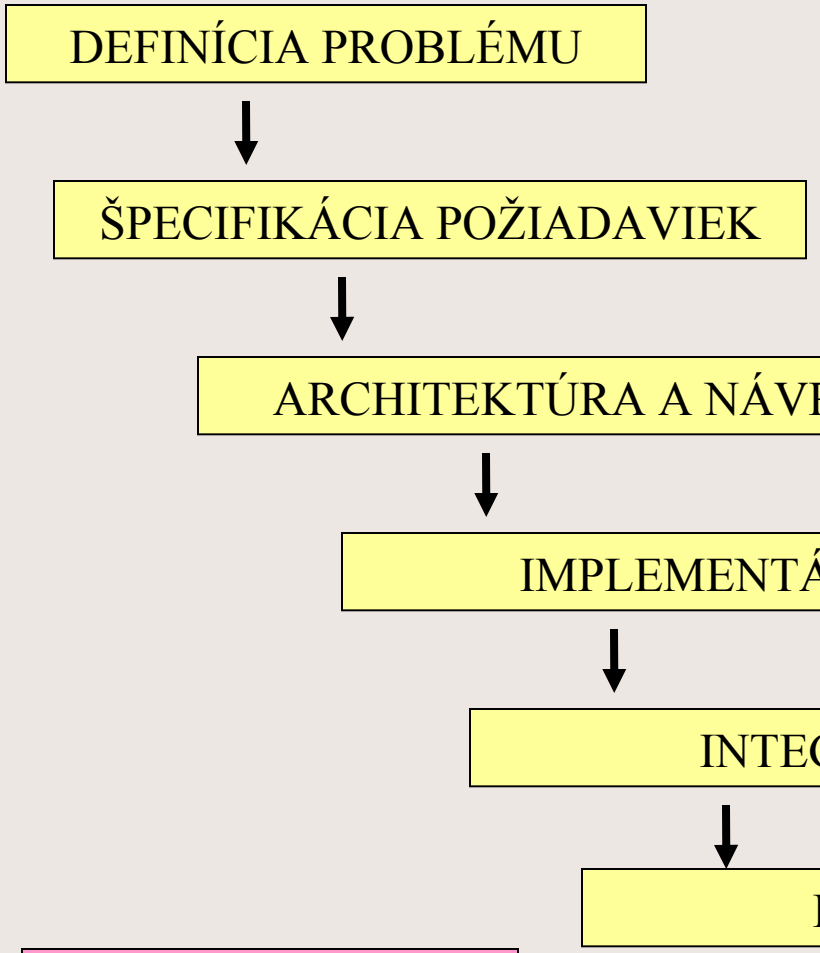
- **Neúplnosť alebo nejasnosť požiadaviek – 13.1%**
- **Nedostatok záujmu a podpory zo strany používateľa – 12.4 %**
- **Nedostatok zdrojov, podhodnotený rozpočet alebo nedostatočné termíny – 10.6 %**
- **Nerealistické očakávania – 9.9 %**
- **Nedostatočná podpora zo strany manažmentu dodávateľa alebo odberateľa – 9.3 %**
- **Zmena požiadaviek a špecifikácií – 8.7 %**
- **Nedostatočné plánovanie – 8.1 %**
- **Strata potreby softvéru – 7.5 %**

Code and Fix Model

Na počiatku programovania v 50tych rokoch dvadsiateho storočia sa ujala metodológia **Naprogramuj a opravuj:**



Stagewise Model

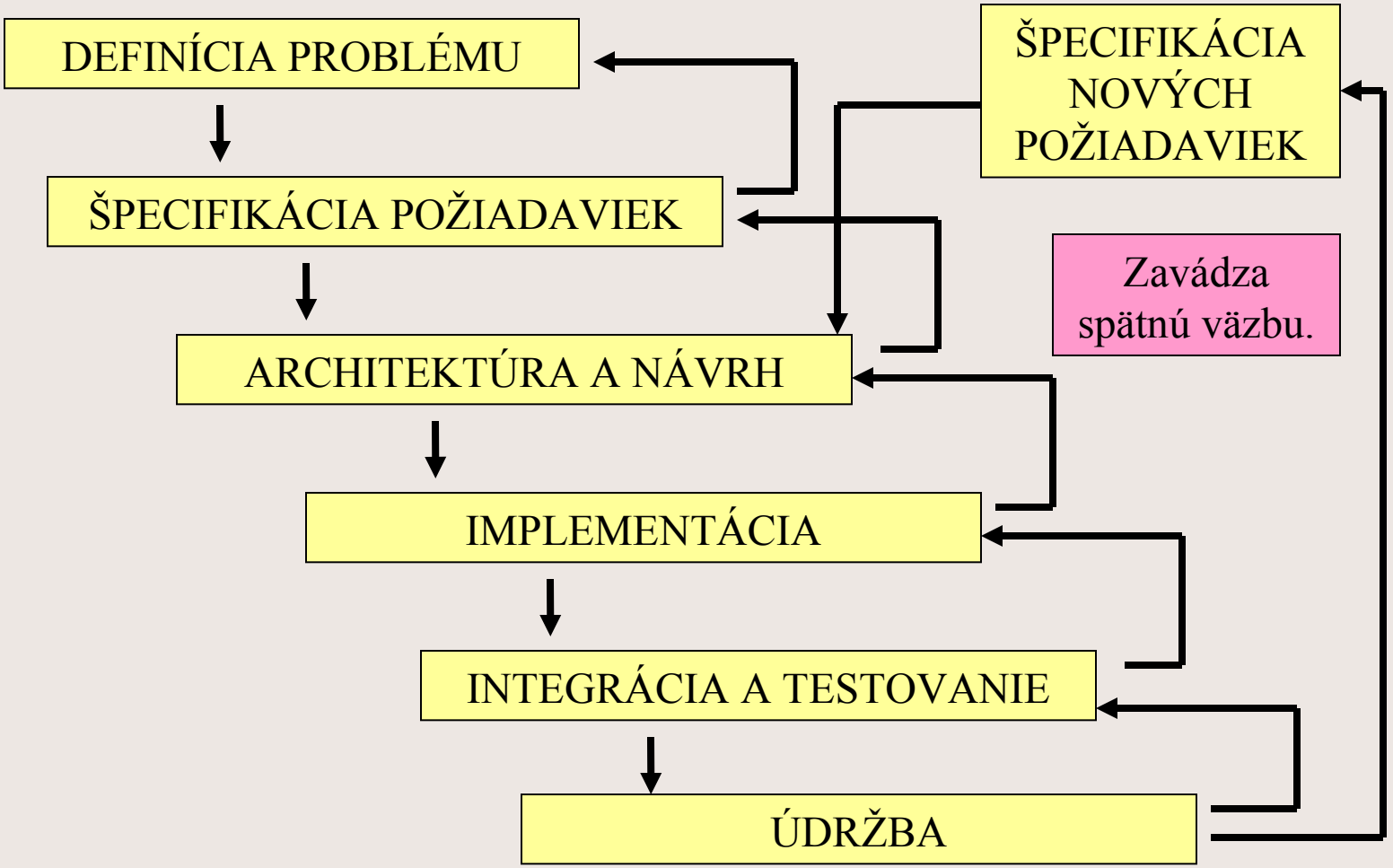


V roku 1957 bol definovaný **stagewise model,** založený na striktnej postupnosti jednotlivých fáz vývoja.

Fatálna je absolútna absencia spätnej väzby.

Waterfall Model

V roku 1970 bol definovaný **vodopádový model**.



Výhody vodopádového modelu

- **Je jednoduchý, priamočiary a vedúci k cieľu.**
- **Je skvelý pre riadenie projektov. Umožňuje sledovanie vývoja systému.**
- **Vnáša do procesu vývoja softvéru disciplínu.**
- **Každá fáza končí úplnou dokumentáciou.**

Nevýhody vodopádového modelu

- **Je jednoduchý, priamočiary a vedúci k cieľu.**

- **Je nepružný.** Do vývoja sa dá ťažko vstupovať. Poradie fáz sa musí dodržovať. Ak chce zákazník zmeniť niečo v zadaní, musí sa vrátiť až na začiatok a znovu prejsť všetkými fázami.

- **Dodanie zákazníkovi prebieha formou veľkého tresku.**

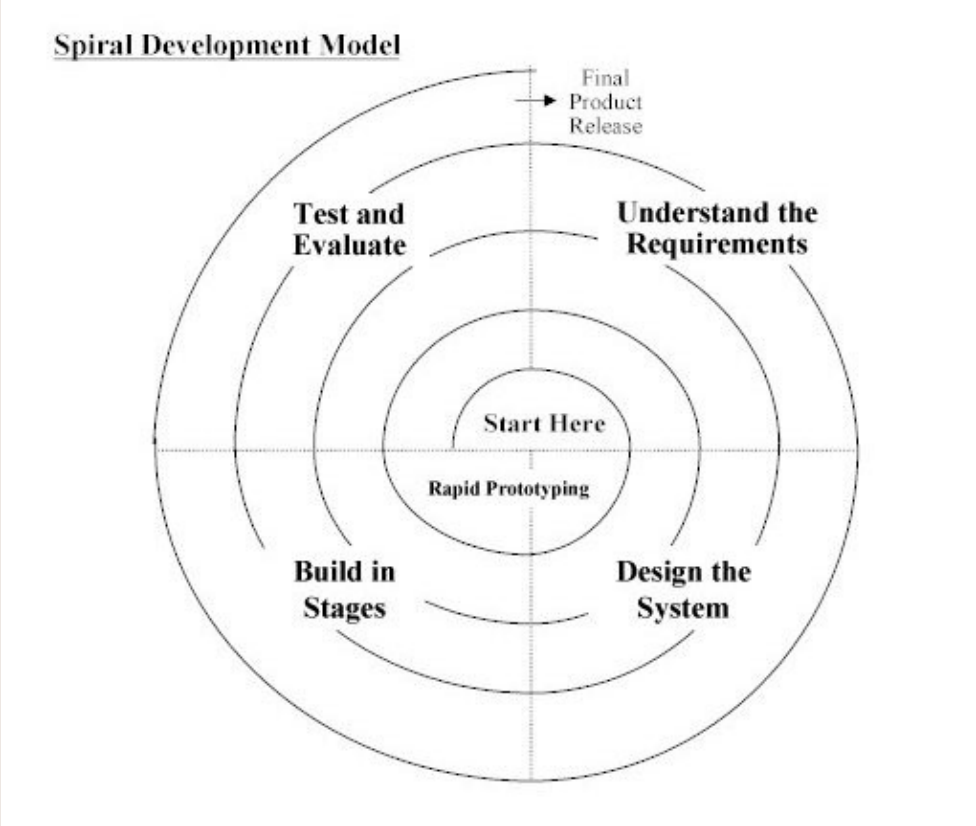
  - Medzi špecifikáciou a dodaním sa z hľadiska zákazníka nič nedeje.
  - Ak sa zákazníkovi medzitým zmenia požiadavky, nakoniec zistí, že dostal, čo nechcel.
  - Zákazník nie je nijako spätý s vývojom. Nemôže priamo ovplyvňovať ani funkcionalitu ani rozhranie.
  - Zákazník necíti žiadnu solidaritu s vývojármi. Nepozná ich, sú pre neho cudzí, má k nim skôr rezervovaný či nepriateľský vzťah.
  - Ak sa projekt z nejakého dôvodu nedokončí, nemá z neho zákazník vôbec nič.
  - Pre zákazníka je obtiažne nasadiť celý produkt naraz.

Špirálový model

V roku 1985 ho navrhol a predstavil Barry Boehm.

Zavádza dva prelomové koncepty:

- **iteratívny prístup**
- **opakovanú analýzu rizík**



Iteratívny prístup

Predstavme si projekt, ktorý vzniká na zelenej lúke – teda nie je to iba inovácia existujúceho prístupu.

Pri takomto projekte je dosť nepravdepodobné, že zadávateľ bude vedieť presne a podrobne špecifikovať, čo vlastne má výsledný systém robiť, aké by mal mať vlastnosti, dátové štruktúry, užívateľské rozhranie.

V takej situácii by bolo podstatne jednoduchšie, a v konečnom dôsledku asi aj efektívnejšie, nesnažiť sa o nesplniteľné, ale iba rámcovo definovať vlastnosti systému, jeho architektúru a postupne došpecifikovávať detaily.

Vývoj aplikácie by teda prebiehal v akýchsi opakovaných cykloch – iteráciach.

Analýza rizík

- **termíny** – stihneme dodať aplikáciu včas?
- **funkcie** – zvládneme implementovať všetko, čo si zákazník požadoval?
- **náklady** – nebudeme mať vyššie náklady, ako je cena?
- **legislatíva** – ktoré zákony a predpisy môžu obmedziť či ohroziť dokončenie a dodanie aplikácie?
- **konkurencia** – nedodá konkurencia podobný produkt skôr a lacnejšie?
- **hardvér** – budeme vedieť náš produkt prevádzkovať na špecifikovanom počítačovom vybavení u zákazníka?

Ciele analýzy rizík

Dopredu odhadnúť možné ohrozenia priebehu projektu

Pripraviť si reakcie na tieto ohrozenia

Typy rizík:

- **projektové riziká –** zníženie rozpočtu, odchod dôležitých ľudí, oneskorenie subdodávky
- **technické riziká –** nevyskúšané alebo nezvládnuté metodiky alebo technológie, výpadky a zlyhanie hardvéru či podporného softvéru, problémy s vývojovými nástrojmi, s operačným systémom, databázami a pod.
- **obchodné riziká –** zlý odhad odbytu a záujmu, uvedenie konkurenčného produktu, nezvládnutý marketing a pod.

Druhy prototypov

- **Ilustratívny prototyp –** dôraz sa kladie na vzhľad, základnú funkčnosť a návrh užívateľského rozhrania
- **Funkčný prototyp –** implementuje sa jadro systému so základnými funkciami, ktoré sa postupne dodávajú, pričom ich poradie je výsledkom analýzy rizík.
- **Overovací prototyp –** implementuje sa určitá časť systému s účelom ujasniť si požiadavky alebo technológie

Výhody

- **Včasné vylúčenie nevhodných riešení**
- **Nezávislosť na konkrétnej metodike**
- **Vývoj znovu použiteľných komponent** – v každom kroku sa prevádza analýza alternatív, ktorá môže vyústiť do rozhodnutia použiť existujúcu komponentu aj za cenu menších úprav
- **Pre každú aktivitu a zdroj je zodpovedaná kľúčová otázka – koľko je dosť –** vďaka rizikom riadenému prístupu je možné stanoviť, koľko času je nutné venovať analýze problému a špecifikácii požiadaviek, plánovaniu, zaisteniu kvality, testovaniu atď. Množstvo zdrojov potrebných pre každú činnosť vyplynie z množstva a rozloženia rizík odhalených ich analýzou.

Nevýhody

- **Nutná expertíza ľudí – analýza zdrojov rizík**
- **Celková komplikovanosť metodiky**
- **Softvér nie je uvoľnený pred skončením posledného cyklu**
- **Zmeny požiadavkov je možné až po skončení cyklu - iterácie**.

Prvá úloha

Vymyslíte a navrhnete zadanie malého programovacieho problému, ktoré je možné naprogramovať v priebehu jedného seminára.

Zadanie napíšete v tak zrozumiteľnej podobe, aby ho mohli na seminári programovať aj Vaši kolegovia.

• Úlohu pošlite ako **zip súbor** cez úlohu 1. z tohto predmetu na stránku predmetu.

POZOR: Na zaslanie zadanie už musíte prihlásení na predmet a musíte byť mnou akceptovaní.

Termín odoslania: **piatok 5.10.2007 – 24.00**