Čo je informatika?

9.2. informatické vedy, informačné a komunikačné technológie
9.2.1. informatika
9.2.3. teória vyučovania informatiky
9.2.7. kybernetika
9.2.9. aplikovaná informatika
9.2.10. hospodárska informatika

Čo je práca inžiniera?

Inžinier by mal vedieť
pochopiť informáciu,
interpretovať ju a
aplikovať v danom kontexte,
vrátane odovzdania informácie
(formalizovania)

ANALÝZA

NÁVRH

IMPLEMENTÁCIA

Kedy a ako napísať abstrakt?

Čo inžinier informatiky píše?

Ako písať?

To je veľká otázka...

Aké menšie otázky sa za ňou skrývajú?

Plán písania = názov + abstrakt + štruktúra

Pozrime nejaké príklady a identifikujme problémy

Aké majú byť vety?

Pozrime nejaké príklady a identifikujme problémy

2.4 Proces vývoja softvéru pomocou Modelom riadenej architektúry

Najprv sa vytvorí Výpočtovo nezávislý model (Computation independent model - CIM). Tento model má najväčšiu úroveň abstrakcie a nahliada na systém len zvonka. Z tohto modelu

V čom písať?

WYSIWYG
What You See Is What You Get

But...
You Get Only What You See

How about this:
What You See Is What You Mean
WYSIWYM

Podobá sa písanie bežného textu na programovanie?

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možné riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých organizačných jednotkách. Ide o poprepletanie vlastností, ktoré sa tiež označujú ako pretínajúce súvislosti (crosscutting concerns). Odpoveď na riešenie týchto a mnohých ďalších problémov poskytlo práve aspektovo-orientované programovanie, ktoré je považované za istú evolučnú odnož objektovo-orientovaného programovania.(11)
Od začiatku 90. rokov minulého storočia sa začala rozvíjať prvá odnož aspektovo-orientovaného programovania kompozičné filtre. Následne sa začala vyvíjať paradigma adaptívneho programovania. Z tejto paradigmy vznikol jazyk DemeterJ. Začiatky aspektovo-orientovaného prístupu boli položené vo výskumnom centre Xerox PARC. Ide o miesto, kde bol vytvorený momentálne stále najpoužívanejší aspektovo-orientovaný jazyk AspectJ. Následne vznikali ďalšie aspektovo-orientované programovacie jazyky, pre ktoré sa stál základom jazyk AspectJ. Postupne vznikali aj iné jazyky, ktoré používali odlišný spôsob zápisu aspektov a iný celkový pohľad.

4.1 Vznik a história aspektovo-orientovaného programovania

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možné riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých organizačných jednotkách. Ide o poprepletanie vlastností, ktoré sa tiež označujú ako pretínajúce súvislosti (crosscutting concerns). Odpoveď na riešenie týchto a mnohých ďalších problémov poskytlo práve aspektovo-orientované programovanie, ktoré je považované za istú evolučnú odnož objektovo-orientovaného programovania.(11)
Od začiatku 90. rokov minulého storočia sa začala rozvíjať prvá odnož aspektovo-orientovaného programovania kompozičné filtre. Následne sa začala vyvíjať paradigma adaptívneho programovania. Z tejto paradigmy vznikol jazyk DemeterJ. Začiatky aspektovo-orientovaného prístupu boli položené vo výskumnom centre Xerox PARC. Ide o miesto, kde bol vytvorený momentálne stále najpoužívanejší aspektovo-orientovaný jazyk AspectJ. Následne vznikali ďalšie aspektovo-orientované programovacie jazyky, pre ktoré sa stál základom jazyk AspectJ. Postupne vznikali aj iné jazyky, ktoré používali odlišný spôsob zápisu aspektov a iný celkový pohľad.

Prednáška 2:

Inžinierska práca v informatike a písanie technického textu

Valentino Vranić

Ústav informatiky a softvérového inžinierstva

STU FIIT

vranic@stuba.sk fiit.sk/~vranic

MIP 2015/16

1. 10. 2015

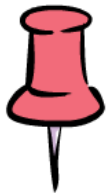
Čo je informatika?

Sústava študijných odborov SR
http://www.minedu.sk/index.php?lang=sk

9.2. informatické vedy, informačné a komunikačné technológie
9.2.1. informatika
9.2.2. teoretická informatika
9.2.3. teória vyučovania informatiky
9.2.4. počítačové inžinierstvo
9.2.5. softvérové inžinierstvo
9.2.6. informačné systémy
9.2.7. kybernetika
9.2.8. umelá inteligencia
9.2.9. aplikovaná informatika
9.2.10. hospodárska informatika
9.2.11. kognitívna veda

Univerzity poskytujú **študijné programy** v stanovených študijných odboroch
https://www.portalvs.sk/sk/studijne-odbory/podskupina/902

Čo je práca inžiniera?



Inžinier by mal vedieť

pochopiť informáciu,

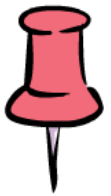
interpretovať ju a

aplikovať v danom kontexte,

vrátane odovzdania informácie

(formulovania)

D. Messer et al. Engineering Information Literacy and Communication. In Proceedings of the 12th International Conference on Learning, Granada, Spain, 2005. http://eprints.qut.edu.au/1606/1/1606.pdf



Inžinier by mal vedieť

pochopiť informáciu,

interpretovať ju a

aplikovať v danom kontexte,

vrátane odovzdania informácie (formulovania)

ANALÝZA

NÁVRH

IMPLEMENTÁCIA

Čo inžinier informatiky píše?

Ako písať?

To je veľká otázka...

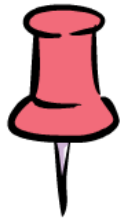
Aké menšie otázky sa za ňou skrývajú?

Plán písania =

názov +

abstrakt +

štruktúra



Kedy a ako napísať abstrakt?

Kedy a ako napísať abstrakt?

Pozrime nejaké príklady a identifikujme problémy

Práca sa zaoberá využitím Model Driven Architecture pri návrhu aspektovo-orientovaných programov. Porovnáva vlastnosti a možnosti niektorých aspektovo-orientovaných jazykov. Analyzuje možnosti ich reprezentácie modelmi v jazyku UML. Následne sa zaoberá možnosťami ich transformácie. Výsledkom práce je navrhnúť spôsob reprezentácie niektorých aspektovo-orientovaných jazykov v modeloch použiteľných v MDA. Práca obsahuje neformálne a formálne návrhy transformácií medzi týmto všeobecným modelom a modelmi konkrétnych jazykov. Opisuje spôsob pridávania nových analyzovaných aspektovo-orientovaných jazykov do výsledného grafu metamodelov aspektovo-orientovaných jazykov.

Neustále je potrebná tvorba nového softvéru pre rôzne mobilné zariadenia. Platforiem na ktorých môže aplikácia bežať je veľké množstvo a každá z nich má svoje vlastné špecifické vlastnosti. Už dlhšie existujú nástroje, ktoré umožňujú vývoj týchto aplikácií pre rôzne platformy naraz. Na vytváranie aplikácií pre rôzne platformy slúži modelom riadená architektúra. V tejto práci sme oboznámený so základným princípom používania modelom riadenej architektúry pri vývoji aplikácií pre rôzne platformy. Práca analyzuje použitie modelom riadenej architektúry pri vývoji multiplatformových mobilných aplikácií. Práca sa zaoberá inými prácami, ktorý použili tento princíp pri vývoji rôzneho softvéru. V tejto práci je za použitia modelom riadenej architektúry vytvorené používateľské rozhranie bližšie špecifikovanej mobilnej aplikácie pre rôzne mobilné platformy, pre ktoré je následne napísaná evaluácia.

Organizational patterns are the key to a stepwise adoption of agile and lean approaches and to a piecemeal growth of agile and lean organization of work. However, their text description is not easy to comprehend. In this paper, we introduce our initial efforts towards establishing an approach to animate organizational patterns as text adventure games. Players pass through a series of scenes described using Erickson's conversational hypnosis language patterns in order to better evoke their experience. The game scenario space is expressed using UML state machine diagrams. The approach is presented on adventure games we created for the Architect Also Implements organizational pattern.

Kde všade v texte možno sledovať štruktúru?

Aké majú byť vety?

Aké majú byť vety?

Pozrime nejaké príklady a identifikujme problémy

2.4 Proces vývoja softvéru pomocou Modelom riadenej architektúry

Najprv sa vytvorí Výpočtovo nezávislý model (Computation independent model - CIM). Tento model má najväčšiu úroveň abstrakcie a nahliada na systém len zvonka. Z tohto modelu

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možne riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých organizačných jednotkách. Ide o poprepletanie vlastností, ktoré sa tiež označujú ako pretínajúce súvislosti (crosscutting concerns). Odpoveď na riešenie týchto a mnohých ďalších problémov poskytlo práve aspektovo-orientované programovanie, ktoré je považované za istú evolučnú odnož objektovo-orientovaného programovania.(11)
Od začiatku 90. rokov minulého storočia sa začala rozvíjať prvá odnož aspektovo-orientovaného programovania kompozičné filtre. Následne sa začala vyvíjať paradigma adaptívneho programovania. Z tejto paradigmy vznikol jazyk DemeterJ. Začiatky aspektovo-orientovaného prístupu boli položené vo výskumnom centre Xerox PARC. Ide o miesto, kde bol vytvorený momentálne stále najpoužívanejší aspektovo-orientovaný jazyk AspectJ. Následne vznikali ďalšie aspektovo-orientované programovacie jazyky, pre ktoré sa stál základom jazyk AspectJ. Postupne vznikali aj iné jazyky, ktoré používali odlišný spôsob zápisu aspektov a iný celkový pohľad.

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možne riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých organizačných jednotkách. Ide o poprepletanie vlastností, ktoré sa tiež označujú ako pretínajúce súvislosti (crosscutting concerns). Odpoveď

4.1 Vznik a história aspektovo-orientovaného programovania

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možne riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých organizačných jednotkách. Ide o poprepletanie vlastností, ktoré sa tiež označujú ako pretínajúce súvislosti (crosscutting concerns). Odpoveď na riešenie týchto a mnohých ďalších problémov poskytlo práve aspektovo-orientované programovanie, ktoré je považované za istú evolučnú odnož objektovo-orientovaného programovania.(11)
Od začiatku 90. rokov minulého storočia sa začala rozvíjať prvá odnož aspektovo-orientovaného programovania kompozičné filtre. Následne sa začala vyvíjať paradigma adaptívneho programovania. Z tejto paradigmy vznikol jazyk DemeterJ. Začiatky aspektovo-orientovaného prístupu boli položené vo výskumnom centre Xerox PARC. Ide o miesto, kde bol vytvorený momentálne stále najpoužívanejší aspektovo-orientovaný jazyk AspectJ. Následne vznikali ďalšie aspektovo-orientované programovacie jazyky, pre ktoré sa stál základom jazyk AspectJ. Postupne vznikali aj iné jazyky, ktoré používali odlišný spôsob zápisu aspektov a iný celkový pohľad.

4.1 Vznik a história aspektovo-orientovaného programovania

Dlhšiu dobu sa menili a vylepšovali aj techniky prístupu k návrhu a implementácii. Prvotné programy sa vyvíjali v strojovom jazyku. Síce to viedlo k efektívnym programom, ale na úkor pomalého vývoja, pretože programátor musel premýšľať na úrovni inštrukčnej sady procesora. Následne sa začali používať vyššie programovacie jazyky, ktoré už umožňovali lepšiu abstrakciu. Tie rozkladali systém na procedúry riešiace jednotlivé problémy. Každá novšia programovacia paradigma umožňovala premýšľať na abstraktnejšej úrovni, a tým bolo možne riešiť zložitejšie úlohy. (10)
Pred desiatkami rokov sa objavilo objektovo orientované programovanie. Do programátorského prostredia sa dostal mocný prostriedok, ako vizualizovať vznikajúci systém, dekomponovať ho na jednotlivé entity a modelovať vzťah medzi nimi. Základným problémom objektovo-orientovaného programovania ostáva skutočnosť, že sa jedna v podstate o statický prístup. Akákoľvek zmena objektu, tak vyžaduje zásah do vnútra, čo môže byť pri zložitých systémoch prácne a nepohodlné. Zároveň sa programátori stretávajú s vlastnosťami alebo črtami systému, ktoré nekorešpondujú priamo so štruktúrami objektovo-orientovaného programovania, čím sa stávajú jeho organizačnými jednotkami ťažko opísateľné. Dochádza k pretínaniu dizajnu celého softvérového systému, pretože implementačný kód sa nachádza vo viacerých

V čom písať?

WYSIWYG

What You See Is What You Get

But...

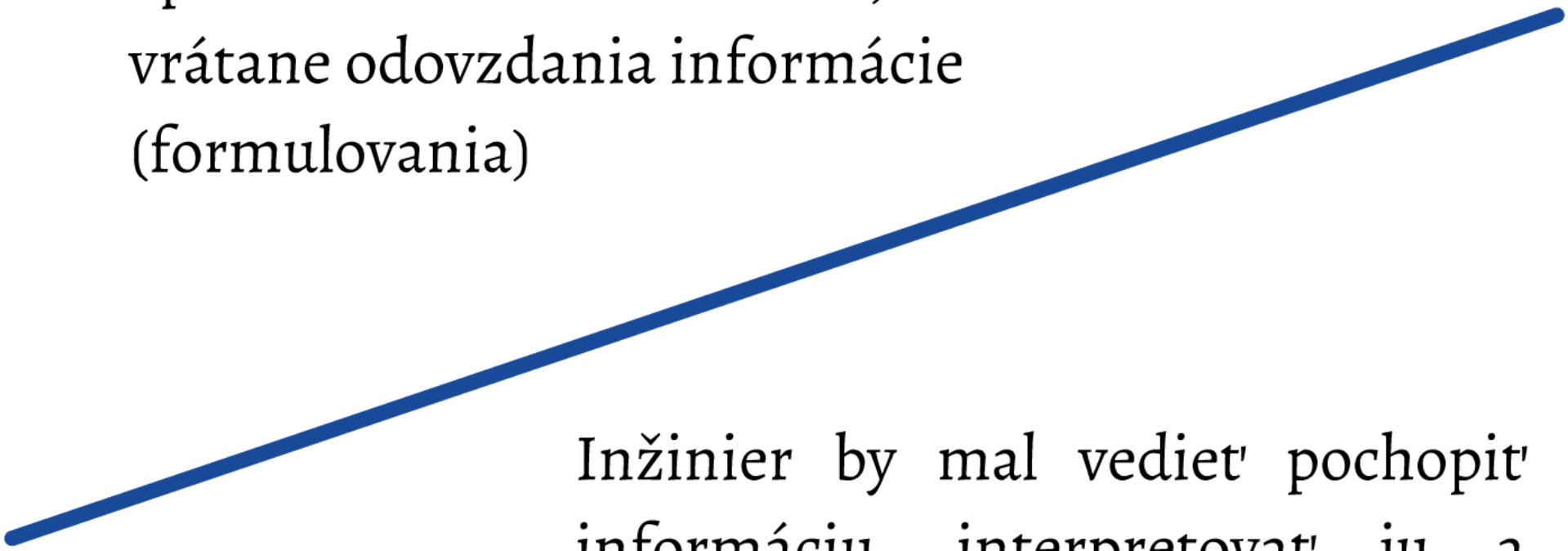
You Get Only What You See

How about this:

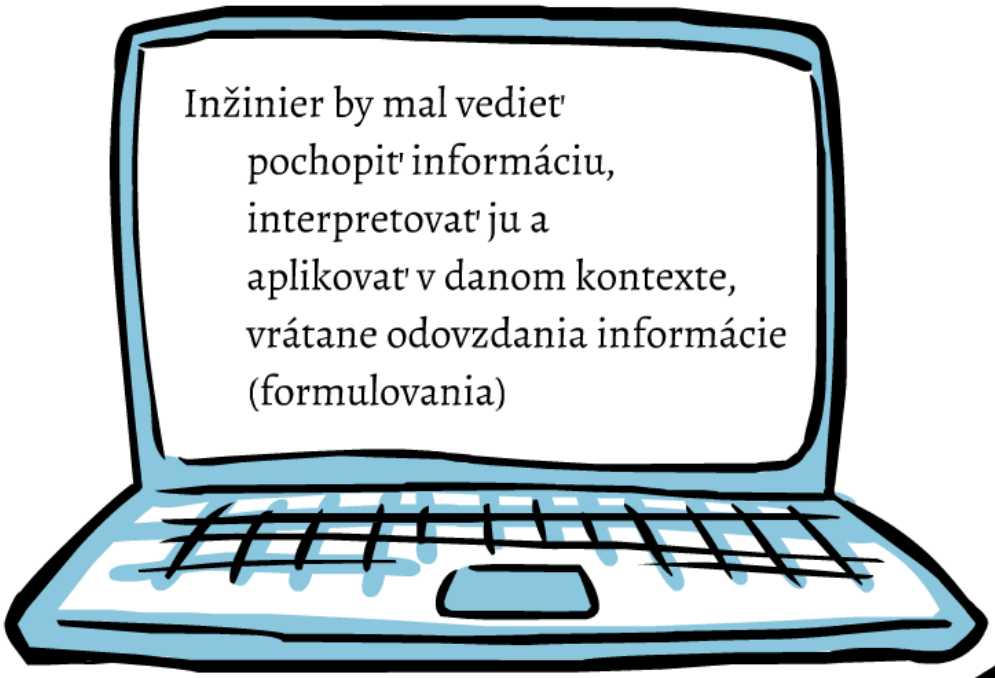
What You See Is What You Mean

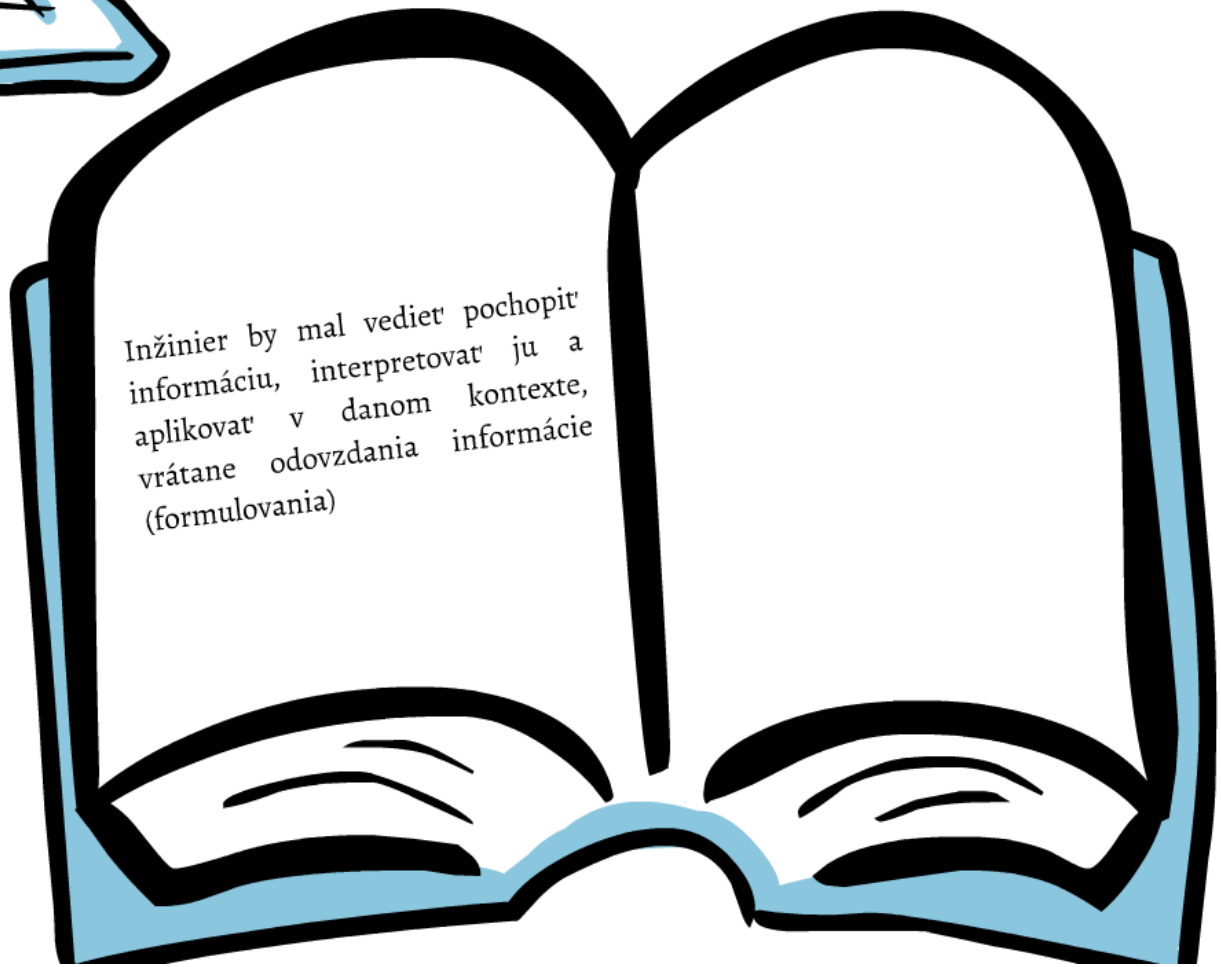
WYSIWYM

Inžinier by mal vedieť
pochopiť informáciu,
interpretovať ju a
aplikovať v danom kontexte,
vrátane odovzdania informácie
(formulovania)

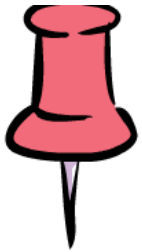


Inžinier by mal vedieť pochopiť informáciu, interpretovať ju a aplikovať v danom kontexte, vrátane odovzdania informácie (formulovania)


Inžinier by mal vedieť
pochopiť informáciu,
interpretovať ju a
aplikovať v danom kontexte,
vrátane odovzdania informácie
(formulovania)


Inžinier by mal vedieť pochopiť
informáciu, interpretovať ju a
aplikovať v danom kontexte,
vrátane odovzdania informácie
(formulovania)

Podobá sa písanie bežného textu na programovanie?



> Inžinier analyzuje, navrhuje a implementuje

> Inžnier spolupracuje

> Inžinier v značnej miere píše

> Plán písania = názov + abstrakt + štruktúra

> Štruktúra textu: nadpisy, odseky a vútorná štruktúra vety

> Konzistencia, presnosť a koncíznosť vo vyjadrovaní

> V čom písať

Dotazník:
http://tinyurl.com/mip-pred02

Práca sa zaoberá využitím Model Driven Architecture pri návrhu aspektovo-orientovaných programov. Porovnáva vlastnosti a možnosti niektorých aspektovo-orientovaných jazykov. Analyzuje možnosti ich reprezentácie modelmi v jazyku UML. Následne sa zaoberá možnosťami ich transformácie. Výsledkom práce je navrhnúť spôsob reprezentácie niektorých aspektovo-orientovaných jazykov v modeloch použiteľných v MDA. Práca obsahuje neformálne a formálne návrhy transformácií medzi týmto všeobecným modelom a modelmi konkrétnych jazykov. Opisuje spôsob pridávania nových analyzovaných aspektovo-orientovaných jazykov do výsledného grafu metamodelov aspektovo-orientovaných jazykov.