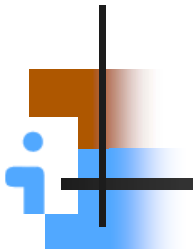
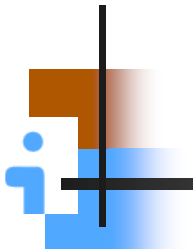


# Počítačové hry

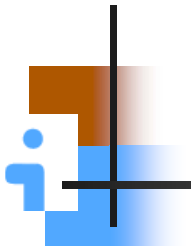
Martin Klíma

martin@idea-games.com



# Úvod

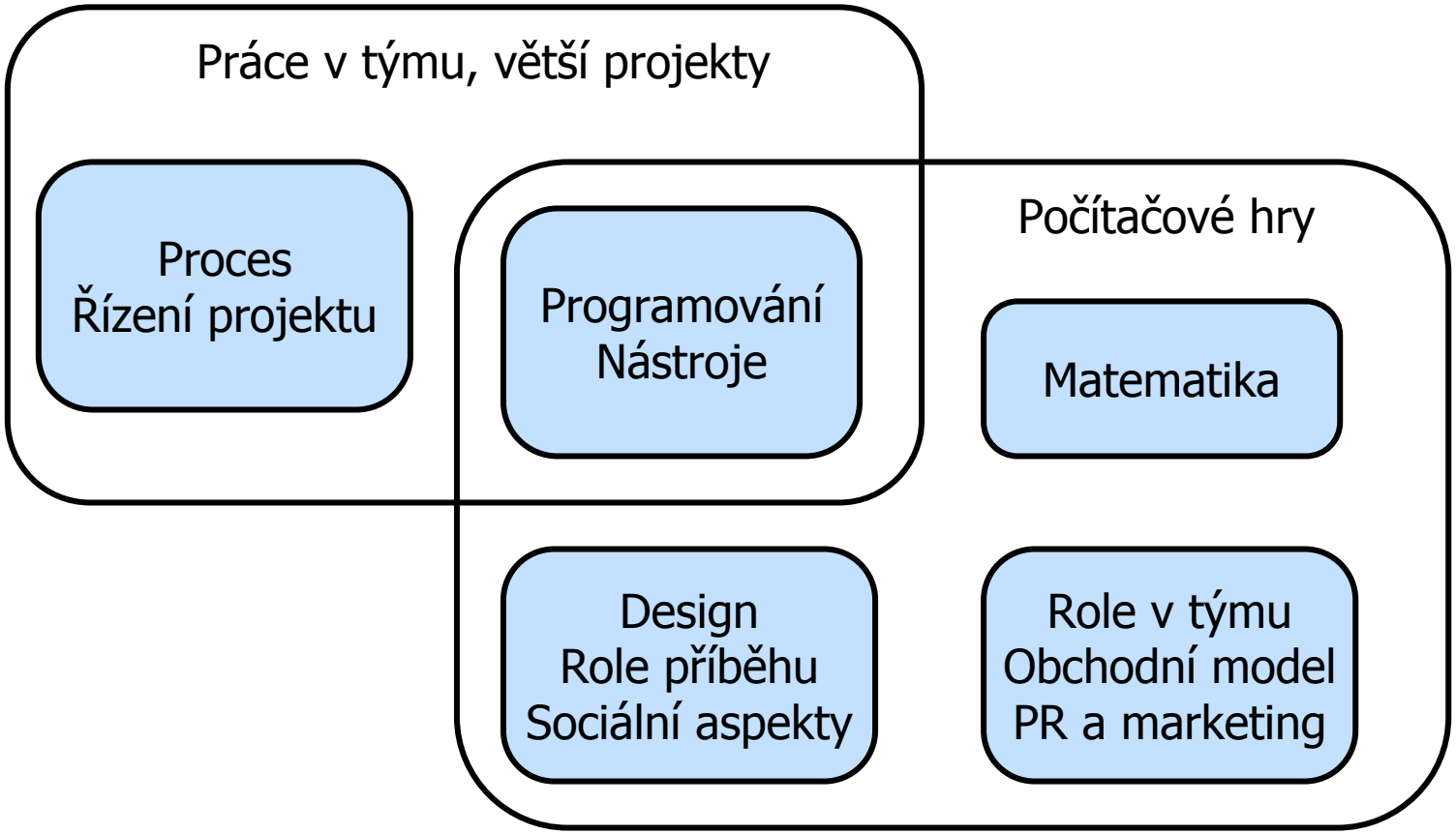
Pro koho je tento kurs určen; za co se dostávají kredity

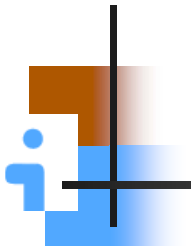


# Pro koho je tato přednáška

- Pracovat v týmu
- Pracovat na vývoji her
- Nikoliv nutně pro programátory

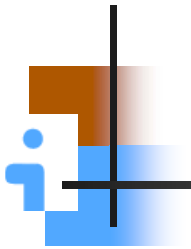
# Okruhy přednášky





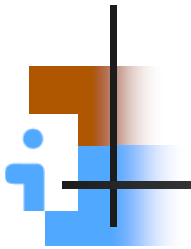
# Cíle přednášky

- Zvýšit vaši cenu na trhu práce
- Vytvořit seminární práci a zkusit si:
  - Práci v týmu
  - Práci simulující reálné podmínky
  - Celý vývojový cyklus počítačové hry



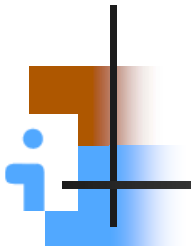
# Seminární práce

- Týmová práce
  - Tým 3-4 lidí, různé role
  - Termín pro sestavení týmu je 20. 10.
  - Tým je možno měnit jen do 21. 12.
  - Poznávací setkání: 18. 10. v 17:20, S5
  - http://braille.ms.mff.cuni.cz/mailman/listinfo/swi115
  - http://ksvi.mff.cuni.cz/~brom
- Cvičení
  - Začínáme v 27. října v 10:40 (pátek) v SW1



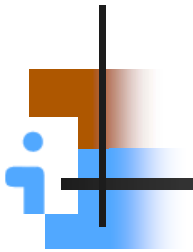
# Seminární práce

- Termíny
  - Design, Plán, Architektura
    - do 21. 12.
  - Finální verze (gold)
    - do 28. 2.
- Hodnocení
  - Tým jako celek
  - Kritéria
    - Dokumentace
    - Presentace
    - Funkčnost/vzhled/úplnost



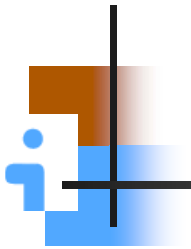
# Design, Plán, Architektura

- Design hry ~ 10 stran
  - vize projektu
  - žánr, podobné hry/odlišnosti, unikátní vlastnosti
  - ovládání, interface
  - herní mechanismy
- Plán vývoje ~ 10 stran
  - specifikace prototypu
  - předběžné odhady času
- Architektura ~ 5 stran
  - základní rozhodnutí o architektuře projektu
  - použitá platforma, middleware, aj.
  - workflow



# Počítačové hry…

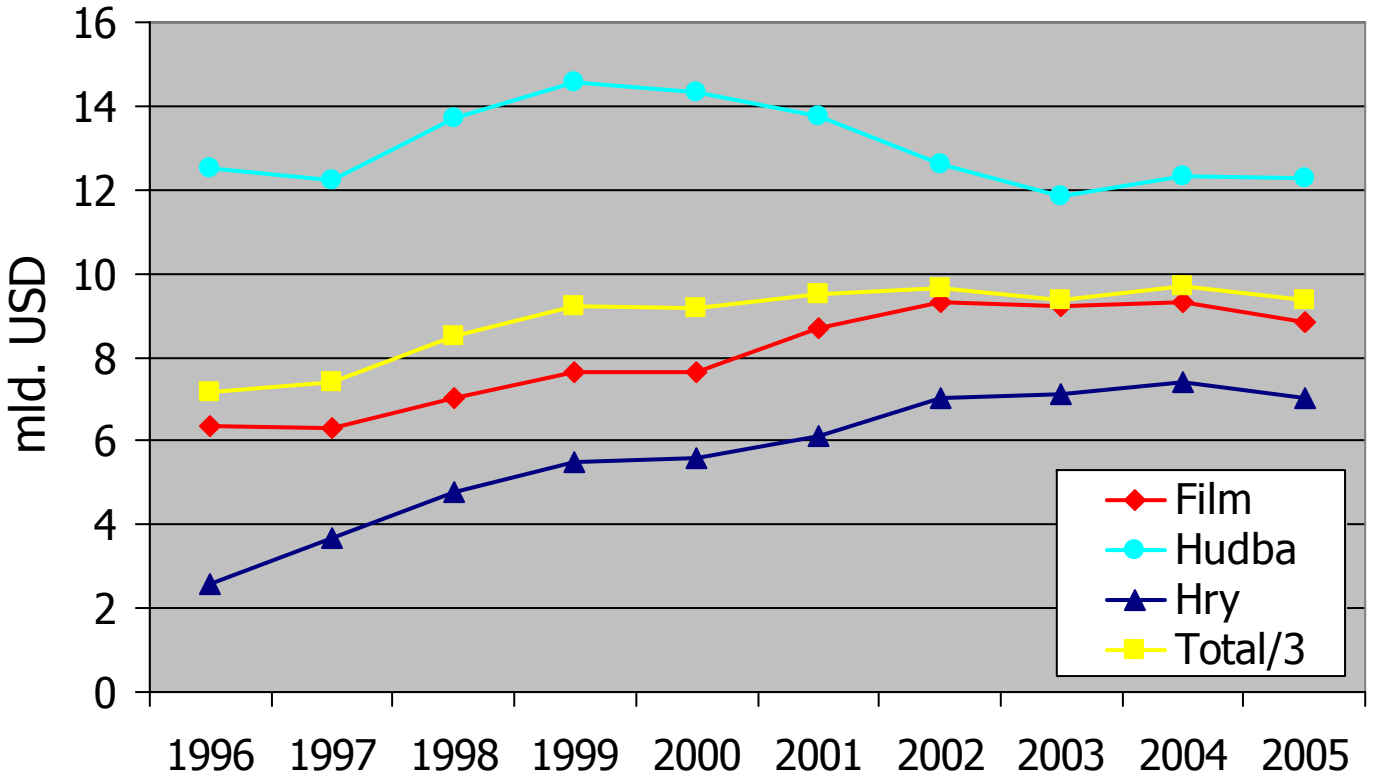
Co to je, co vás čeká, jak to funguje.



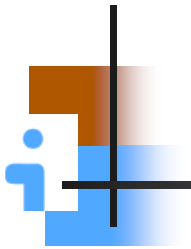
# Co to je?

- 69% Američanů hraje video hry.
- Věk průměrného hráče je 33 let.
  - 25% je starší než 50 let
  - Průměrný věk toho, kdo kupuje hry je 40 let
- V průměru hrají 6,8 hodiny týdně
- 58% online hráčů jsou muži, 42% ženy
- Nejoblíbenější žánry:
  - Na PC: strategie (31%), dětské (20%)
  - Na konzoli: akce (30%), sporty (17%)

# Jak se jim daří?

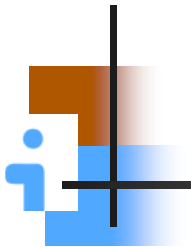



Pramen: ESA, showbizfacts.com, RIAA



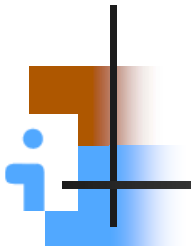
# Co vás čeká?

- Stress
- Málo peněz
- Malá společenská prestiž
- Dlouhá pracovní doba
- Žádné databáze



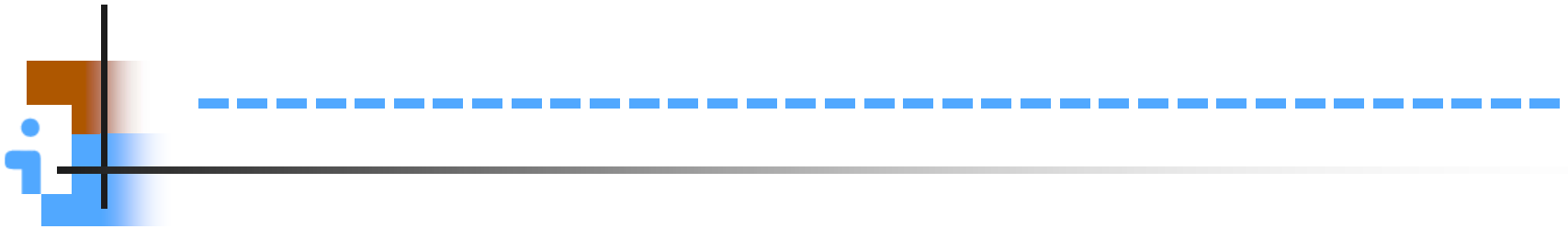
# Co vás čeká kromě toho?

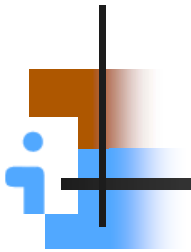
- V ČR je několik vývojářů na světové úrovni:
  - ALTAR, BES, BIS, Illusion Softworks, Mindware
- A pár už taky není:
  - Hra01, Plastic Reality, Pterodon, 7fx
- Mobilní hry
  - Nostromo, Redboss
- We are hiring!
  - Zkušený programátor si celkem snadno najde práci v zahraničí
  - Což lze vzhledem k celkovému vývoji jen doporučit



# Jak to funguje

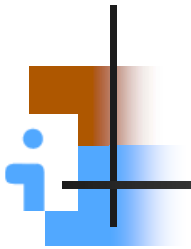
- Velikost týmu (v ČR)
  - IS - ~100 lidí, sdílí enginový tým
  - ALTAR, BIS - ~30 lidí, vlastní engine
    - 5 – 7 programátorů
    - 8 – 15 grafiků (žádný outsourcing)
    - 3 – 5 designérů
    - vedoucí projektu, zvukař, technik, admin, atd.
- Next gen týmy
  - Větší (~100 lidí), ale větší outsourcing





# Proces a řízení projektu

Práce v týmu; větší projekty

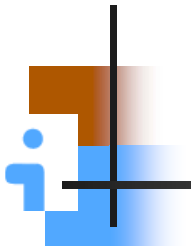


# Literatura


- Steve McConnell. *Software Project Survival Guide*, Redmond, Washington USA: Microsoft Press, 1998
- Alistair Cockburn. *Surviving Object-Oriented Projects*, Indianapolis, Indiana USA: Addison-Wesley, 1998
- Tom DeMarco a Timothy Lister. *Peopleware: Productive Projects and Teams 2nd Ed.*, New York, New York USA: Dorset House, 2001
- Tom DeMarco a Timothy Lister. *Waltzing With Bears.*, New York, New York USA: Dorset House, 2001


---

- Na MFF UK se o procesu vývoje software přednáší v těchto kurzech:
  - *Vedení databázových aplikací a jazyk UML*
    Tomáš Rubač, zimní semestr
  - *Softwarové inženýrství*
    Jan Pavelka, letní semestr

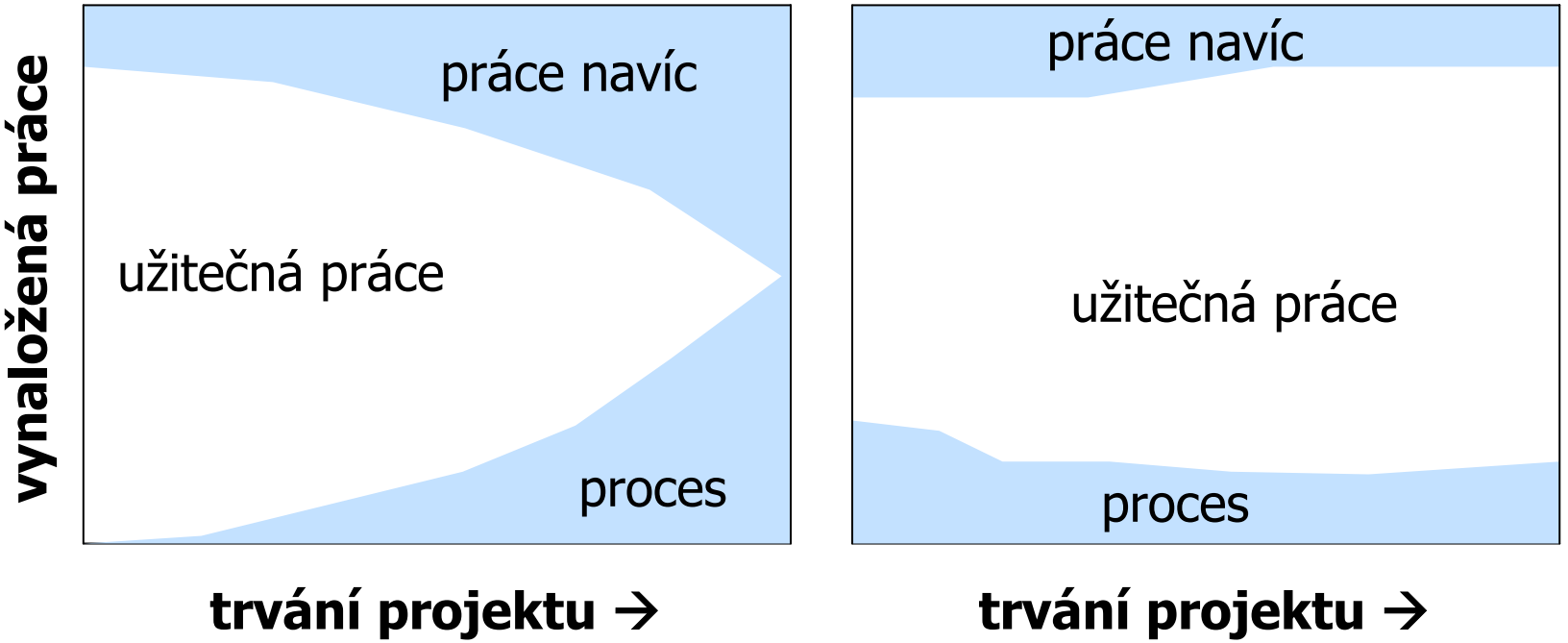


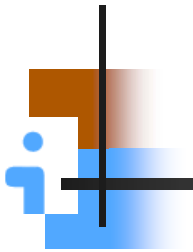
# Proces

- Formalizované postupy
- Práce navíc, která se dá plánovat, místo práce navíc, která je zbytečná
- Proces jsou dvě různé věci:
  - lineární proces, od začátku projektu po jeho dokončení
  - cyklický proces, 'metodika', způsob, jak se na projektu pracuje

# Proces vs. zbytečná práce

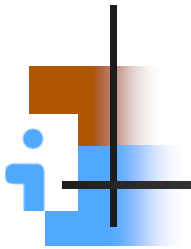
- Každý projekt používá proces; přijde na to, kdy





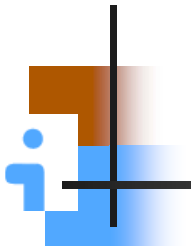
# Metodika

Každodenní práce na projektu



# Součásti metodiky

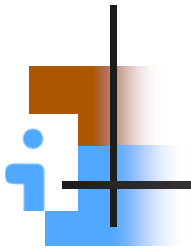
- metodika vs. Metodika
- Metodiku tvoří:
  - Standardy
  - Nástroje
  - Postupy a techniky
  - Role, týmy a dovednosti



# Role, týmy a dovednosti

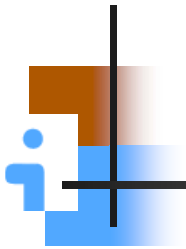
- každý člověk může mít více rolí a být součástí více týmů
- Typické role a dovednosti
  - Architekt
  - Vedoucí projektu
  - Hlavní programátor/návrhář* (kódu, ne hry!)
  - Programátor/návrhář
  - Tester
  - Návrhář interfacu

*Budeme odlišovat návrh (programu) a design (hry)



# Role, týmy a dovednosti

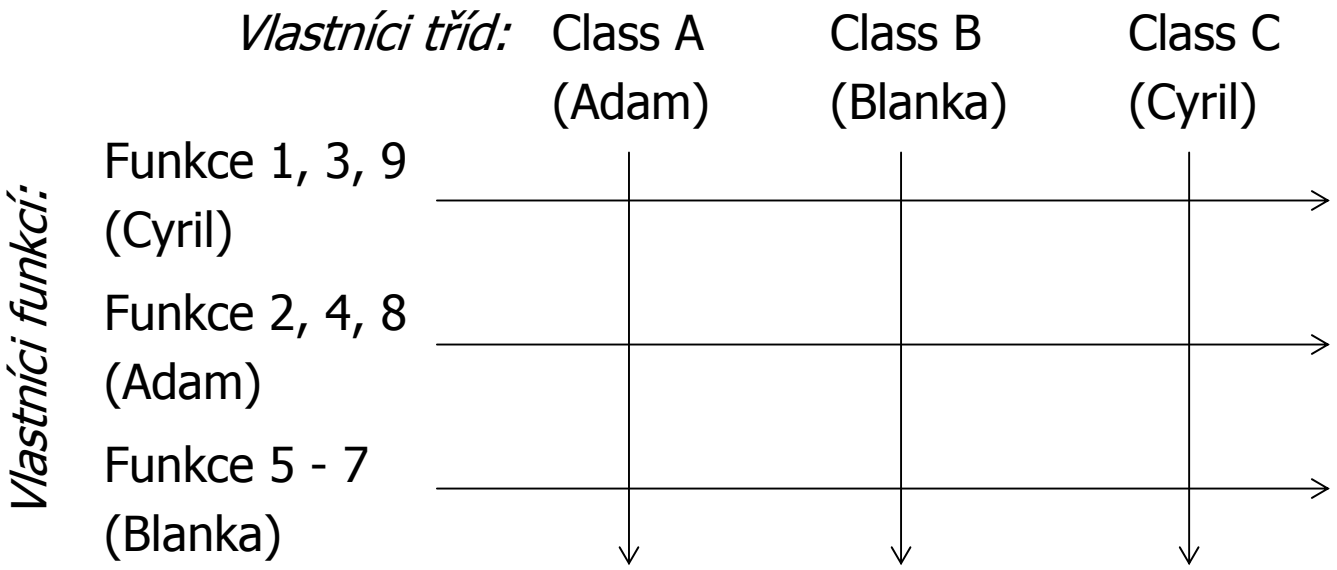
- Typické týmy
  - Funkční týmy
  - Infrastrukturní týmy
  - Testovací tým
  - Sledování projektu
  - Architektonický tým
  - …

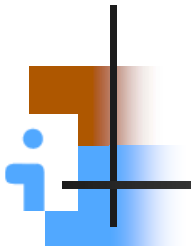


# Vlastnictví

- Každý modul (třída) má svého vlastníka
- Každý výstup (požadovaná funkčnost) má vlastníka

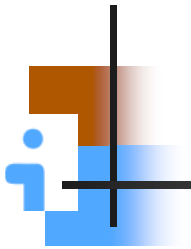
**Matice vlastnictví: funkce x komponenty**





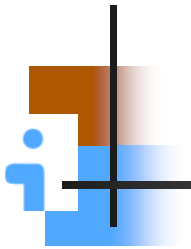
# Postupy a techniky

- Plánování
- Postupné odevzdávání – fáze, iterace
- Řízení rizik
- Kontrola změn
- Testování – QA
- Průhlednost projektu
- Pracovní prostředí
- Evidence času, měření produktivity
- Zapojení uživatelů



# Plánování

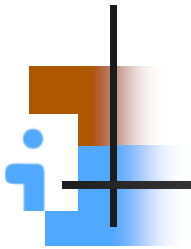
- Plánování má dvě části:
  - Plán procesu
  - Odhady času – více v části o Vývoji projektu
- Proces
  - Plán vývoje
  - Plán testování
  - Struktura týmu
  - Zdroje a kde je vzít
  - Proces tvorby odhadů, kontroly změn
  - Nástroje, metody a techniky



# Postupné odevzdávání*

- 'Fáze' a 'inkrement' je totéž: jeden ucelený úsek projektu.
- 'Iterace' je jeden pokus o dodání požadované funkčnosti.

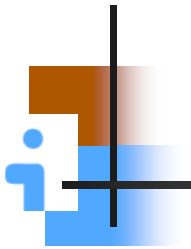
*Staged delivery



# Fáze* a inkrementy

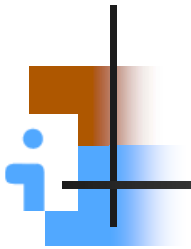
- Součásti fáze/inkrementu
  - Specifikace
  - Návrh
  - Konstrukce
  - Test
- Mezi fázemi
  - Zhodnocení průběhu projektu
  - Učení se
  - Změny kursu

*Stage



# Iterace

- Iterace
  - vyhodnocení a přehodnocení dokončené práce
  - Následuje buď po každém kroku fáze nebo jako opakování celé fáze
- Problémy
  - Ulpívání na prototypech
  - Spirála smrti



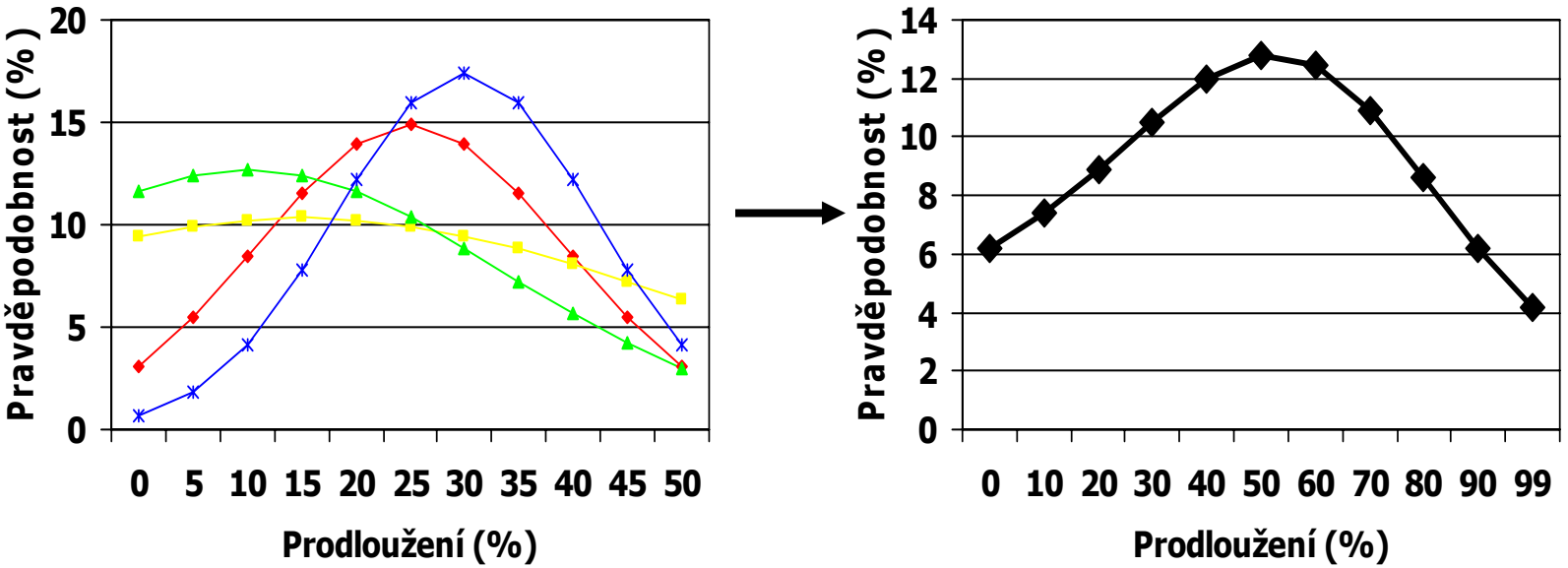
# Řízení rizik*

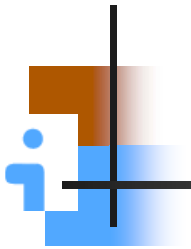
- Riziko je problém, který ještě nenastal, zatímco problém je riziko, které se realizovalo
- Inventarizace rizik
- Pro každé riziko určíme:
  - odhad pravděpodobnosti
  - způsob detekce
  - preventivní opatření a jejich cenu
  - cenu a způsob řešení následků
  - rozhodnout, zda je výhodnější podnikat preventivní opatření nebo ne

*Risk management

# Riziko a plánování

- Datum dokončení projektu známe jen s jistou pravděpodobností

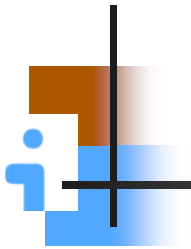




# Kontrola změn*

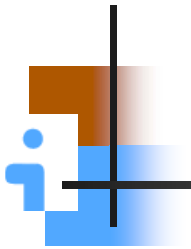
- Vývoj probíhá ve dvou fázích:
  - Prvotní vývoj (bez kontroly změn)
  - — Odborné posouzení —
  - Úpravy (schválené VPKZ)
- Výbor pro kontrolu změn (VPKZ)
  - Zástupci všech 'stakeholders' projektu
  - 1-3 lidi
  - Má finální slovo ohledně všech dodatečných změn

*Change control



# Postup při změně

- Návrh na změnu obsahuje:
  - Co se bude měnit
  - Důvody změny
  - Cena a výhody změny z hlediska navrhovatele
- Výbor určí, koho se změna týká a pošle jim návrh na posouzení
- Oslovení určí cenu a výhody změny ze svého hlediska
- Výbor vyhodnotí tato data a rozhodne se změnu zamítnout, odložit nebo provést
- Výbor uvědomí všechny zúčastněné o svém rozhodnutí

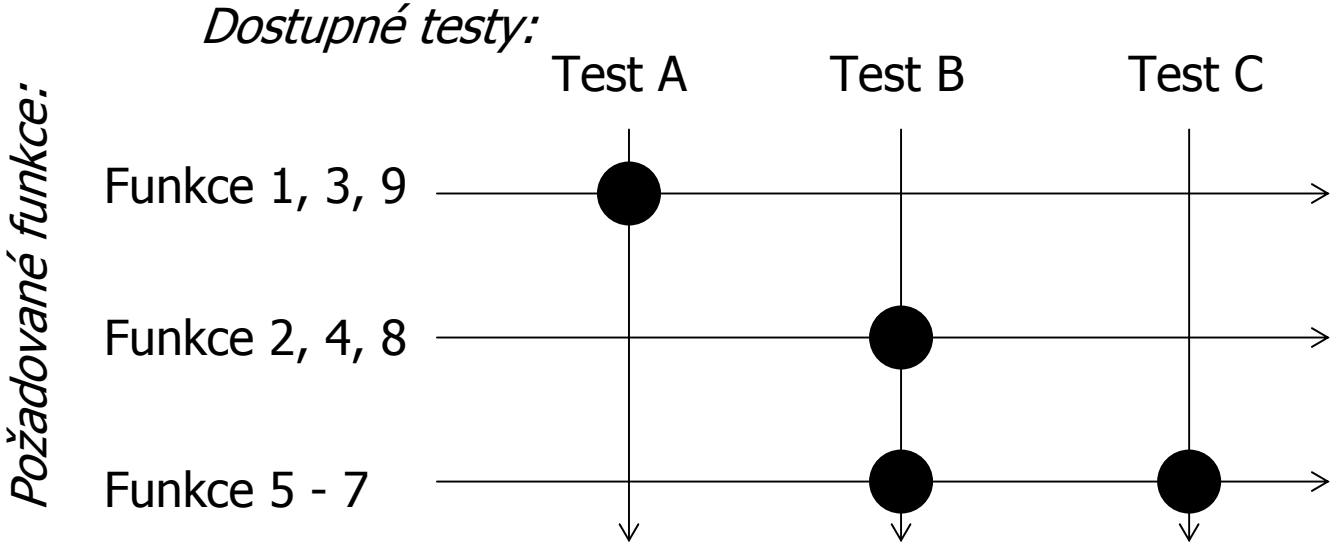


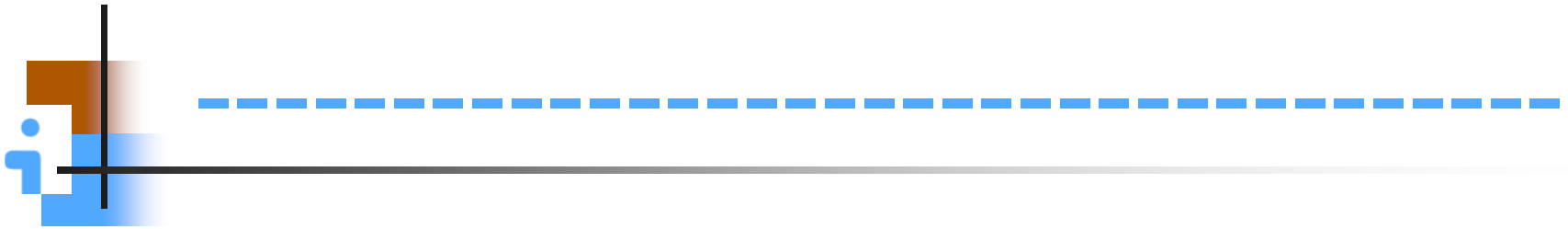
# Testování/QA

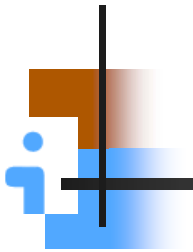
- Posuzování (review)
  - rozeslání
  - příprava
  - schůzka
  - zpráva
  - implementace
- Testování
  - provádí sám autor
- Systémové testování
  - 'klasické' testování

# Systémové testování

- Součástí vývoje je specifikace 'testovacích případů'
- Korespondence mezi požadavky a testy:

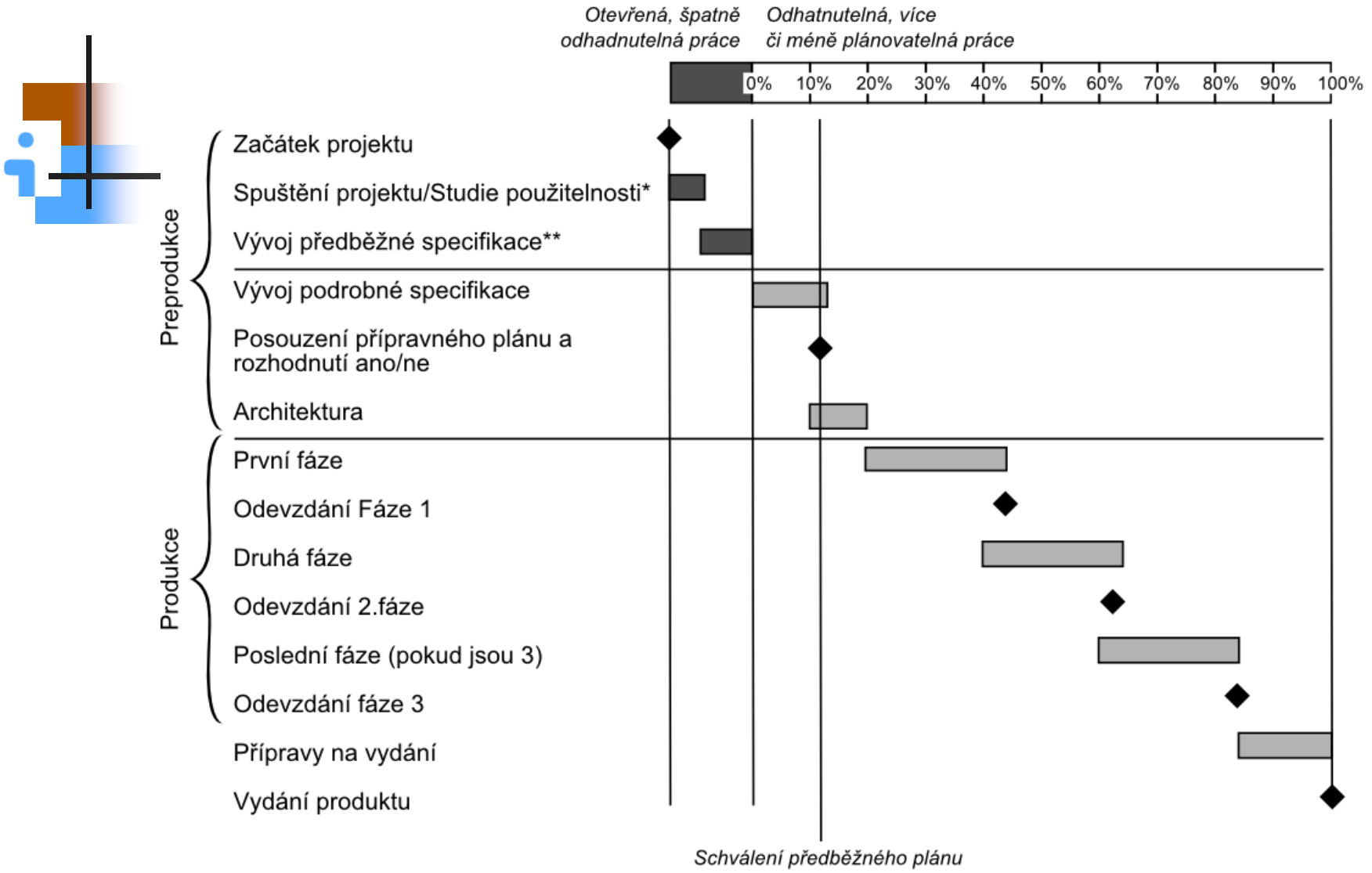






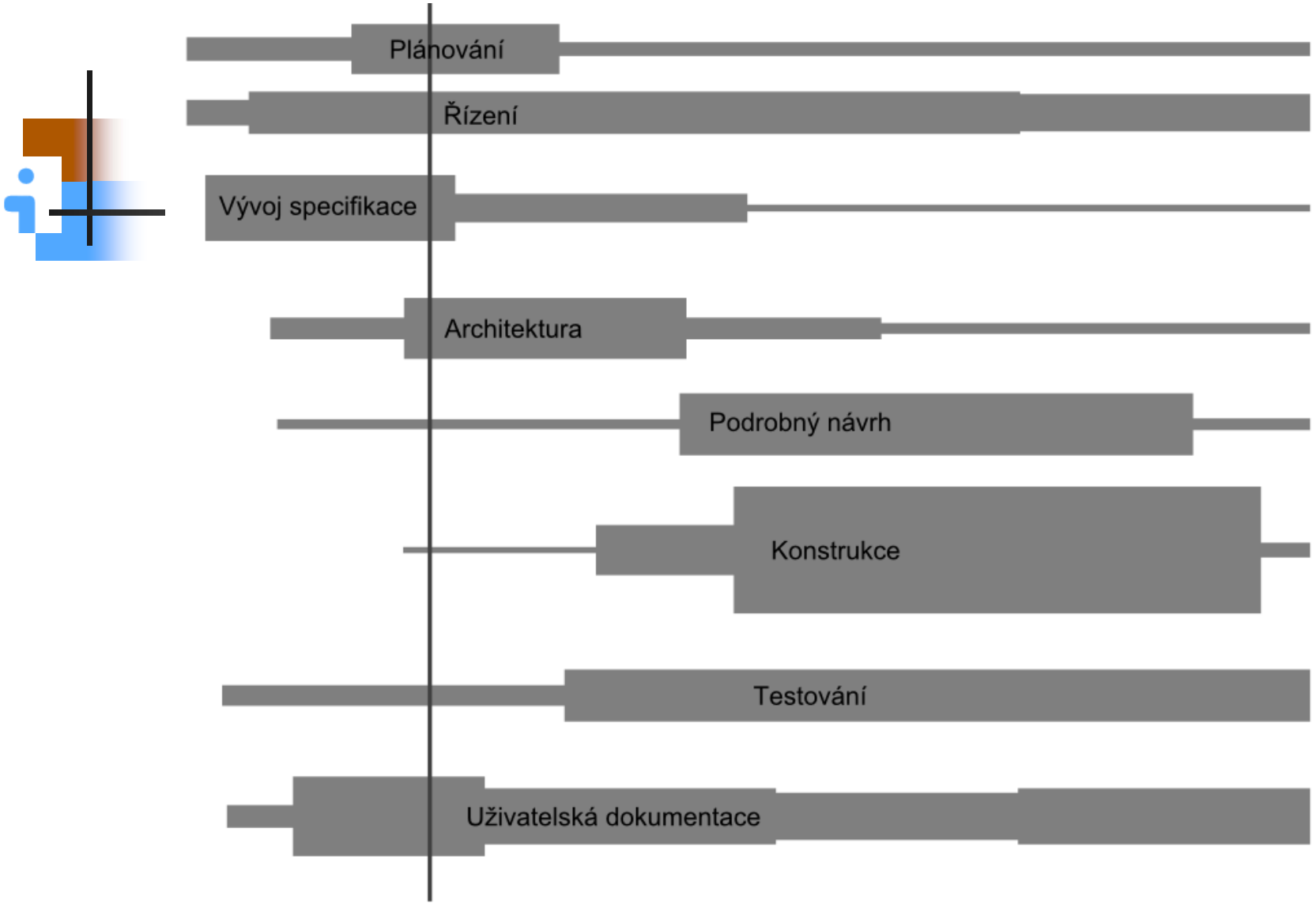
# Vývoj projektu

Softwarový projekt od začátku do konce




*Feasibility study

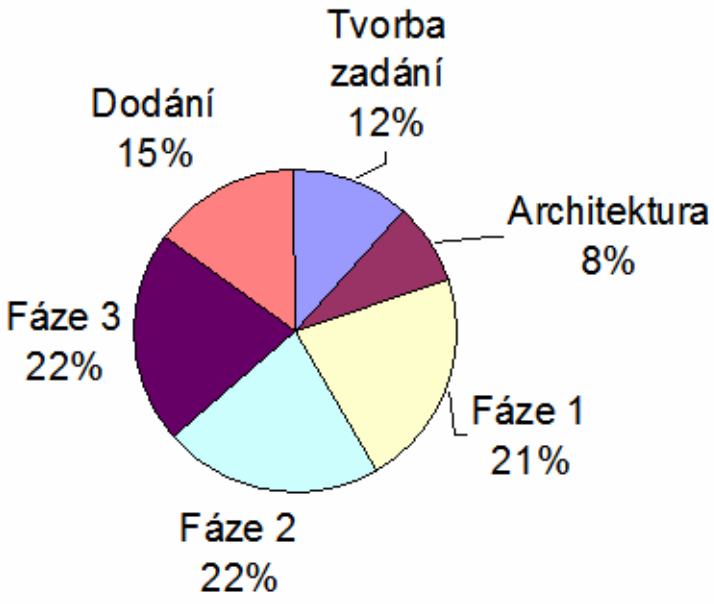
**Requirements development

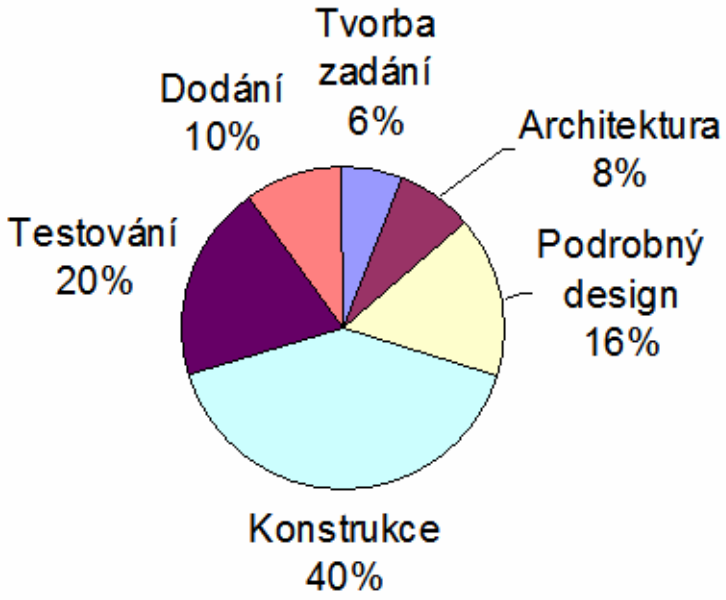



*Schválení předběžného plánu*

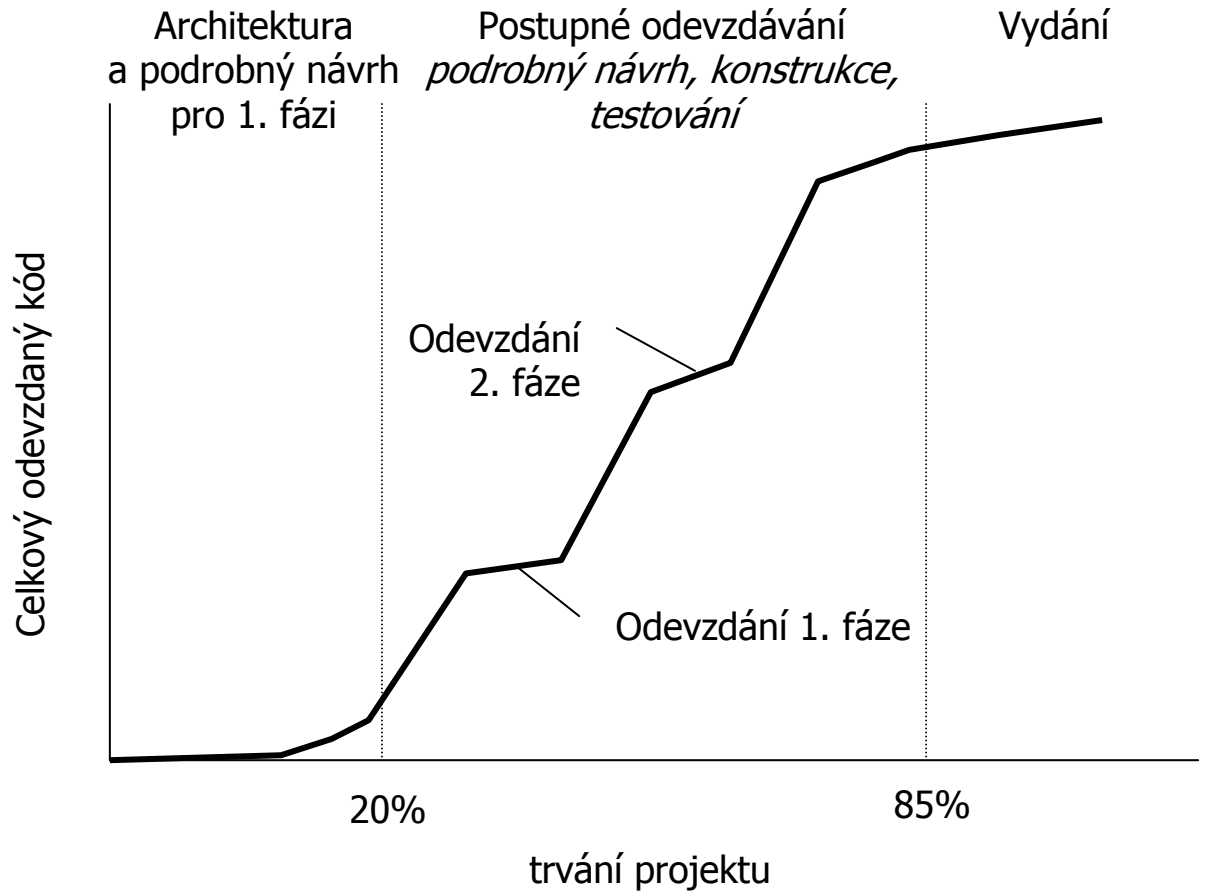
# Čas versus práce

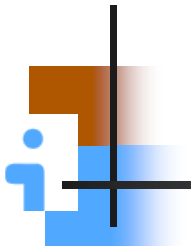
- V levém grafu je podíl různých fází na celkovém čase
- V pravém grafu podíl různých činností na celkové práci



# Nárůst kódu

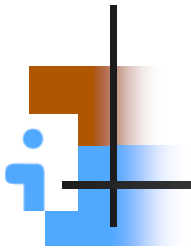

Architektura
a podrobný návrh
pro 1. fázi
Postupné odevzdávání
*podrobný návrh, konstrukce, testování*
Vydání
Celkový odevzdaný kód
Odevzdání
2. fáze
Odevzdání 1. fáze
20%
85%
trvání projektu



# Posouzení přípravného plánu*

- Jméno člověka s konečnou rozhodovací autoritou
- Vize projektu
- Obchodní rozvaha projektu
- Předběžné odhady objemu práce a termínů
- Seznam hlavních risik
- Definice stylu uživatelského rozhraní (Style guide)
- Podrobný prototyp UR
- Manuál/Specifikace zadání
- Podrobný plán vývoje softwaru

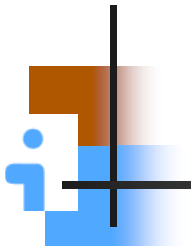
*Planning checkpoint review



# Hlavní milestony 1/5

- Spuštění projektu
  - Rozhodovací autorita
  - Vize
  - Obchodní koncept (business case)
  - Předběžné odhady rozsahu a trvání
  - Tým 2-3 vývojářů
  - Základní plány a dokumenty

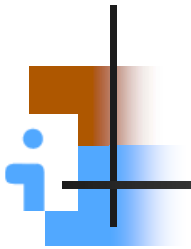
*Práce v této části projektu nemá pevný rámec. Zahrnuje, mimo jiné, i odhad toho, jak velký projekt vlastně je. Následkem toho se rozsah i trvání této části velmi liší projekt od projektu.*



# Hlavní milestony 2/5

- Vývoj předběžné specifikace
  - Hlavní tester a tvůrce dokumentace pracují
  - Jednoduchý prototyp uživatelského rozhraní
  - Odhady času s přesností na +100%, -50%
  - Předběžný plán vývoje softwaru
  - Aktualizace

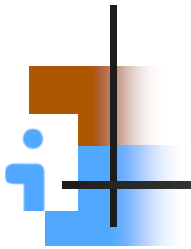
*Rozsah i trvání této části se také velmi liší projekt od projektu*



# Hlavní milestony 3/5

- Vývoj podrobné specifikace
  - Detailní prototyp uživatelského rozhraní
  - Manuál/Specifikace zadání
  - Plán testování a plán vývoje softwaru
  - Odhady času s přesností na +75%, -45%
  - Aktualizace
- Rozhodnutí ano/ne

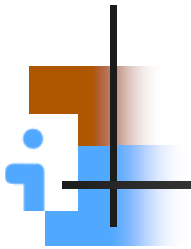
  *V tomto okamžiku má projekt za sebou přibližně 12% celkového času a 6% celkové práce. Tato procenta nezahrnují čas a práci vynaloženou na spuštění projektu a vývoj předběžné specifikace.*



# Hlavní milestony 4/5

- Architektura
  - Většina vývojářů a testerů pracuje
  - Softwarová architektura
  - Plán postupného dokončování
  - Testovací případy pro 1. fázi
  - Odhady času s přesností na +40%, -30%
  - Aktualizace

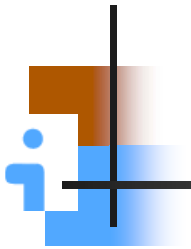
*V tomto okamžiku má projekt za sebou přibližně 20% celkového času a 14% celkové práce.*



# Hlavní milestony 5/5

- První fáze
  - Všichni vývojáři a testeři pracují
  - Podrobný návrh a plán konstrukce pro 1. fázi
  - Testovací případy pro 2. fázi
  - Zdrojový kód pro 1. fázi
  - Kompletní (z hlediska vlastností) produkt 1. fáze
  - Odhady času s přesností na +30%, -20%
  - Aktualizace

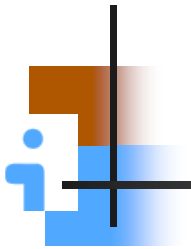
*V tomto okamžiku má projekt za sebou přibližně 45% celkového času a 40% celkové práce (za předpokladu, že jsou tři fáze).*



# Předběžné plánování

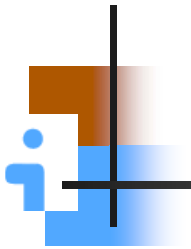
- Planování určuje:
  - Vize
  - Rozhodovací pravomoc*
  - Hrubý rozsah projektu
  - Strategie zviditelnění projektu
  - Řízení rizik
  - Personální strategie
  - Evidence času
- Na základě těchto úvah se vytvoří plán vývoje

*Executive sponsorship



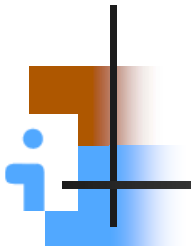
# Předběžné plánování

- Vize
  - Krátká a výstižná charakteristika projektu: oč nám jde, jaký produkt vytváříme
  - Realistická a současně motivující
- Rozhodovací pravomoc
  - Ideálně jeden konkrétní člověk, ale může to být i skupina lidí, např. Výbor pro kontrolu změn
  - Rozhoduje o rozsahu, podobě a harmonogramu projektu
  - Rozhoduje, jestli je možné projekt vydat



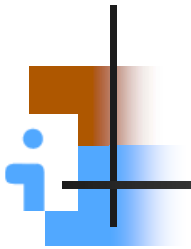
# Odhady času

- Metody:
  - Sofistikovaná metodika:
    - COCOMO, KSLOC, function points
    - PM = A * Size^B → 1 člověko-měsíc ≈ 1 KSLOC
  - Odhady autorů
    - Jednodušší a stejně spolehlivé
- Časté chyby
  - Člověko-dny nejsou pracovní dny
  - Ale pracovní dny jsou peníze



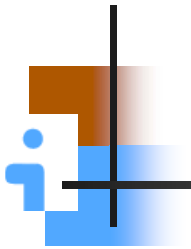
# Vývoj zadání

- Zadání neboli specifikace
  - Není stabilní, ale to neznamená, že neexistuje
  - Ideálně ho má zákazník – velký rozdíl mezi "opravdovým softwarem" a počítačovou hrou
- Počítačové hry
  - USP
  - Positioning
  - Brand



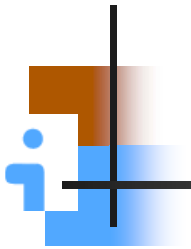
# Předběžné zadání

- Design dokument
  - Popis hry, herních mechanismů, příběh (pokud je), jak vypadá jedna hodina hry
  - Nemusí obsahovat seznam všech jednotek, levelů, předmětů atd., ani přesné vzorce
- Prototyp
  - Neobsahuje finální grafiku, možná vůbec žádnou grafiku
  - Demonstruje klíčové nové prvky hry nebo technologie
  - Pro interní potřebu



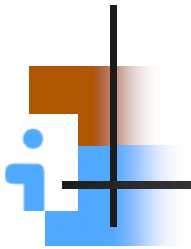
# Podrobné zadání

- Interface
  - Všechny obrazovky hry/aplikace
  - Nemusí mít žádnou funkčnost, může být i na papíře
- Kompletní zadání
  - Pro grafiky a designéry:
    - kompletní soupis herních entit a technologické nároky
    - workflow
  - Pro programátory:
    - kompletní popis herních mechanismů, vzorečky



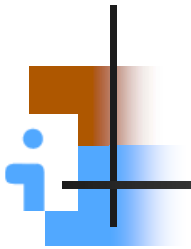
# Architektura

- Architekt
  - Není to vedoucí projektu ani hlavní designér
  - I u středně velkých projektů je to tým
- Požadavky
  - Najednou, nikoli ve fázích
  - Korespondence s požadavky
  - Podpora postupného odevzdávání
- Dobrá architektura
  - Jednoduchá a zřejmá
  - Homogenní
  - Stavový automat aplikace



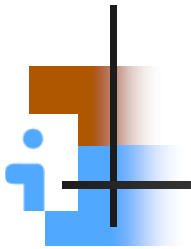
# Preprodukce

- Prototyp/Design dokument
  - Nahrazuje specifikaci od uživatele
  - Demonstruje základní gameplay
  - Typicky je to vzorový level
- Testovací aplikace
  - Prokazuje schopnost pracovat na dané platformě
- Nástroje a Workflow
  - Editory, pluginy do komerčních editorů
- Kompletní zadání pro grafiku a pro programátory



# Průběh fáze/inkrementu

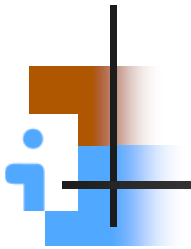
- Součásti
  - Podrobný návrh kódu
  - Konstrukce kódu
  - Tvorba testovacích případů
  - Odborné posouzení
    - součást Podrobného návrhu i Konstrukce
  - Oprava chyb
  - Integrace a vydání
  - Úklid a dokončení
- Souběžně se provádí
  - Aktualizace specifikace
  - Aktualizace uživatelské dokumentace
  - Technická koordinace
  - Řízení rizik
  - Sledování projektu
    - Minimilestony



# Minimilestony

*"Jak se projekt zpozdí o rok? Pokaždé o další den." – Frederick P. Brooks, Mythical Man-Month*

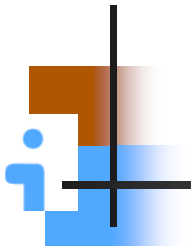
- Krátkodobé úkoly nebo cíle
  - Nejdéle jeden za týden
  - Nejméně jeden za den
- Jsou binární: buď jsou hotové nebo ne
- Definují se dvakrát během fáze
  - Pro Podrobný návrh
  - Po jeho skončení až do konce fáze
- Během fáze by nikdo neměl dělat na ničem, co není minimilestone



# Podrobný návrh

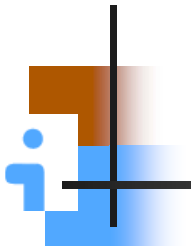
- Podrobné rozpracování programu na úrovni metod
- Nedílné součásti:
  - Odborné posouzení*
  - Plán konstrukce – minimilestony
- Formálnost
  - Velmi formální
    - diagram pro každou třídu, všechny funkce v pseudokódu
  - Neformální:
    - design postupuje paralelně s konstrukcí
    - většina kódu se rovnou píše 'načisto'

*Technical review



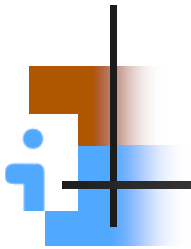
# Podrobný návrh – posouzení

- Posuzují
  - 2-3 vývojáři
  - Odděleně, společná schůzka není nezbytná
- Chyby
  - *špatná funkce*, nedělá to, co by mělo
  - *neúplnost*, nepokrývá to všechny případy
  - *nesrozumitelnost*, nikdo kromě autora to nechápe
  - *neodpovídá zadání*, neplní požadavky zadání
  - *přesahuje zadání*, dělá něco, co zadání nepožaduje



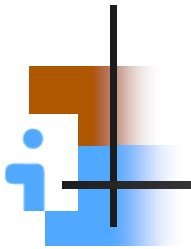
# Konstrukce

- Relativně snadná a předvídatelná
- Příležitost ke zjednodušení projektu
- Díky minimilestonům se dá snadno pozorovat průběh práce
- Příliš pozdě na implementaci změn
- Dodržovat 'code standard'
  - To je víc, než kde se píší závorky



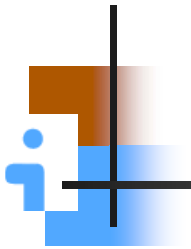
# Co je správně?

```
//varianta a
if (x == 1) {
      a = 0;
}
a++;

//varianta b
if (x == 1)
{
      a = 0;
}
a++;

//varianta c
if (x == 1)
      {
      a = 0;
      }
a++;
```



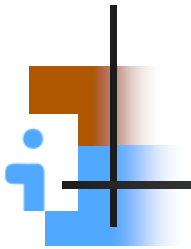
# Code standard

- Layout tříd, metod, atd.
- Komentáře
- Jména proměnných
- Jména funkcí
- Jména funkcí typu Get a Set
- Maximální délka metody
- Maximální počet metod ve třídě
- Maximální stupeň složitosti
- Verse nástrojů a knihoven
- Dodržování arch. standardů
  - přístup k paměti
  - zacházení s chybami
  - přístup ke stringům, atd.
- Konvence pro soubory
  - jména
  - počet tříd v souboru
  - hlavičkové soubory
- Způsob, jak se označuje nedodělaný kód (např. //TBD)



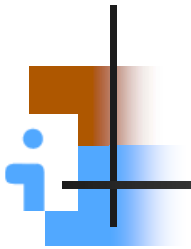
# Integrace

- Postup integrace
  - Vývojář si přeloží celý projekt u sebe a vyzkouší ho
  - Pak poskytne kód k posouzení
  - Současně vytvoří (nebo nechá vytvořit) testovací případy pro nový kód
  - Vývojář opraví chyby, které se objevily při posouzení
  - Vývojář znovu zbuilduje celý projekt u sebe a vyzkouší ho
  - Pak teprve může komitnout nový kód do hlavní repository
- Denní build a automatické testy
  - Projekt se každý den zbuilduje
  - Provede se sada automatických testů



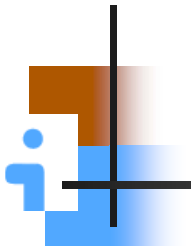
# Konstrukce v první fázi

- Vyvíjí se kostra aplikace:
  - Uživatelské rozhraní
    - zatím nic nedělá, jsou to jen menu a tlačítka
  - Infrastruktura
    - přístup k paměti
    - ošetřování vyjímek, atd.
- Nepodlehnout pokušení vybudovat infrastrukturu kompletní
  - Jen tolik, co potřebuje první fáze



# Testování a vydání

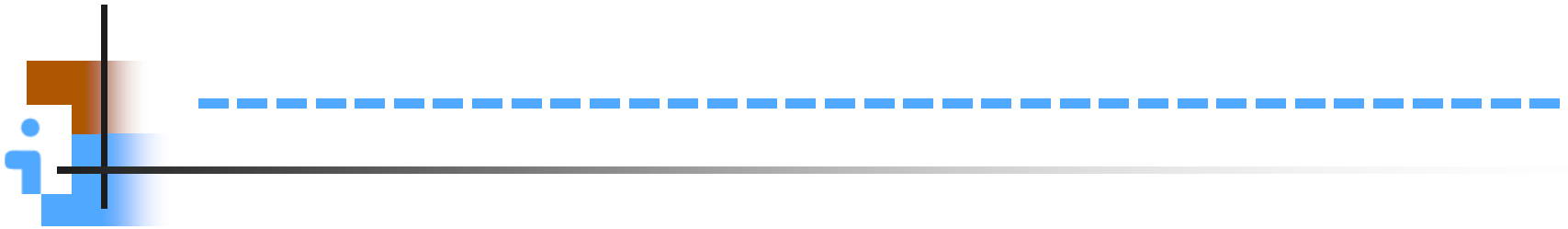
- Quality assurance je víc než testování
- Ve fázi spustitelného kódu je odstraňování chyb nejdražší
  - Testeři jsou až ta poslední možnost jak najít chybu
  - Závislost na testerech je nezdravá
- Na konci fáze
  - Vyhodnocení minulé fáze
  - Aktualizace předpokladů a odhadů
  - Záznam okamžitého stavu

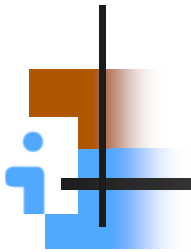


# Konec projektu

*"Držet tištěnou, svázanou knížku s historií projektu dá člověku pocit, že něco dokončil, a každý člen týmu by měl dostat svůj výtisk." – Steve McConell*

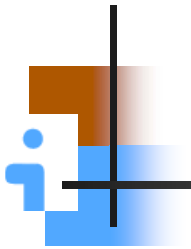
- Debriefing týmu
- Sepsání historie projektu
  - Objektivní dokumenty
    - Záznamy o času
    - Plnění miletonů, atd.
  - Subjektivní doklady
    - Názory jednotlivých členů týmu
    - Názory klienta a/nebo rozhodovacího týmu





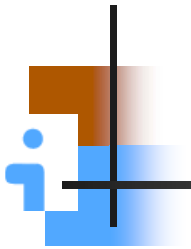
# Programování

Práce v týmu, rozdělení rolí, práce na větším projektu



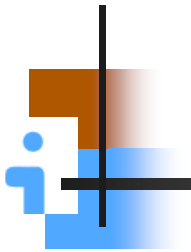
# Předpoklady

- Projekt má ~6 programátorů
- Jazyk projektu je C++
- Trvání projektu je ~18 měsíců
- Vývoj pro Windows



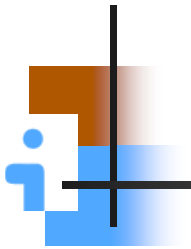
# Obsah

- Nástroje pro týmovou práci
- Middleware
- Skriptovací jazyky
- Specifika většího projektu



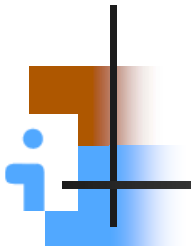
# Nástroje pro týmový vývoj

Co používají profesionální týmy



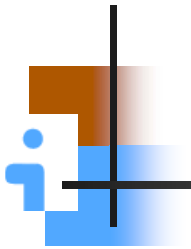
# Nástroje

- IDE
- Compiler a linker
- Profiler
- Správa verzí (Version control)
- Správa dat (Asset management)
- Bugtracker
- Dokumentace
- Wiki



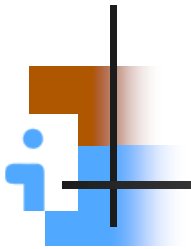
# IDE

- Editor, compiler, debugger
- MS Visual Studio
  - nejrozšířenější
  - užitečné pluginy
  - nejlepší debugger
  - používá se i pro Xbox i pro PS3
  - na druhou stranu podporuje závislost na kompilaci
- Linuxové editory
- Ostatní (Eclipse, Bloodshed Dev-C++)



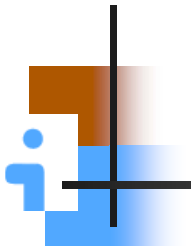
# Profiling a tuning

- Měření rychlosti
- Kontrola paměti
- Komerční projekty
  - VTune, Gprof
  - Memory Validator, BoundsChecker, Valgrind
- Nejlepší je vlastní
  - Profiler: Sean Barrett, Game Developer 08/04
  - Vlastní memory manager



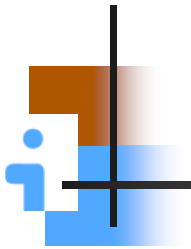
# Správa verzí

- Řešení
  - CVS, Subversion, SourceSafe
- Aspekty
  - zamykání souborů (check in/out)
  - podpora binárních dat
  - srovnání lokálně nebo na serveru
  - databáze nebo soubory



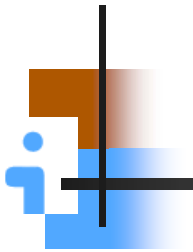
# Správa dat

- Data (assety)
  - Modely, textury, zvuky, animace, skripty
  - Zdrojová data
    - 3DS Max, Photoshop, Sound Forge, ...
  - Nativní data
    - 'čitelná' pro používaný engine
    - nedají se přímo editovat
- Speciální, drahé systémy
  - Alien Brain, Perforce
- Systémy správy verzí



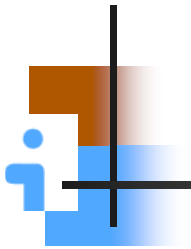
# Ostatní

- Bugtracker
  - Mantis, Bugzilla, Trac
- Dokumentace
  - Doxygen, Sandcastle
- Wiki
  - Twiki



# Middleware a smrt algoritmů

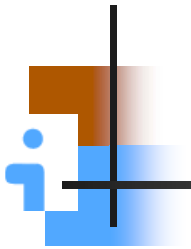
Všechno už někdo naprogramoval



# Co je to middleware

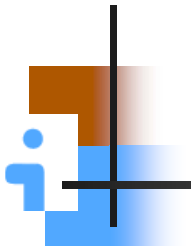
- Cizí kód v naší aplikaci
  - Výhody
    - Rychlejší a levnější
    - Spolehlivější
    - Podpora
  - Nevýhody
    - Dražší úpravy a údržba
    - Záludné chyby
    - Závislost na 'cizích'
- Klasifikace
  - Podle stupně integrace (od zdrojáků po OS)
  - Podle účelu (fyzika, AI, ...)

*Middleware je outsourcing pro programátory*



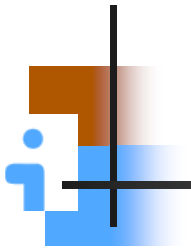
# Middleware reloaded

- Pole vlastního programování se zmenšuje...
  - Abstrakce hardwaru
  - Delegace odpovědnosti
    - toto je opravdový 'middleware'
  - Integrace
- ...a současně prohlubuje
  - Zvětšuje se podíl tvůrčí práce
  - Komplexnější mechanismy, sofistikovanější světy
- Totální middleware?



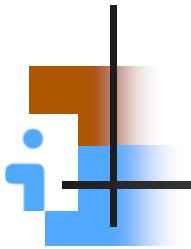
# Fyzikální middleware 1/2

- Využití fyziky
  - ragdoll, animace smrti
  - ničení objektů, což obnáší další zásadní problémy
  - animace vody a kapalin
  - brodění se krabicemi
- Kritéria pro hodnocení fyz. mw
  - rychlost, přesnost (stabilita), API
  - rozsah – tuhá tělesa, kapaliny, ragdoll atd.



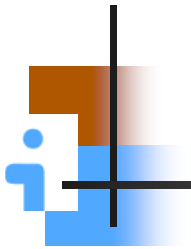
# Fyzikální middleware 2/2

- Problémy s fyzikou
  - HW akcelerace dává výsledky z minulého snímku, navíc vyžaduje dvojí data
  - Musí se simulovat kontinuálně i na místech, které hráč nevidí
  - Často je jen fyzika pro fyziku, herní význam je nejasný nebo žádný
- Není jasné, zda je to spíše grafický nebo spíše designový efekt



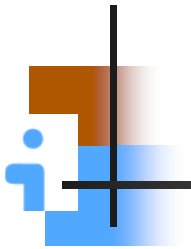
# Fyzikální mw – příklady

- Havoc
  - drahý
  - legacy API
- Ageia
  - není Meqon
  - HW akcelerace
- ODE
  - takřka použitelné
  - nemá sweep test (CCD)
- Bullet
  - původně z Blenderu
  - má CCD a používá ODE solver



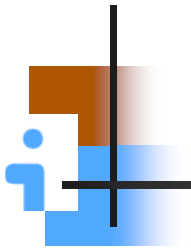
# AI middleware

- Path-finding, steering, konečné automaty
  - PathEngine
- Komerční = drahé
  - AI.Implant
  - Kynapse
  - X-AItment (i neuronové sítě)
- Akademické
  - FEAR
  - ScriptEase (skriptování RPG)
- IGDA AI standardy (výsledky sporadické)



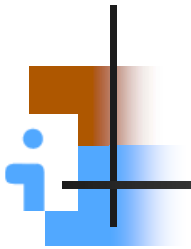
# Engine jako middleware 1/3

- Co je to engine
  - Správa scény
    - Hierarchie objektů, světel, kamer, render targetů
    - Viditelnost, kolize
    - Materiály, shadery
  - Řešení osvětlení a stínů
  - Animační systém
  - Podpora skriptování
  - Formát dat
  - …



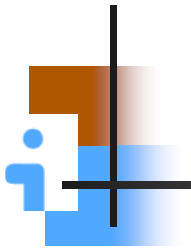
# Engine jako middleware 2/3

- Všechna studia v ČR používají vlastní engine
- Proč mít vlastní engine:
  - je to naše (cena)
  - je to naše (vyznáme se v tom a dokážeme to upravit a vylepšovat)
  - je to naše (nejvhodnější pro náš typ hry)
- Vývoj vlastního enginu trvá dlouho
  - 2 – 4 roky
  - 1 – 6 programátorů



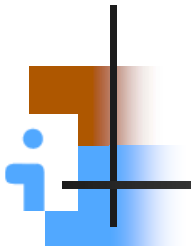
# Engine jako middleware 3/3

- Kritéria pro výběr enginu
  - Komerční vs. OS
  - Workflow a nástroje pro něj
  - API
  - Podpora skriptování (integrace s herním kódem)
  - Multiplatformnost
  - Výkon



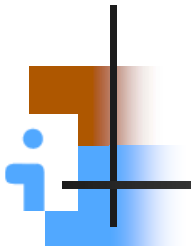
# Engine mw – příklady

- Komerční enginy
  - Doom3, Half Life, Unreal…
- OS enginy
  - Ogre: rozumný kompromis mezi složitostí a funkčností
  - Irrlicht: hezký menší, ale funkční
  - CrystalSpace: příliš velký pro své vlastní dobro
  - XEngine, atd…



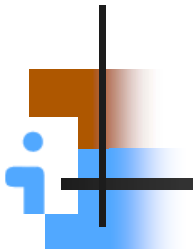
# Ostatní middlewarek

- Audio a Video
  - Bink/Miles
  - Ogg/Theora/Vorbis/FLAC
  - OpenAL
- Grafika
  - FreeImage
  - FreeType
- Ostatní
  - zLib



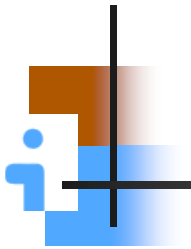
# Jak zvolit middleware?

- Čím blíže standardu, tím lépe
  - Stabilní, neměnná specifikace
  - Popularita, tj. počet uživatelů
- Opatrně na závislost na proprietárním řešení
  - Open Source není automaticky správná odpověď
- Postupujte racionálně:
  - Nebojte se cizího kódu
  - Nebojte se delegovat zodpovědnost



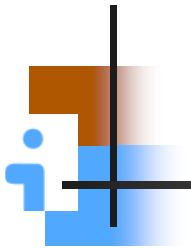
# Skriptovací jazyky

V čem se píší hry



# Skriptování obecně 1/3

- Překlad do bajtkódu
  - Virtuální procesor
    - může se zrychlit generátorem assembleru podle platformy
    - výhoda virtuálního procesoru je vlastní scheduling, správa stacků
  - Překladač je součástí jazyka
    - Pro vlastní jazyk lze použít OS lexer a parser (lex, yacc, bison)

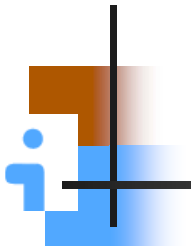


# Skriptování obecně 2/3

- Multitasking
  - Kooperativní – coroutines (Lua)

```
int sound_handle = GetBigFish().Say("thing1");
if(sound_handle > 0)
  coroutine.yield("sound_handle", sound_handle);
```

    - jednodušší pro programátory, zodpovědnější pro skriptaře
  - Preemptivní
    - Vlastní thread manager
    - Těžší naprogramovat, skriptování není jednodušší
    - Aplikace je odolnější



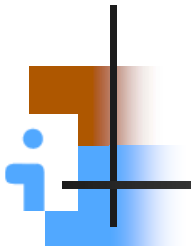
# Skriptování obecně 3/3

- Propojení s aplikací
  - Voláním nativního kódu aplikace
    - jména funkcí se při překladu vyhledají v tabulce

```
if (!IsInArea(soldier1,area1) {
  SetCommandMoveToArea(soldier1,area1);
  yield(1);
}
...
```

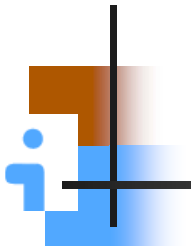
  - Voláním eventů
    - pro určité eventy se zaregistrují funkce, které se mají volat

```
SetCommandMoveToArea(soldier1,area1);
OnEvent UnitInArea(soldier1,area1) {
  ...
}
```



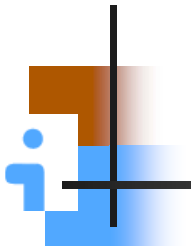
# Skriptovací příklady 1/2

- Lua
  - Jednoduchá syntaxe
  - Není debugger
  - Rozporuplná rychlost
- Python
  - Nemáme praktické zkušenosti
  - Expresivnější syntaxe
  - Prý pomalejší než Lua



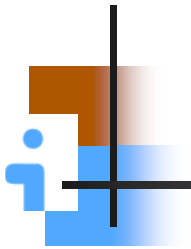
# Skriptovací příklady 2/2

- AngelScript, GameMonkey
  - OpenSource
  - Integrovatelné do C++, C++ podobná syntaxe
  - Nemáme praktické zkušenosti
- Unreal script, Enforce script
  - (Enforce engine = Black Element Studio)
  - 2,5 MB zdrojáků pro ES se zkompiluje za 0,5s
  - C++ podobná syntaxe
  - Rychlejší interpretace



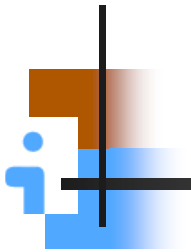
# Výhody skriptování 1/2

- Bezpečnější
  - runtime kontrola herního kódu
  - kontrola všech operací
- Jednodušší ukládání hry
  - ukládají se všechny objekty skriptu
  - handle pointer (místo C-pointeru)
- Debugování
  - v případě pádu je kompletní informace o stacku, stavu atd. i v release verzi



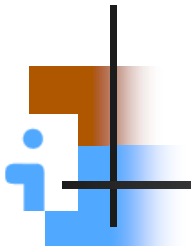
# Výhody skriptování 2/2

- Čistší design, rychlejší vývoj
  - herní programátoři jsou oddělení od enginu
  - hru není nutné kompilovat a linkovat s enginem
  - i v releasu jsou dostupné všechny debugovací informace
  - viz kapitola o middlewaru
- Jednodušší a lepší multitasking
  - virtuální stroj může skript libovolně přerušit a pouštět jiné thready aplikace



# Specifika většího projektu

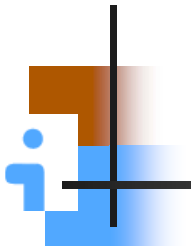
Velikost je rozdíl kvality



# Specifické oblasti

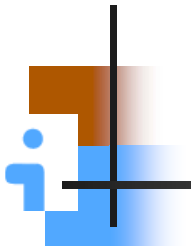
Povšechně jde o zcela subjektivní výběr okruhů a problémů

- Psání kódu
- Projekt a projekt v MSVC
- Stavový automat
- Memory manager
- Ostatní



# Psaní kódu

- Lidé si musí být schopni číst navzájem kód
  - viz Posuzování v kapitole o řízení projektu
  - dodržování Coding standardu
  - sdílení a šíření informací
  - Odhalování chyb
- Wrappery kolem systémových funkcí
  - systémové funkce (např. strcpy) nepoužívat přímo
  - zjednodušuje portování
- Standardní makra
  - délka pole, preprocesorová makra, aserty, profiler

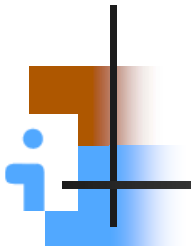


# Projekt jako projekt v MSVC

- Čas na zbuildování enginu

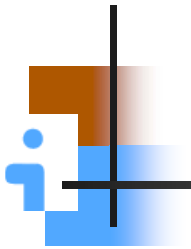
| | | |
|---|---|---|
| BES | ~5 min | pořádek v includech |
| Mindware | ~5 min | předkompilované headery |
| ALTAR | ~45 min | Incredibuild |
| IS | ~90 min | nic |

- Rozdělení na *.lib
  - Nefunguje dobře, nesmí se přehánět
  - Umožňuje např. jinou utility library pro nástroje a pro engine



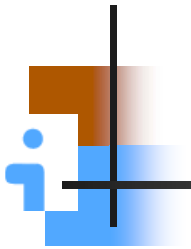
# Hlavičkové soubory

- Problém:
  - Jednoprůchodový překladač → forward deklarace → olbřímní textový soubor
  - Deklarace třídy na jednom místě → private funkce a proměnné ve stejném souboru jako public → změna private části vynutí kompilace všude



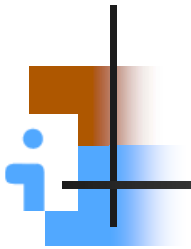
# Jak includovat

- Includy v *.h souborech?
  - Vysoce praktické
  - Includovat jen to, co skutečně potřebuješ
  - Je lepší mít .h menší, i když bude includovat jiná .h
- Jiné tipy
  - Jeden class může být implementován ve více souborech
    - přepínání mentálního kontextu programátora je drahé
    - může používat jiná .h
  - Include guard vs. #pragma once



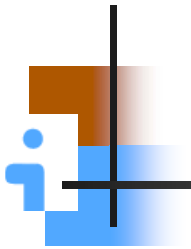
# Propojení .h ↔ .cpp

- Pure virtual interface
  - Vhodné jen pro 3rd party knihovny
- V public .h používat jen forward deklarace
- Předkompilované headery



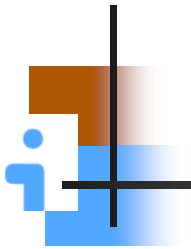
# Interface přes pure virtual třídu

```
//CMyFunc_ifc.h -- START
class CMyFunc_ifc
{
    public:
        virtual double Sin(double x) = 0;
};

CMyFunc_ifc* MakeFunc();
//CMyFunc_ifc.h -- END

//CMyFunc_imp.h -- START
class CMyFunc_imp: public CMyFunc_ifc
{
    public:
        double Sin(double x);
    private:
        double ThirdPower(double x);
};
```



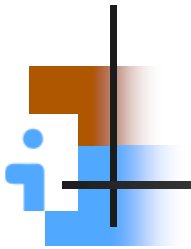
# Interface přes fwd. deklarace

```
//CMyFunc_ifc.h -- START
struct SBiVec;
class CGAlgebra
{
    public:
        SVec4 GetRotationAxis(double EulerAngX, EulerAngY, EulerAngZ);
    private:
        SBiVec MakeBiVec(SVec4 vec1, vec2);
};
//CMyFunc_ifc.h -- END

//CMyFunc_imp.h -- START
struct SBiVec
{
    double xy, yz, zx;
    SBiVec()
    {
        ...
    }
};
```



# Využití namespacu

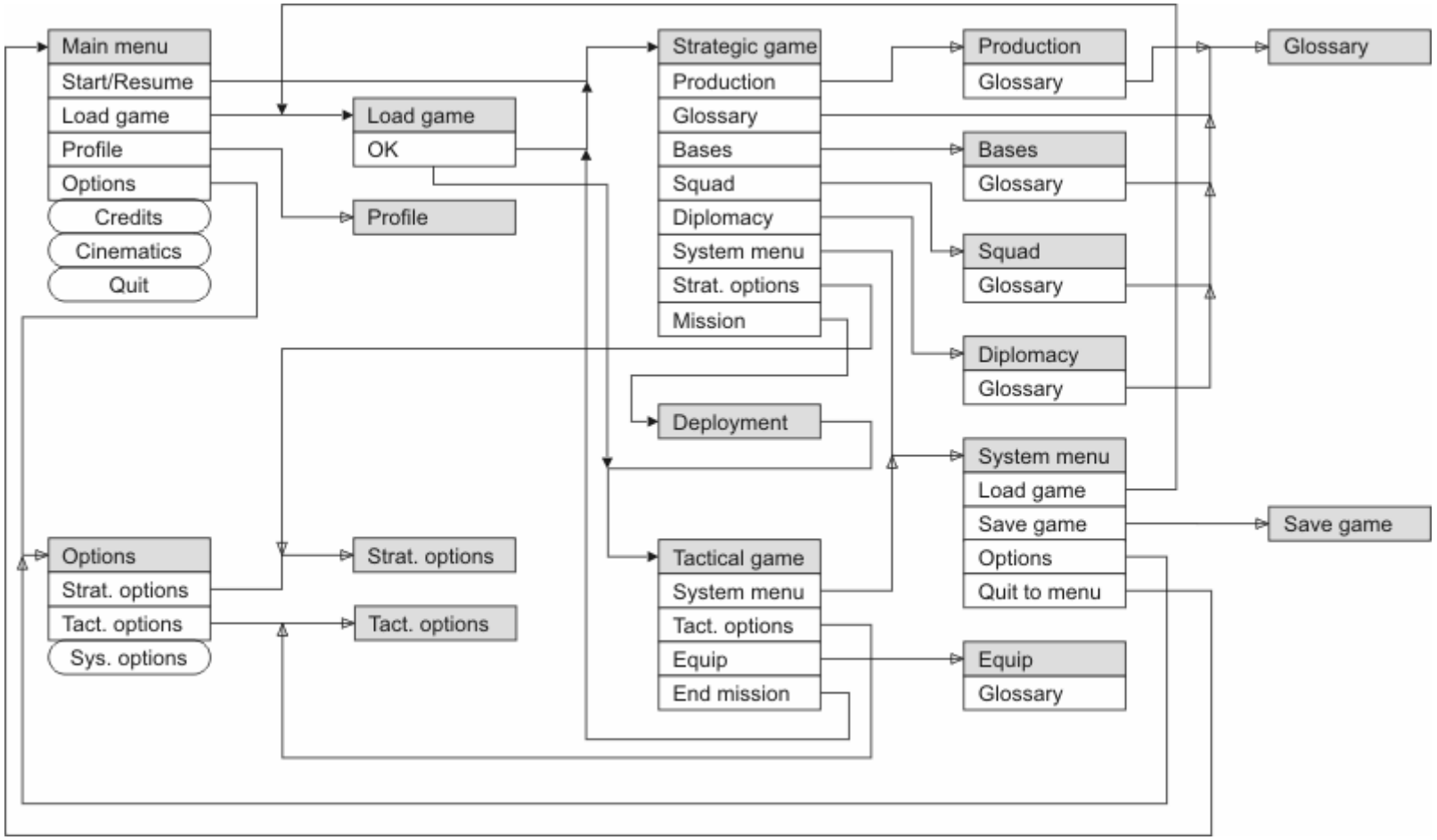
```
//MyFunc_ifc.h -- START
namespace MyFunc {
    double Sin(double x);
}
// MyFunc_ifc.h -- END

//MyFunc_imp.h -- START
namespace MyFunc {
    double ThirdPower(double x);
}
//MyFunc_imp.h -- END
```

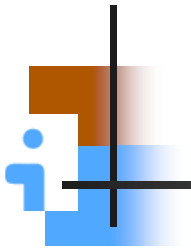


# Stavový automat

- Aplikace je vlastně stavový automat
- Na začátku projít možné stavy a definovat přechody
- Neřídit aplikaci okny ale okna aplikací
  - Stav aplikace by neměl záviset na tom, které okno je zrovna nahoře
- Jednodušeji se rozšiřuje a adaptuje na zvláštní situace

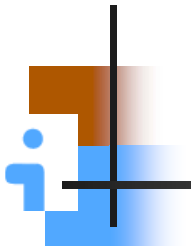
# Stavový automat – příklad





# Memory manager

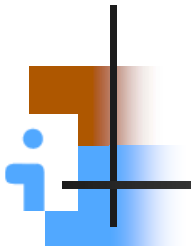
- Několik paradoxů:
  - Konzole a Windows: jiná motivace, stejný výsledek
  - Zásadní rozhodnutí, 'neviditelné' důsledky
  - Nemoderní, ale populární
- Výhody vlastního memory manageru:
  - Je lepší alokovat staticky
  - Vlastní 'sesypávání' paměti
  - Možnost plnit paměť vlastním vzorkem
  - Kontrola memory leaků



# Ostatní.ostatní

- Dodržování const-antnost u metod a funkcí
- Používání assertů
- Pečlivost při pojmenovávání proměnných
- Používání namespaců
- Omezení vnoření, limitování složitosti
- Plánování času na mergování kódu
- Optimalizovat přístup do paměti, ne assembler

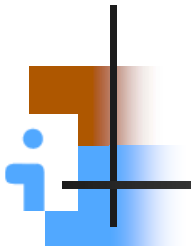
- Hodně genericky je dobře podle učebnice, ale všeho s mírou; všechno nemusí být objekt



# Const-ness v C++

```
class Person
{
   public:
      ...
      char* GetName() {       //kolikrát patří 'const' na tento řádek?
         return m_szName;
      };
   private:
      char* m_szName;
};
```

```
class Person
{
   public:
      ...
      const char* GetName() const {
         return m_szName;
      };
   private:
      char* m_szName;
};
```



# Programování v praxi

- Filip Doksanský
  - Black Element Software
- Jan Kratochvíl
  - Illusion softworks
- Otakar Nieder
  - ALTAR games
- Patrik Rak
  - Mindware studios