RUP - Motivace, principy

Jaroslav Žáček
jaroslav.zacek@osu.cz
http://www1.osu.cz/~zacek/infs1/

Tradiční vs. iterativní přístupy

Vodopádové principy	**Iterativní (agilní principy)**
Zaměřen na procesy, předpokládá jejich opakovatelnost.	Zaměřen na lidi – motivace, komunikace prvořadá.
Pevné, podrobné plány definovány na úvod, kdy je spousta nejasností.	Pro celý projekt pouze road map. Detailní plány jen iterace (kratší úseky, 2 měsíce).
Rizika jsou často překvapení, přináší problémy.	Řízen riziky – nejrizikovější věci řešíme nejdříve.
Integrace a testování až na konci.	Průběžná integrace a testování.
Změny nejsou vítány.	Počítá se změnami, přijímá je.
Často zaměřen na tvorbu dokumentů bez přidané hodnoty a jejich revize.	Zaměřen na fungující SW (hodnota pro zákazníka).
Buildy a testy až na konci, často přeskočeno nefunkční testování.	Automatizované buildy a testy.
Za kvalitu odpovědní pouze testeři, QA manažeři nebo často nikdo.	Všichni (celý tým) odpovědní za kvalitu produktu.

Výzkumy

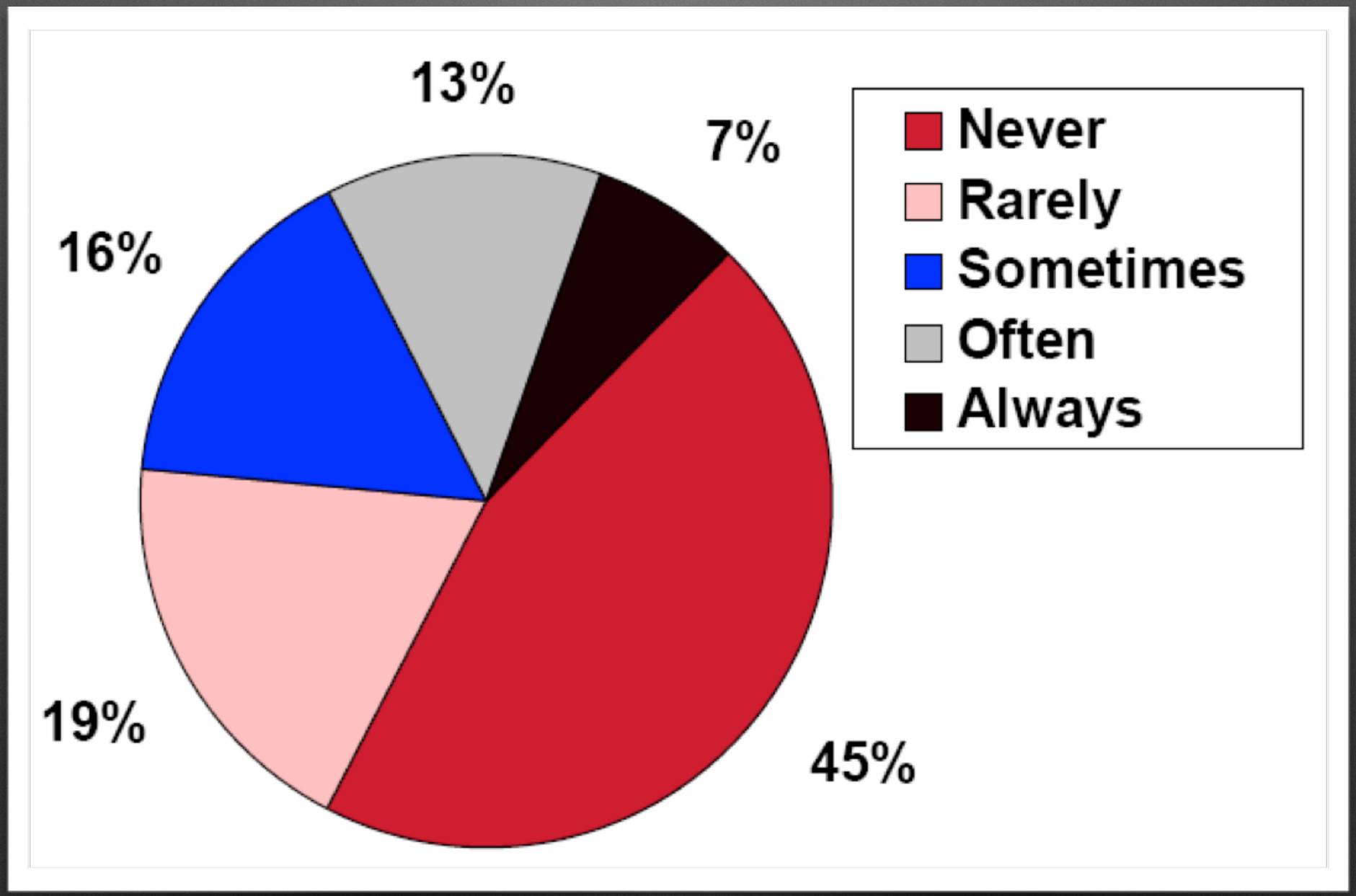
Zákaznické projekty vyvíjené na zelené louce pomocí tradičních metod

Size	Successful
Under $500K	68%
$501K - $3M	22%
$3M - $6M	9%
$6M - $10M	1%
Over $10M	0%
Total	100%

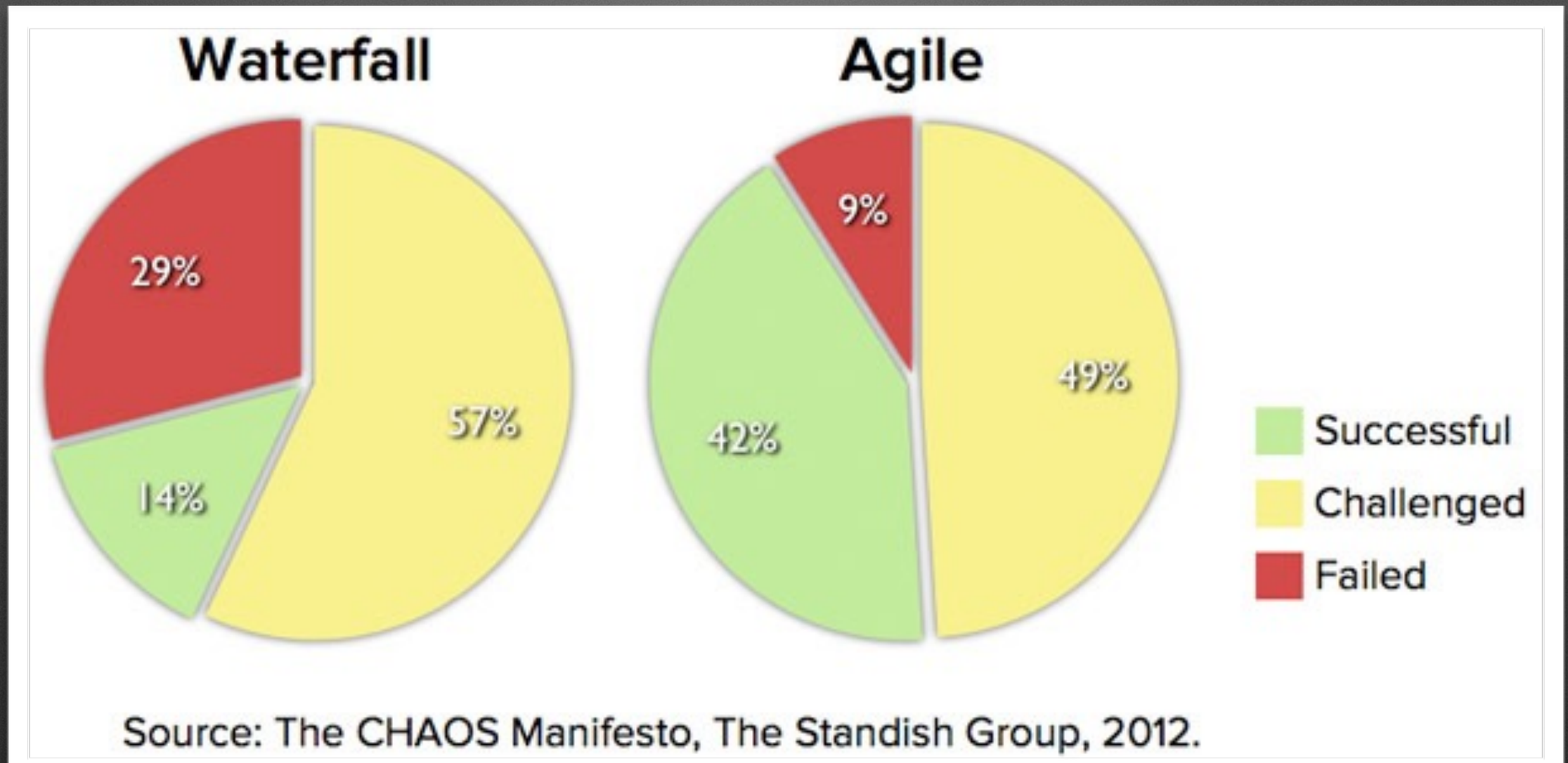
Úspěšnost vodopádového přístupu na velkých projektech?

- McKinsey - 17% velkých IT projektů skončí katastrofou
- Geneca - Velké IT projekty překračují rozpočet o 45%, doručí polovinu toho, co zákazník původně chtěl.
- 75% účastníku na projektu v něj nemá důvěru

Výzkum Standish



Výzkum Standish



Výzkum Standish

Table 1

Standish project benchmarks over the years

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	**100%**	**100%**	**100%**

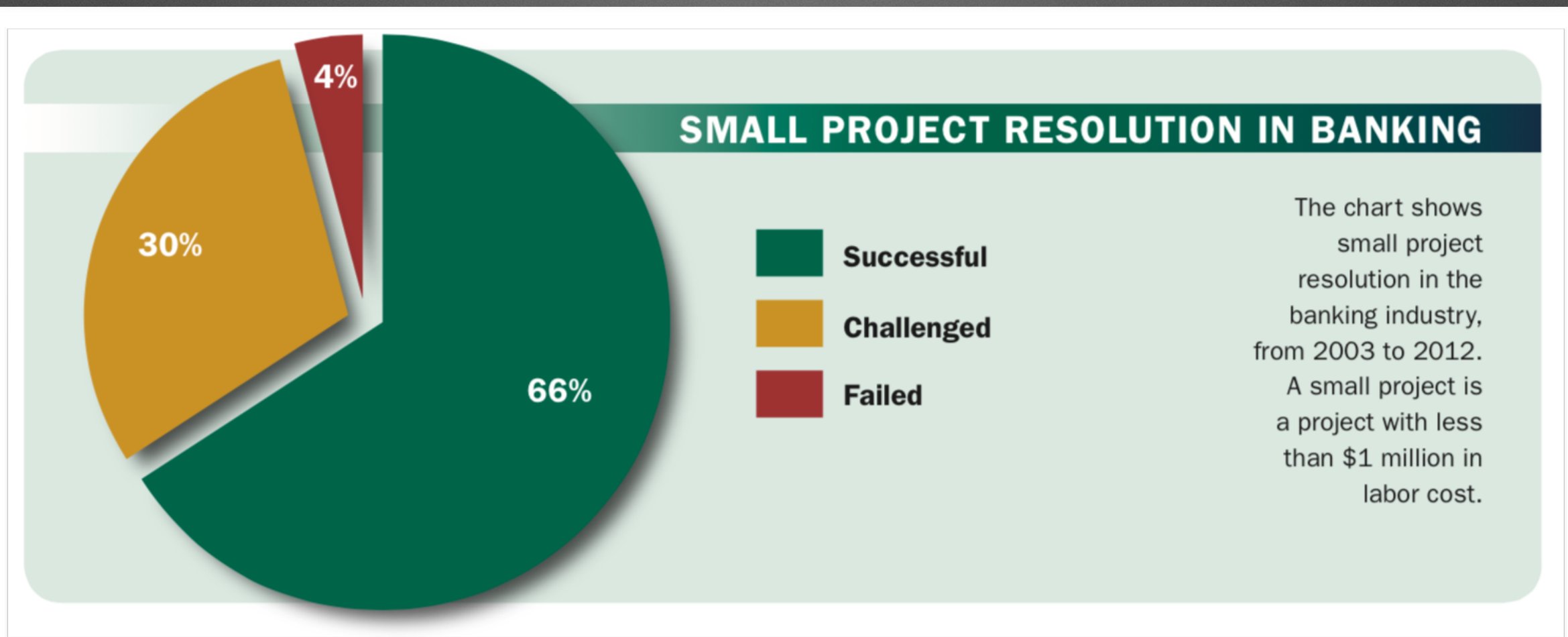
The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

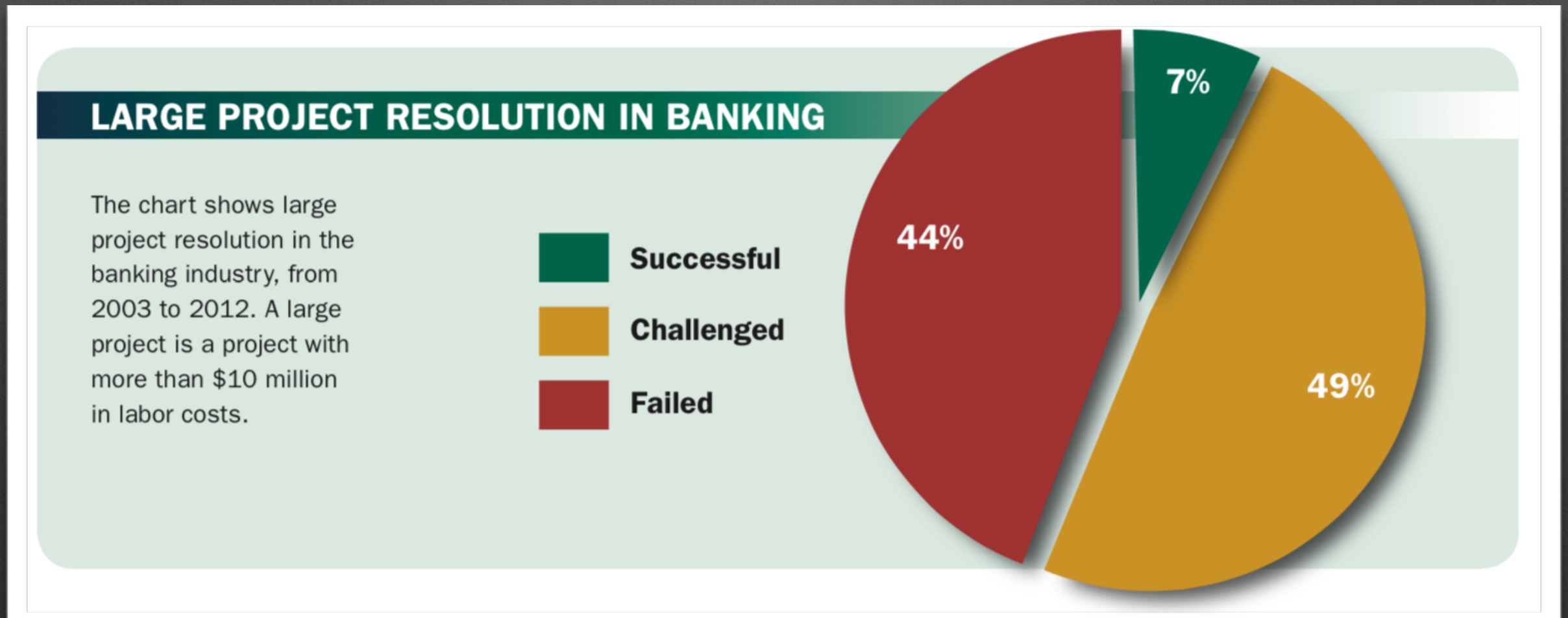
SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000

Malé projekty jsou vcelku úspěšné



U větších už je to horší…



Unified Process

- Agile Unified Process
- Basic Unified Process
- Enterprise Unified Process
- Essential Unified Process
- Rational Unified Process
- Oracle Unified Method
- RUP-Systems Engineering

Co je RUP

- RUP - Rational Unified Process
- Původně Rational, od roku 2002 pod značkou IBM
- Produkt - procesní framework
    - Webová aplikace
    - Rational Method Composer (RMC)

Historie

- Vychází ze spirálového modelu (B. Boehm) a z Objectory (I. Jacobson) =>Rational Objectory Process
- 1998 - první verze Rational Unified Process, architekt P. Kruchten
- Aktuální verze 7.5.3 (2018) (Rational Method Composer)
- Open-source verze EPF (součástí je OpenUP)

Klíčové aspekty

- Risk driven
    - Risk management je součásti procesu vývoje
- Use-Case driven
    - Jsou podkladem pro vývojový proces
    - Vyjadřují požadavky na systém a zároveň přidanou hodnotu pro zákazníka (byznys)
- Architecture-centric design
    - Architektura je základní kámen pro konceptualizaci systému
    - Skládá se z více koordinovaných pohledů (modelů)

Rozdíl mezi RUP a OpenUP

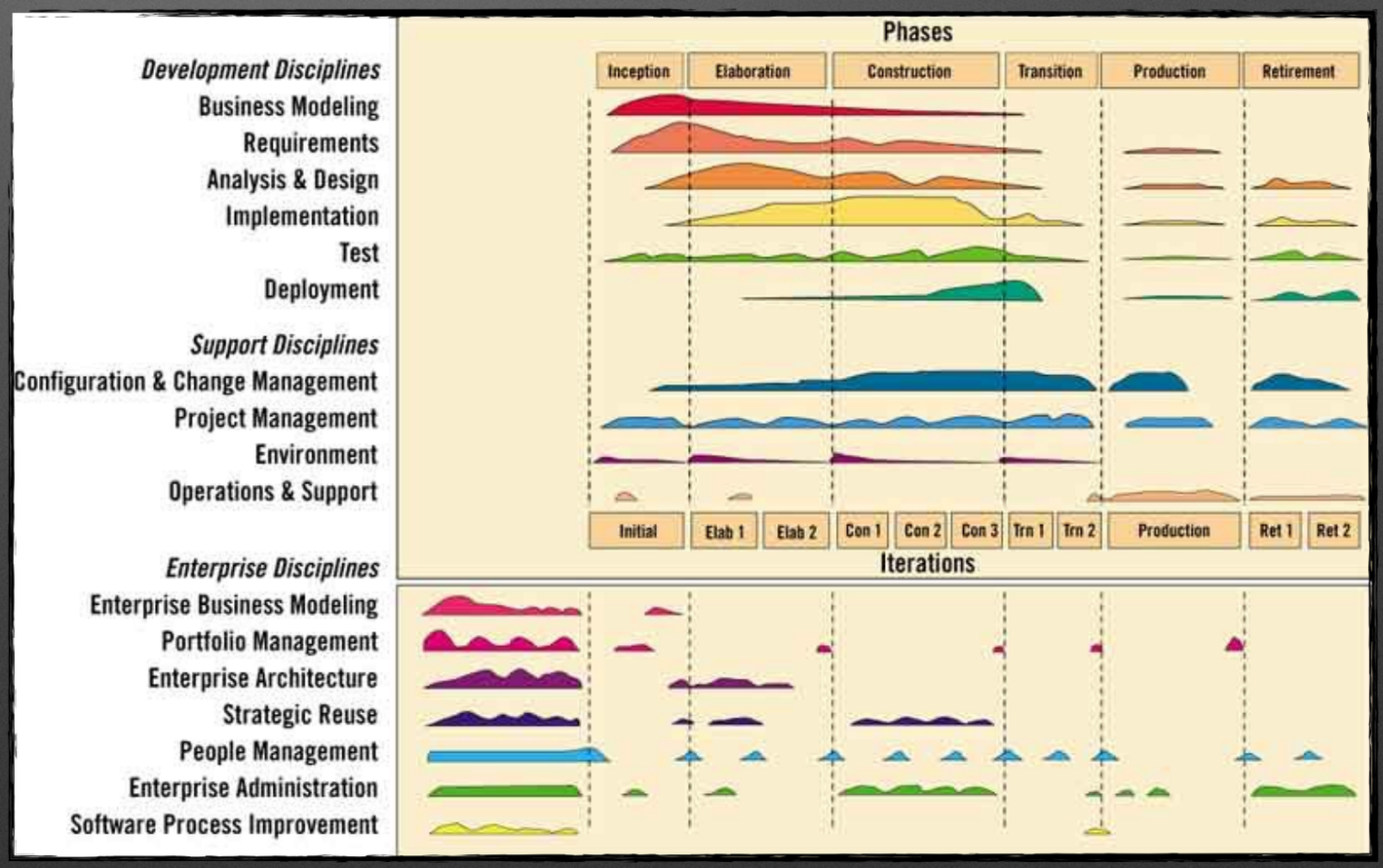
- RUP - maximalistický, obsahuje spoustu artefaktů, aktivit, snaží se o co nejširší pokrytý procesu vývoje SW.
    - Komerční = placené licence
    - Pro správné použití je potřeba výborná znalost frameworku a předchozí zkušenosti.
- OpenUP (EPF) - minimalistický, obsahuje pouze jádro RUP a agilní techniky z XP, Scrum, Lean Development
    - Zdarma = volně ke stáhnutí
    - Umožňuje plynulý přechod k RUP
    - Základní startovní bod pro menší agilní týmy

Fáze

- Inception - pochopení rozsahu projektu a jeho ceny
- Elaboration - architektura, rizika
- Construction - transfer požadavků do přidané hodnoty
- Transition - nasazení v prostředí zákazníka
- (Production)
- (Retirement)

Disciplíny

- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment



RUP vs. Agile

Vodopád

Méně procedur, formálnosti

Standard DoD

Více procedur, formálnosti

RUP

Open Up

XP, Scrum, Lean Development

Iterativní přístup

Principy RUP

- Adapt the Process
- Balance Competing Stakeholder Priorities
- Collaborate Across Teams
- Demonstrate Value Iteratively
- Elevate level of Abstraction
- Focus Continuously on Quality

Struktura principů

- **Vzor (pattern)** - vychází z ověřeného způsobu, jak řešit konkrétní problém
- **Anti-vzor (anti-pattern)** - příklad toho, jak by řešení problému vypadat nemělo

Adapt the Process

- **Přínos** (benefit): Větší efektivnost životního cyklu, lepší komunikace rizik

- **Vzor**: Preciznost a formálnost se vyvíjí od nízké po vysokou v průběhu životního cyklu tak, jak jsou odstraněny nejasnosti. Přizpůsobte proces velikosti a distribuci projektového týmu, složitosti aplikace. Neustále vylepšujte Váš proces.

- **Anti-vzor**: Precizní plány, odhady a postupování podle nich. Mít více procesu (artefaktů, aktivit, rolí) je vždy lepší. Vždy v průběhu životního cyklu udržujte stejnou úroveň formálnosti a preciznosti.

How Much Process is Necessary?

Simple upgrades
R&D Prototypes
Static web apps

Dynamic web apps
Packaged applications
Component based (J2, .Net)

Legacy upgrades
Systems of systems
Real-time, embedded
Certifiable quality

When is Less Appropriate?

- Co-located teams
- Smaller, simpler projects
- Few stakeholders
- *Early life-cycle phases*
- Internally imposed constraints

When is More Appropriate?

- Distributed teams
- Large projects
- Many stakeholders
- *Later life-cycle phases*
- Externally imposed constraints
  - Standards
  - Contractual requirements
  - Legal requirements

Příklady

- Jiný proces je potřeba pro malý tým vývojářů, jiný pro velký tým.
- Jiný proces (volnější) třeba v úvodu projektu, kdy je třeba spousta kreativity, spolupráce pouze několika klíčových lidí. Jiný v Construction – zahrnuto více lidí, třeba dodržovat postupy, využívat architektonických mechanismů.
- Jiný proces třeba pokud známá technologie a stejný tým, jiný pokud nová technologie, nový tým.

USER STORIES & TECH. AN.
- PRIORITIES (PO)
- JIRA
Tom. PO
ZADANI

ESTIMACE
- TEAM
- PO (GROOMING)

RELEASE PLAN
- TEAM
- PO

SPRINT PLANNING
- TASKS

2 WEEKS

TASKING & TECHNIQUES

CONSULTING

DAILY

- PROGRAMMING
- PROTOTYPES

CONTINUOUS INTEGRATION

QA

- DAILY MEETING
- ~~CONFIN~~

DEMO
- CUSTOMER

RETROSPECTIVE

RELEASE

Balance Competing Stakeholder Priorities

- **Přínos:** Vývoj aplikace podle potřeb uživatelů, redukce vývoje na zakázku, optimalizace přínosů pro byznys.

- **Vzor:** Definuj a porozuměj podnikovým procesům a potřebám uživatelů; snaž se porozumět, které komponenty můžeš znovupoužít.

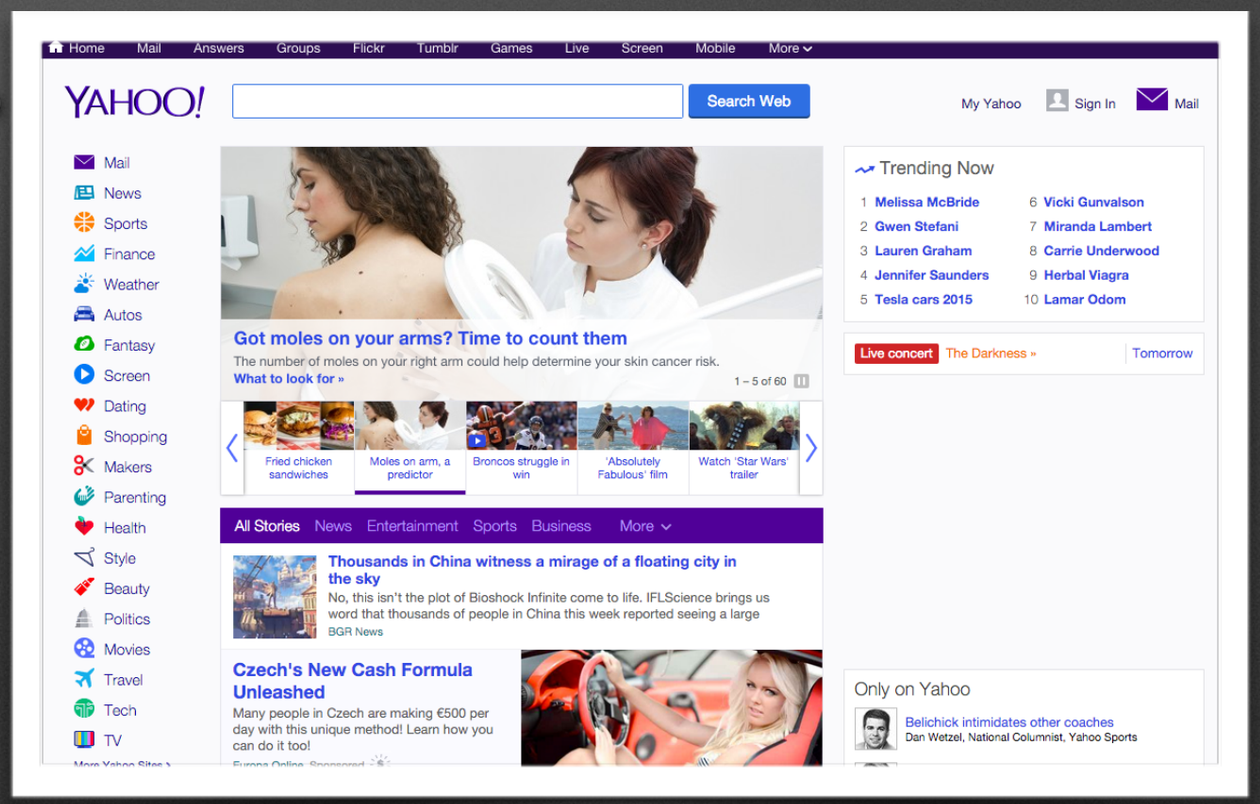
- **Anti-vzor:** Zachyť precizně veškeré požadavky předtím, než začnou veškeré práce na projektu. Definuj požadavky tak, aby byl celý vývoj zákaznický (bez znovupoužitých komponent).

Balance Competing Stakeholder Priorities

- Jádrem tohoto principu jsou dvě roviny:
  - Identifikace klíčových potřeb/požadavků uživatelů.
  - Zakázkový vývoj vs. znovupoužitelnost existujících komponent

Kdo používá Google pro vyhledávání?

Proč?

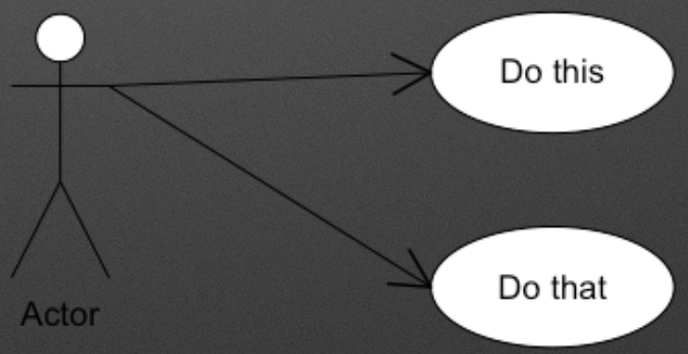


Vzor: popis požadavků v jazyce uživatele

Tradiční (dokumentový) způsob

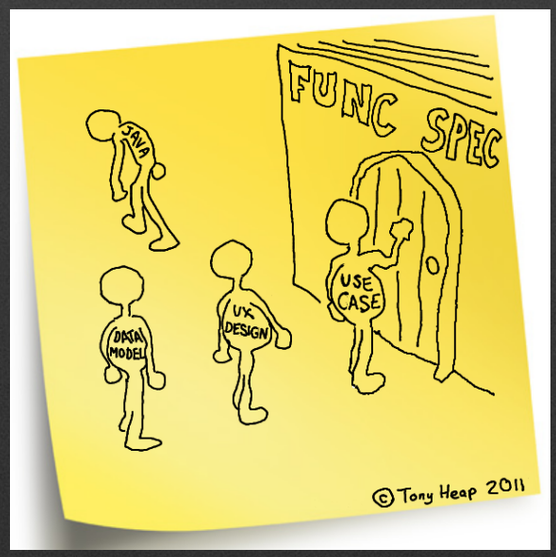
- Detailní dekompozice (funkce, rysy dohromady)
  - Nejasné závislosti
  - Nejasná struktura

Systém bude dělat to ...
Systém bude umět ono...
Systém bude produkovat ...



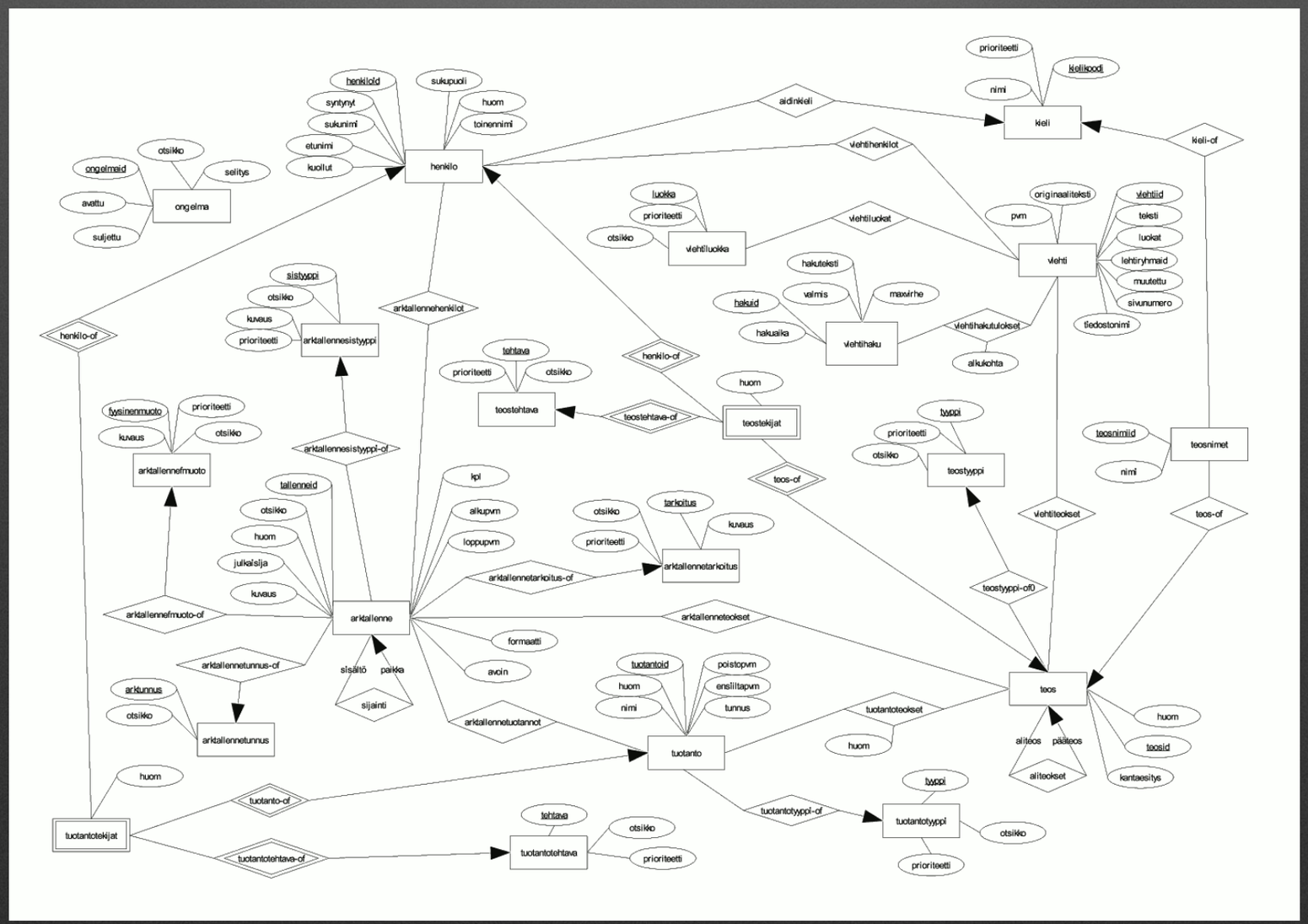
Lepší způsob

- Požadavky definovány ve formě funkcí, (UC+scénáře)
- Popis systému z pohledu uživatele



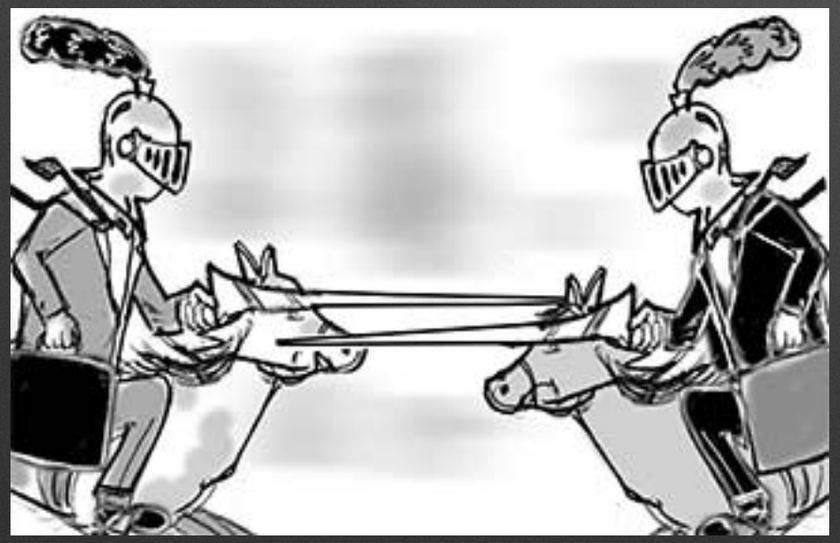
Anti-vzor

- Precizně a detailně specifikuj požadavky předem.



Anti-vzor

- Nechej je schválit zadavatelem a pak vyjednávej jejich každou změnu.



Anti-vzor

- Ber v úvahu pouze požadavky nejhlasitějších uživatelů.

Collaborate Across Teams

- **Přínos:** Produktivita týmu, lepší spojení mezi byznysem, vývojem a provozem software.

- **Vzor:** Motivuj lidi, aby pracovali nejlépe, jak umějí. Spolupráce mezi jednotlivými funkčními celky, analytik, vývojář a tester pracují dohromady. Ujisti se zda byznys, vývojové a provozní týmy pracují efektivně jako integrovaný celek.

- **Anti-vzor:** Vychovávej heroické jednotlivce a vybav je mocnými nástroji.

Týmová práce

- Efektivní spolupráce - samo-řízené týmy (self-managed teams)

- Tým seznámíme s tím, co chceme zákazníkovi doručit a delegujeme na něj odpovědnost a rozhodování.

- Přínos: Pocit odpovědnosti za týmový výsledek, lepší motivace lidí.

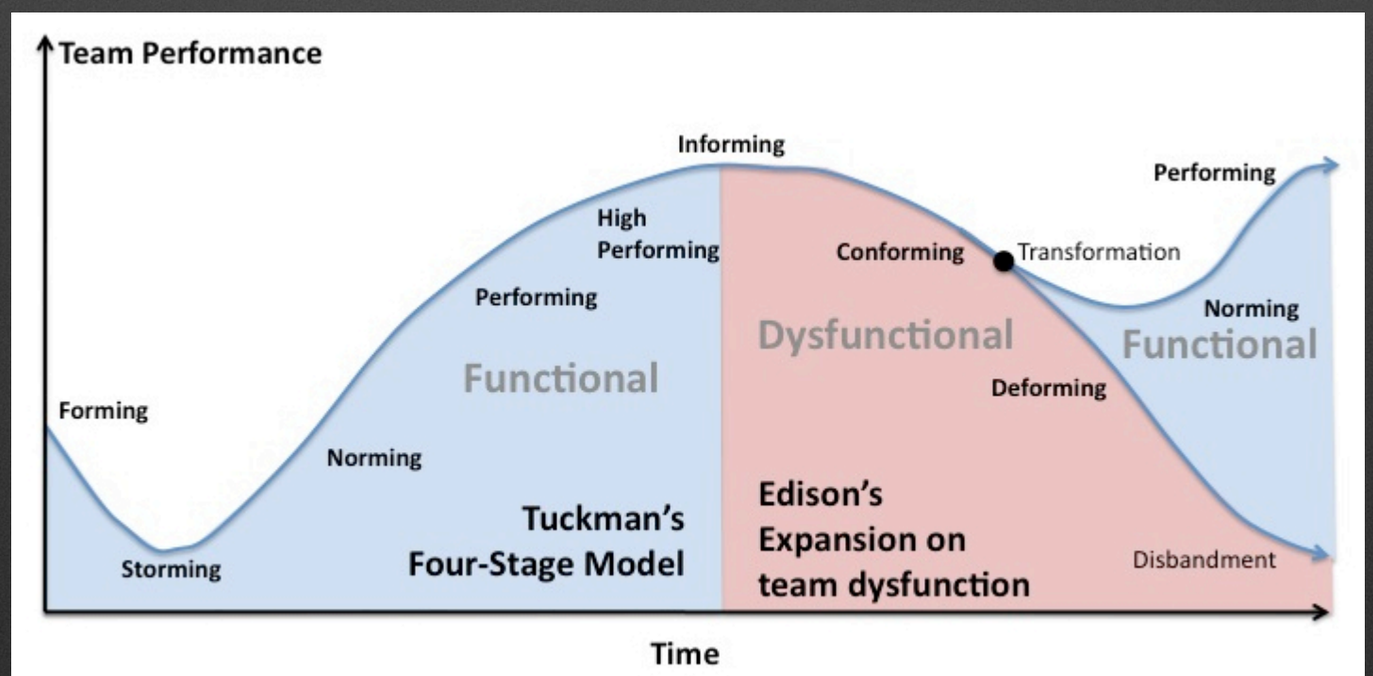
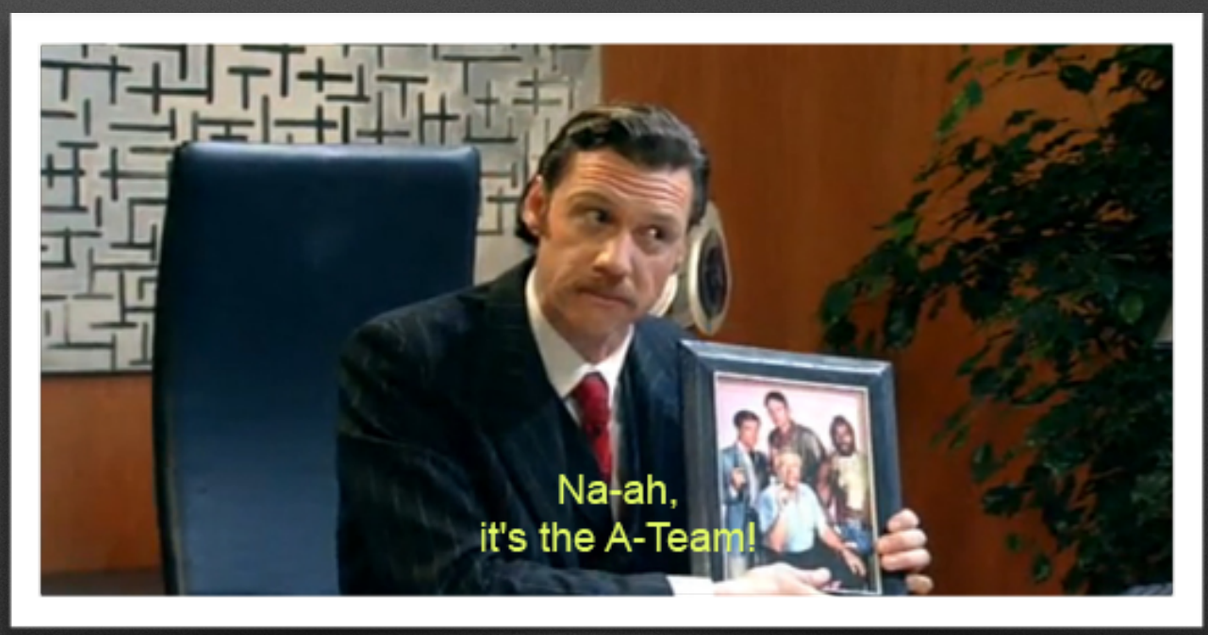
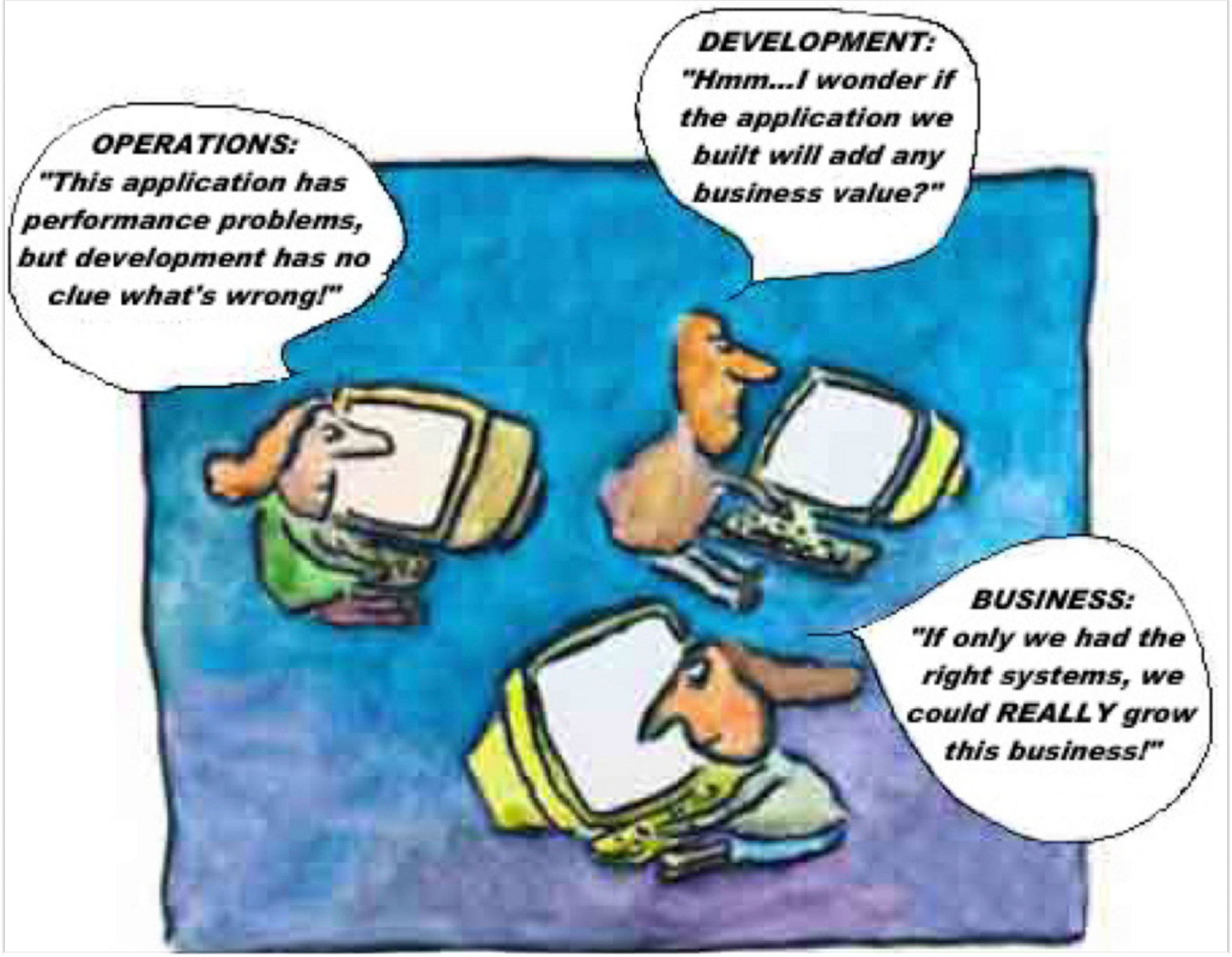


Figure 1 – This illustration shows the full life-cycle of a team's development, from its initial formation to its potential decline unless there is a transformation initiated (adapted from Edison [6])

Příklad

- Cítíte se motivování, abyste v ROPR pracovali nejlépe, jak dovedete?
- Jak byste motivovali ostatní členy týmu, aby pracovali také nejlépe, jak dovedou?

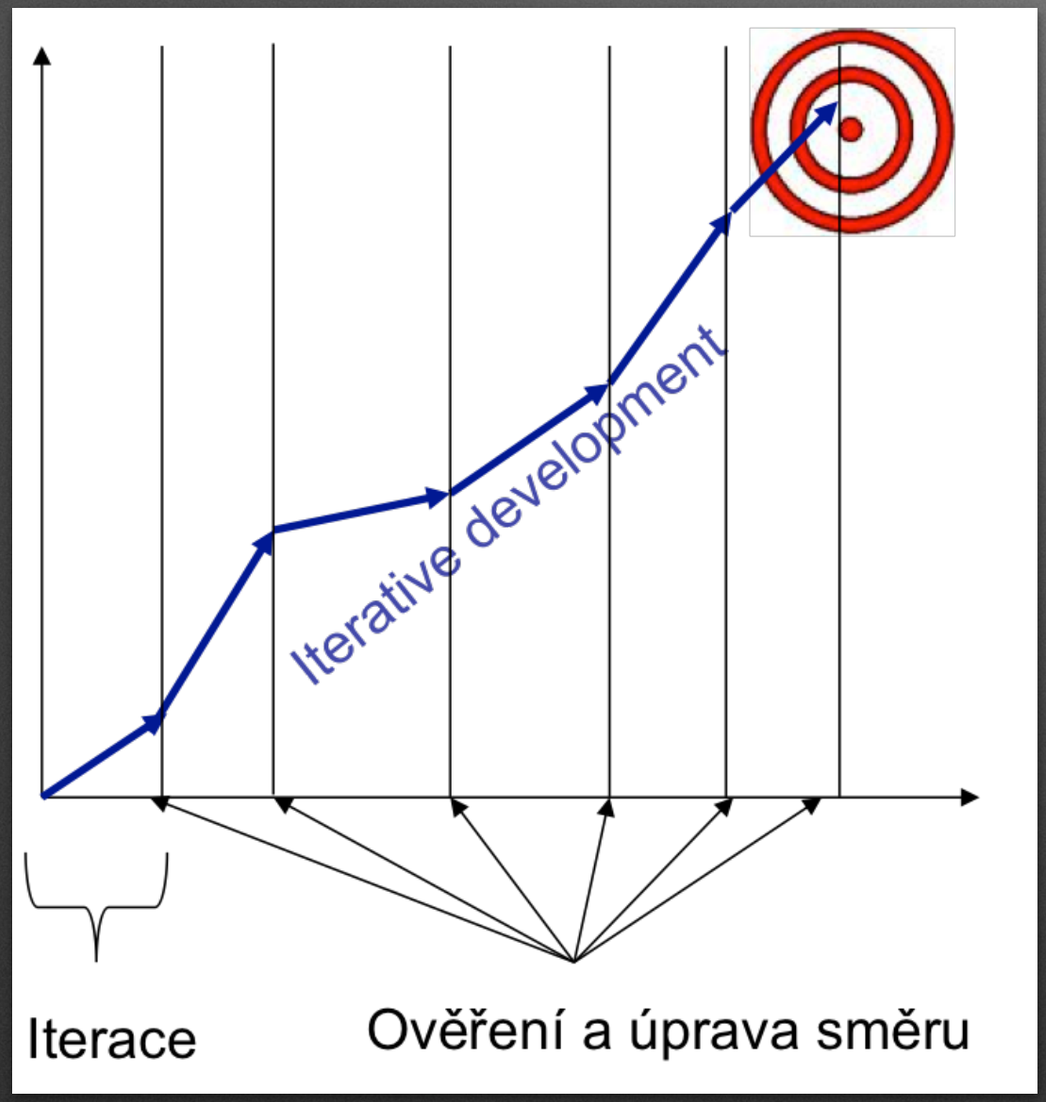


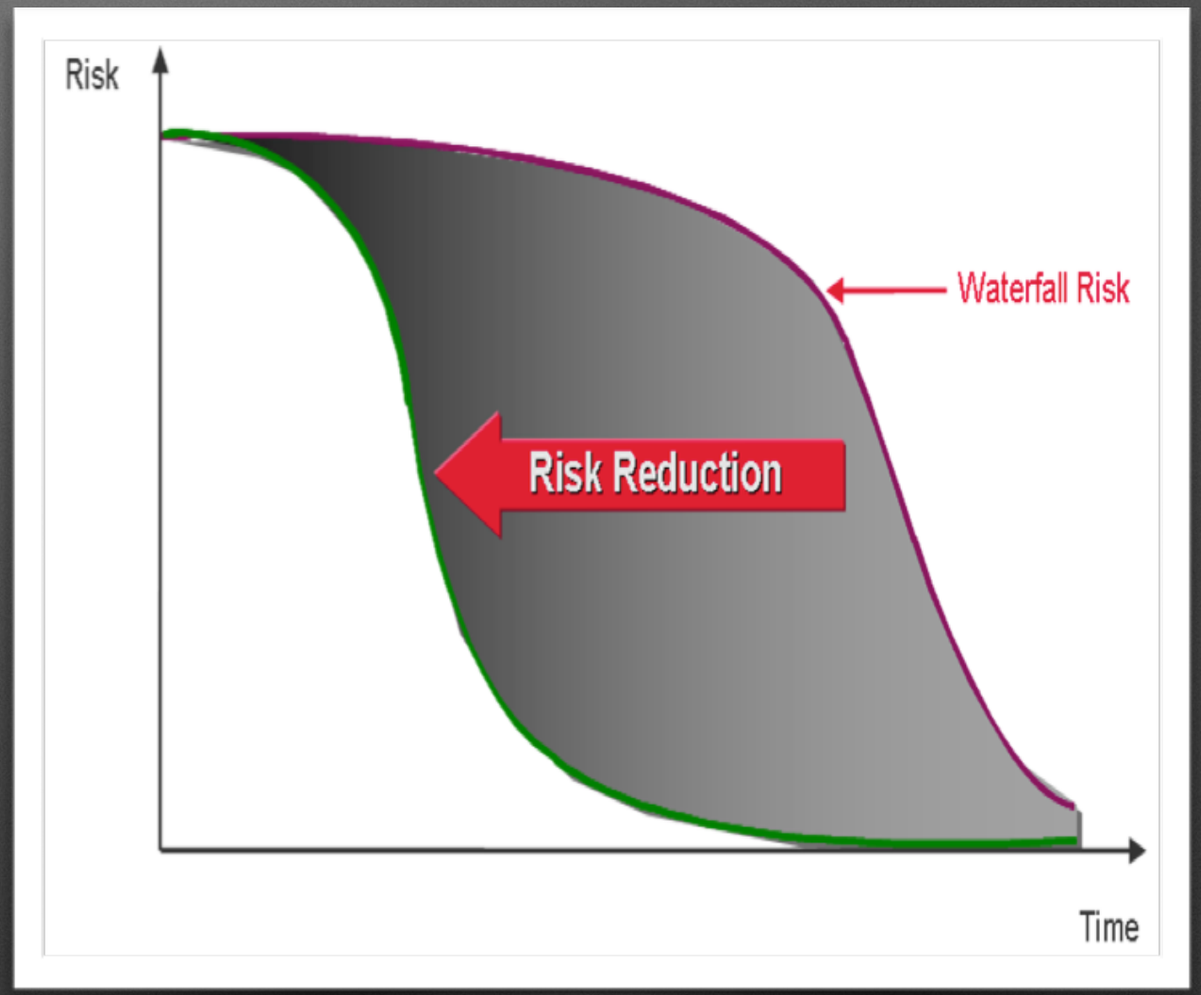

DEVELOPMENT:
"Hmm...I wonder if the application we built will add any business value?"
OPERATIONS:
"This application has performance problems, but development has no clue what's wrong!"
BUSINESS:
"If only we had the right systems, we could REALLY grow this business!"

Demonstrate Value Iteratively

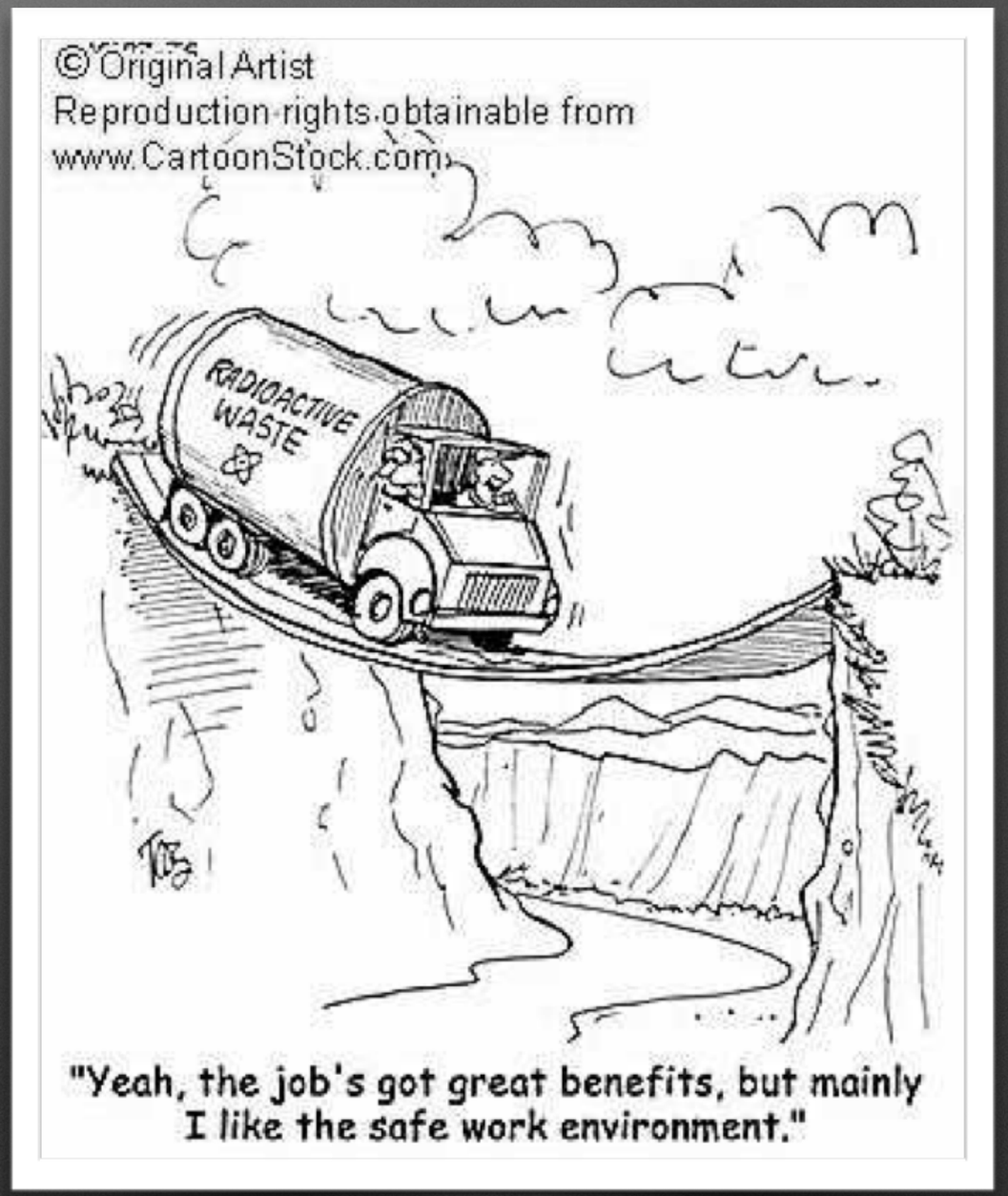
- **Přínos:** Brzké zmírnění rizik, vyšší předvídatelnost vývoje, důvěra mezi všemi účastníky projektu.
- **Vzor:** Adaptivní management s použitím iterativního vývoje. Atakuj významná technická, byznys a programátorská rizika co nejdříve. Získej zpětnou vazbu od uživatele tím, že v každé iteraci doručíš nějakou hodnotu.
- **Anti-vzor:** Naplánuj detailně celý životní cyklus, sleduj změny proti tomuto plánu. Detailní plány jsou lepší plány. Posuzuj status projektu revizí specifikací.

Redukce rizik iterativním vývojem





Příklad


©Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
RADIOACTIVE WASTE
"Yeah, the job's got great benefits, but mainly I like the safe work environment."

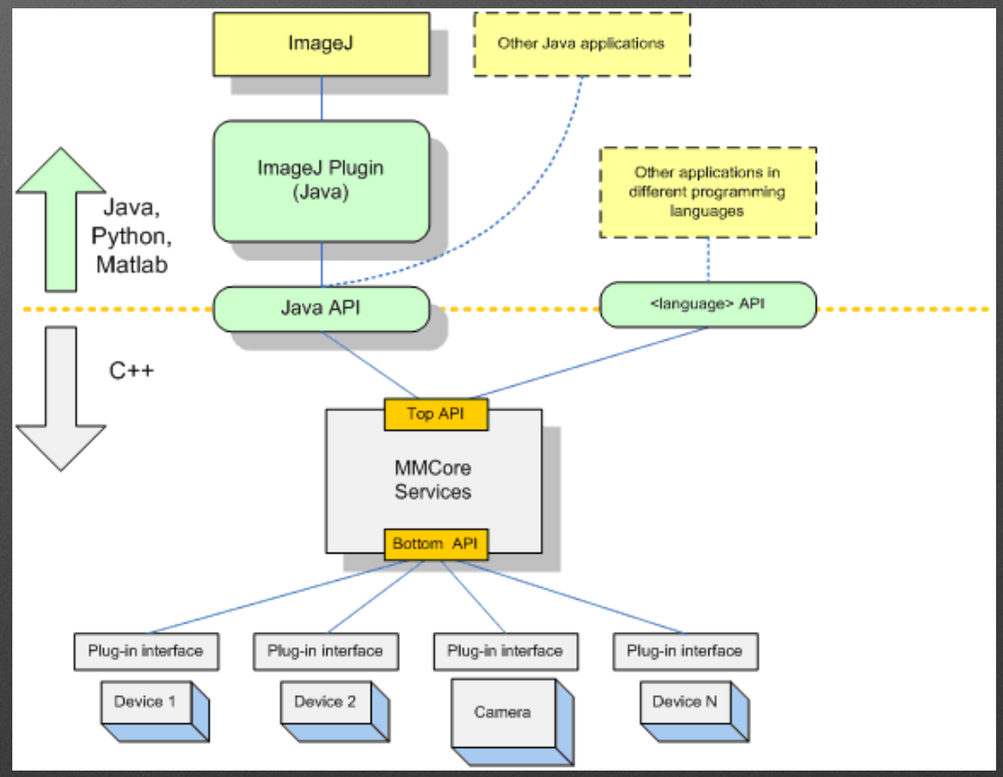
Reaktivní či proaktivní přístup

Reaktivní čí proaktivní přístup

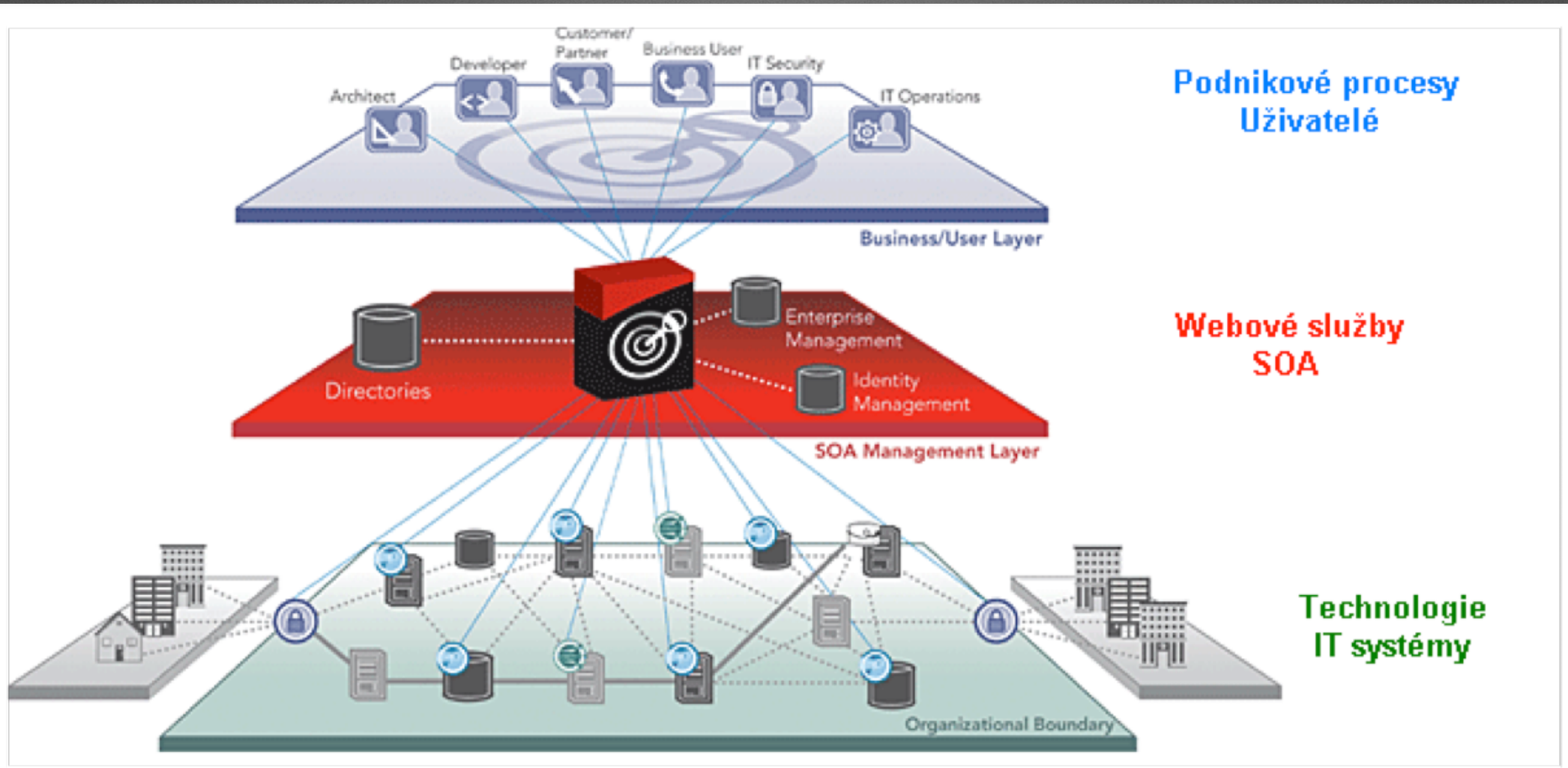
Reaktivní čí proaktivní přístup

Elevate level of Abstraction

- **Přínos:** Produktivita, nižší komplexnost.
- **Vzor:** Znovupoužij již existující komponenty, redukuj objem ruční práce použitím nástrojů a jazyků vyšší úrovně, navrhuj pružnou, kvalitní a srozumitelnou architekturu.
- **Anti-vzor:** Jdi přímo od vágních, vysokoúrovňových požadavků uživatelů k psaní kódu celé aplikace.



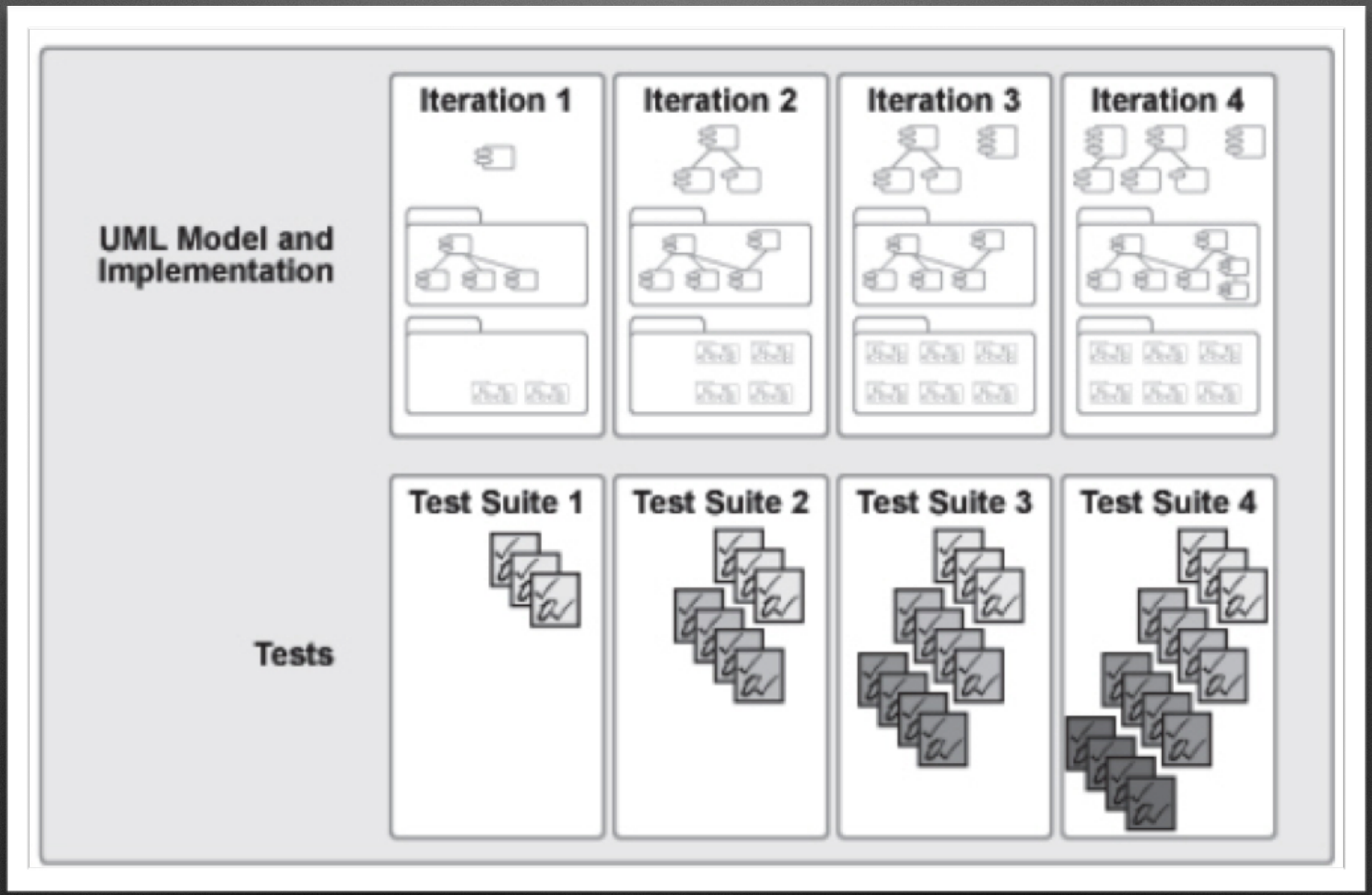
Příklad


Customer/
Partner
Developer
Business User
IT Security
Architect
IT Operations
Business/User Layer
Enterprise
Management
Directories
Identity
Management
SOA Management Layer
Organizational Boundary
Podnikové procesy
Uživatelé
Webové služby
SOA
Technologie
IT systémy

Focus Continuously on Quality

- **Přínos:** Vyšší kvalita, rychlejší pokrok.
- **Vzor:** Odpovědnost celého týmu za výsledný produkt. Testování a průběžná integrace se stávají prioritními. Inkrementálně zlepšuj testy a automatické testování.
- **Anti-vzor:** Posuň integrační testování až do fáze, kdy je celý produkt naprogramovaný a otestovaný unit testy. Prováděj revizi všech artefaktů raději než ověřování a testování částečně implementovaného řešení za účelem nalezení problémů.

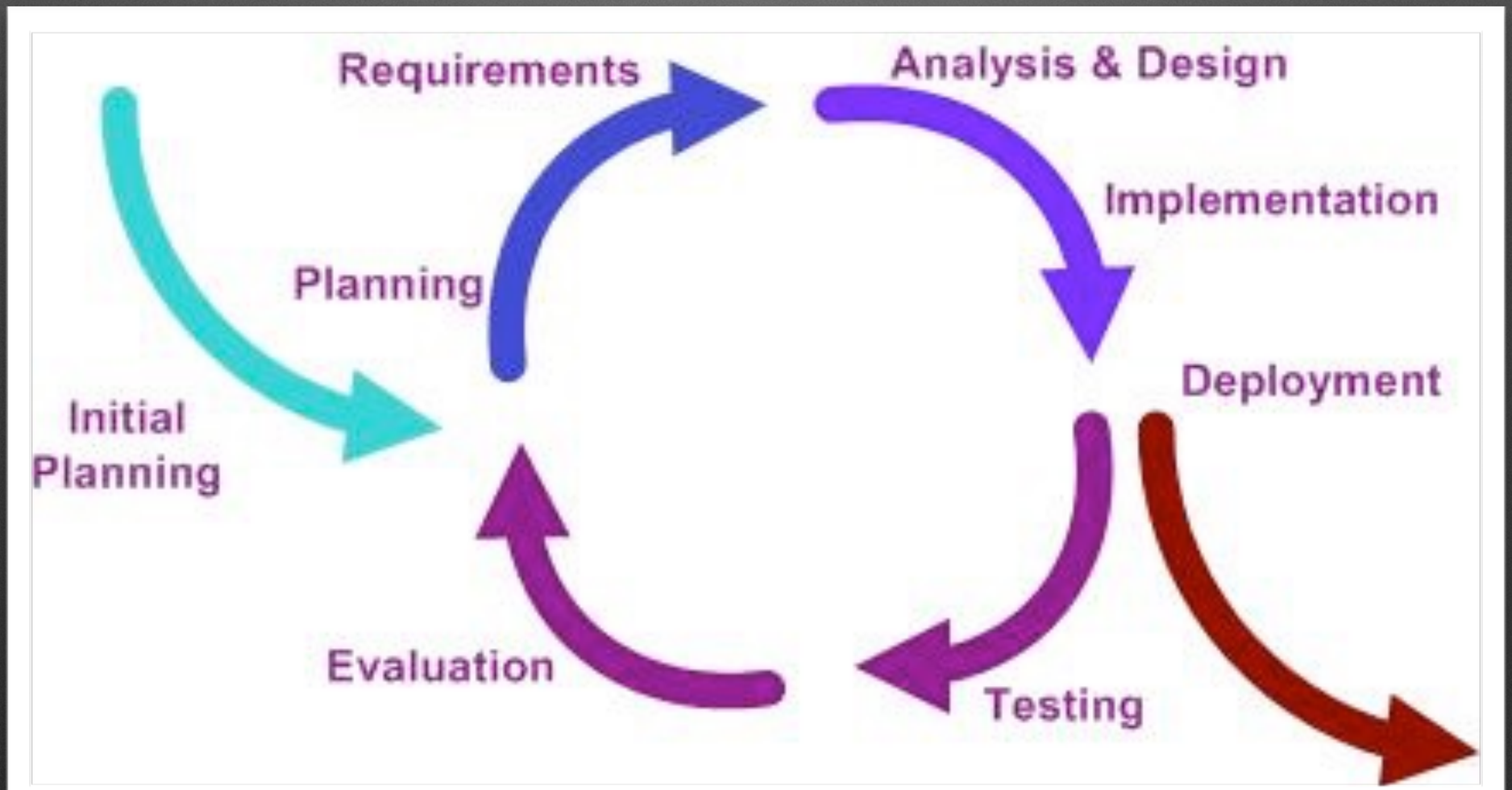
Příklad


Iteration 1
Iteration 2
Iteration 3
Iteration 4
UML Model and Implementation
Test Suite 1
Test Suite 2
Test Suite 3
Test Suite 4
Tests

Best practices

- Develop Iteratively
- Manage requirements
- Use Component Architectures
- Model Visually
- Continuously Verify Quality
- Manage Changes

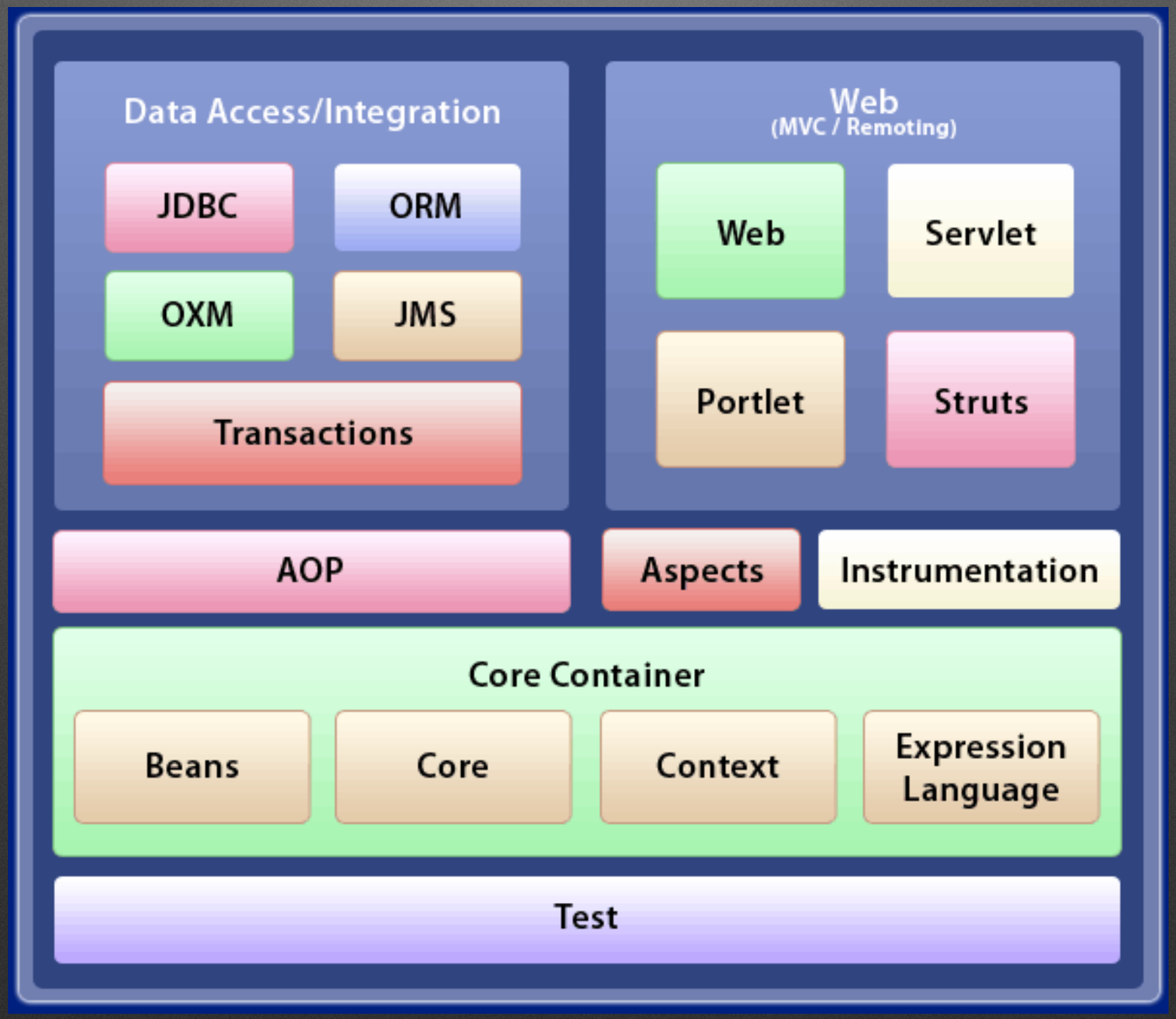
Develop Iteratively



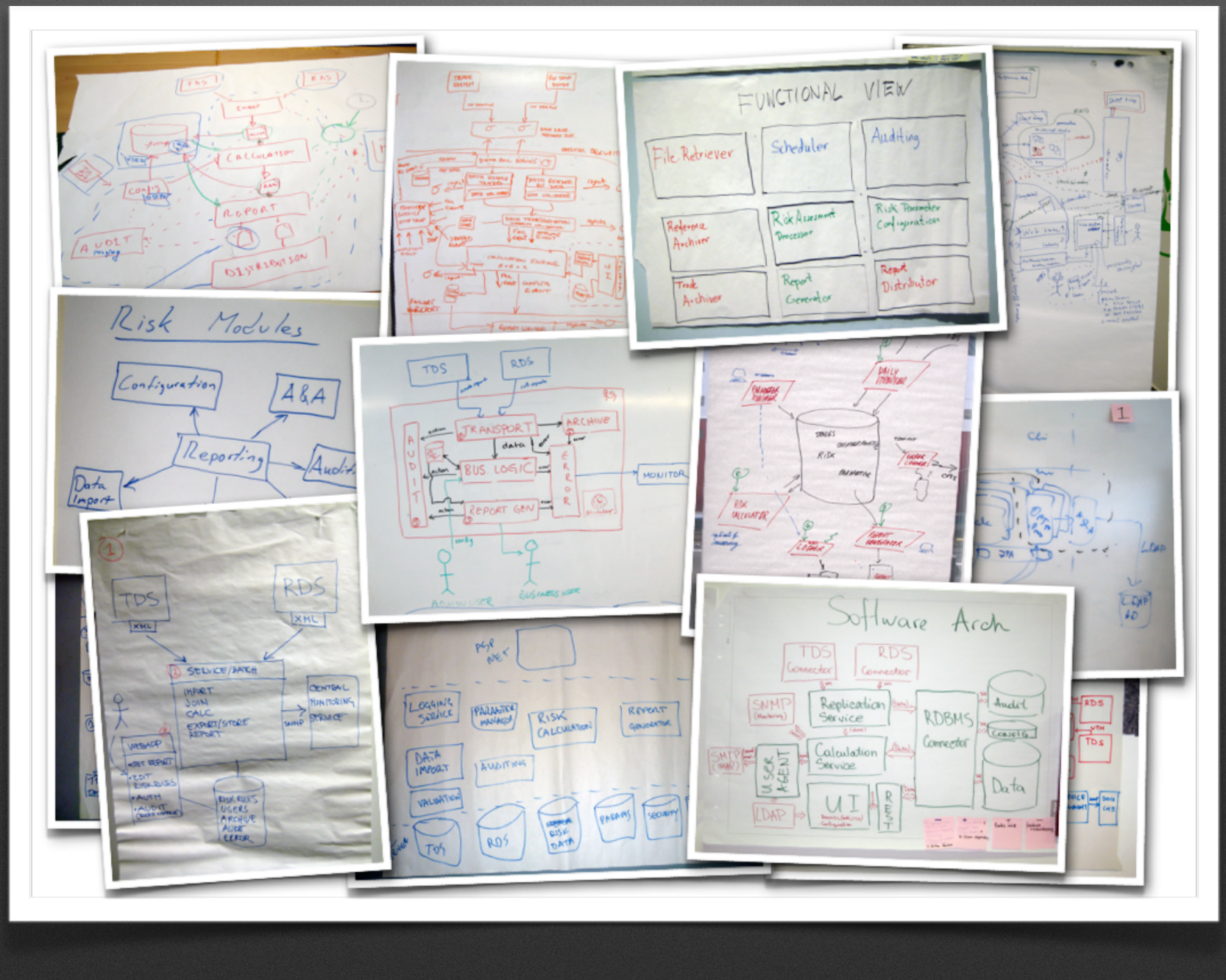
Manage requirements

- Analýza problému
- Pochopení problému
- Definování systému
- Určení rozsahu projektu
- Upřesnění systému
- Správa změn požadavků

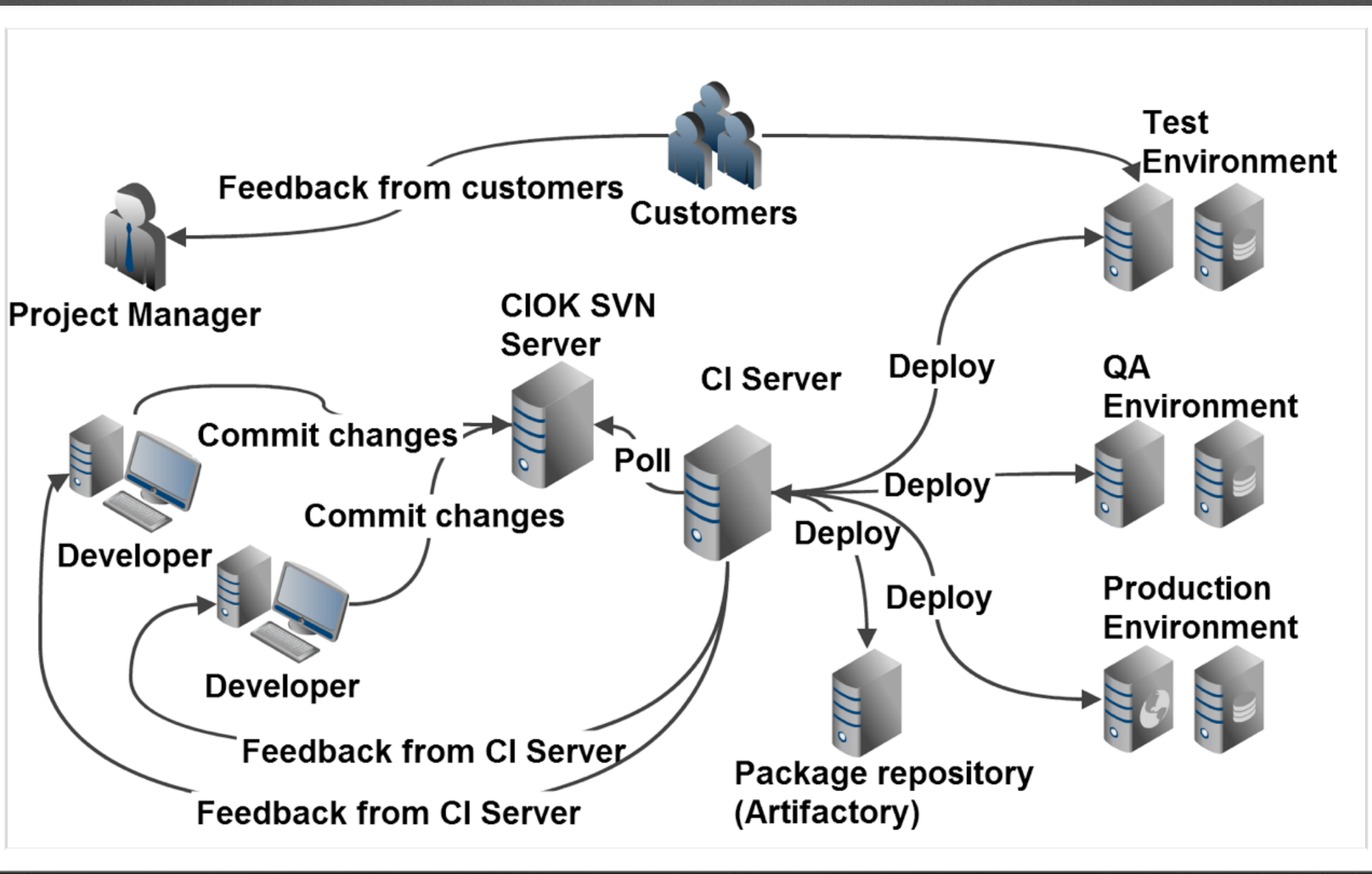
Use Component Architectures



Model Visually



Continuously Verify Quality



Manage changes