AHEAD
THROUGH
KNOWLEDGE

PROFINIT

Softwarový proces a jeho zlepšování

Martin Hlavatý

Květen 2022

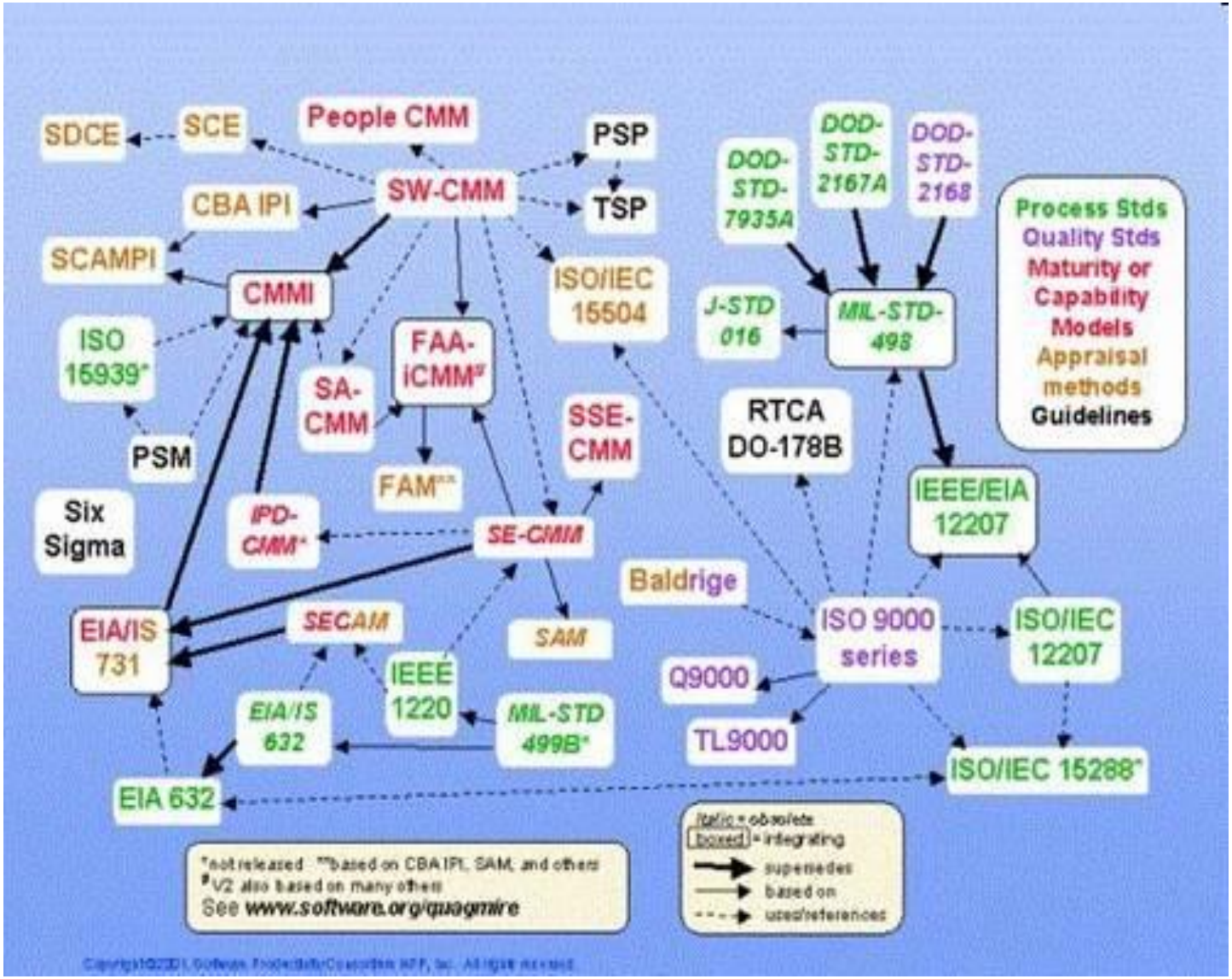
Základní koncept SPI

PDCA model (Deming cycle)

- Plan
  - Prověřit současnou výkonnost
  - Posoudit problémy, omezení
  - Navrhnout řešení
  - Naplánovat provedení
- Do
  - Otestovat účinnost řešení
- Check
  - Zhodnotit výsledky testu
  - Posoudit dosažení výsledků
  - Zaměřit se na překážky bránící zlepšení
- Act
  - Rozpracovat konečné řešení, aby bylo kdekoli použitelným přístupem

PROFINIT

Mnoho přístupů



Základní přístupy

- Systematický, dlouhodobý přístup
  - ISO, CMM, CMMI, … (prescriptive)
    - Preskriptivní přístupy vychází z premisy, že správné praktiky vývoje SW jsou známy, a tyto předepisují.
  - SEL/NASA, … (inductive)
    - Induktivní vycházejí z premisy, že zlepšovat se má to, co odpovídá stavu, cílům, problémům, atd. konkrétní dané organizace.
- Best practices

Rodina ISO 9000

ISO 9001:2008

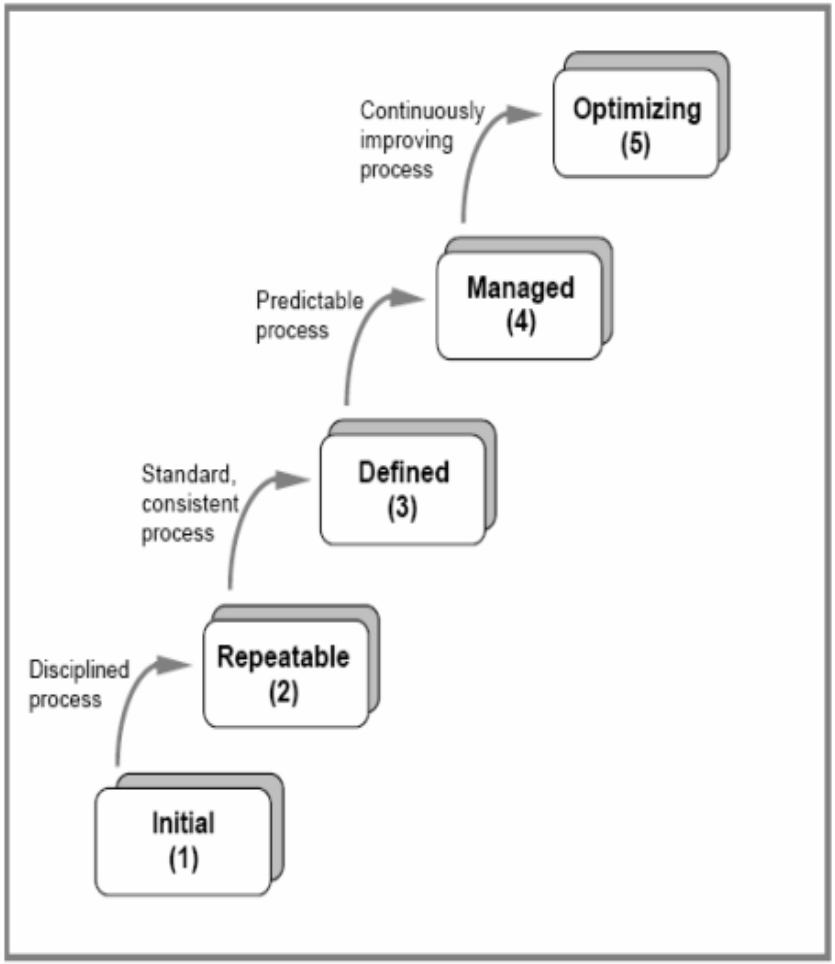
- Požadavky na Quality Management System
- Standard

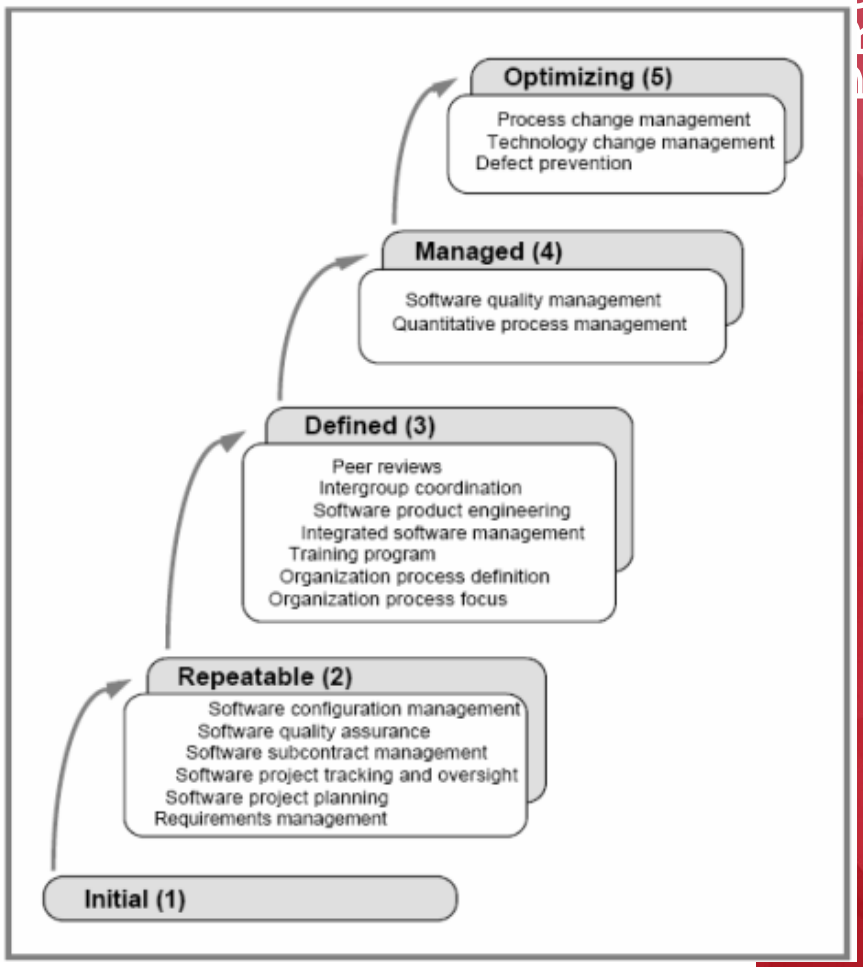
ISO/IEC 90003:2004

- Návod pro aplikaci ISO 9001 na SW

PROFINIT

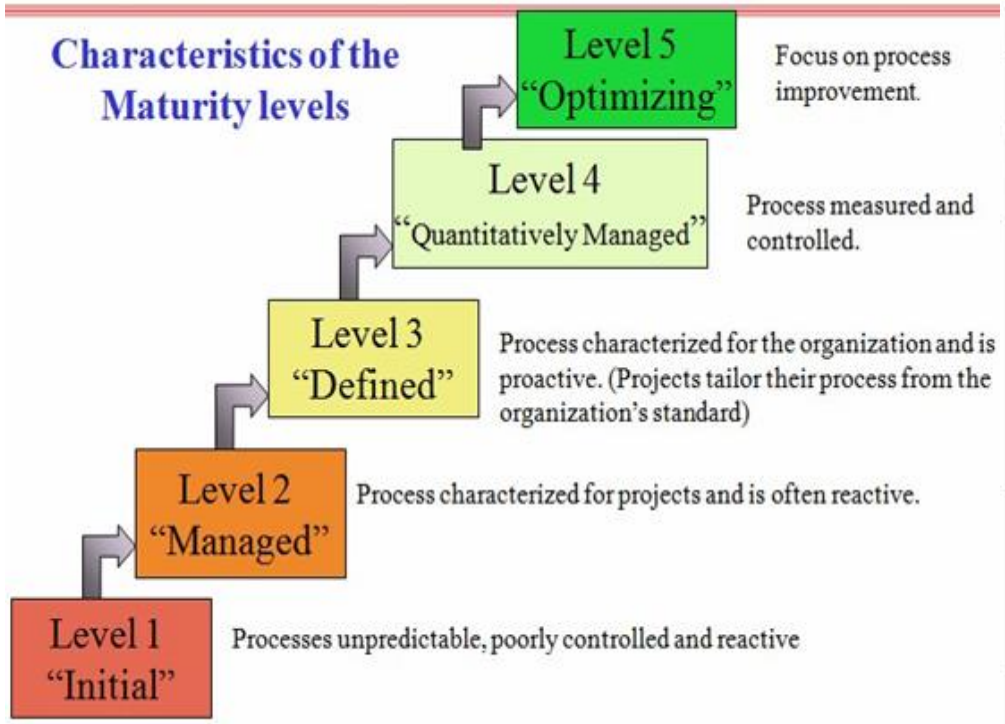
CMM


Continuously improving process
Optimizing (5)
Predictable process
Managed (4)
Standard, consistent process
Defined (3)
Disciplined process
Repeatable (2)
Initial (1)


Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)

CMMI

- Nástupce staršího CMM
- 5 úrovní, 22 key process areas (KPA)
- Staged vs. Continuous
- 3 různé modely – Development, Acquisition, Services



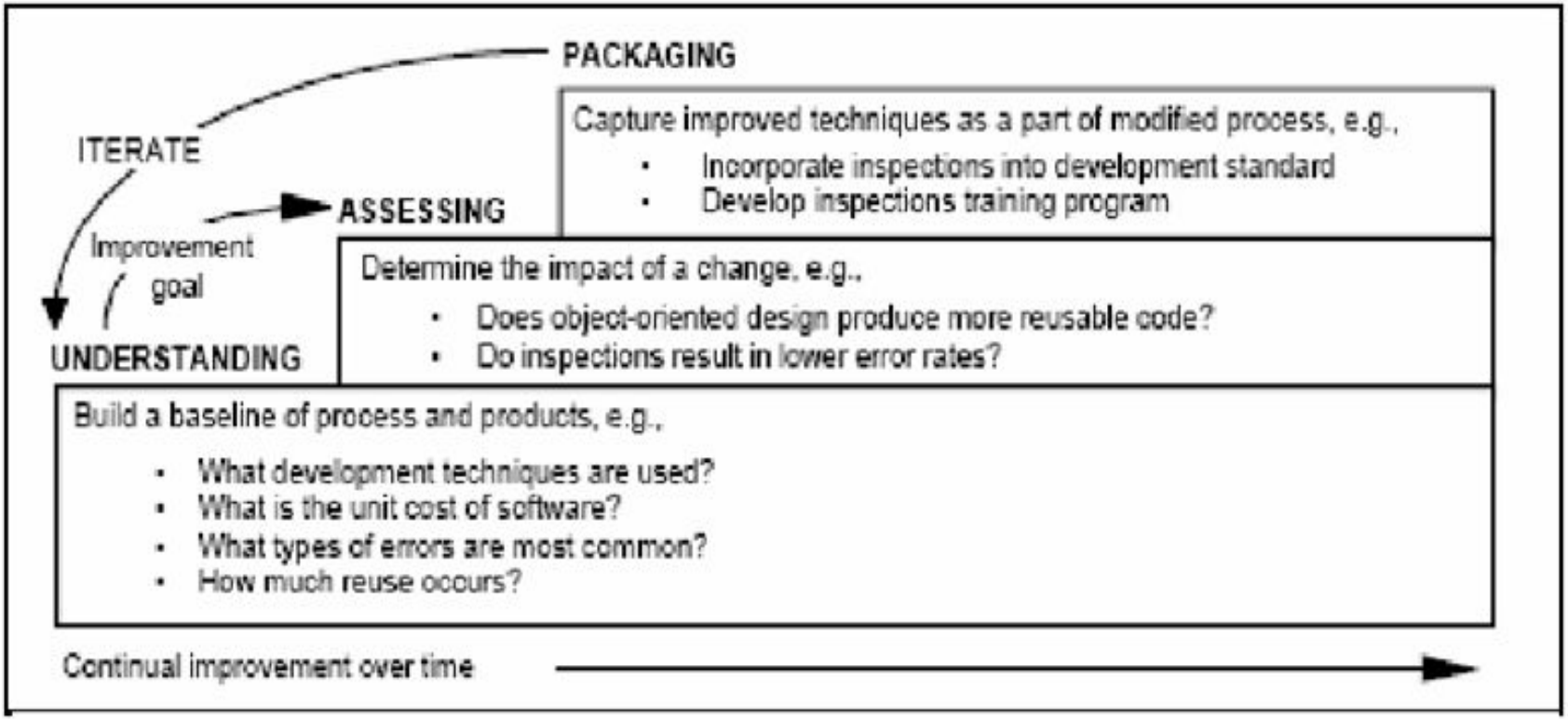
Level	Focus	Process Areas	Result
5 Optimizing	*Continuous process improvement*	Organizational Innovation & Deployment Causal Analysis and Resolution	Productivity & Quality
4 Quantitatively Managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management	
3 Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Risk Management Decision Analysis and Resolution	
2 Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring & Control Supplier Agreement Management Measurement and Analysis Process & Product Quality Assurance Configuration Management	
1 Initial	*Competent people and heroics*		

SEL / NASA

Základní premisa

Vývojová organizace musí úsilí o zkvalitňování zaměřovat na

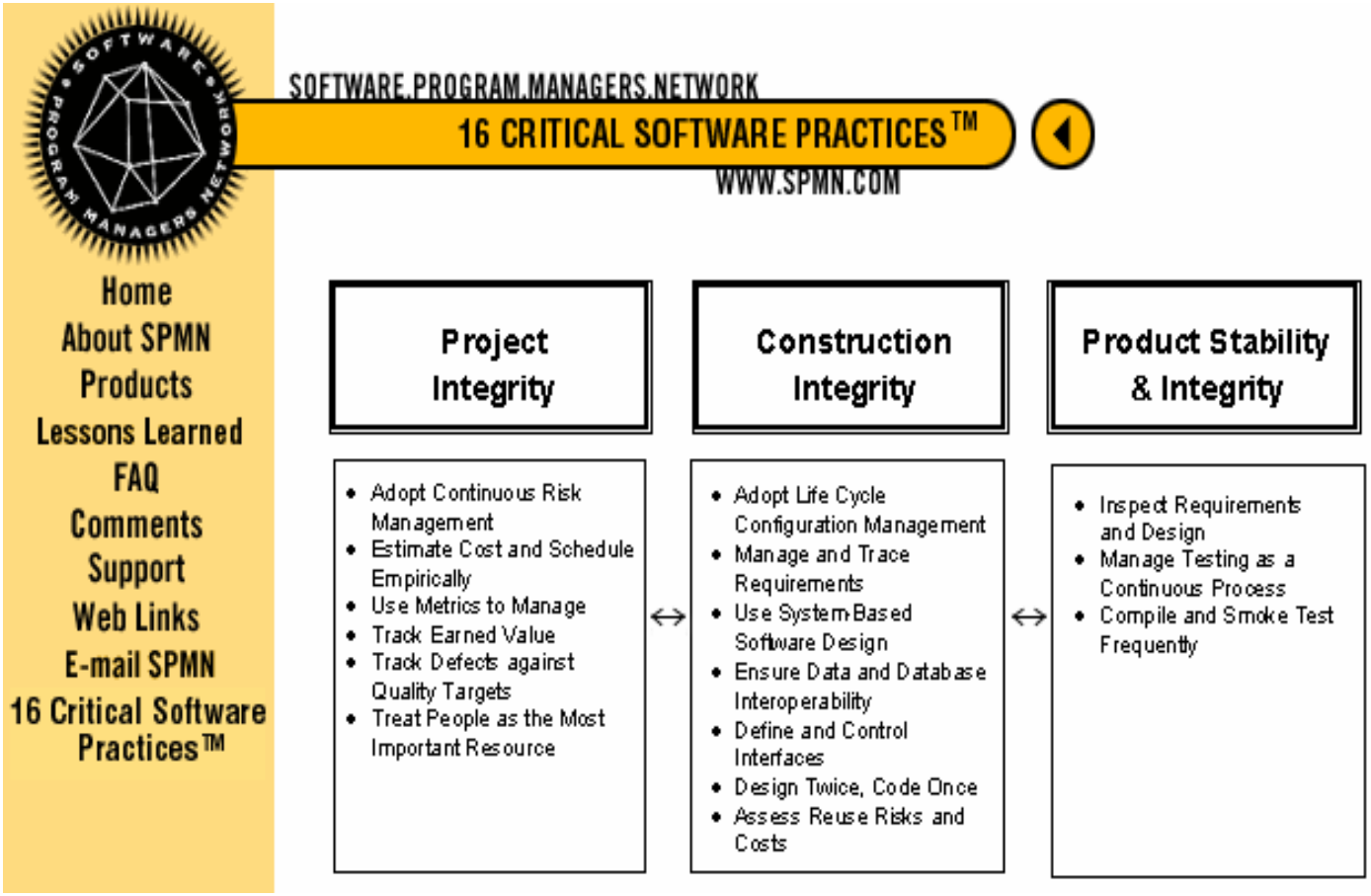
- zamezení minulých problémů a
- opakování minulých úspěchů.



PROFINIT

SPMN best practices

- Reakce na problémy se zvládáním velkých projektů
- Přímo použitelné strategie, techniky a praktiky
- „SPMN books“ (nutná registrace)



Shrnutí

Je třeba:

- znát současný stav vlastního procesu vývoje a jeho charakteristiky
- znát problémy s ekonomickým projevem a jejich vážnost
- mít názor, které problémy je nutné a možné odstranit
- mít názor, jak modifikovat proces vývoje
- mít prostředky, jak tuto modifikaci prosadit

... a znovu na začátek …

AHEAD
THROUGH
KNOWLEDGE

PROFINIT

SW process improvement

Pavel Mrázek

Prosinec 2020

Obsah

1. Kontext přednášky
2. Vývoj SW není jen programování
3. Proč
4. Jak na to?
5. Příklad: Issue tracking
6. Příklad: Project management
7. Příklad: Testování
8. Doporučení na závěr

PROFINIT

Kontext přednášky

- 2012 interní Profinit přednáška – že by stále aktuální téma?
- 2010–2019 Různé projekty a role

- 2019–2020 Manta (PM, SPI)

- Manta
  - Produkt pro analýzu datových toků: https://getmanta.com/
  - 2012 – 2016 startup
  - 2017-2019 ze 6 vývojářů na 40
  - Zákazníci TOP 500
  - 7 týmů
  - Technologická pestrost (graph DB) + alg. složitost
  - Podpora 6 produkčních verzí nazpět
  - 60% studentů

Kontext přednášky

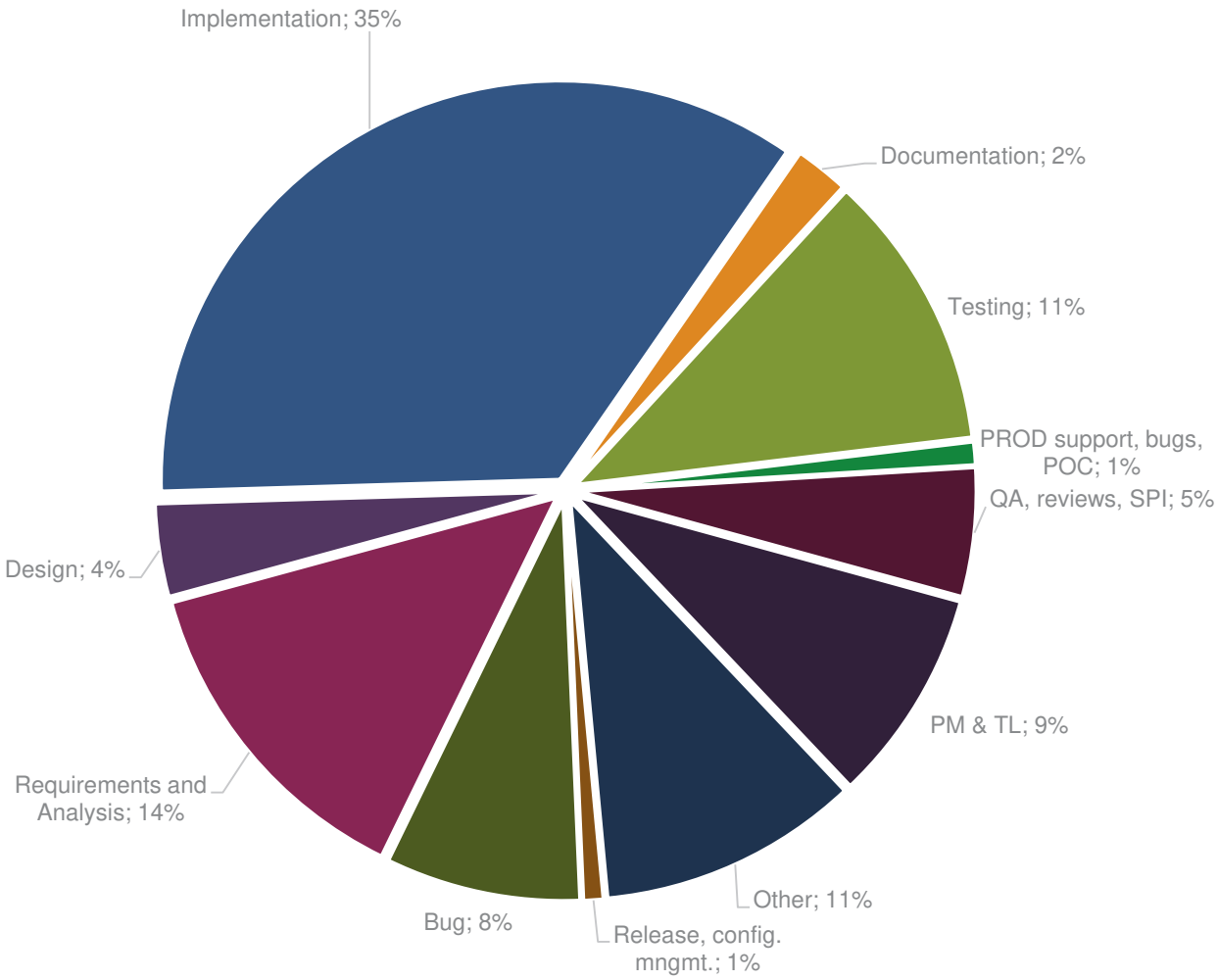
- Šéf: „Všechno už funguje“
  - Dodáno 24 verzí
  - Nastavený process
  - Vysoce modulární architektura
  - Build, dependency management (Nexus), CI
  - Release, update proces
  - Statická analýza kódu
  - Unit testy
  - Pomocné tooly (validátor Maven POM, automatizace release, OWASP kontrola zranitelností v ext. knihovnách)
  - Pořádek v konfiguračním managementu
  - Vykazování propojené s issue trackingem

- Šéf: „…a dá se to ladit do nekonečna“
  - Dlouhodobý vývoj dává možnost pozorovat, měřit, porovnávat opakující se momenty a jemně ladit a vyhodnocovat dopady změn

PROFINIT

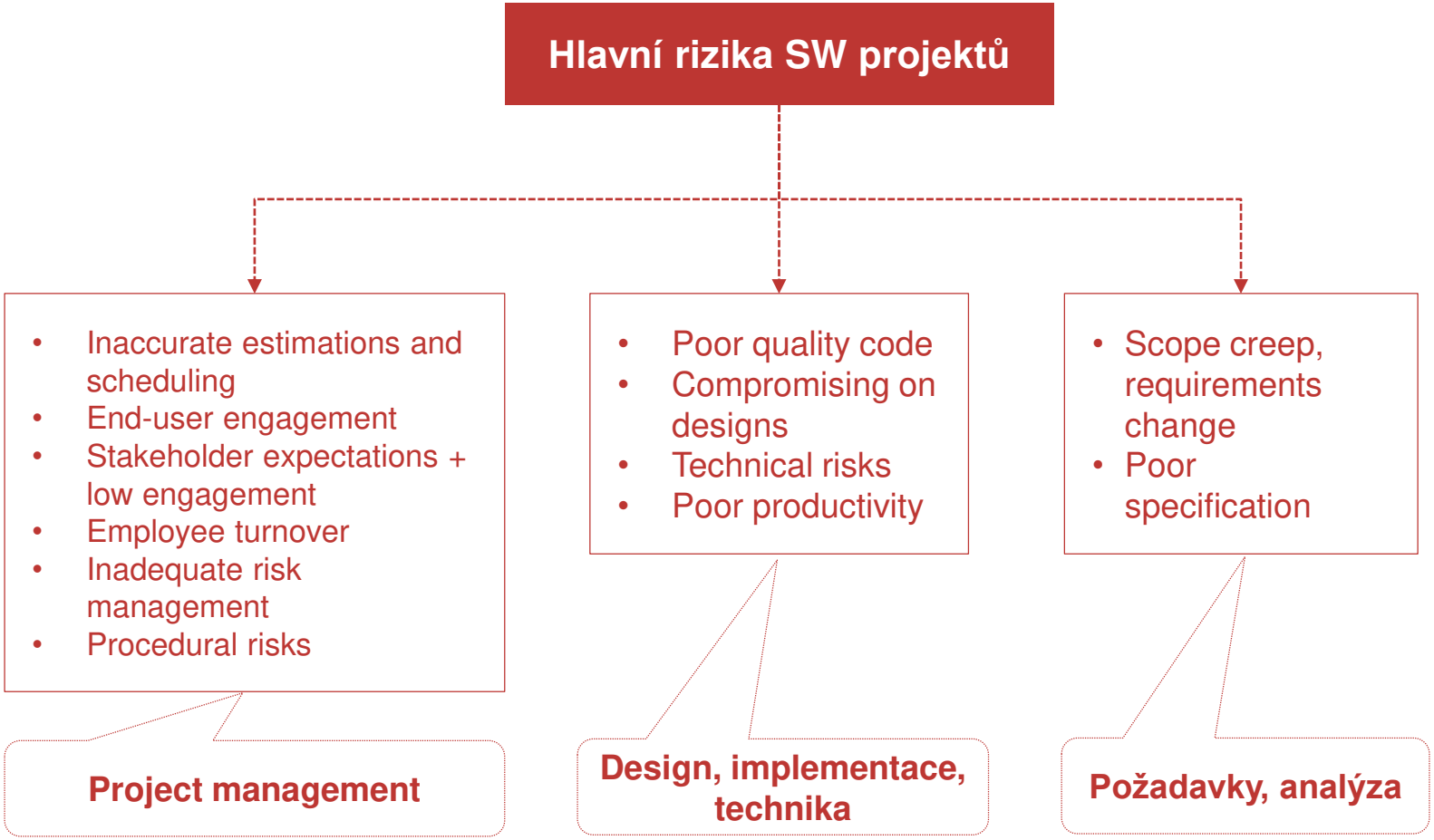
Vývoj SW není jen programování

- Podíl implementace 30-50%
- QA: kvalitní proces -> kvalitní produkt



PROFINIT

Vývoj SW není jen programování



Poznámka: pro ilustraci sjednoceno z více zdrojů

PROFINIT

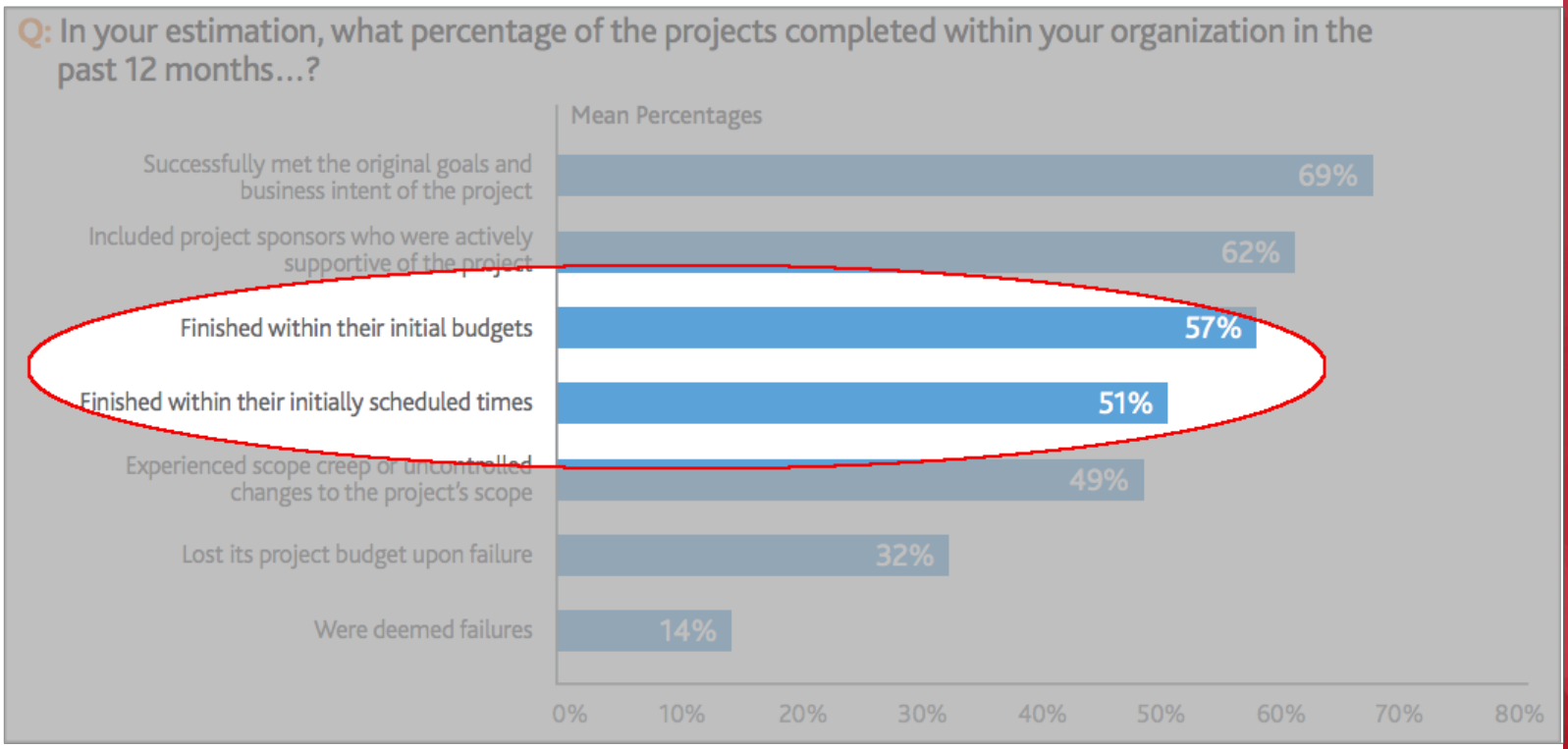
Vývoj SW není jen programování

- Zeptejte se vývojářů, co je na vašem projektu největší problém
  - Vývojář A: „Nelíbí se mi duplicity v CSS. Jo a třída DoItAllServiceImpl má 3000 řádků“
    - Detailní konkrétní problém
  - Vývojář B: „Zavedeme check-style pravidlo. Při nedodržení formátování kódu neprojde build“
    - Zlepšovák typu zhoršovák
  - Vývojář C: „Co je design? Vše je v pohodě“
    - Neví, že neví
- Typicky se málo mluví o analýze, designu, dokumentaci, testování, CI, release, organizaci práce, počtu regresních chyb ...?
- Takže si vyhrneme rukávy a jdeme vymyslet nějaký zlepšovák…

PROFINIT

Proč zlepšovat

- Důvod: Protože musíme, abychom uspěli

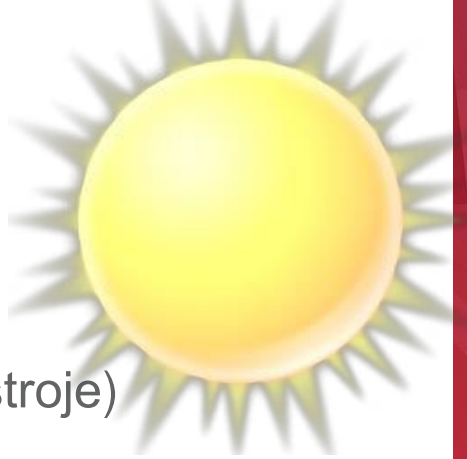


Zdroj: https://www.atspoke.com/blog/it/reasons-for-it-project-failure/

- Důvod: Technický/procesní dluh
  - pořád to tak nějak jde, až to nakonec opravdu nejde

PROFINIT

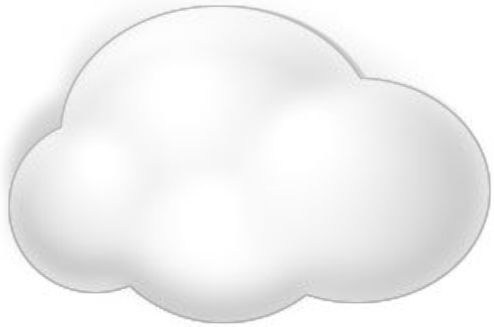
Proč zlepšovat



- Poznám projekt (procesy, architektura)
- Naučím se technologie (knihovny, frameworky, nástroje)
- Dostanu se mezi lidi
- Prozkoumám slepé uličky
- Ale nakonec ukážu, že to jde
- Pomůžu tím kolegům
- Založím si na tom kariérní postup

PROFINIT

Proč to nejde
Rozšířené výmluvy



- „Uděláme to, až na to bude víc času“
- „Víš, tenhle projekt je hodně specifický“
- „Nemůžeme mít proj. plán, protože se pořád mění zadání“
- „Přijde mi, že všechno funguje dobře“
- „Už je to rozbitý dlouho“
- „Takhle jsme to vždycky dělali“
- Ok důvod: Build feature větví, až po migraci na Git a zavedení code review

PROFINIT

MANTA

Jak na to?

Důležité předpoklady

- Zvládáme svou práci, zlepšováky jsou něco navíc
- Troufneme si na to
  - Kolik znáte odborníků na konfiguraci Maven repository?
- Tušíme, že se nám to vyplatí
- Máme podporu
  - Čas, pomoc
  - Vedení projektu, tým, zákazník
  - Získat si podporu. Nepodceňujte diskuze v kuchyňce u kafe a společné obědy
- Ke zvážení
  - Jednorázový krátkodobý projekt, tlak na čas a cenu, nebudeme dál rozvíjet

    X
  - Interní dlouhodobý produktový vývoj

PROFINIT

Jak na to?

Fáze 1: Pozorovat, poznávat

- Chápat vývojové procesy, než začnu něco měnit
- Identifikovat nedostatky
- Navrhnout řešení a obhájit ho (ideálně kvantifikovat přínos)
- Prioritizovat
  - Výběr závažnějších nedostatků řešitelných v rozumném čase
  - Quick win

PROFINIT

Jak na to?

Fáze 1: Pozorovat, poznávat

- **Příklad**: Seznam cca 40 návrhů na zlepšení
  - 21 done, 13 in progress
- Struktura: Závažnost, náročnost, oblast, aktuální stav, důsledek, návrh řešení, kdo může řešit, závislosti, hrubý odhad pracnosti

ID	Výběr	Závažnost	Náročnost	Oblast	Podoblast	Aktuální stav	Důsledek	Návrh řešení	Kdo může řešit	Závislost na	Odhad pracnosti [MD]	Stav
10	x	střední	střední	Konfigurační management a release proces	Continuous integration	Na jenkinsu se nebuildí automaticky feature větve ani release větve	Můžeme rozbít build nebo testy a dozvíme se o tom za delší dobu nebo při pokusu o merge do trunku. Aktuálně se většina kódu implementuje do společného trunku a merge probíhá zpětně do starších verzí kvůli patchi, ten se testuje samostatně. Bylo tu historicky jen několik feature větví pro experimenty, ty se ale mergovaly "pod dohledem". Workflow bude při použití GITu jiné.	Viz migrace na Git - bod 13 a Git flow - bod 36. Zajistit automatický vznik build konfigurace, jakmile vznikne feature větev. Zamezit push do dev a master větve.	IT admin	13, 36	1	in progress
12	x	střední	střední	Konfigurační management a release proces	QA	Do minor dodávky se dodávají kromě bugfixu i features, ale nedělá se endzend akceptační test a je to na tom vývojáři, že si to po sobě otestoval. Prý to "nemůže nic rozbít".	Do dodávky se může dostat chyba.	Před dodávkou by měl být stanoven code freeze - povinnost test managera a DM + TL. Před dodávkou by mělo probíhat standardní kvalifikační testování s vybranou množinou testovacích dat a scénářů - test manager určí a bude evidovat tuto množinu. Toto nevylučuje, že by se mělo testovat průběžně a ne jen před dodávkou, aby byl čas na opravu chyb a aby se stihlo otestovat vše, co se má otestovat (viz test plan). V první vlně můžeme aspoň dělat ruční smoke test - cca 2MD Ve druhé vlně chceme tyto testy co nejvíce automatizovat, aby nám zůstávala velká flexibilita v dodávání minor	testeři	44, 45	2MD na manuální smoke test až 30MD na automatizaci	done
15	x	vysoká	snadná	PM	Issue tracking	Tasky v Jira mají velkou granularitu, některé nejsou evidovány vůbec. Některé úkoly nemají časový odhad	- Nelze sledovat progres, nelze zajistit adekvátní množství času spáleného při plnění úkolu. - K velkému úkolu je těžké napsat relevantní zadání, vůči kterému by se dal validovat výstup. - Vznikají úkoly, které nebyly plánovány v projektovém plánu. - Release nemá zaručen termín dokončení nebo není zaručeno, že se stihne to, co se chtělo.	-Všechny úkoly by měly být evidovány v Jira a mít odhad pracnosti v rozsahu cca 1-5MD. - Odhad by měl odpovídat evidenci v projektovém plánu - Odhady pro režijní činnosti a bugfixing bude postupně laděn dle metrik z předešlých releasů. - Větší úlohy dekomponovat na menší. - Provázat je pomocí vazeb a nastavit jim komponenty, tagy a prioritu. - Nemělo by se pracovat na ničem, co nemáme v projektovém plánu (vyjma bugfixů a úkolů, které nemohly být předvídány, nicméně na nepředvídatelné úkoly by měl existovat buffer v projektovém plánu). - Je nutné nechat možnost založit nepředvídatelný úkol a prioritizovat ho do stejného releasu na úkor jiných úkolů, aby nebobtnala pracnost na zadaných úkolech, které s tím nemají nic společného. - Se všemi proběhla schůzka na založení tasků, rozpad	PM, TL, DEV	-	20 MD (20 lidí * 2 plánovací schůzky po hodině * 4 lidi na schůzce)	done

Jak na to?

Fáze 2: Plánování změny

- Vyčíslení nákladů a přínosů
  - Ušetřit něco stojí
  - Příklad: Zakoupení IntelliJ Idea
  - Příklad: Redesign modulu pro sémantickou analýzu Java byte-kódu

Jak na to?

Fáze 2: Plánování změny

- **Příklad vyčíslení nákladů a přínosů: Zakoupení IntelliJ Idea**

Problém: Vývoj FE + BE ve 2 IDE; build a redeploy celé aplikace 10 * 3 minuty => 30 minut denně

Náklady

Za rok vývojář odpracuje 12 * 20dní v release * 80% AVG úvazek * 80% utilizace = 153dní

Cena licence/den = 12.300Kč/153dní = 80Kč/den

Vyplatí se to?

1 IDE místo 2, hotswap místo build + deploy.

Za den musí ušetřit min. 80Kč -> Ano, ušetří půl hodiny.

Efektivita: 80Kč/denní plat XXXKč ~ malé jednotky procent

I pokud to zlepší efektivitu o malé jednotky %, tak se to vyplatí.

Jiný zdroj: ROI studie https://www.jetbrains.com/lp/idea-forrester-tei (návratnost do půl roku)

PROFINIT

Jak na to?

Fáze 2: Plánování změny

- **Příklad vyčíslení nákladů a přínosů: Redesign Java kódu**

Problém: experimentální kód (čti extrémně složitý, nezdokumentovaný, neudržovatelný, bez testů, absence OOP)

Přínos: možnost dál pokračovat ve vývoji

Náklady

20 kLOC

Odpracováno: 2 FTE na 1,5 roku = 2 * 1,5 * 12 * 21 * 80% utilizace = 605MD

Aktuální predikce dle proj. plánu: 600MD

Zajímavost: COCOMO man-month: 2.4 * kLOC^1.05 -> 2,4 * 20^1,05 = 56MM -> 56MM * 21dnů = 1.100MD

Vyplatí se to?

Srozumitelný, kvalitní, robustní, otestovaný kód. Původní kód bychom pravděpodobně zahodily později a vývojáři by byli nespokojení.

PROFINIT

Jak na to?

Fáze 2: Plánování změny

- Jde vždy vyčíslit přínos?
- Nastoupil jsem -> změřil jsem -> navrhl jsem opatření -> zavedl jsem
- Příklad: Zavedení statické analýzy kódu, zavedení code review, zlepšení kvality dokumentace, dlouhodobější aktivity, spokojenost práce s moderními technologiemi a nástroji
- Citovka a selský rozum

Jak na to?

Fáze 2: Plánování změny

- Zlepšení je taky projekt – business case, scope, zdroje, čas, fáze, milníky, rollout
- **Ukázka**: Plán migrace na Git

Fáz	Podfáze	Oblast	Detailní úkol	Kdo řeší	Odhad [MD
2	2	Analýza - rozdělení do více repo	schůzka / připomínkování	Architects	0,8
2	3	Analýza - rozdělení do více repo	zapracování připomínek	MT	0,5
3	1	Analýza - Branching model	Analýza jaký zvolit branching model - trunk-based-developement vs. git flow vs. something else, bude se to odvíjet taky od struktury složek. - s ohledem na vývoj, IDE, code review, testing, build v Mavenu, release, CI, hotfixy - ovlivní postup migrace - Zohlednit diskuze v DEV-3949	MT	1,5
3	2	Analýza - Branching model	schůzka / připomínkování	Architects	2,0
3	3	Analýza - Branching model	zapracování připomínek	MT	0,8
4	1	Analýza - struktura složek	Analýza struktury složek - už nebude checkoutováno branches / trunk / tags pro každý "modul" - vadí to něčemu? To by znamenalo, že branch bude vždy na nejvyšší úrovni - ovlivní postup migrace - je požadována jiná změna struktury složek? Podle mě ne. - dopady do buildu, releasu, CI, utilit	MT	1,5
4	2	Analýza - struktura složek	nastavení oprávnění v Gitu na úrovni složek - zadání kdo kam, pokud se bude měnit struktura, jinak zůstane jako v SVN	Admin	0,7
4	3	Analýza - struktura složek	schůzka / připomínkování	Architects	1,0
4	4	Analýza - struktura složek	zapracování připomínek	MT	0,6
5	1	Analýza - dopady na Maven a release proces	- změna SCM tagu - šlo by nechat jen v rodičovském POMu?	MT	0,3
5	2	Analýza - dopady na Maven a release proces	- popis dopadů na release proces - major, minor, hotfix - aktuální stav vs nový stav	MT	2,5
5	3	Analýza - dopady na Maven a release proces	- jak se budou buildovat jednotlivé typy větví (develop a feature branch)	MT	0,0
5	4	Analýza - dopady na Maven a release proces	- POC - vyzkoušet lokální build a testy z gitu pro různé branche, tranzitivní závislosti	MT	2,0
5	5	Analýza - dopady na Maven a release proces	připomínky a schválení	Architects	1,0
5	6	Analýza - dopady na Maven a release proces	zapracování připomínek	MT	0,6
6	1	Analýza - dopady na CI	- popis buildu různých typů větví (develop a feature branch ano, pull request není podmínkou) automaticky? jenkins file?	MT	2,0
6	2	Analýza - dopady na CI	- jak na build všech tranzitivních závislostí (chceme zachovat AS-IS stav)	MT	3,0

WBS | Pracnost po lidech | Pracnost po fázích | Prerekvizity

WBS plánu migrace na Git

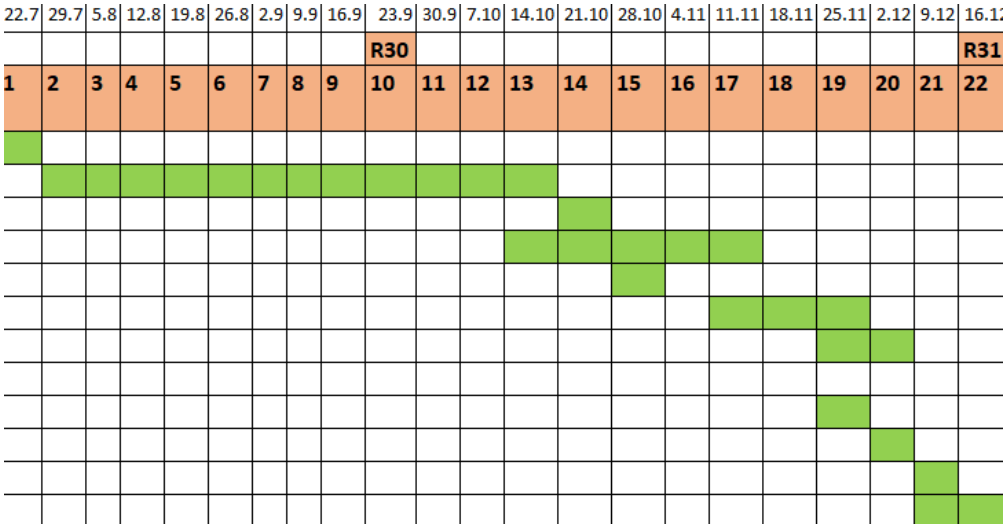
PROFINIT

Jak na to?

Fáze 2: Plánování změny

Fáze	MD	%	Oblast	Hlavní zodpovědnost	Reálná alokace [FTE]	Čas [T]
1	10	5%	Plán migrace	DM	0,4	6
2 - 9	43	23%	Analýza	MT	0,9	12
10	36	19%	Školení na git (a gitflow)	DEV	16,2	1
11	19	10%	Implementace - dopady do utilit	MT	0,9	5
12	3	1%	Administrace	Admin	0,15	4
13	11	6%	Migrace	MT	0,9	3
14	5	2%	Po migraci	MT	0,9	2
15	0	0%	Out of scope: Integrace nástrojů		-	-
16	22	12%	Lokální instalace Gitu	DEV	16,2	1
17	31	16%	Poinstalační podpora při řešení problémů	DEV	16,2	1
18	2	1%	Zpětná vazba	DEV	16,2	1
19	7	3%	Dokumentace na Confluence	MT	0,9	2
Total	**186**	**100%**				

Pracnost fází



Harmonogram

Jak na to?

Fáze 3: Rollout

- U rizikových změn ideálně postupný
- S možností rollbacku
- Příklad: Zavést Git nejprve u jednoho vývojáře / v 1 týmu

PROFINIT

Jak na to?

Anti-patterns: Jak naštvat kolegy

- „Zlepšováky jsou nejdůležitější. Vývoj počká“
  - Respektovat i jiné priority
- „Chápu, že opravujete chyby do release. Ale chci, abyste taky začali dělat průběžné code review“
  - Zvolit vhodné načasování
- „Proč jedete na Javě 8? Upgradujeme na Javu 11“
  - Chápat důvody proč je to zrovna takhle, vyslechnout si a rozmyslet
- „Prostě to tak dělejte“
  - Vysvětlit důvod a ozřejmit přínosy

- Někdy to může být i destruktivní (např. zahození 2 let práce na Java scanneru)
- „Proč mám mít task na každou drobnost a psát k tomu odhady?“
  - Nezavděčíte se všem

PROFINIT

Jak na to?

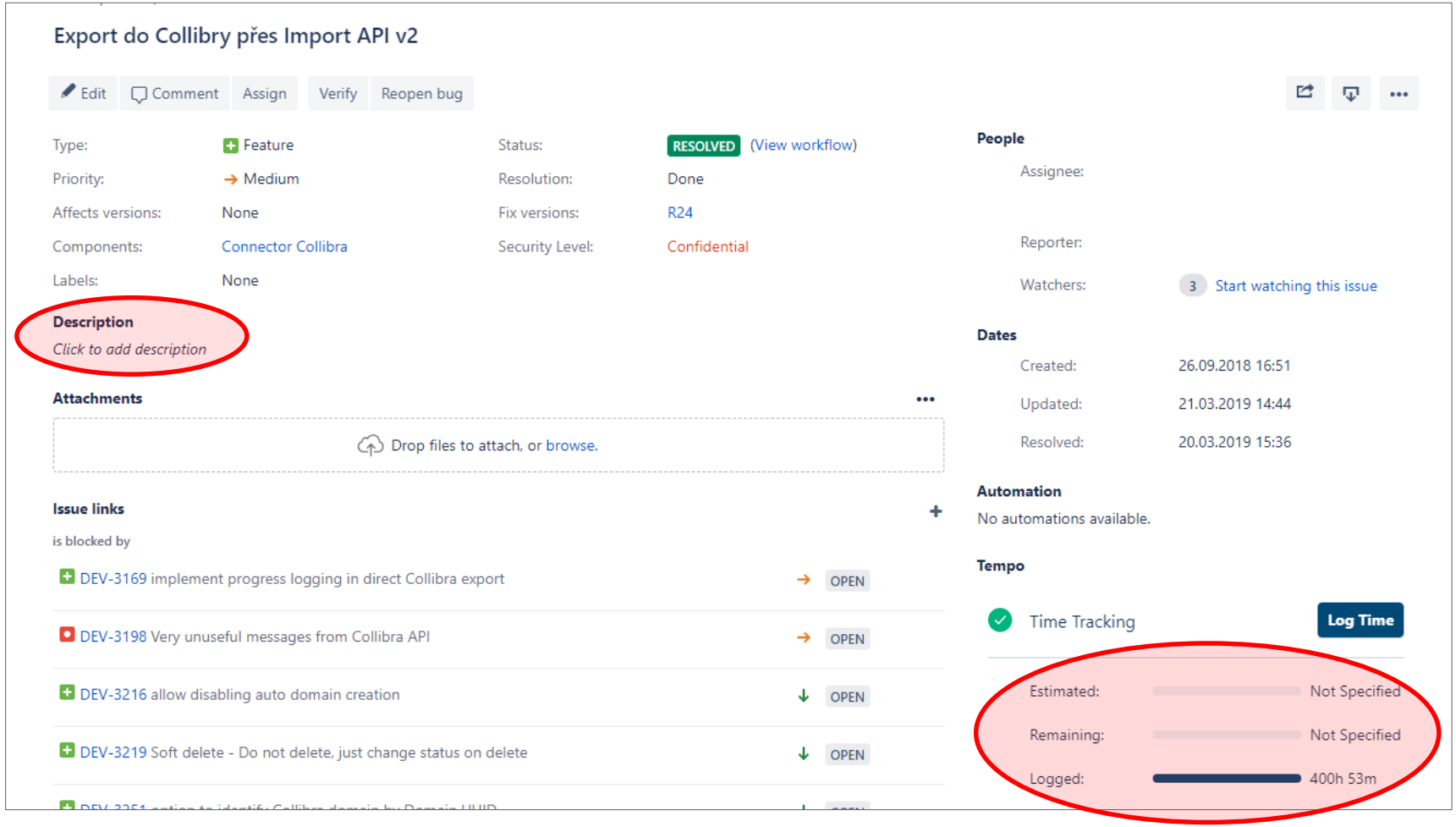
Fáze 4: Vyhodnocení změny

- Žije se nám s tím lépe?
- Šetří nám to čas / peníze / problémy?
- Změnit a přizpůsobit (někdy i individuálně)
  - Ukázka

PROFINIT

Příklad 1: Issue tracking

- Hlavní motto: „Vědět, co se má dělat“
- **Původní stav:**



Příklad 1: Issue tracking

- Není zadání
  - Vývojář neví přesně, co se po něm chce a co je cílem jeho práce
  - Jak zkontroluju splnění úkolu?
  - Podle čeho mám napsat analýzu a testovací scénář?
- Odhady
  - Velká granularita
  - Neodhaduje se, neměří se, neví se, co se stihne
  - Už jsi vykázal? Aktualizoval jsi zbývající odhady?
- Scope
  - Neřiditelný
  - Úkoly vznikají průběžně dle potřeby
  - Některé tasky nejsou evidovány vůbec
- Workflow
  - V různých týmech se s tasky pracuje různým způsobem
  - Proběhl zde design?
  - Proběhne zde testování?
  - Bude to někdo revidovat?

Příklad 1: Issue tracking

Výsledek

- Všechny úkoly evidovány v Jira
- Stanoveno jednotné Jira workflow a nové stavy
  - TO ACCEPT – úvodní prioritizace tasku
  - IN REVIEW – revize výstupů
  - Nové pole SDLC phase – dekompozice tasku na analýzu, design, implementaci, testování, dokumentaci
- Odhady pracnosti
  - V rozsahu cca 1 až 5MD
  - Původní, aktuální, odpracováno
  - Pravidelné vykazování a aktualizace odhadů
- Samostatné tasky pro režijní činnosti
- Jira filtry na nekorektně vyplněná pole + nově vzniklé issues zařazené do release
  - Např. v R28 stav cca 3 tasky z 600 bez zadání

PROFINIT

Příklad 2: Project management

- Hlavní motto: „Dodat vše včas a v odpovídající kvalitě“
- PM by měl umět odpovědět na otázky:
  - Čím se na projektu tráví čas?
  - Kolik času potřebujeme k vývoji funkcí?
  - Co zvládneme dodat v příštím releasu?
  - Máme pro to dostatek vývojářů?
  - Kdo bude co dělat?
  - Jaká jsou rizika a jak je budeme řešit / předcházet jim?
  - Jak snížit počet produkčních chyb?
  - Stíháme vše do dalšího milníku?
  - Máme kapacitu na testování?
  - Testujeme dost nebo málo?
  - Kolik bude chyb z vývoje / QT / produkce vzhledem k objemu?
  - Jak velký to byl release?
  - Kde jsme nejvíc nestíhali a proč?

Příklad 2: Project management

PROFINIT

Příklad 2: Project management

- **Původní stav**:
  - TLs jsou zahlcení vývojem a opravou chyb a nemají čas na TL
  - Není detailní proj. plán -> nedokážeme systematicky, opakovaně a přesně odpovědět na výše uvedené otázky

Příklad 2: Project management

- **Výsledek:**
  - Změřeny režie – víme čím trávíme kolik času
  - Reaktivní způsob změněn na plánovaný
    - Uvolnili jsme si ruce pro TL, testování (2 týdny QT), code review, support, snižování tech. dluhu
  - Zaveden detailní projektový plán
  - WBS a odhady pracnosti vychází od týmů
  - Nepracuje se na ničem, co není v projektovém plánu (na bugfixy a nepředvídatelné úkolů vyhrazen buffer)
  - Zavedeny týmové kick-offs a MVP mindset
  - Týdenní review vývoje s každým TL (odhady vs realita, termíny, chyby, kvalita, scope)
  - TLs program
  - Historie release: metriky + lessons learned (zdroj: Q2 2020)

PROFINIT

Příklad 2: Project management

	Priority	SDLC phase	Status	Jira key	Jira link	Summary	Description	Assignee	Milestone	Done [%]	Worked [MD]	Remaining [MD]	Current estimate [MD]	Original estimate [MD]	Diff [MD]
706						DEV - Reporting									
707	High	Implementation	Verified	DEV-8121	DEV-8121	Support for Relationship/Redefine in SSAS Dimension	SSAS, in review	YPE	Feature Deadline	100%	16,0	0,0	16,0	4,0	12,0
708	Medium	Testing	Done	DEV-9724	DEV-9724	Bugs root cause analysis workshop - Reporting team		PKO	Development Freeze	100%	1,7	0,0	1,7	0,6	1,1
709	Highest	Testing	Done	DEV-4149	DEV-4149	Tests for current resolving and data flow (Cognos)	patch R29.1	PKO	Feature Deadline	100%	3,1	0,0	3,1	1,0	2,1
710	Medium	Documentation	Done	DEV-4328	DEV-4328	Configuring new cube nodes (Cognos)	patch R29.1	PKO	Release	100%	0,6	0,0	0,6	0,8	-0,1
711	High	Implementation	Done	DEV-9690	DEV-9690	Dataflow from physical sources with deduction (Cognos)	patch R29.1	PKO	Feature Deadline	100%	2,0	0,0	2,0	2,0	0,0
712	Medium	Implementation	Done	DEV-9691	DEV-9691	Virtual cubes data flow (Cognos)		PKO	Feature Deadline	100%	4,4	0,0	4,4	2,0	2,4
713	Medium	Implementation	Done	DEV-9694	DEV-9694	Remaining flows between cube objects (Cognos)	patch R29.1	PKO	Feature Deadline	100%	1,7	0,0	1,7	1,5	0,2
714	Medium	Implementation	Done	DEV-9695	DEV-9695	Nested and aggregate cubes (Cognos)	patch R29.1	PKO	Feature Deadline	100%	0,8	0,0	0,8	1,0	-0,2
715	High	Requirements and Analysis	Done	DEV-7065	DEV-7065	Talend - Manual columns mapping functionality analysis		MBU	Feature Deadline	100%	0,8	0,0	0,8	0,5	0,3
716	High	Implementation	Done	DEV-7066	DEV-7066	Talend - Manual columns mapping functionality		MBU	Feature Deadline	100%	1,7	0,0	1,7	0,8	1,0
717	High	Documentation	Done	DEV-7067	DEV-7067	Talend - Manual columns mapping functionality		MBU	Release	100%	0,2	0,0	0,2	0,1	0,1
718	Medium	Testing	Done	DEV-7068	DEV-7068	Talend - Manual columns mapping functionality testing		MAB	Feature Deadline	100%	0,3	0,0	0,3	0,3	0,0
719	Medium	Implementation	Done	DEV-7738	DEV-7738	Update Talend to use the new QueryService API		MBU	Feature Deadline	100%	0,7	0,0	0,7	0,5	0,2
720	Low	Implementation	Verified	DEV-8485	DEV-8485	StreamSets Transforming/non-transforming flag for		MBU	Feature Deadline	100%	0,8	0,0	0,8	1,5	-0,7
721	Medium	Testing	Done	DEV-9652	DEV-9652	StreamSets Extractor tests - register testing pipelines on SDC DEV-7		MBU	Feature Deadline	100%	0,4	0,0	0,4	0,4	0,0
722	Low	Implementation	Verified	DEV-9653	DEV-9653	StreamSets Dataflow Generator - fix JDBC stage		MBU	Feature Deadline	100%	0,3	0,0	0,3	0,3	0,0
723	High	Implementation	Verified	DEV-9336	DEV-9336	[StreamSets] Refactor logging to use new framework	Streamsets - logging	MBU	Feature Deadline	100%	3,1	0,0	3,1	2,5	0,6
724	High	Implementation	Verified	DEV-9337	DEV-9337	[Talend] Refactor logging to use new framework	Talend - logging	MBU	Feature Deadline	100%	5,5	0,0	5,5	2,5	3,0
725	Low	Implementation	Verified	DEV-4914	DEV-4914	XML reader to suggest encoding as per DEV-4026		JPO	Feature Deadline	100%	1,4	0,0	1,4	0,6	0,8
726	Medium	Design	Verified	DEV-6315	DEV-6315	Mapping to 3rd party tools (QlikSense)		AJU	Feature Deadline	100%	0,7	0,0	0,7	0,8	-0,1
727	Low	Implementation	Verified	DEV-6352	DEV-6352	Filtertask (Qlik Sense)		AJU	Feature Deadline	100%	0,3	0,0	0,3	0,5	-0,2
728	High	Implementation	Verified	DEV-6358	DEV-6358	Validation of connection (Qlik Sense)		AJU	Feature Deadline	100%	0,6	0,0	0,6	0,5	0,1
729	Low	Documentation	Verified	DEV-6361	DEV-6361	Knowledge base documentation (Qlik Sense)		AJU	Release	100%	1,0	0,0	1,0	1,0	0,0
730	Low	Requirements and Analysis	Done	DEV-6655	DEV-6655	Finding minimal permissions for extraction (Qlik Sense)		AJU	Feature Deadline	100%	0,4	0,0	0,4	0,5	-0,1
731	Medium	Implementation	Verified	DEV-8228	DEV-8228	Add Aggregation Scope keyword recognition in parser (QlikSense)		AJU	Feature Deadline	100%	0,5	0,0	0,5	0,5	0,0
732	High	Requirements and Analysis	Verified	DEV-8418	DEV-8418	Resolver completeness check (Qlik Sense)		AJU	Feature Deadline	100%	0,5	0,0	0,5	0,8	-0,3
733	Medium	Implementation	Verified	DEV-8610	DEV-8610	Add Expression Evaluator feature to process expressions in parentheses (Qlik Sense)		AJU	Feature Deadline	100%	0,3	0,0	0,3	0,3	0,0
734	Medium	Testing	Done	DEV-8744	DEV-8744	Verification of DEV-6355 and DEV-8142 (Qlik Sense)		ANO	Release	100%	0,5	0,0	0,5	0,4	0,1
735	Highest	Testing	Done	DEV-8754	DEV-8754	Verification of Qlik Sense Generator - Load Script and Presentation Layer		AJU	Feature Deadline	100%	1,6	0,0	1,6	1,3	0,3

... | Milestones | WBS | Jira | Resources | Commitments vs resources | Plan vs reality | Risks | Configuration

WBS

Příklad 2: Project management

	Původní odhad %	Realita %
Requirements and Analysis	6%	6%
Design	4%	5%
Implementation	32%	40%
Documentation	3%	3%
Testing	11%	10%
PROD support (bugs, POC, HD)	6%	2%
QA, reviews, SPI	6%	5%
PM & TL	8%	8%
Other	9%	9%
Release, config. mngmt.	1%	1%
Risk	4%	4%
Bug	9%	8%

Pracnost SDLC fází – plán vs realita

PROFINIT

Příklad 2: Project management

Metrika	Hodnota	Komentář
FTE	30	Někteří lidé najati a někteří odešli. Toto je průměr. Zahrnuje testery, PM.
Lidí	40	
Celkem MDs	1 751	
Implementační MDs	700	Čas odpracovaný na implementaci. Opravy chyb nejsou zahrnuty
Zdrojáků celkem	21 288	
kLOC celkem	1 600	
kLOC přírůstek	166	SLOC-P přírůstek (bez komentářů, prázdných řádků, unit testů, testovacích dat, generovaných zdrojáků)
Přírůstek features	286	Celkem implementovaných funkcí
Nárůst scope po začátku vývoje	23%	
Přetečení odhadů	15%	Vůči celému scope
Přesnost odhadů	63%	Zahrnuta nepřesnost oběma směry (přestřelení, podstřelení)

Základní projektové metriky

PROFINIT

Příklad 3: Testování

Původní stav

- Testují jen vývojáři, nejsou dedikovaní testeři – problém end to end funkčnost
- Průběžné testování se neprovádí nijak systematicky
- Kvalifikační testování
  - Málo organizované
  - Vývojáři si rozeberou k testování, kdo co chce
  - Bugy se někdy neevidovaly
  - Jen 1 týden
  - 5 vývojářů
  - Občas se nestihne vše otestovat ani opravit
  - Do minor dodávky se dodávají kromě bugfixu i features, ale nedělá se end2end kvalifikační test a je to na konkrétním vývojáři, aby si to po sobě otestoval po vývoji. "Nemůže to nic rozbít"
- TC nejsou
- Testovací prostředí shodné s DEV prostředím
- Metriky – neměří se

Příklad 3: Testování

Výsledek

- 3-4 dedikovaní testeři
- Průběžné testování (dry run)
  - Feature i bug má testovací subtask, otestuje se ihned po vývoji
- Kvalifikační testování
  - Plán QT – scope, zdroje, tasky, testovací kick-off a ranní standupy lidí z test poolu
  - Prodlouženo na 2 týdny
  - Evidence bugů
  - Testeři + zapojeno cca 10 vývojářů
  - Stíháme otestovat a opravit min. 90%
  - Před minor dodávkou end2end test

PROFINIT

Příklad 3: Testování

Výsledek

- TC – vznikají pro nový vývoj, evidence v TestLink
- Testovací prostředí odděleno od DEV a zdokumentováno
- Metriky
  - Nová pole polí v Jira, která umožňují systematicky měřit
  - Vyhodnocovat a zlepšovat metriky z testování
    - Velikost vývoje (kLOC, počty modulů, změn, FTE)
    - Počty chyb v různých fázích (dry run / QT / produkce)
    - Zastoupení severit
    - Nejchybovější komponenty
    - V závislosti na velikosti releasu
  - Podniknout opatření
  - Predikovat počty chyb
- Automatizace testování (end 2 end) teprve v začátcích

PROFINIT

Příklad 3: Testování

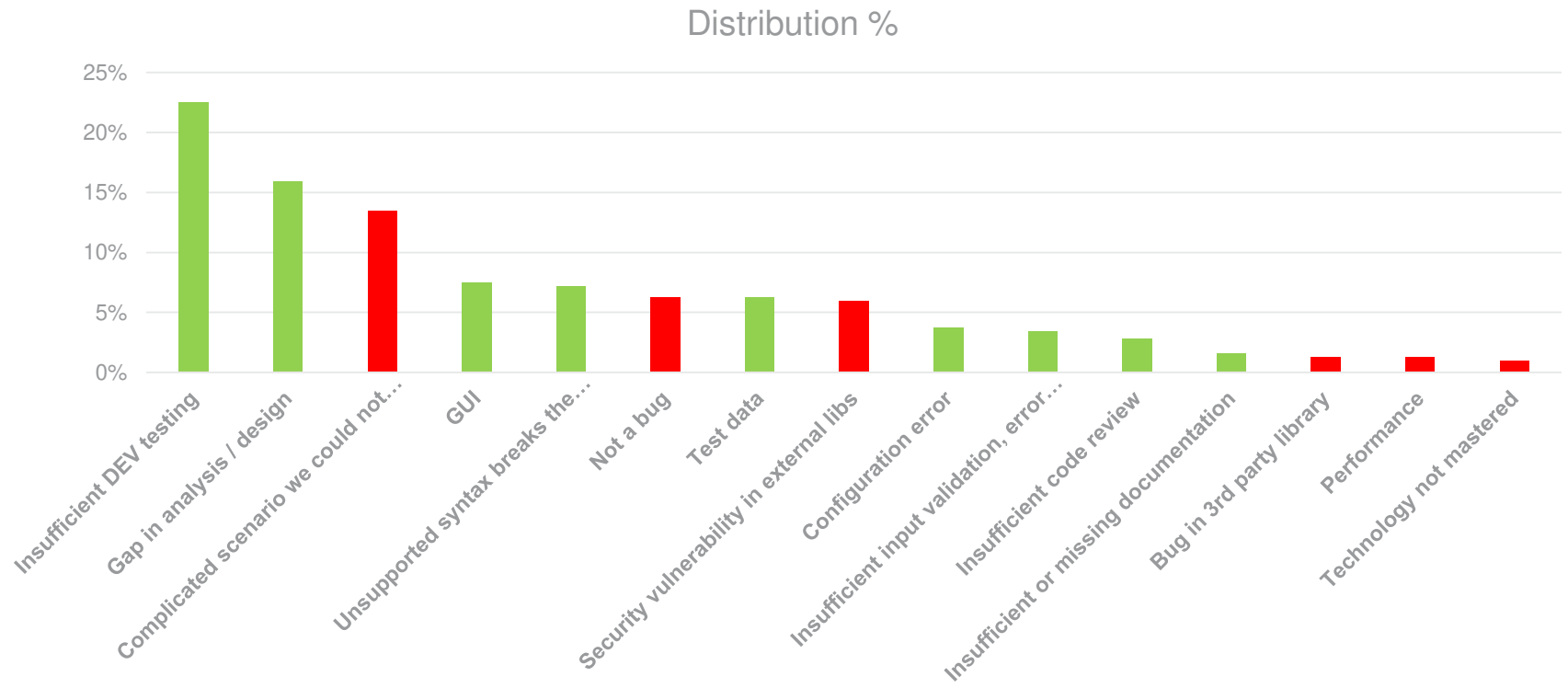
- Ukázka: Sledujeme cca 40 metrik

Metrics / Release	R27	R28	R29
	222	158	324
QT bugs			
# QT valid bugs	72	43	36
# fixed	50	17	36
# not finished	22	26	0
# QT bugs invalid	7	6	14
Comments	*QT length 2 weeks paid off, more developers joined testing. Many bugs were not fixed. Feature freeze established although not fully followed.*	*Less bugs found because the delivery was smaller (no scanners delivered) and thus the testing effort was intentionally smaller. Many low-severity bugs (mainly Postgres) revealed but not fixed (deliberately postponed).* *- Feature freeze practiced and almost completely respected (very much limited last minute development was allowed). Eliminate even these rare examples (better planning).*	*;- Less bugs found despite the big production delivery* *- Good job done during dry run testing.* *- First time in hoistory, all QT bugs have been fixed.* *- Quite a lot invalid bugs. No single cause found.*
Production bugs and DRE			
# PROD bugs (valid bugs from PROD)	46	38	91
# all bugs (valid bugs from PROD & DEV)	222	158	324
DRE	79%	76%	72%
Comments	*Significantly less PROD bugs probably caused by much intense DEV testing and lower usage by our customers (only few upgraded to R27). Much better DRE. Target min DRE is 90-95%.*	*- We follow the trend of reasonable number of production bugs. We have even improved slightly in DRE.* *- We would like to predict the number of POC which produce big part of the PROD bugs.*	*- We have probably improved the DRE slightly if the number of PROD bugs don't exceed the expected amount of 50 bugs.* *- we had huge amount of bugs, mostly Dry run*

PROFINIT

Příklad 3: Testování

- **Ukázka: Analýza příčin chyb**



- 320 chyb z vývoje a PROD
- Vývojáři kategorizovali chyby a navrhovali protiopatření
- Dlouhodobý cíl: snížit počet chyb o 30%

PROFINIT

Doporučení na závěr

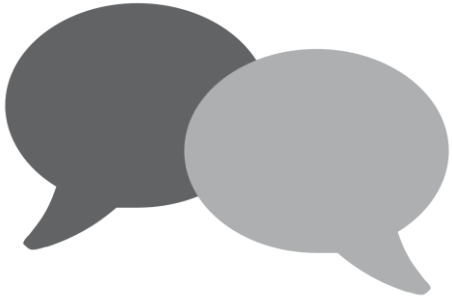
- Pohled: teď nás to zdrží, ale ve výsledku to pomůže
- Každý projekt je unikátní, technologie přichází a odchází, principy jsou pořád stejné
- Poznejte celý SW proces, nejen implementaci
  - Začněte co nejdřív, dokud jste si nezvykli nešvary přehlížet
- Ptejte se na názor více lidí, otevřenými ale konkrétními otázkami
  - Nefunguje: „Co bys zlepšil?“
  - Lepší: „Kolik bylo chyb z vývoje a kolik jste jich stihli opravit a otestovat do konce release?“
  - Lepší: „V jakém modulu máte nejmenší pokrytí testy a kolik je to procent?“
- Používejte checklisty – např. Profinití minimální nároky
- Retrospektiva: tvořte si svůj vlastní checklist
  - Z každého projektu si odneste získané poznatky a ponaučení

PROFINIT

Doporučení na závěr

- Zjistěte, kde vás tlačí bota nejvíc
- Podpořte svůj návrh jasnými argumenty a/nebo vyčíslete přínos
- Je skvělé mít v týmu „rýpala“ - vrátí vás na zem
- Přínos si vydělte dvěma
- Zavedením změny to nekončí, vymyslete kontrolní mechanismus
- Jedno po druhém
- Můžete začít v malém, ve svém bezprostředním okolí a postupně
- Zapojte do hry kolegy

PROFINIT



Diskuze

Děkuji za pozornost

PROFINIT
AHEAD THROUGH KNOWLEDGE

Profinit EU, s.r.o.
Tychonova 2, 160 00 Prague 6 | Phone + 420 224 316 016

Web
www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU

Facebook
facebook.com/Profinit.EU

Youtube
Profinit EU