

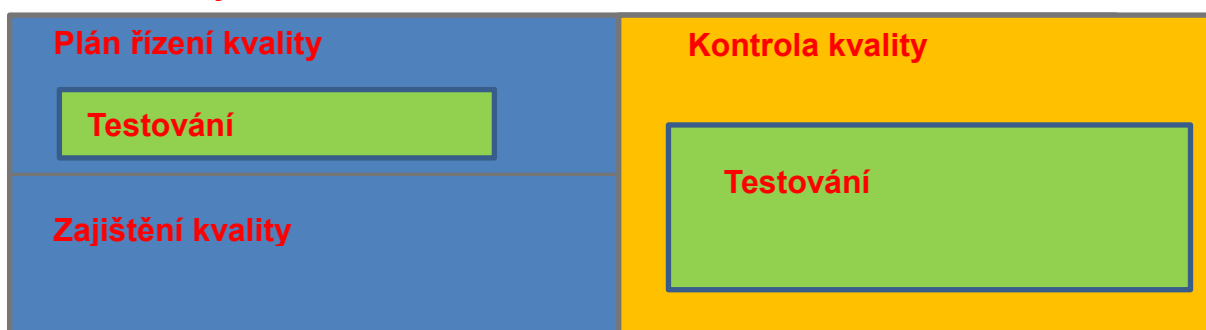
Testování v procesu implementace informačního systému

V minulém čísle jsme se zabývali řízením kvality v procesu implementace informačního systému. Z důvodu rozsahu jsme pouze velmi letmo zmínili testování. Protože je to jedna z klíčových činností při implementaci informačního systému (IS) věnuji mu samostatný článek.

V článku vycházíme z předpokladu, že implementace informačního systému je v drtivé většině implementací standardního software dodávaného softwarovou společností mnoha zákazníkům a při implementaci dochází k jeho parametrizaci a pouze z malé části k vytváření nového či změně dodaného standardního programového kódu. Většina používaného kódu pochází tedy ze standardní dodávky, prošla již interními testy softwarového dodavatele a případné chyby v něm nalezené jakýmkoli zákazníkem jsou opravovány prostřednictvím dodávaných patchů. To samozřejmě neznamená, že ve standardním kódu nemohou být chyby, jejich výskyt je však méně pravděpodobný než u kódu vytvořeného během implementace.

Existuje mnoho definic *testování* případně *testování softwaru*. Jedna z nejčastěji užívaných se objevuje i na wikipedii (byť jinak je tam heslo „Testování softwaru“ popsáno dost zmatečně): „Testování softwaru je empirický technický výzkum kvality testovaného produktu nebo služby prováděný za účelem poskytnutí těchto informací všem zainteresovaným (=stakeholdrům).“ Jak už to obvykle s takovými definicemi bývá, není na první pohled příliš srozumitelná. Podstatné je, že se zkoumá **kvalita** testovaného produktu, tj. v našem případě informačního systému. A kvalita je stupeň splnění požadavků, které jsou na tento informační systém kladeny. Z pohledu metodiky projektového managementu je tedy testování součástí řízení kvality a to v procesech Plán řízení kvality (plánování testování) a Kontrola kvality (angl. Quality Control). Viz obrázek.

Řízení kvality



Obrázek 1 Testování jako součást řízení kvality

V souvislosti se sílícím tlakem na kvalitu systémů roste i „atraktivnost“ testování pro IT management. Na druhou stranu se testování v praxi často ukazuje jako drahé, vyžadující velké úsilí (někdy až neproveditelné) a někdy neúčinné (chyby systému nejsou odhaleny před zahájením produktivního provozu). Tyto negativa lze však výrazně omezit použitím správné metodiky testování a vhodného nástroje na plánování, řízení, provádění a vyhodnocování testů a na řízení defektů (chyb).

Testování software je svébytným oborem, o kterém byla v posledních letech napsána řada knih a zabývá se jím řada norem. Za všechny jmenujme alespoň ISO/IEC 29119 Software Testing. My se v tomto článku budeme věnovat spíše praktickým stránkám testování při implementaci IS.

Mýty o testování software

Kolem testování existuje řada mýtů a zavádějících tvrzení. Navíc projevy řady chyb v řízení testování často ani nebývají spojovány právě s test managementem, ale jejich příčiny se hledají jinde.

Proberme si alespoň nějaké mýty související s testováním při implementaci IS:

- Řízení kvality = testování. Realita: testování je pouze jedna součást řízení kvality (viz obrázek 1)
- Cílem testování je zajistit, aby software byl bez chyb. Realita: cílem testování je minimalizovat počet chyb a zajistit, že software je v souladu s požadavky. Nedává však nikdy jistotu, že je bez chyb. Stará programátorská pravda říká, že v každém programu je chyba.
- Každý může testovat. Realita: testování je disciplína, která vyžaduje určité znalosti a dovednosti.
- Pokud chyba není objevena při testování, je to vina testera. Realita: za kvalitu IS je zodpovědný celý implementační tým (analytici, vývojáři, autoři test skriptů, klíčoví uživatelé, management projektu,...). Záleží na konkrétní situaci.
- Automatizované testy stoprocentně nahradí manuální testy. Realita: v některých případech se manuálním testům nelze efektivně vyhnout.
- Testování je vždy drahé. Realita: Vhodná organizace testů a použité nástroje mohou výrazně snížit náklady na testování. Navíc chyba nalezená v testech je mnohem lacinější než chyba nalezená v produkci.
- Testování může začít až po dodání aplikace. Realita: je tomu právě naopak. Kvalita je integrální součástí činností (není to revize) a tedy testování se provádí již v průběhu vývoje.

Typy testů

Existuje mnoho typů testů dělených z různých hledisek. Testy se dělí například na statické a dynamické. Statické jsou takové, při kterých není potřeba software spustit. Patří k nim například revize dokumentace, code review apod. Používají se zejména při vývoji software na zakázku. To není typický scénář implementace IS, proto se nadále budeme věnovat pouze dynamickým testům, kdy je software spuštěn a je posuzováno jeho chování.

Další dělení testů je Whitebox vs. Blackbox. Blackbox testy přistupují k aplikaci výhradně jako k celku, testuje se jen přes uživateli přístupná rozhraní (GUI, webservices,...) a nesbírají se informace o vnitřním chování aplikace. Whitebox testy využívají informace o vnitřním programovém uspořádání aplikace. Většinou se také používá pouze standardní rozhraní aplikace, ale znalost vnitřního řešení se využívá pro optimalizaci sady testů tak, aby byly pokryty všechny větve/cesty v kódu. To při blackbox testingu nelze, tam se musí procházet všechny varianty dle obchodních procesů. Z principu jsou tedy whitebox testy doménou vývojářů a uplatňují se zejména při vývojářských unit testech.

V dalším textu se budeme zabývat typickými typy testů při implementaci IS: funkčními (unit testy, testy scénářů a integrační testy, regresní testy) a technickými (zátěžové, backup/restore, security, tisky,...).

Strategie a plán testování

Pojďme se nyní podívat na činnosti související s testováním dle typického průběhu projektu. Přestože v posledních letech je velmi populární Agilní projektový management, nejčastější jsou stále projekty typu waterfall. V tomto článku budeme předpokládat, že se jedná o waterfall, pro agilní projekty by se však postupovalo obdobně.

Kdy v projektu začít myslet na testování? Určitě ne v době, kdy už je systém nastaven. Čas začít s testováním je již na samém počátku projektu. Plán testování je jako součást kvality management plánu součástí celkového plánu projektu.

Někdy se rozlišuje strategie testování a plán testování. Ve většině běžných projektů však dle mých zkušeností stačí připravit a schválit pouze jeden dokument a není nutno věnovat zbytečně mnoho času vytváření strategického dokumentu. V každém případě je vstupem pro plán testování analýza stavu projektu v oblastech:

- Obecné požadavky a standardy testování v organizaci
- Harmonogram projektu
- Kritičnost aplikace / riziko z hlediska dopadu na provoz organizace
- Nástroje, které jsou k dispozici pro řízení testů (včetně defektů) a pro vlastní testování
- Disponibilita testovacího prostředí a testovacích dat (v době, na kterou je testování plánováno)
- Kapacita a zkušenosti týmu testerů

Plán testování by měl obsahovat kompletní informaci o tom, jak se bude testovat. Měl by obsahovat:

- Cíle a požadavky testování v projektu
- Předpoklady
- Rozsah testování
- Typy testování
- Přístup k testování
- Nástroje pro provádění a řízení testů
- Pokrytí byznys požadavků
- Kritéria úspěšnosti testů
- Testovací prostředí
- Testovací data
- Defekt Management
- Role a zodpovědnosti

Podívejme se na některé body detailněji.

Rozsah testování: ne vždy je nutno testovat všechny funkcionality a někdy to ani z důvodu rozsahu není možné. Je však třeba na základě analýzy rizika předem stanovit, jaké funkcionality není nutno

otestovat (nebo je otestovat pouze namátkově) a jaké dopady mohou mít chyby v neotestovaných funkcionalitách.

Typy testů a Defekt Management rozebíráme v samostatných kapitolách.

Přístupem k testování miníme organizační stránku a postupy provádění testů.

Nástroje pro řízení testů: Pro řízení testů je potřeba mít nástroj, ve kterém se bude plánovat a evidovat seznam testovacích případů, stav jejich přípravy, přiřazení testerů, výsledky testů, evidenci testovacích defektů (chyb) a průběh jejich řešení. Pokud se jedná o malé množství testovacích případů, dá se jako nástroj použít vhodně navržená excelovská tabulka. Pokud množství testů překročí stovku, pak evidence v excelu je práce pro otrlé jedince (vím to z vlastní zkušenosti). Naštěstí existuje několik softwarových nástrojů vyvinutých pro řízení testů, které ušetří značné množství času a lidských chyb. Je jen otázkou analýzy, kolik testů se bude v budoucnu provádět a zda se tedy pořízení a implementace specializovaného nástroje vyplatí.

Nástroje pro provádění testů: existují nástroje pro různou míru automatizace testů. Od přípravy seznamu práce pro jednotlivé testery, přes automatizaci provádění testovacích skriptů až po kompletní automatizaci. Špičkové nástroje umí nejenom provést skript v testované aplikaci, ale též výstupní data (jako například čísla vytvořených dokladů) předat jako vstupní data dalšímu skriptu, který je automaticky spuštěn na základě výsledku předchozích skriptů. V ideálním případě je možno připravit automatizovaný plán spouštění řetězců skriptů, večer jej spustit a ráno si v logu přečíst výsledky.

Automatizované testování – zní to skvěle, že? U mých klientů to zpočátku většinou budilo nadšení. Při implementaci IS však není nic zadarmo. Implementovaný software musí umožňovat spouštění automatických skriptů, musíme pořídit nástroje pro jejich tvorbu a spouštění a hlavně musíme skripty vytvořit. Na rozdíl od manuálně prováděného testování, kde se dá předpokládat, že dobrý tester použije též hlavu a tedy při popisu skriptů nemusíme být tak detailní, u automatických skriptů musíme předem přesně specifikovat, co se má provést a jak ověřit správnost. No a když už tohle všechno máme a těšíme se, že už můžeme opakovaně testovat bez nákladů, tak se aplikace změní (nahraje se patch, provede upgrade,...) a skript možná přestane fungovat.

Přestože to podle předchozího odstavce nevypadá, nejsem proti automatizovanému testování.

Naopak, znám z praxe několik případů, kdy se s úspěchem používá a znamená významnou úsporu nákladů. Opět, jako mnohokrát v tomto článku řekneme, že je třeba zanalyzovat, zda se to v konkrétní situaci vyplatí. A ještě jedna poznámka: v případě SAP a použití nástrojů SAP pro automatické testování se uvádí, že vytvoření automatických skriptů se vyplatí při více než třech opakování testů, což v praxi docela běžně nastává. Obvykle však od klientů slyším: „Je to skvělé a určitě se do toho pustíme. Teď ale budeme testovat manuálně, protože na vytvoření automatických skriptů nemáme čas“. A co se ozve při příštím testování, můžete hádat.

Kritéria úspěšnosti testů: Je třeba předem stanovit pro každé testy v projektu, za jakých podmínek je prohlásíme za úspěšné, a projekt se může posunout do další fáze. Jak jsme se již zmínili, není realistické očekávat, že všechny testy skončí bez chyb. Kritéria úspěšnosti se obvykle stanovují v závislosti na kategorii zjištěných chyb (viz níže).

Testovací prostředí se může lišit podle jednotlivých typů testů. Výhodou je, pokud máme k dispozici dedikované testovací prostředí, které je odlišné od vývojového i produktivního a v něm nastavení, které bude odpovídat produktivnímu prostředí v době spuštění aplikace. To může být mnohem komplikovanější, než se na první pohled zdá. Většinou nemáme k dispozici identickou kopii produktivního HW, problematique jsou rozhraní na další systémy a problém je řádově složitější, pokud provádíme implementaci do prostředí, kde již některé aplikace / procesy běží v produktivním

provozu a paralelně k našemu projektu běží jiné projekty a podpora provozu. Existují postupy, jak požadavky na testovací prostředí co nejlépe splnit i za takovýchto podmínek, ale to by vydalo na samostatný článek a nemáme zde pro to prostor.

Testovací data jsou další důležitou oblastí, kterou je třeba dobře naplánovat a provést. V průběhu implementace je nutno opakovaně testovat, přičemž řada testů může být vůči připraveným datům destruktivní. Například pokud testujeme platbu faktury, pak k jedné připravené faktuře lze úplnou platbu provést pouze jednou. Zejména je nutno testovací data dobře vyřešit pro vývojářské a unit testy, kdy dochází k řadě neúspěšných testů a pokusů a systém se tak zanáší chaotickými daty. Důležitá je též kvalita testovacích dat, aby se při testování jednoduše odlišila chyba aplikace od chyby dat. Univerzální návod bohužel opět neexistuje, možnosti závisí mimo jiné na systémovém prostředí a použitých technologiích. Většinou se používá sada „nedotknutelných dat“ v kombinaci s opakovaným generováním dat, na což dnes již také existují nástroje (byť se nedají použít vždy).

Role a zodpovědnosti: Klíčovou rolí je role test managera, který zodpovídá za všechny aktivity týkající se testování, provádí plánování testů a jejich monitoring a řízení. Dále je nutno definovat kdo a na základě jakých podkladů vytváří testovací scénáře respektive skripty, kdo je schvaluje, kdo testuje, kdo je zodpovědný za opravu nalezených chyb atd. Já považuji za klíčové, že testovací skripty schvalují (když už ne vytváří) klíčoví pracovníci klienta (vlastníci procesů, klíčoví uživatelé atd.). Pokud je připravuje dodavatel, připraví skripty tak, jak si myslí, že by systém měl fungovat a ne tak, jak si to představuje klient, což může být dost rozdílné. Jestliže jsou skripty napsány velmi podrobně, může testování provést i třetí strana. Většinou je však lepší, když i testeři jsou pracovníci klienta.

Rozsah dokumentu Strategie testování, respektive Testovací plán musí být samozřejmě úměrný velikosti a složitosti projektu. Nemá smysl pro malý projekt vytvářet stostránkový dokument a v něm sáhodlouze popisovat něco, co všichni účastníci projektu dobře vědí. Řídl jsem například testování na projektu, kde byly desítky členů projektového týmu a stovky testů. Přesto jsem si díky tomu, že projektový tým byl velmi zkušený, vystačil s třístránkovým hlavním dokumentem, ke kterému pro každý z třech typů testů (unit, integrační a akceptační testy) vznikla jedna jednostránková příloha. Což mě vede k další poznámce: jako u jiných oblastí plánování i u plánování testování není nutné vytvořit detailní plán hned na začátku projektu, ale plán se postupně detailizuje v průběhu projektu (technika „rolling wave planning“).

Berte tedy prosím tuto kapitolu jako seznam věcí, na které je při plánování testování dobré pamatovat.

Test skript

Tady si musíme udělat trochu pořádek v pojmech. Existují pojmy *test skript*, *testovací scénář*, *testovaný případ* (anglicky test script, test scenario, test case), které se používají v různém významu a často jsou zaměňované. Aby byl další výklad jasný, tak je budeme používat v následujícím významu:

Testovaný případ je testovaná jednotka v byznys významu. Například objednávka spotřebního materiálu, objednávka na sklad, zadání služební cesty atd. Víceméně mu odpovídá unit test.

Test skript je návod (data, postup kroků, ...) jak příslušný případ otestovat. Často se v tomto smyslu používá též pojem testovací scénář, což může vést k nedorozuměním.

Testovací scénář je posloupnost funkčních kroků, které na sebe navazují a tvoří logický řetězec procesních kroků. Může to například být pořízení skladového materiálu (objednávka -> příjemka -> likvidace faktury -> platba).

Ještě definujme pojem **Testovací protokol**, což je dokument, do kterého se zapisují výsledky testování podle příslušného skriptu.

Z praktického hlediska je nejlepší mít testovaný případ jako co nejmenší funkční jednotku a tedy test skript co nejkratší. Setkal jsem se s test skripty v délce několika desítek kroků. Pak, pokud se v jednom z dejme tomu padesáti kroků vyskytne chyba, je celý test logicky označen jako chybný. Jednak to pokrývá report o (ne)úspěšnosti testování a navíc, pokud chce vedení projektu zjistit, kolik je vlastně skutečných chyb, odhadnout jak náročná bude jejich oprava a retestování, nestačí mu přehled na úrovni testovaných případů, ale musí detailně analyzovat protokoly neúspěšných testů – což ho jistě nepotěší.

Pokud se testuje delší scénář, řeší se to tak, že se jednotlivé testované případy (respektive skripty) zřetěží, což dává možnost v reportingu testů vykazovat výsledky testování scénáře na úrovni testovaných případů.

Pro automatické testy je test skript popsán skriptovacím jazykem daného testovacího nástroje. I v takovém případě je dobré mít k dispozici nějaký lidský popis toho, co se testuje. Potřebujeme mít přece souhlas vlastníka procesu (nebo někoho, kdo tuto roli supluje) s tím, že aplikaci je možno nasadit do produktivního provozu – a podle čeho by poznal, že funguje správně?

Pro manuální testování je příklad formuláře pro test skript (ze skutečného projektu) na obrázku 2.

TESTOVACÍ PŘÍPAD		<NÁZEV >			
Krátký popis:					

IDENTIFIKACE PROJEKTU					
Jméno projektu		Jméno zákazníka		Test Manager	
				Jiří Otta	
Project Manager Dodavatel	Project Manager Zákazník	Zahájení	Plánované dokončení		

Scénář za zákazníka schválen ¹ :			Datum:		

PROVEDENÍ A VYHODNOCENÍ TESTU					
Datum ² :	DD.MM.RR	Celkový výsledek ² :	Akceptován bez připomínek <i>Eventuelně Akceptován s výhradou, Neakceptován</i>	Test provedl ² :	
Připomínky ² :					
Schválil ² :		Datum ² :		Podpis ² :	

Nastavení testovacích dat			
#	Datový objekt	Hodnoty / vstupní soubor / ...	Popis
1.			
2.			

Testovací kroky						
#	Cinnost	Vstupní data	Očekávaný výsledek	Transakce / Report	Výstup – číslo dokladu / záznamu, doba běhu ²	Výsledek ² : (OK nebo číslo chyby)
1.						
2.						
3.						

¹ Jedná se o schválení posloupnosti kroků testování, nikoli vyplnění scénáře
² Vyplní se při provedení testu.

Obrázek 2 Formulář pro test skript

Názvy polí ve formuláři jsou víceméně samovysvětlující. Poznamenejme pouze, že většina polí se vyplňuje před provedením testu (s výjimkou těch označených), důraz jsme v projektu kladli na schválení skriptu ze strany odborných útvarů (vlastníků procesů) a že dokument obsahuje též pole pro identifikaci testovacích dat.

Vývojářské testy, unit testy

Pojďme se teď podívat na nejdůležitější typy testů. K základním a často podceňovaným patří při implementaci IS tzv. unit testy, někdy též jednotkové testy. Jedná se o ověření, zda individuální funkcionality je nastavena a/nebo vyvinuta dle požadavků stanovených v definici požadavků. Testují se individuální konfigurační elementy a procesní kroky většinou asociované s transakcí nebo reportem, což odpovídá testovaným případům.

Provádějí se co nejdříve po dokončení příslušné funkcionality. To plyne z potřeby, že testování by mělo naplňovat princip integrální a nikoli revidované kvality. V průběhu vývoje se tak testují malé samostatné části softwaru nejdříve implementačním týmem a poté klíčovými uživateli klienta tak, jak jsou postupně dokončovány. Například se otestuje založení objednávky, i když k ní ještě není možno záúčtovat fakturu. Nebo pořídit příjem na sklad, i když systém ještě špatně počítá účetní hodnotu zboží atd. Tento postup má řadu výhod:

- Vady jsou odhaleny relativně brzy a je tedy čas je opravit.
- Pokud ze strany implementačního týmu došlo k nějakému zásadnímu nepochopení nebo k chybě, která se může projevit ve více částech aplikace, je tento nedostatek odhalen relativně brzy a není nutno přepracovávat velkou část aplikace.
- Projektoví manažeři mají přesnější informaci o pokroku projektu. Pokud totiž implementační tým nahlásí, že příslušná funkcionality je dokončena, z pohledu projektového manažera to vůbec nemusí být pravda – kde je jistota, že je to správně a že se to nebude ještě třikrát předělávat? (A pokud je to nově napsaný program, klidně i vícrát?) Jestliže funkcionality projde úspěšně unit testem, pak se prudce zvýší pravděpodobnost, že se předělávat nebude.

Předpokladem je, že aplikace vzniká postupně po jedné funkcionalitě (transakci, reportu,...). Existují vývojáři a konzultanti, kteří jsou zvyklí připravovat celou aplikaci najednou a teprve poté ji předat k testování a budou vás přesvědčovat, že v daném případě to jinak není možné. Nevěřte jim, není to pravda. Je to pouze otázka jejich pohodlnosti či chybějících schopností.

V průběhu testování se vytváří testovací protokol, kde se zaznamenávají výstupy aplikace (pro pozdější ověření), výsledek testování a chyby, pokud jsou nalezeny. Většinou se pro protokol používá stejný formulář, ve kterém je popsán test skript. Tak tomu bylo i v případě z obrázku 2.

V ideálním případě jsou test skripty připraveny a schváleny ještě před zahájením unit testů. V praxi ale není projektový tým často schopen test skripty napsat předem, protože chybí přesná představa o tom, jak bude aplikace vypadat. Několikrát se mi osvědčilo, že test skripty byly připraveny a schváleny klientem v průběhu unit testů.

Připomeňme, že do unit testů patří též testy zakládání kmenových dat, testy konverzí (programy pro nahrání dat před spuštěním provozu – datové migrace) a základní testy rozhraní v tom smyslu, že rozhraní je technicky funkční.

Vraťme se ještě k vývojářským testům. Jsou to testy, které provádí vývojář aplikace před tím, než předá příslušnou část k testování testerům klienta. Tyto testy se sice nedělají protokolárně, nicméně v případě, že se část aplikace programuje na projektu, jsou v podstatě povinné. Několikrát v životě jsem zažil, že programátor předal k testování aplikaci, která měla zásadní chybu již na první obrazovce. Bylo jasné, že si to ani jednou nespustil – myšlenka na středověké mučení šla v té chvíli těžko zahnat.

Procesní testy, Integrační testy, Akceptační testy

Jak jsme se již naznačili, procesní testy (anglicky scenario tests, string tests,...) je posloupnost kroků, které na sebe navazují, tvoří provázaný řetězec procesních kroků (z pohledu byznys procesů) a jsou provázány daty. Například založení prodejní zakázky, k níž se provede částečné vychystání a odeslání zboží a fakturace dle skutečně expedovaného množství. V praxi se realizuje jako posloupnost testovacích případů respektive skriptů pro unit testy s tím, že některá výstupní data unit testů se předají jako vstupní data pro následující testy. Procesní testy z principu následují po unit testech. O procesních testech se opět vytváří protokol jako o unit testech.

Netestujeme tedy funkčnost jednotlivých reportů, transakcí, aplikací atd. (to už jsme otestovali v unit testech), ale pouze jejich provázanost. To neznamena, že nemůžeme najít chybu při pořízení faktury, ale v této fázi projektu to skoro jistě znamená, že chyba je důsledkem předchozích kroků ve scénáři.

Procesní testy by se měly opět provádět hned, jak je daný procesní řetězec v systému funkční. V praxi jsem se však často setkal s tím, že samostatné procesní testy (resp. testy scénářů) nebyly prováděny, ale návaznosti kroků byly testovány až v rámci integračních testů. Pro implementace malého a středního rozsahu je dle mého názoru toto zjednodušení akceptovatelné.

Integrační testy, jak sám název napovídá, by měly sloužit k testování integrace celého řešení a to jak vnitřní (jednotlivé kroky jsou vzájemně integrovány), tak vnější (integrace na systémy a funkcionality, které nejsou součástí naší implementace). Testují se takzvané end-to-end scénáře v plné délce. Testovací scénář vznikne stejně, jako jsme popsali výše pro procesní testy – jedná se de facto o procesní testy v maximální možné délce včetně rozhraní na externí systémy. Právě s externími systémy bývá někdy potíž, pokud pro ně nemáme k dispozici dedikované testovací prostředí a tedy nelze plně otestovat funkčnost rozhraní bez dopadů na produkční systémy. O integračních testech se opět vytváří protokol.

Často bývá výsledek akceptačních testů brán jako podklad pro akceptaci projektové fáze. Pro dodavatele je to svým způsobem pohodlné. Být na straně klienta, byl bych však zásadně proti – součástí akceptace projektové fáze musí být všechny její plánované výstupy a integrační testy jsou pouze jedním z nich.

Název Akceptační testy také může vést k myšlence, že slouží k akceptaci řešení předávaného dodavatelem klientovi. Není tomu tak. Tento pojem se používá pro akceptaci řešení zástupci odborných útvarů klienta formou testů na prostředí, které funkčně i datově co nejvíce odpovídá budoucímu produktivnímu. Následují po integračních testech. Je to předávání řešení celým projektovým týmem, nikoli dodavatelem. Většinou se nepostupuje protokolárně podle scénářů a na projektech v Čechách jsem se s nimi setkal jen velmi málo.

Regresní testy

Pokud implementujeme úplně nový informační systém, je svět v podstatě krásný a tuto kapitolu můžeme s klidem vynechat. Protože však platí, že dobře už bylo, probíhá velká většina dnešních implementací do systémů, které již běží produktivně a přidávají se pouze nové oblasti funkcionalit, nebo probíhají velké změny z důvodu změny byznysu (malé změny se neřeší projektově a mohly by být námětem na samostatný článek).

V takovém případě potřebujeme vědět nejen to, že nová funkcionalita funguje tak, jak jsme si definovali, ale že zároveň nepřestala fungovat již produktivní funkcionalita, na které jsme nic měnit nechtěli. Jinými slovy, nic jsme nerozbili. Musíme si totiž uvědomit, že dnešní informační systémy jsou z principu velké, komplexní a provázané celky. Ta provázanost a integrovanost je sice jedna z jejich největších výhod, na druhou stranu má své nepříjemné důsledky. Testování toho, co již bylo funkční, se nazývá **regresní testy**. Opět z nich vzniká protokol.

Uveďme si příklad. V produktivním systému nám k naší spokojenosti běží prodej hotových výrobků. Provádíme implementaci pro procesy prodeje náhradních dílů. Ve většině IS budou obě prodejní funkcionality využívat stejné části programového kódu a stejné konfigurační tabulky. V projektu musíme tedy otestovat, že funguje prodej náhradních dílů, ale že též stále funguje prodej hotových výrobků.

Další příklad, kdy je třeba provést regresní testy, je patchování nebo upgrade systému.

Kardinální otázka u regresních testů je: Co všechno musím otestovat? Ideální je otestovat všechny funkcionality produktivního systému. Kromě toho, že je to ideální, je to bohužel často i nereálné. Málo kdy máme k dispozici takové prostředky (technické a zejména lidské zdroje), abychom si to mohli dovolit. A to musíme ještě zdůraznit, že základem pro jakékoli řízené regresní testování je mít aktuální dokumentaci toho, co je v systému produktivní, až do úrovně jednotlivých procesních kroků, nejlépe i s test skripty. A jak řekl klasik: „dámy a pánové, kdo z vás to má?“. (Co a jak dokumentovat pro produktivní systém je opět námětem na samostatný článek.)

Pro některé informační systémy existují nástroje, které poskytnou podklady pro určení rozsahu potřebných regresních testů. Já sám mám například poměrně dobré zkušenosti s nástrojem SAP Solution Manager. Pokud takový nástroj není k dispozici, nezbyvá, než potřebné testy odhadnout na základě znalosti daného IS a jeho konkrétní instalace.

Technické testy

Při implementaci IS existuje celá řada takzvané „technických“ testů. Ne všechny jsou relevantní pro každý projekt. Tuto kapitulu tedy pojmem jako přehled těch nejčastějších.

Objemové testy (někdy zaměňované za zátěžové) ověřují schopnost systému (případně transakce) zpracovat požadované velké množství dat v požadovaném čase. Typicky následují až po integračních testech, ale nemusí tomu tak být vždy. Osobně jsem zažil situaci, kdy měl klient tak velké nároky na zpracování velkého objemu dat v určitých transakcích, že jsem rozhodl o provedení objemových testů těchto transakcí ihned po jejich unit testech. A vyplatilo se. Objemové testy neprošly a bylo nutno udělat takové zásahy do programů, že jsme poté museli unit testy opakovat. Kdybychom objemové testy provedli až po integračních, museli bychom opakovat i integrační testy.

Zátěžové testy ověřují schopnost systému zvládnout velké množství současně pracujících uživatelů a spouštěných programů. Následují po integračních testech. Pokud test neprojde, spočívá řešení nejdříve v ladění systému a teprve poté v navýšení HW.

Problém se zátěžovými a objemovými testy je v testovacím prostředí. Aby byl test plně vypovídající, je třeba ho provést na HW, které je konfigurací obdobné jako produktivní systém. Takové často nebývá k dispozici. Pak se testuje na slabším HW a je třeba se pokusit o aproximaci. U zátěžových testů je problémem též zajistit dostatečný počet uživatelů pro práci v systému během testů. Pokud to není možné, používají se specializované nástroje na simulaci uživatelů.

Důležitým testem je **test zálohování**. Nejde o vlastní zálohování, ale o reload zálohy. Tento test je mimochodem potřeba opakovat pravidelně i u běžícího systému. Zjistit při havárii systému, že zálohu mám, ale nejsem schopen ji přečíst, je dost smutná historka.

Tisky je třeba testovat na všech typech tiskáren. Zejména na místech, kde je tisk kritický. Oblíbené historky jsou o tom, jak se nákladové listy místo ve skladu tiskly v projektové místnosti.

Testy autorizací jsou často odsunovány na pozdní fázi projektu a pak na ně není čas. Je třeba je však brát vážně a testovat je stejným způsobem jako unit testy. Je třeba provést nejen pozitivní testy (autorizace umožňuje akce, které má umožnit), ale i negativní testy (neumožní, co je zakázáno).

Další důležité testy jsou testy datových konverzí, tj. programů pro upload dat před zahájením provozu. Je třeba otestovat funkčnost programů pro jejich nahrání na plném rozsahu dat.

Defekt management

Poslední důležitou oblastí, které se budeme věnovat, je defekt management. Jedná se o to, jak postupovat, pokud je v testu nalezena chyba (defekt).

Pro defekt management je potřeba systém na jeho řízení (databáze chyb). Většinou se s výhodou dá použít jakýkoli klasický helpdeskový systém, kde se dají chyby z testů oddělit od ostatních hlášení. Pokročilé systémy na řízení testů v sobě obsahují aplikaci na řízení defektů a propojení defektů s testovacími protokoly. Pokud není k dispozici žádný nástroj, dá se využít excel, ale platí stejná poznámka jako pro využití excelu pro řízení testů.

Pokud se vyskytne chyba (defekt), který nebyl opraven v průběhu testování, je tato chyba zaznamenána testerem do databáze chyb. V záznamu je uvedeno:

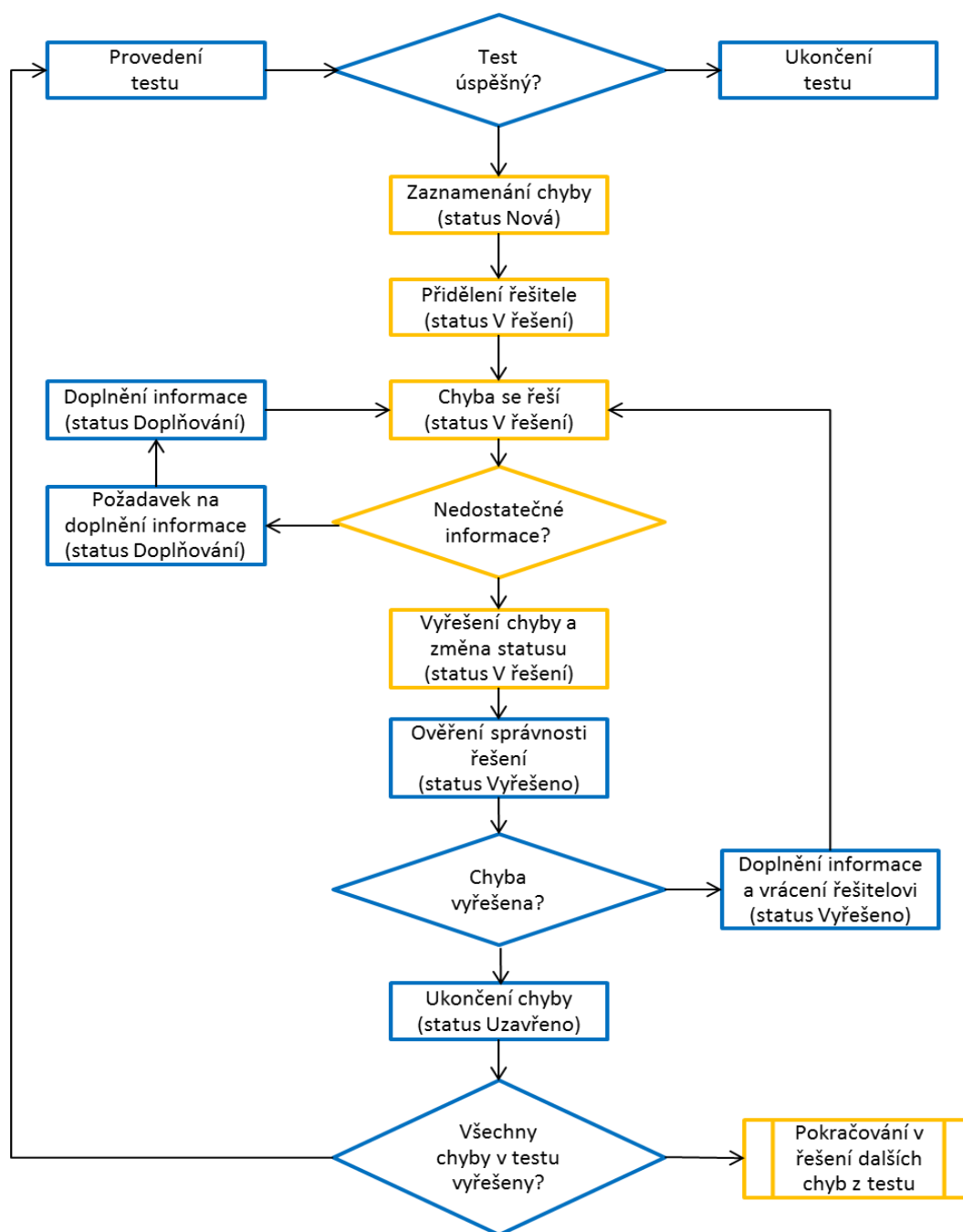
- Jméno testera, který chybu reportuje
- Datum, případně čas vytvoření záznamu
- Popis chyby, případně s odkazem na externí dokument (otisk obrazovky,...)
- Identifikaci testu, ve kterém chyba vznikla
- Priorita
- Status
- Jméno řešitele (může být doplněno dodatečně)
- Další textové informace z průběhu řešení chyb a návrh řešení
- Datum a čas poslední změny záznamu

Do jednoho hlášení se uvádí pouze jedna chyba. K jednomu testu tedy může existovat více hlášení.

Priorita chyb může být definována různě, uvádíme zde pouze příklad:

- A = Vysoká – chyba způsobuje naprostou nefunkčnost aplikace
- B = Střední – chyba způsobuje, že aplikace je jen obtížně použitelná a obejítí chyby je časově velmi náročné a pracné
- C = Nízká – všechny ostatní chyby.

Postup zpracování chyby je definován posloupností statusů. Opět uvádíme pouze příklad jednoho zjednodušeného schématu (modře označené činnosti provádí tester, žluté dodavatel):



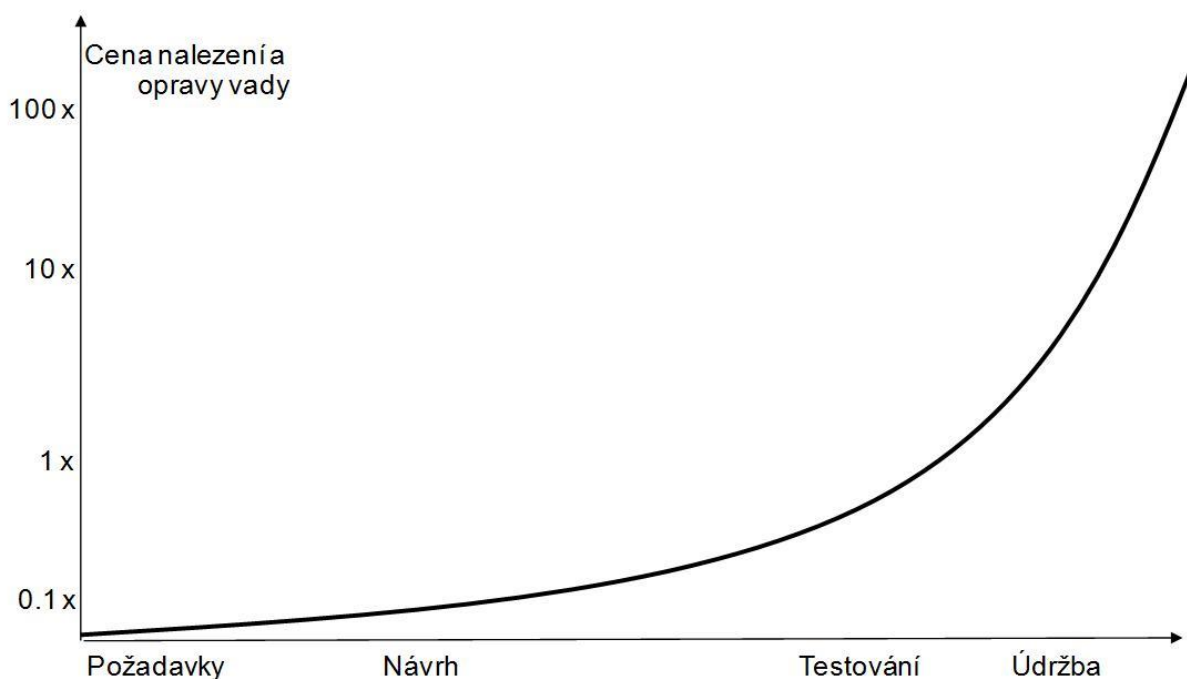
Obrázek 3: Schéma řešení chyb

Závěr

Vypadá testování jako složitá, pracná a drahá záležitost? Složitost je relativní pojem. Když to člověk umí, složitě to nevypadá. Je třeba si uvědomit, že tento článek má za cíl obsáhnout pokud možno všechny aspekty testování na projektech implementace informačního systému. A jak se liší jednotlivé typy projektů, tak je potřeba vybrat správné typy testů a procesy testování – a jsme opět u toho, že o testování je třeba něco vědět, než projekt začneme, aby výběr byl správný.

Testování nemusí být pracné. Naopak, vhodným řízením testů se může ušetřit čas, který se jinak věnuje chaotickému testování, ale hlavně se ušetří čas na řešení problémů, které by se dali včas odhalit vhodnými testy a vyřešit je. Uvědomme si, že na projektech se vždy dělá nějaké „ověřování, kontrolování, zkoušení,...“. Jde jen o to tyto aktivity řídit tak, aby z nich byl maximální užitek.

Je testování drahé? Ano, něco to stojí. Na druhou stranu může ušetřit spoustu peněz, které nás stojí vada nalezená ve finální fázi projektu, nedej bože v produktivním provozu. Ohledně těchto nákladů obecně platí pravidlo 1:10:100. Schematicky to ukazuje obrázek.



Obrázek 4: Cena opravy vady

Někdy je z hlediska nákladů možné i netestovat. V takovém případě je však potřeba vědět, jaká jsou rizika.

Závěrem vám nepopřeji testy bez chyb, protože o tom to není. Popřeji vám, aby vaše testy co nejdříve odhalily co nejvíce chyb, které při implementaci vznikly. Pak bude vaše testování úspěšné.