MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Systém pro podporu nasazení CMMI

DIPLOMOVÁ PRÁCE

Bc. Lukáš Strmiska

Brno, podzim 2009

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Jan Pavlovič, Ph.D.

Shrnutí

Tato diplomová práce se zabývá analýzou a vytvoření nového nástroje pro pomoc při zavádění systému řízení kvality. Analýza je provedena podle standardů UML 2.1. Implementace je provedena v jazyce Java s využitím moderních technologií, jako je například Spring 3.

Klíčová slova

CMMI, SCAMPI, QMS, Java, Spring

Obsah

Kapitola 1

Úvod

CMMI je zkratkou z *Capability Maturity Model®Integration*, což by se dalo velice volně přeložit jako „Stupňovitý model vyspělosti“. CMMI je model kvality organizace práce určený pro vývojové týmy. Definuje procesní oblasti, které musí tým realizovat a cíle, kterých musí v každé oblasti dosahovat [15]. Byl vynalezen v softwarovém institutě na univerzitě v Carnegie Melon. Jedná se o sadu praktik, které pokud jsou dodržovány při celém životním cyklu softwarového produktu, pak se dá u tohoto produktu dosáhnout zaručené kvality. Pod životním cyklem se myslí všechny fáze projektu, počínaje specifikací požadavků, jejich analýzou, návrhem systému, samotného vývoje a provozu u koncových zákazníků. Pod jednotlivými praktikami si můžeme představit třeba projektové řízení, samotný vývoj a tak podobně. CMMI neříká, jaké nástroje se mají používat, nedefinuje přesně co a jak se má provádět. Definuje pouze různé cíle, kterých by měl vývojový tým dosahovat a identifikuje jednotlivé artefakty, které by měly v nějaké formě existovat a být pravidelně sbírány a kontrolovány. Existuje celkem 5 úrovní CMMI. Nejvyšší úroveň má pouze několik organizací na světě. Tyto organizace jsou zejména z těch, které vyvíjí software pro naprosto kritické záležitosti, jako jsou například raketové systémy nebo systémy pro řízení leteckého provozu. To ukazuje vysokou kvalitu těchto metodik.

Systémů kvality existuje více, vedle CMMI existuje též ISO 20000:2005, který pochází z rodiny ISO a stalo se celosvětovým standardem, který se vztahuje k managementu služeb IT a zaměřuje se na zlepšování kvality, zvyšování efektivity a snížení nákladů u IT procesů. Mimo totoho systému neexistoval dříve žádný systém, který by se zabýval přímo oblastí IT. Nevýhodou tohoto systému je fakt, že jako CMMI nenabízí úrpvně vyspělosti a tak nenutí firmu k trvalému zlepšování svých procesů. Obecně mezi systémy kvality lze nalézt mapovaní, které umožňuje jednodušší zavádění dalšího systému kvality, v případě, ze firma má již nějaký systém kvality zaveden. Tato práce se zabývá primárně systémem CMMI. Otázka mapovaní na jiné systémy může být součástí pokračování tohoto projektu, případně může být aplikace rozšířena a doplněna o možnosti provádět zavádění přímo modelů ISO. Návrh aplikace má umožňovat jakékoliv sledování procesů a plnění cílů.

Pro zavádění CMMI se používají metody, které říkají, jak se má CMMI zavádět, případně co se má sledovat a jak se má postupovat při hodnocení. Nejrozšířenější jsou metody SCAMPI. Podle toho, jak moc jsou detailní a časově náročné, tak jsou rozděleny do několika kategorií – A, B, C. S významem, že A je nejkomplexnější a umožňuje zavádět CMMI až do nejvyšší úrovně, kdežto C je nejméně náročná a je více omezená. Tyto metodiky kontroly pocházejí od stejných autorů, jako samotné CMMI a jsou úzce provázány. Aplikace CMMI

Tool bude umožňovat definici modelu nejpreciznější metodiky – SCAMPI A.

1.1 Systém řízení kvality

Systém řízení kvality, přeloženo z anglického *Quality management system* (QMS), dále zmiňován pouze jako systém kvality nebo systém jakosti, je sadou pravidel, kontrolních bodů, kontrolních postupů, případně i množinou doporučených činností, které by se měly dodržovat, a seznamu artefaktů (například dokumenty, nebo entity v nějakém systému), které musí v nějaké podobě vznikat a být archivovány. Tyto artefakty musí být dohledatelné a zatříděné. Při přípravě zavedení systému je nutné identifikovat, zda při podnikových procesech jsou plněny dané kontrolní body, v jaké jsou artefakty kvalitě a zda obsahují potřebné náležitosti. Například daný artefakt může vznikat, ale jeho obsah může být naprosto bezcenný. Kontrola provádění systému kvality pak spočívá zejména v kontrole těchto artefaktů, neboli důkazů o tom, že se dané činnosti dějí. Tyto důkazy pak zaručují, že je daný systém naplňován a výrobky nebo výstupy, které vznikly v rámci tohoto systému mají zaručenou danou úroveň kvality oproti situaci, kdy by se vše řešilo tzv. „ad hoc“. V takovém případě je totiž vysoká pravděpodobnost, že nějaká důležitá činnost zůstane opomenuta.

1.2 Motivace pro vývoj nástroje

Pro zavedení a pro udržování a používám systému řízení kvality (QMS), je třeba mít někde evidovaný seznam výše uvedených důkazů. Existuje několik možností, jak toto dělat. Nejjednodušší formou je papírová evidence (nebo evidence v textovém souboru), která má výhodu v počáteční investici, která je nulová, ale pak už následují jen samé nevýhody. Data nelze třídit, řadit, efektivně je prohledávat, nelze jednoduše získaná data upravit. Nelze data nijak agregovat a získávat z nich další informace, například kolik procent je již splněno, co se ještě očekává a kde jsou nejslabší částí systému. Na papír se spíše tisknou různé zprávy a sestavy o tom, jak je systém plněn. Zdrojem sestav na papíře je však nějaký software.

Další možností je k evidenci využít tabulkový procesor, jako je například *Microsoft Excel*. Tento obecný nástroj umožňuje zkušenému člověku efektivně provádět evidenci téměř čehokoliv s možností různé agregace údajů a hledání v nich, mizí všechny nevýhody uvedené u předchozího textového řešení. Velkou nevýhodou je počáteční časová investice do vytvoření šablon, chybějící validace dat zadávaných dat, chybějící automatické průvodce, které uživateli napovídají a posunují ho dopředu, tak aby svoji práci zvládl efektivně. Dalším obrovským nedostatkem (společným i pro předchozí variantu) je fakt, že všechny informace jsou uloženy v jednom souboru, jež mezi sebou tým, který provádí systém řízení jakosti musí nějakým způsobem sdílet. Zde už nastávají netriviální problémy s tím, kdo má aktuální dokument, může vzniknout více „posledních“ verzí, mezi kterými je třeba netriviálně synchronizovat a podobně. Řešením by zde mohl být například *Microsoft Sharepoint*, který umožňuje živé sdílení Excel tabulek, ale toto je nemalou investicí pro firmu, která tento nástroj ke své další práci nepotřebuje. *Google Documents* naopak mají velkou výhodu ve sdí-

lení dokumentů v reálném čase, avšak v současné době neposkytují takovou funkcionalitu s potřebným komfortem, která by byla pro tuto evidenci postačující.

Poslední možností je použít nějaký již existující nástroj, který je specializován pro sběr těchto důkazů a pro zavedení systému řízení jakosti. V současné době jsou nejrozšířenější dva, které umožňují plné zavádění systému CMMI. Jsou popsány v kapitole 3. – *Současné systémy*. Každý z nich má ovšem svoje nevýhody a oba společně spojuje fakt, že nemají otevřené zdrojové kódy a nejsou zdarma. První problém je zcela zásadní, protože není možno software modifikovat a případně dodat další potřebnou funkcionalitu. Druhý problém, že nejsou zadarmo anebo mají svazující licenční podmínky už není tak zásadní, protože zavedení systému řízení kvality je nemalá investice, ve které se cena nástroje ztratí. Nicméně mít k dispozici nástroj, který bude mít potřebnou funkcionalitu a bude zcela zdarma může být nemalou výhodou, např. pro menší společnosti nebo společnosti, které nemají potřebné prostředky anebo si zavedení systému chtějí ověřit na nástroji, který je k dispozici zadarmo.

Kapitola 2

CMMI a SCAMPI

CMMI 1.2 for Development (CMMI-DEV) je označena verze CMMI, která vychází z verze 1.1 a je speciálně zaměřena na vývoj a údržbu softwarových produktů a služeb. Oproti předchozí verzi došlo k zpřesnění, zjednodušení a odstranění některých pokročilých vlastností praktik. Dále došlo k drobným reorganizacím v rámci cílů a praktik. Rozdělení je provedeno do tří základních oblastí: vývoj, služby a akvizice. Tato práce se zabývá primárně oblastí vývoje a služeb. Účel této kapitoly není kompletní průvodce CMMI, to by bylo duplikování reference CMMI, jde spíše o vysvětlení základních pojmů a principů v CMMI používaných. V druhé podkapitole jsou zevrubně popsány metody SCAMPI, které se používají pro vyhodnocování CMMI. Vše s cílem pro pochopení funkcionality aplikace.

2.1 CMMI 1.2-DEV

2.1.1 Vznik CMMI

V současné době se softwarové společnosti specializují na dodávku software, služeb a kladou se velké nároky na kvalitu těchto dodávek. Dalším faktem je, že se nic ve firmě nevyvíjí od nuly, ale často se používají již existující části, které se integrují a poté dodávají jako celek. To je též netriviální problém. Jiné normy se spíše zabývají byznysem a už je tak nezajímá, jak probíhá vývoj, integrace, testování, apod. Obchodování je samozřejmě důležité, ale aby společnost mohla dodávat kvalitní služby a software, tak se musí zaměřit na tyto primární činnosti.

CMMI pro vývoj (CMMI for Development) se primárně zaměřuje na 3 základní okruhy. Jsou to lidé, metody a nástroje. Dohromady tyto tři okruhy spojují procesy.

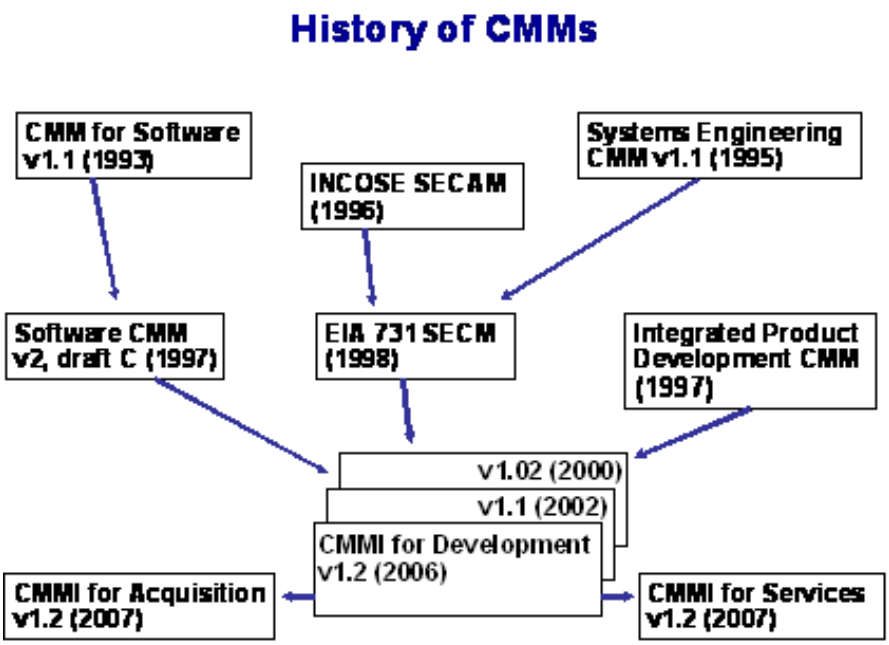
CMMI má mnoho společného s CMM. CMM je zkratkou z *Capabality Maturity Model*. Modelů CMM existuje mnoho, pro každou část průmyslu existuje jeden model. Netriviálním problémem je integrace těchto modelů z různých průmyslových disciplín do jednoho modelu, protože typicky v jedné organizaci se prolíná více disciplín a proto i více modelů. Úkolem CMMI bylo tyto zvolit několik modelů používaných v IT společnostech a tyto integrovat. Konkrétně byly integrovány následující modely:

1. The Capability Maturity Model for Software (SW-CMM) v2.0 draft C

2. The Systems Engineering Capability Model (SECM)

3. The Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

Tyto modely byly vybrány na základě jejich rozšířenosti a adaptacemi mnoha organizacemi. [2]



Obrázek 2.1: Historie CMMI [2]

2.1.2 Přístup k CMMI

Původní rozdělení přístupu k CMMI na průběžné (continous) a režírovanou (staged) bylo spojeno od verze 1.2 dohromady do jedné specifikace. Dříve byly specifikace pro tyto odlišné přístupy dvě. Rozdíl je následující:

- Průběžné

  Zvolí se jedna nebo více procesních oblastí, a ty se vylepšují. Měří se způsobilost procesní oblasti. Tento způsob nabízí maximální flexibilitu, tzn. v různých procesních oblastech může být organizace různě vyspělá (na jiné úrovní). Použití tohoto způsobu však vyžaduje znalost závislosti procesních oblastí ve společnosti.

- Režírované

  Pro zlepšování je zvolena předdefinovaná skupina procesních oblastí, která patří do dané úrovně vyspělosti organizace, a ta se zlepšuje. Měří se úroveň vyspělosti

organizace. Jedná se o kontinuální zlepšování všech procesů v organizace a zvyšování úrovně vyspělosti napříč celou organizací.

2.1.3 Model CMMI

Komponenty modelu CMMI jsou rozděleny do tří kategorií [2]:

1. Vyžadované (required)

   Tyto komponenty musí být viditelně implementovány v organizačních procesech. Jde o generické a specifické cíle.

2. Očekávané (expected)

   Jde o praktiky (generické nebo specifické), které organizace může implementovat pro dosažení daného generického nebo specifického cíle.

3. Informativní (informative)

   Jedná se o informativní podklady pro začátek. Aby se vědělo odkud začít, co kde hledat, jaké nástroje a postupy používat. Jedná se o artefakty, typicky pracovní produkty (work products), poznámky, reference a podobně.

2.1.3.1 Procesní oblast (Process area)

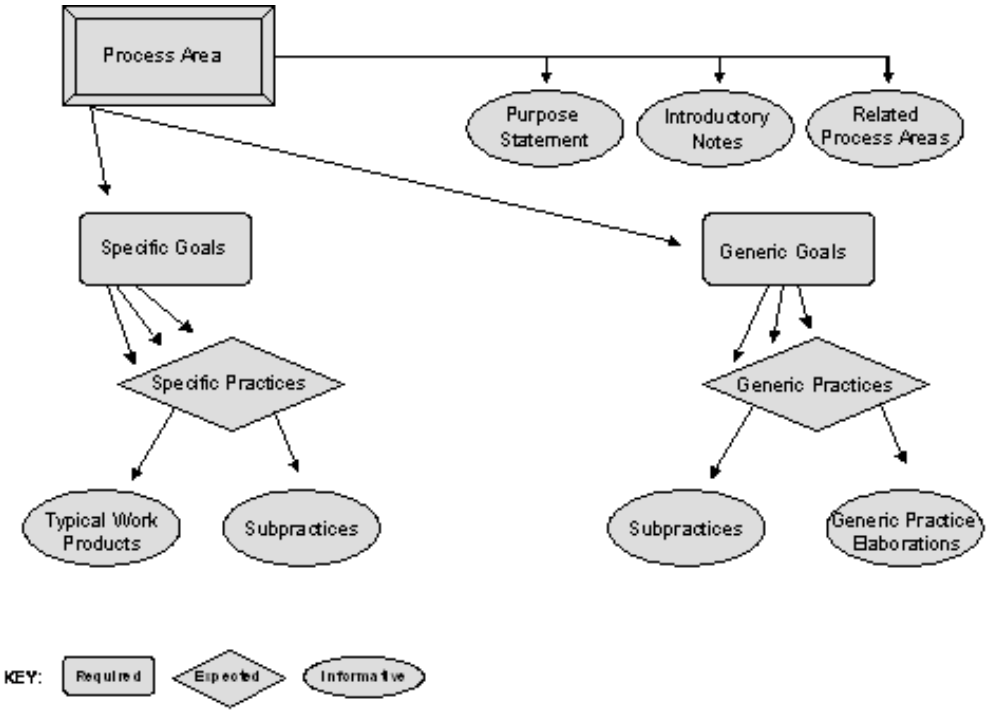
Procesní oblast je množinou příbuzných praktik, které když jsou dohromady implementovány, tak zajistí dosažení množiny cílů, které jsou důležité pro zlepšení organizace v dané procesní oblasti.

V modelu CMMI existuje celkem 22 procesních oblastí, jako například Plánování projektu, Správa rizik, Verifikace, atd.

Každá procesní oblast má informativní část. Ta obsahuje popis svého účelu, popis hlavních konceptů, které pokrývá a seznam příbuzných procesních oblastí. Každá procesní oblast obsahuje stěžejní část a tou je seznam specifických nebo generických cílů, které do ní spadají.

2.1.3.2 Specifický cíl (Specific goal)

Specifický cíl popisuje charakteristiky, které musí být viditelné (dostupné), aby byla splněna procesní oblast. Jsou to vyžadované komponenty modelu a jsou primárně používané pro rozhodnutí při hodnocení, zda je daná procesní oblast splněna. Z hlediska hodnocení je důležitý pouze název cíle. To je většinou nějaké prohlášení, např. „Existuje evidence nemocných.“ Může být doplněn o doplňující popis, který však není směrodatný pro rozhodnutí o splnění cíle.



Obrázek 2.2: Komponenty CMMI [2]

2.1.3.3 Generický cíl (Generic goal)

Cíle jsou nazývány generické z toho důvodu, že stejné prohlášení platí pro více procesních oblastí současně. Prohlášení popisuje obvykle charakteristiku, která musí být splněná, aby bylo dosaženo hodnocení procesní oblasti. Příkladem takového prohlášení může být například „Proces je definován a funguje.“ Stejně jako specifický cíl může mít nezávazný popis.

2.1.3.4 Specifická praktika (Specific practice)

Jde o popis konkrétní aktivity, která je považována za důležitou pro dosažení specifického cíle. Specifické praktik\definují všechny aktivity, kterých je třeba dosáhnout pro splnění specifického cíle. Může to být například „Monitorujte commity do SVN oproti projektovému plánu.“

2.1.3.5 Pracovní produkt (Work product)

Je o seznam typických výstupů z nějaké specifické praktiky. Jde o informativní příklady. Například záznamy z verzovacího systému SVN.

2.1.3.6 Generická praktika (Generic Practice)

Pro generickou praktiku platí totéž, jako pro generický cíl – vztahuje se k více procesním oblastem. Jde opět jako u specifické praktiky o činnost, která je považována za důležitou pro dosažení daného generického cíle. Důležitý je pouze název, stejně jako další objekty v modelu CMMI může obsahovat i popis, který je ovšem nezávazný.

2.1.3.7 Rozpracování generické praktiky (Subpractices)

Je na stejné úrovni v modelu jako Pracovní produkt. V aplikaci CMMI Tool jsou tyto listy modelu obecně nazývány artefakty. Dává v podstatě pouze informativní radu, jak má být generická praktika aplikována.

2.1.4 Úrovně CMMI

Existují dvě škály pro hodnocení úrovně a to úroveň způsobilosti (capabality level) pro průběžnou reprezentaci a úroveň vyspělosti (maturity level) pro režírovanou verzi.

Úroveň	Úroveň způsobilosti	Úroveň vyspělosti
0	Nekompletní (Incomplete)	není (N/A)
1	Provedená (Performed)	Počáteční (Initial)
2	Spravovaná (Managed)	
3	Definovaná (Defined)	
4	Kvantitativně spravovaná (Quantitatively Managed)	
5	Optimalizující (Optimizing)	

Tabulka 2.1: Porovnání úrovní způsobilosti a úrovní vyspělosti [2]

Názvy úrovní jsou od 2. úrovně stejné, rozdíl je v počátku. Průběžná reprezentace počítá se zlepšováním jedné procesní oblasti. Proto je důležité, zda je proces neproveden nebo proveden. Úroveň vyspělosti na druhou stranu hodnotí vyspělost celé organizace a tak není důležité, zda je nějaký proces proveden nebo ne. Až jsou všechny procesy, které patří do dané skupiny, provedeny pak je dosažena úroveň vyspělosti organizace.

Obě dvě měřítka však hodnotí, jak je organizace schopna zlepšovat a zlepšuje svoje procesy, i když přístup je odlišný.

V následujících dvou podkapitolách je stručně popsáno, co zhruba která úroveň znamená v obou reprezentacích.

2.1.4.1 Úrovně způsobilosti

Jsou určeny pro hodnocení průběžné reprezentace. Když se splní všechny cíle a jejich praktiky na danou úroveň, pak má celá procesní oblast splněné hodnocení a organizace získává benefity ze zlepšení procesů.

- Úroveň 0: Nekompletní

  Jde o proces, který je nesplněn, nebo jen částečně splněn.

- Úroveň 1: Provedená

  Provedený proces je proces, který se provádí a plní daný cíl. Přesto, že proces na 1. úrovni je důležitým zlepšením, dosud není zaveden do závazných organizačních pokynů, a tak se časem může být ztracen.

- Úroveň 2: Spravovaná

  Jde proces první úrovně, který je v organizaci zaveden a je plánován. Je monitorován, kontrolován a revidován. Jsou pro tyto činnosti alokovány zdroje. Toto zaručuje, že i během náročnějších situacích (jak finančních, tak časových), nebude proces opomenut.

- Úroveň 3: Definovaná

  Je proces druhé úrovně, který je navíc standardizován a ve všech instancích je stejný. Je zařazen do standardních procesů organizace. Tyto procesy jsou mnohem více popsány, včetně přesné definice, jaké jsou vstupy, výstupy, měřítka, role, výstupní kritéria. Mezi procesy jsou definovány vztahy a je jim rozumět.

- Úroveň 4: Kvantitativně spravovaná

  Jde o procesy třetí úrovně, které jsou hromadně zpracovány a řízeny nějakým nástrojem.

- Úroveň 5: Optimalizující

  Jsou to procesy 4. úrovně, u kterých se organizace zaměřuje na zlepšování a měření výkonu procesu.

Organizace může zvyšovat úrovně procesů pouze postupně, pokud jsou splněny a zavedeny jednotlivé podmínky na jednotlivé úrovně. Vždy je třeba, aby byl management organizace nakloněn CMMI a byly pro tyto činnosti dostupné prostředky. Bez nich nelze skutečně CMMI zavádět.

2.1.4.2 Úrovně vyspělosti

Jsou určeny pro vyjádření celkové úrovně organizace. Skládají se z cílů a praktik sjednocených v jednotlivých procesních oblastech. Každá úroveň má definován seznam procesních oblastí, které je nutno splnit, aby byla úroveň dosažena. Procesní oblasti jsou inkluzivně zahrnuty, čili pro dosažení úrovně 3, je potřeba aby organizace adaptovala všechny praktiky obsažené v procesních oblastech z úrovně 2 a 1.

- Úroveň 1: Počáteční

  Na této úrovní jsou procesy obvykle prováděny „ad hoc“ a chaoticky. Organizace obvykle neposkytuje stabilní prostředí pro provádění procesů. Na 1. úrovni jsou organizace schopné dodávat produkty a služby, ale často nedodrží rozpočet a termíny. Typické je střídaní krize s úspěchem a organizace sama o sobě nedokáže tento stav výrazně ovlivnit a zopakovat svoje úspěchy.

- Úroveň 2: Spravovaná

  Organizace na této úrovní dokáží zajistit provádění procesů v projektech. Jsou k dispozici adekvátní prostředky, procesy jsou prováděny i v krizovějších situacích. Management organizace má přehled, co je v jakém stavu a produkty jsou kontrolovány. Dodávky splňují očekávání na ně kladené.

- Úroveň 3: Definovaná

  Procesy jsou velmi dobře definované a pochopené napříč organizací, jsou popsány ve standardech a odráží se v používání správných nástrojů, postupů a metod. Rozdílem oproti předchozí úrovni je fakt, že jsou používány napříč celou organizací, ve všech projektech. Procesy jsou podrobněji definované.

- Úroveň 4: Kvantitativně spravovaná

  Oproti předchozí úrovni jsou procesy hromadně spravované, sbírají se z nich různé statistiky, které se analyzují, procesy se kontrolují. Na základě těchto informací se dá předpovídat výkonnost procesů.

- Úroveň 5: Optimalizující

  Na základě znalostí z předchozí úrovně má organizace možnost tyto svoje procesy optimalizovat. To je hlavním cílem této úrovně. Organizace je schopná odstraňovat problémy, které způsobují to, že nejsou plněny definované cíle procesů.

2.2 Metody SCAMPI

Metody SCAMPI jsou obecně široce akceptovány jako metody pro hodnocení CMMI. Existují tři varianty: A, B a C. Pouze metoda SCAMPI A může vyústit v hodnocení úrovně vyspělosti. Ostatní dvě (B, C) povolují měnit rozsahy v rozsahu hodnocení a změnu měřítek pro konkrétní případ. Slouží tedy spíše pro zjištění, jak na tom organizace je, nebo pro méně nákladnou variantu přípravy na CMMI a hledání slabších míst.[11]

Existuje více metod na provedení hodnocení CMMI, avšak metody SCAMPI jsou hodnoceny nejlépe, co se týče nutných investic a zisku. Metodiky kombinují postupy známe z hodnotících metod CBA-IPI a EIA 731-2.

Hodnocení je založeno na tzv. *PII* a *PIID*. PII je zkratkou z *Proces Implementation Indicators*, což by se dalo přeložit jako indikátor implementace procesu a PIID (Process Implementation Indicators Description) znamená popis tohoto indikátoru. Jsou to popisy na základě

kterých se dá měřit, jak proces probíhá. V seznamu jsou uvedeny typické důkazy o probíhajícím procesu (např. jeho výstupy). U každé organizace je však třeba nalézt ty správné.

2.2.1 Provedení hodnocení

Celé hodnocení pomocí SCAMPI se skládá ze dvou fází. V první fázi probíhají následující kroky:

1. Analýza požadavků
2. Vytvoření plánu hodnocení
3. Výběr a příprava týmu
4. Získaní a analýza prvotních důkazů
5. Příprava na sběr důkazů

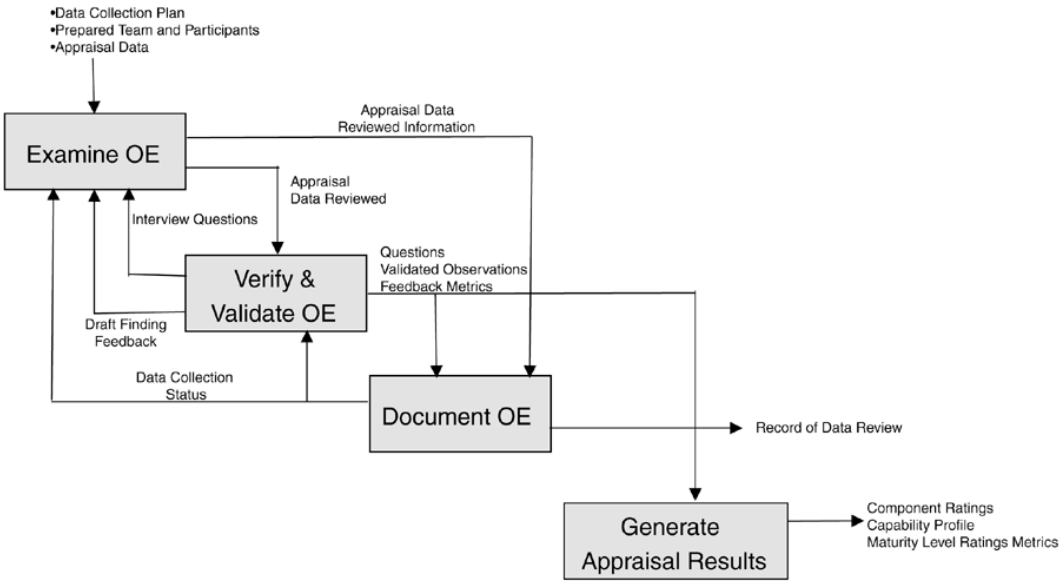
Po těchto krocích první fáze začíná samotné hodnocení organizace, které je názorně popsáno na obrázku 2.3 a sestává se z těchto kroků:

1. Prozkoumání důkazů
2. Ověření a validace důkazů
3. Dokument důkazů
4. Vygenerování výsledků hodnocení

Cílem aplikace CMMI Tool je primárně tato druhá fáze spojená se sběrem a zkoumáním důkazů, validací důkazů, generováním dokumentu důkazů a výsledků hodnocení. Pro všechny tyto činnosti jsou analyzované případy užití.

Po té, co je organizace připravena a tým sestaven, začíná sbírání přímých a nepřímých důkazů a artefaktů. Tyto se váží k jednotlivým praktikám a jejich účelem je dokázat, že se daná praktika provádí. Přímý důkaz může být nahrazen nějakým nepřímým artefaktem, který říká, že se něco takového děje, ale není přímým výsledkem té činnosti. Ke každé praktice může být vyžadován jeden nebo více důkazů. SCAMPI nabízí tzv. *PIID* (viz výše), které nabízí seznam doporučení, jaké důkazy by se měly hledat. Protože na začátku je jasně daný počet praktik, které se hodnotí, pak se dá v aplikaci lehce sledovat, jak hodnocení pokračuje, kolik procent je splněno, ke kterým již byly nalezen dostatečný počet důkazů a podobně.

SCAMPI nabízí sadu různých agregačních pravidel na základě kterých navrhuje úroveň vyspělosti u praktik a procesních oblastní na základě hodnocení těchto praktik. Finální rozhodnutí je však vždy na týmu, který provádí hodnocení, jelikož v různých specifických případech nemusí nutně existovat všechny důkazy v požadované formě a tým provádějící hodnocení může na základě jiných indikátorů tuto oblast považovat za splněnou.



Obrázek 2.3: Provedení hodnocení [11]

Všechny sebrané důkazy by měly být překontrolovány a ohodnoceny. Důkaz sám o sobě má z tohoto důvodu dva stavy – *nový důkaz* a *ohodnocený důkaz*. Z těchto ohodnocených důkazů je pak vygenerován dokument důkazů a na základě tohoto dokumentu jsou k dipozici výsledky hodnocení. Tyto výsledky se negenerují pouze na konci hodnocení, ale průběžně v čase projektu hodnocení, aby bylo zřejmé, jak na tom projekt hodnocení právě je.

2.3 Provedení hodnocení úrovně CMMI

Pro každý projekt hodnocení CMMI musí být zváženy a rozhodnuty následující body:

- Který model CMMI bude použit
- Musí být rozhodnuto, jaký bude rozsah, co vše se bude hodnotit a v jakém časovém rozmezí
- Je třeba zvolit metodu hodnocení
- Musí se též určit členové týmu a jejich role
- Dále definovat, kdo bude dotazován
- A jakým způsobem budou zaznamenávány výsledky

Volbu modelu, metody, vytvoření týmu a zaznamenávání výsledku umožňuje aplikace CMMI Tool, která slouží jako hodnotící systém. Umožňuje definici modelů CMMI, umožňuje definici metod. Mimo jiné též definici týmu a uživatelských rolí v týmu.

Kapitola 3

Současné systémy

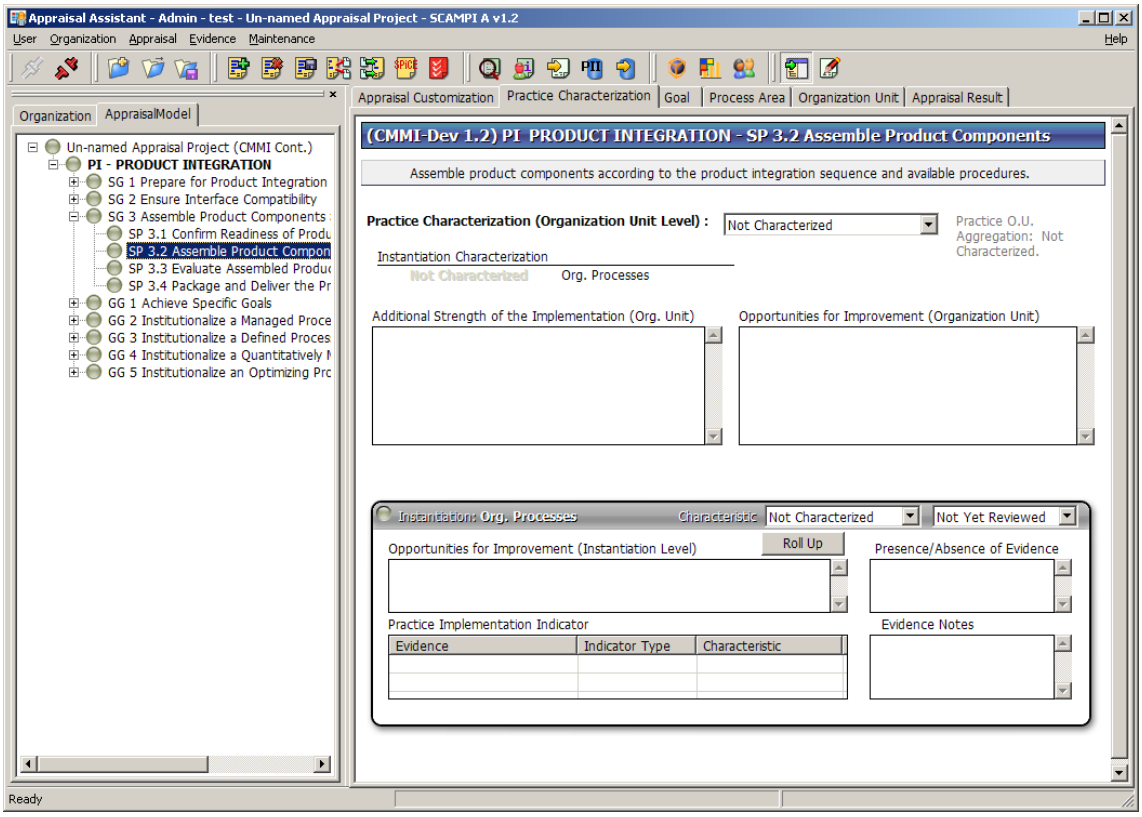
V současné době jsou nejrozšířenější dva hlavní nástroje pro zavádění QMS. Oba dva však jsou z nějakých důvodů nepostačující pro potřeby společností, které se připravují samy na zavedení systému kvality a zatím nechtějí investovat větší prostředky do firem, které zavádění provádějí externě. Z tohoto důvodu byla provedena analýza a započal se vývoj nového nástroje, který má otevřené zdrojové kódy, je multiuživatelský a má centralizovanou databázi.

3.1 Appraisal Assitant

Software vyvíjel Fred Liang na australské univerzitě v rámci své dizertační práce. Vývoj aplikace bohužel skončil v betaverzi v roce 2007 a od té doby nebyla vydána žádná další verze. Aplikace tedy není dále vyvíjena a není k ní žádná podpora. Aplikace nemá k dispozici zdrojové kódy, a tak není možné na dílo navázat, navíc není k dispozici jasná licenční politika. Jde o aplikaci pro Windows XP, interně využívající MS Access databázi kombinovanou se souborovým systémem. Mezi její hlavní nevýhody patří složité sdílení databáze důkazů napříč více uživateli a její vázanost na konkrétní Windows. Funkcionalita aplikace je rozsáhlá, umožňuje provádět hodnocení jak pomocí všech metod SCAMPI hodnotit několik modelů CMMI (včetně CMMI 1.2 for Development), tak umí i hodnotit ISO 9001. Práce s aplikací je příjemná, k dispozici je nápověda i uživatelský manuál. Je k dispozici všechna potřebná funkcionalita včetně generování rozsáhlých a podrobných zpráv a výstupů hodnocení. [1]

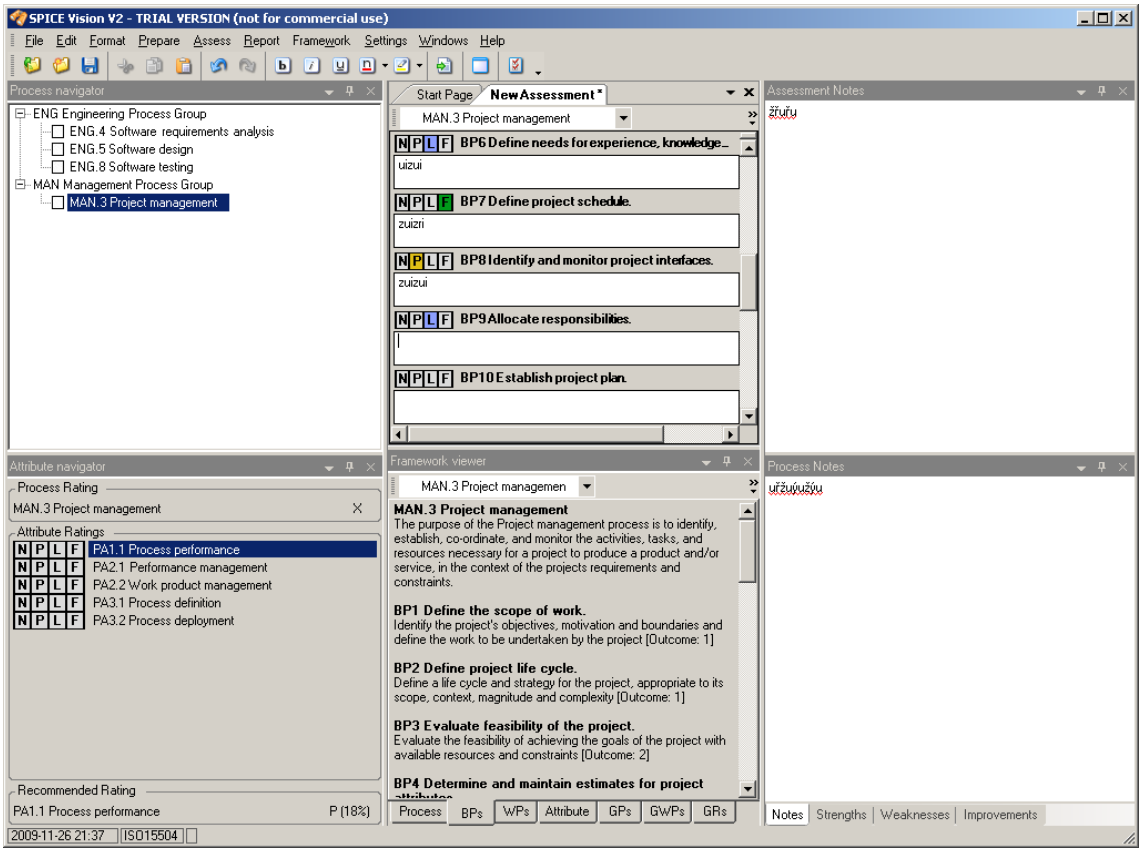
3.2 Spice Vision

Jedná se o komerční aplikaci vyvíjenou za účelem provádění hodnocení ISO. Nevýhodou je existence pouze placené verze na druhou stranu vyvážené stálou podporou ze strany výrobce. Z pochopitelných důvodů k aplikaci nejsou zdrojové kódy a tak není možno aplikaci jakkoliv modifikovat. Aplikace je postavena na .net frameworku a tak je vázána systém MS Windows, díky frameworku by však měla být bez problému provozovatelná nad všemi Windows, které tento framework podporují. Aplikace je nabízena ve dvou edicích, v levnější nemůže uživatel editovat modely, což může být poměrně svazující omezení. Další nevýhodou je omezení počtu procesních skupin na jednu a další zamknuté funkce, které pak



Obrázek 3.1: Appraisal Assistant

donutí uživatele pro seriózní práci pořídit plnou verzi. Oproti Appraisal Assistant chybí import prvotního seznamu důkazů z Excelu. Výhodou oproti Appraisal Assitant je dostupnost mnohem více modelů ISO v kontrastu, že úplně chybí modely pro CMMI – tyto si uživatel může vytvořit sám v profesionální verzi aplikace. Další výhodou jsou rozsáhlejší exporty, hlavně v profesionální edici. Tato je vhodná hlavně pro větší týmy. Celkově se mi uživatelské rozhraní jevilo jako méně komfortní a přehledné oproti Appraisal Assistant. Aplikaci stejně jako předchozí chybí centrální databáze hodnocení, data jsou uložena lokálně v souborovém systému. [12]



Obrázek 3.2: Spice Vision

Kapitola 4

CMMI Tool

Tato kapitola popisuje model navrženého nástroje pomocí metodiky UML 2.1. Nejdříve jsou definovány role v systému a následně uvedeny diagramy případů užití. Každý případ užití je pak podrobně definován buď detailnějším diagramem případů užití nebo popisem a podrobným scénářem formou toku událostí (Flow of events, FOE). Následují analytické diagramy tříd. Další důležitou kapitolou je popis implementace (obsahující i diagram nasazení), který je důležitý zejména pro vývojáře, kteří v systému budou pokračovat.

4.1 Role v systému

Systém obsahuje kompletní správu uživatelských účtů. Uživatelé aplikace mají tedy možnost si uživatelské účty spravovat sami. Pokud tohoto nechtějí využít, tak díky použité technologii je možné systém spojit s nějakým existujícím zdrojem uživatelů, např. LDAP. Do systému má přístup pouze registrovaný uživatel. Každý uživatel má přidělenu právě jednu aplikační roli, která určuje jeho práva a úkoly v systému. Poté, pokud je přiřazen na nějaký projekt, má definovánu i projektovou roli, která se váže k tomuto projektu a ta určuje jeho práva v konkrétním projektu. Uživatel může mít v různých projektech různé projektové role.

4.1.1 Aplikační role

V systému existují právě 3 následující role, z nichž každý uživatel v systému má právě jednu:

- **Administrátor**

  Hlavním úkolem je vytváření a správa uživatelských účtů, vytváření a správa modelů CMMI a definice hodnotících metodik SCAMPI. Administrátor taktéž spravuje organizace, pro které jsou prováděny projekty hodnocení. Rozšiřuje roli Uživatel.

- **Manažer kvality**

  Manažer kvality se zabývá správou projektů hodnocení v organizacích, definicí členů týmu a jejich rolí v těchto projektech. U projektu definuje procesní instance. Rozšiřuje roli Uživatel.

- **Uživatel**

  Uživatel nemá v systému žádná speciální práva, jeho hlavním úkolem je provádět hodnocení, tj. být zařazen v nějakém projektu.

4.1.2 Projektové (týmové) role

Každý uživatel zařazený na projektu má právě jednu z následujících rolí.

- **Člen týmu**

  Úkolem člena týmu je spravovat a zadávat důkazy a generování zpráv o průběhu hodnocení projektu.

- **Auditor**

  Tato role rozšiřuje Člena týmu zejména o následující úkoly spojené s hodnocením. Jedná se o charakterizaci důkazů a hodnocení jednotlivých komponent modelu (organizace, procesní oblast, praktika a cíl).

4.2 Případy užití (Use Cases)

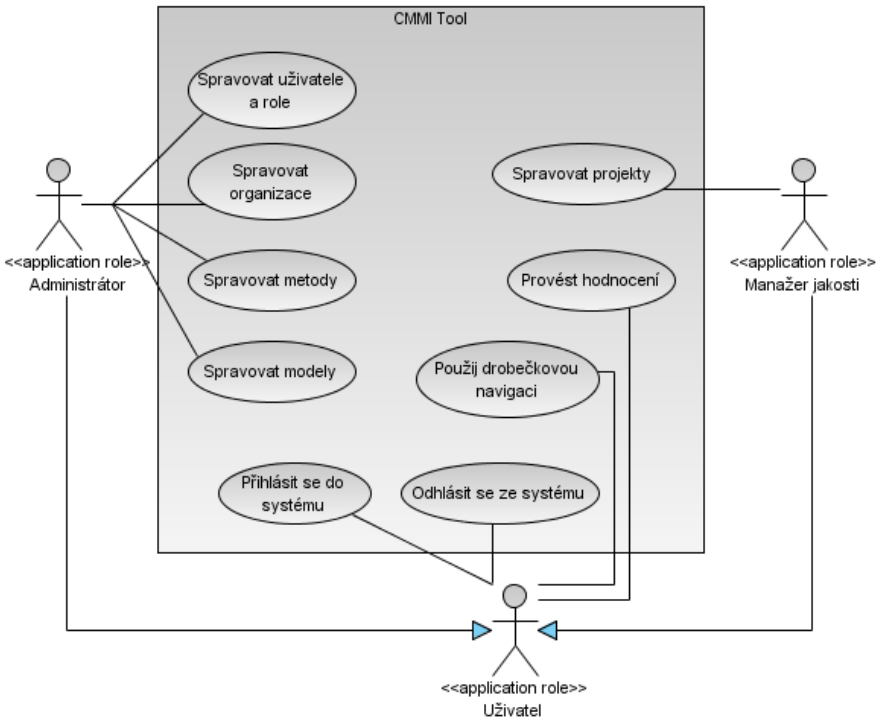
V následující části jsou analyzovány případy užití. Vždy je nabídnut diagram případů užití a pak buď podrobnější diagram popisující případ užití detailněji anebo přímo textový popis (včetně toku událostí) daného případu užití. Vždy pokud není uveden textový popis UC, tak se UC rozpadá další diagram, který je uveden dále.

4.2.1 Hlavní diagram případů užití

4.2.1.1 Přihlásit se do systému

Jde o proces autentizace uživatele. S aplikací smí pracovat pouze přihlášení uživatelé. při zadání jakékoliv URL na aplikaci systém zobrazí přihlašovací dialog pro vložení jména a hesla, pokud uživatel není přihlášen. Po úspěšném přihlášení je zobrazeno hlavní okno aplikace (anebo obsah původního URL). Bez správného jména a hesla uživatel není oprávněn nijak pracovat s aplikací. Uživatel smí při přihlášení zvolit možnost „Pamatuj si mne“, která zabezpečí, že uživatel zůstane přihlášen i po vypršení sezení (session).

Vstupní podmínka
Uživatel není přihlášen.



Obrázek 4.1: Hlavní diagram případů užití

Tok událostí

1. Případ užití začíná, když uživatel zadá do prohlížeče adresu systému
2. Systém zobrazí formulář pro zadání jména a hesla a volbu **Zapamatovat si mě**
3. Uživatel zadá jméno a heslo a stiskne **Přihlásit**
4. Systém ověří platnost údajů a dále ověří, že účet není uzamčen, není expirován a je povolen
5. KDYŽ údaje jsou platné a účet je v pořádku POTOM Systém přihlásí uživatele a zobrazí hlavní obrazovku aplikace s rozcestníkem k jednotlivým úlohám
6. JINAK Systém vypíše chybové hlášení a pokračuje se bodem 2.

Výstupní podmínka

Uživatel je přihlášen.

4.2.1.2 Odhlásit se ze systému

Tento scénář může uživatel zavolat z jakékoliv obrazovky systému. Dojde k okamžitému odhlášení uživatele a odstranění případného příznaku „zapamatuj si mne“.

Vstupní podmínka
Uživatel je přihlášen.

Tok událostí

1. Případ užití začíná, když uživatel klikne na **Odhlásit se** na jakékoliv obrazovce systému
2. Systém odhlásí uživatele

Výstupní podmínka
Uživatel není přihlášen.

4.2.1.3 Použij drobečkovou navigaci

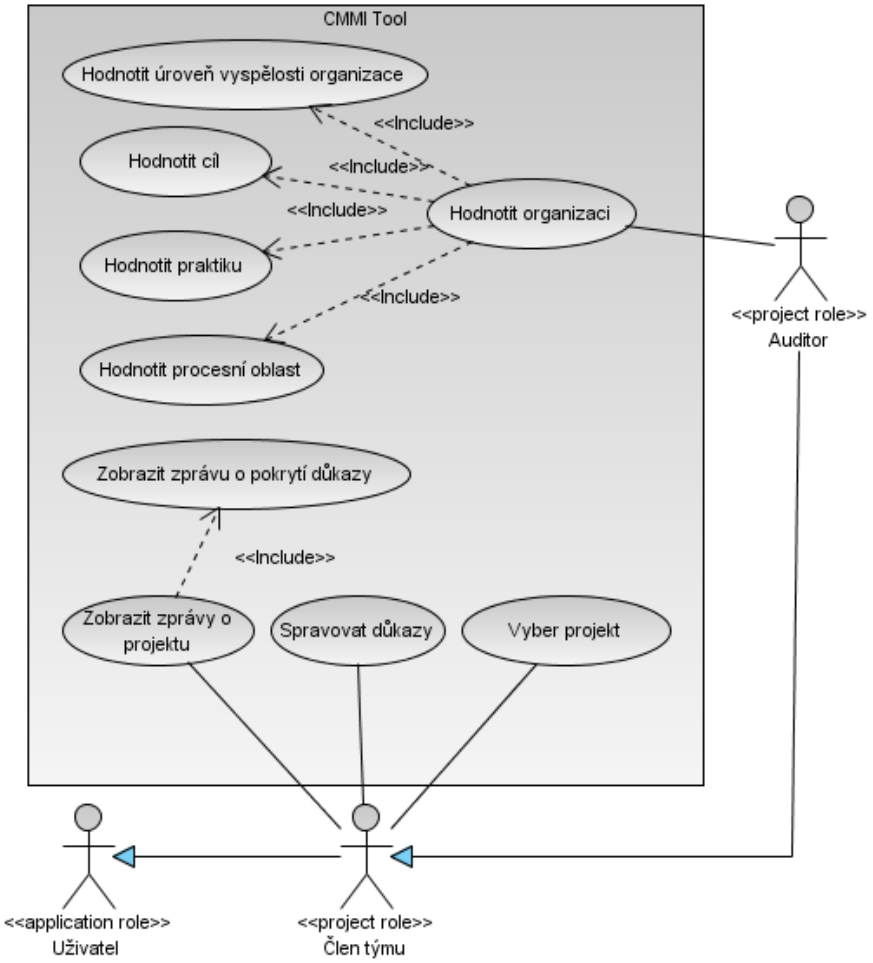
Tento scénář může uživatel zavolat z jakékoliv obrazovky systému. Drobečková navigace je zobrazena vždy nahoře a ukazuje zanoření uživatele v sytému do hloubky. Kliknutím na ni se uživatel může vrátit na jakoukoliv výše položenou obrazovku. Příklad drobečkové navigace na stránce o Zobrazení pokrytí důkazy: *Hlavní Strana >> Provést hodnocení >> Zobrazit zprávy o projektu*. Systém zde zobrazuje všechny přímé nadstránky aktuální stránky, které jsou v přímé cestě ke kořenové stránce, která zobrazuje hlavní obrazovku systému.

Vstupní podmínka
Uživatel je přihlášen.

Tok událostí

1. Případ užití začíná, když uživatel klikne na nějakou stránku zobrazenou v drobečkové navigaci
2. Systém zobrazí zvolenou stránku.

Výstupní podmínka
Uživateli se zobrazí zvolená stránka.



Obrázek 4.2: Provést hodnocení

4.2.2 Provést hodnocení

4.2.2.1 Hodnotit úroveň vyspělosti organizace

Jedná se o celkové hodnocení úrovní vyspělosti organizace na stupnici úrovně vyspělosti 0 - 5.

Vstupní podmínka

Uživatel je v roli auditor v daném projektu.

Tok událostí

1. Případ užití začíná, když uživatel zvolí kořenový strom hodnocených entit, tj. název metody.

2. Systém zobrazí aktuální hodnocení organizace (viz popis) a také všechny procesní oblasti s výsledkem jejich hodnocení a uspokojení

3. Uživatel dle informací zvolí nové hodnocení (viz popis) a stiskne **Uložit**.

4. Systém uloží hodnocení do databáze

Výstupní podmínka
Hodnocení je uloženo a zobrazeno.

4.2.2.2 Hodnotit cíl

Jedná se o hodnocení cíle (generického nebo specifického) na základě zvolené hodnotící metody projektu. Dále uživatel smí zadat následující víceřádkové textové informace / komentáře, v čem je implementace tohoto cíle silná a kde má slabiny.

Vstupní podmínka
Uživatel je v roli auditor v daném projektu.

Tok událostí

1. Případ užití začíná, když uživatel zvolí jako entitu nějaký cíl (tj. jeden z poduzlů procesních oblastí)

2. Systém zobrazí aktuální hodnocení cíle s možností toto hodnocení změnit (viz popis). Měřítka hodnocení jsou definována zvolenou metodou projektu. Dále systém zobrazí navrhované agregované hodnocení vypočítané z hodnocených praktik daného cíle, pouze pokud tyto hodnocení a tyto agregační pravidla jsou definována a lze navrhnout výsledek. Systém také jako podpůrné informace zobrazí hodnocení všech praktik daného cíle. Systém zobrazí textová pole (viz popis) s případným předchozím obsahem nebo prázdné..

3. Uživatel z možností vybere hodnocení, může vyplnit komentáře k silným a slabým stránkám cíle a stiskne **Uložit**

4. Systém uloží hodnocení do databáze

Výstupní podmínka
Hodnocení je uloženo a zobrazeno.

4.2.2.3 Hodnotit praktiku

Jedná se o hodnocení praktiky. Hodnocení je vybíráno z možností měřítek, která určuje zvolená hodnotící metoda projektu. Dále uživatel smí zadat víceřádkové komentáře k následujícím oblastem: Příležitosti, Chybící/nalezené důkazy, poznámky, silné stránky, slabé stránky.

Vstupní podmínka
Uživatel je v roli auditor v daném projektu.

Tok událostí

1. Případ užití začíná, když uživatel zvolí jako entitu nějakou praktiku (tj. jeden z poduzlů cíle)
2. Systém zobrazí aktuální hodnocení praktiky s možností toto hodnocení změnit (viz popis). Měřítka hodnocení jsou definována zvolenou metodou projektu. Dále systém zobrazí navrhované agregované hodnocení vypočítané z hodnocených procesních skupin (instancí) k důkazům, které se váží na danou praktiku, pouze pokud tyto hodnocení a tyto agregační pravidla jsou definována a lze navrhnout výsledek. Systém také jako podpůrné informace zobrazí hodnocení všech procesních instancí, všech důkazů, které se váží k této praktice (jejich charakterizaci a adekvátnost) rozdělených dle procesních instancí. Systém zobrazí textová pole (viz popis) s případným předchozím obsahem nebo prázdné.
3. Uživatel z možností vybere hodnocení, může vyplnit textové komentáře a stiskne **Uložit**
4. Systém uloží hodnocení do databáze

Výstupní podmínka
Hodnocení je uloženo a zobrazeno.

4.2.2.4 Hodnotit procesní oblast

Jde se o hodnocení procesní oblasti. Procesní oblast má dvě hodnocení a to hodnocení způsobilosti, které má možnosti hodnocení "nehodnoceno" a poté škálu úrovně vyspělosti 0 - 5. Druhým měřítkem je uspokojení procesní oblasti, tyto volby jsou definovány zvolenou hodnotící metodou. Dále uživatel může vložit dva textové komentáře – silné a slabé stránky zvolené procesní oblasti v organizaci.

Vstupní podmínka
Uživatel je v roli auditor v daném projektu.

Tok událostí

1. Případ užití začíná, když uživatel zvolí jako entitu nějakou procesní oblast (tj. jeden z poduzlů kořenového uzlu).
2. Systém zobrazí aktuální hodnocení procesní oblasti s možnosti toto hodnocení změnit (viz popis). Dále je zobrazeno hodnocení všech cílů v tabulce název cíle a hodnocení cíle, které spadají do této procesní oblasti a jsou hodnoceny.
3. Uživatel z možností vybere hodnocení, může vyplnit textové komentáře a stiskne **Uložit**
4. Systém uloží hodnocení do databáze

Výstupní podmínka
Hodnocení je uloženo a zobrazeno.

4.2.2.5 Zobrazit zprávu o pokrytí důkazy

Tento případ užití popisuje obrazovku zobrazující pokrytí důkazy daného projektu. Je zobrazen strom všech hodnocených entit: *Metodika - Procesní oblast - Cíl - Praktika*. U každé praktiky jsou pak jako poduzly vypsány všechny procesní instance. U názvu každé procesní instance je v závorce uvedeno, kolik je k dané praktice a procesní skupině v systému evidováno přímých důkazů, nepřímých důkazů a přímých ujištění. Pokud k dané procesní instanci existuje více nebo rovno jak jeden přímý důkaz a to stejné platí pro přímé ujištění, pak je daná procesní instance označena jako zelená, jinak jako červená. Následně jsou obarveny všechny zbylé uzly stromu podle pravidla: Pokud je více jak polovina poduzlů uzlu A zelená, pak je i uzel A zelený.

Vstupní podmínka
Uživatel je v jakékoli roli zařazen na daný projekt.

Tok událostí

1. Případ užití začíná, když uživatel volí na obrazovce Zprávy o projektu možnost **Zobrazit pokrytí důkazy**
2. Systém zobrazí rozbalený strom všech hodnocených entit (viz popis)
3. Uživatel klikne na **[-]** u nějakého rozbaleného uzlu
4. Systém sbalí podstrom daného uzlu
5. Uživatel klikne na **[+]** u nějakého sbaleného uzlu
6. Systém rozbalí podstrom daného uzlu

Alternativní tok
Uživatel se může vrátit na předchozí obrazovku.

4.2.2.6 Zobrazit zprávy o projektu

Tento případ užití definuje rozcestník pro generování všech možných zpráv o projektu.

Vstupní podmínka
Uživatel je v jakékoli roli zařazen na daný projekt a má vybraný projekt.

Tok událostí

1. Případ užití začíná, když uživatel na obrazovce s výběrem projektu má vybraný projekt a zvolí **Zobrazit zprávy**
2. Systém zobrazí zprávu s dostupnými zprávami pro projekt
3. KDYŽ uživatel zvolí možnost **Pokrytí důkazy** POTOM ZAHRNOUT <<Zobrazit zprávu o pokrytí důkazy>>

Alternativní tok
Uživatel se kdykoliv může vrátit na předchozí obrazovku.

4.2.2.7 Hodnotit organizaci

Tento případ užití popisuje vlastní hodnocení celé organizace a všech entit v modelu. Systém zobrazí strom: *Hlavní kořen (název modelu) - Procesní oblast - Cíl - Praktika*. Ve výchozím stavu sbalený. U každého uzlu reprezentující hodnocenou entitu (pokud je tato hodnocena) zobrazí adekvátní barvu podle hodnotící metody, pokud je tato barva metodou definována.

Vstupní podmínka
Uživatel je v roli auditor v daném projektu.

Tok událostí

1. Případ užití začíná, když uživatel na obrazovce s výběrem projektu má vybraný projekt a zvolí **Hodnotit organizaci**
2. Systém zobrazí všechny hodnocené a hodnotitelné entity modelu podle zvolené metody projektu (viz popis).
3. KDYŽ uživatel zvolí hlavní kořen POTOM ZAHRNOUT <<Hodnotit úroveň vyspělosti organizace>>

4. KDYŽ uživatel zvolí procesní oblast (dítě hlavního kořene) POTOM ZAHRNOUT <<Hodnotit procesní oblast>>
5. KDYŽ uživatel zvolí cíl POTOM ZAHRNOUT <<Hodnotit cíl>>
6. KDYŽ uživatel zvolí praktiku POTOM ZAHRNOUT <<Hodnotit praktiku>>

Alternativní tok
Uživatel se smí vrátit na předchozí obrazovku.

Alternativní tok 2

1. Případ užití začíná, když uživatel klikne na [+] nebo [-] u jakékoliv uzlu ve stromě
2. Systém rozbalí nebo sbalí podstrom daného uzlu

4.2.2.8 Vyber projekt

Tento případ užití popisuje obrazovku s výběrem projektu pro práci s ním. Systém nabízí akce podle role uživatele. Každý přiřazený uživatel smí spravovat důkazy a prohlížet zprávy o projektu. Auditor smí navíc provést hodnocení organizace.

Vstupní podmínka
Uživatel je přihlášen.

Tok událostí

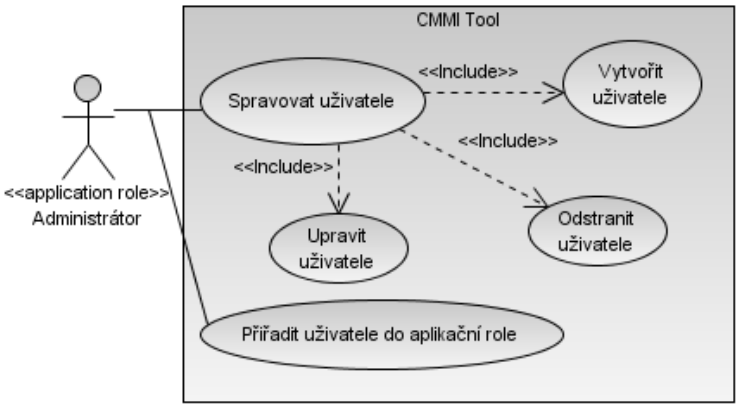
1. Případ užití začíná, když uživatel na hlavní stránce volí možnost *Provést hodnocení* NEBO uživatel se vrátí z jiné obrazovky na tuto obrazovku
2. Systém zobrazí všechny projekty, na které je daný uživatel přiřazen. a pokud již projekt zvolil, tento je označen jako zvolený.
3. Uživatel zvolí projekt
4. Systém nabídne akce, které uživatel smí na daném projektu dělat (viz popis).

Alternativní tok
Uživatel může na této obrazovce zvolit projekt kdykoliv znovu anebo se vrátit na titulní stránku.

4.2.3 Spravovat uživatele a role

Jedná se o běžné *CRUD* (*Create, Read, Update, Delete* – Operace: Vytvořit, Číst, Upravit a Smazat) nad entitou uživatel, proto z pohledu byznys analýzy nemá cenu provádět detailní analýzu případů užití. Podstatné je zmínit, jaké informace jsou u entity uživatel definovány a uvést diagram případů užití. Existuje uživatel *admin*, kterého nelze odstranit. U každého uživatele se evidují následující informace:

- login (povinné, unikátní, nelze později měnit)
- heslo (povinné)
- jméno (povinné)
- e-mail (povinné)
- aplikační role (povinné, výběr z administrátor, manažer kvality a uživatel)
- uzamčen (ano/ne)
- expirován (ano/ne)
- povolen (ano/ne)



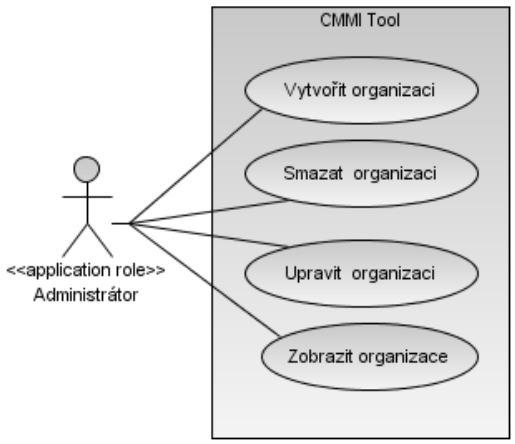
Obrázek 4.3: Spravovat uživatele a role

4.2.4 Spravovat organizace

Jedná se o běžné *CRUD* (viz výše) nad entitou organizace, proto z pohledy byznys analýzy nemá cenu provádět detailní analýzu případů užití.

- Název (povinný údaj, text)

- Adresa (text)
- Kontaktní osoba (text)
- E-mail (text)
- Telefon (text)
- Fax (text)
- Aktivní (příznak ano/ne)



Obrázek 4.4: Spravovat organizace

4.2.5 Spravovat metody

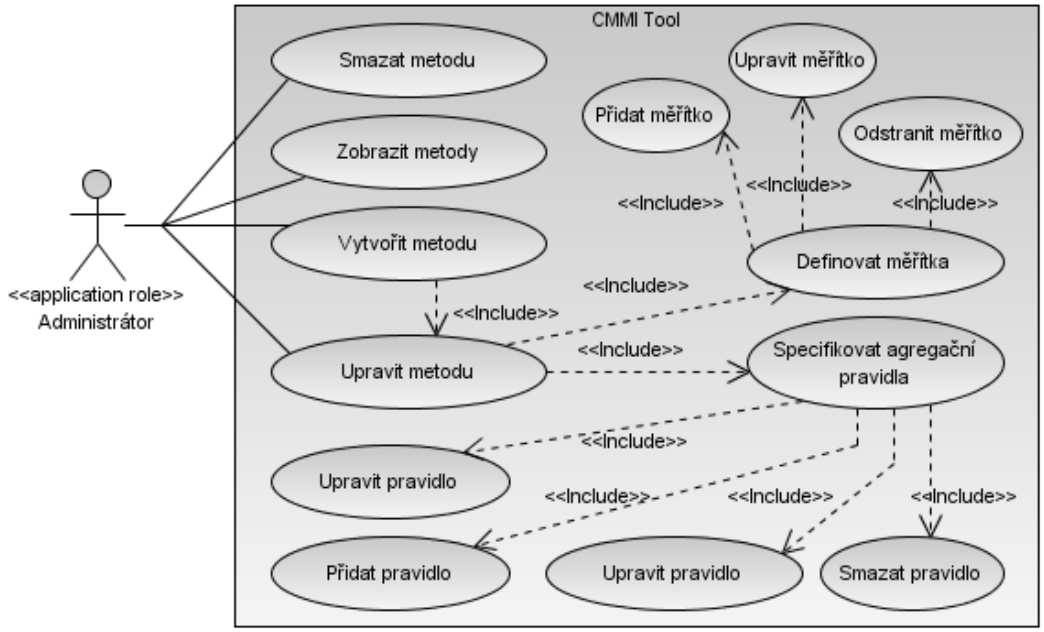
4.2.5.1 Zobrazit metody

Případ užití popisuje zobrazení všech metod v systému.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí na obrazovce Administrátora volbu **Spravovat metody**
2. Systém zobrazí seznam všech metod v tabulce. Je zobrazeno vždy jméno a popis metody a všechny povolené akce – Upravit a Odstranit.



Obrázek 4.5: Spravovat metody

3. Uživatel klikne na sloupec, podle kterého chce tabulku seřadit
4. Systém seřadí tabulku vzestupně nebo sestupně podle toho, zda jde o sudý nebo lichý klik

4.2.5.2 Smazat metodu

Odstranění metody ze systému.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel u metody volí možnost **Odstranit**
2. Systém zobrazí potvrzení
3. Uživatel potvrdí svoji volbu
4. POKUD uživatel potvrdí volbu POTOM je metoda odstraněna JINAK metoda odstraněna není.

4.2.5.3 Vytvořit metodu

Tento případ užití popisuje vytvoření nové metody.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel na obrazovce s přehledem metod volí **Přidat metodu**
2. Systém vytvoří novou metodu a dále ZAHRNOUT <<Upravit metodu>>

Výstupní podmínka
Je vytvořena nová metoda.

4.2.5.4 Upravit metodu

Tento případ užití popisuje upravení stávající metody. U každé metody je v prvním formuláři definovány následující položky

- Jméno (text, povinný)
- Popis (text)
- Zaznamenat nález na (pro následující položky ano/ne)
  - Úroveň organizace
  - Úroveň procesní oblasti
  - Úroveň cíle
  - Úroveň praktiky
- Hodnotit (pro následující položky ano/ne – jedná se o oblasti hodnocení)
  - Úroveň způsobilosti na procesní oblasti
  - Uspokojení procesní oblasti
  - Uspokojení cíle
  - Úroveň vyspělosti organizace
  - Charakterizovat implementaci praktiky

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí u stávající metody *Upravit*.
2. Systém zobrazí formulář metody (viz popis)
3. Uživatel vloží data a stiskne **Další**
4. Systém ověří správnost zadaných údajů podle popisu
5. POKUD údaje nejsou v pořádku, POTOM zobrazí chybové hlášení a pokračuje se bodem 2
6. ZAHRNOUT <<Definovat měřítka>>
7. ZAHRNOUT <<Specifikovat agregační pravidla>>

Výstupní podmínka
Změny jsou persistetně uloženy do databáze.

4.2.5.5 Definovat měřítka

Tento případ užití popisuje vytvoření a definování hodnotících měřítek. Každé měřítko má svůj název (text - povinné), skóre (celé číslo - povinné) a barvu (HTML barva - nepovinné). Měřítka se definují pouze pro ty oblasti hodnocení, které uživatel zvolil v předchozím kroku.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel zobrazí obrazovku Definovat měřítka
2. POKUD jsou nyní zvoleny nějaké nové oblasti hodnocení, která předtím nebyla definována POTOM je systém automaticky doplní měřítky z XML souboru (viz příloha C – *Měřítka*)
3. Systém v podobě stromu zobrazí všechny zvolené oblasti a k nim seznam měřítek
4. Uživatel klikne na **[+] [-]** u uzlu

5. Systém podstrom tohoto uzlu je buď sbalí nebo rozbalí
6. Uživatel klikne na jakýkoliv uzel
7. Systém označí tento uzel jako vybraný

4.2.5.6 Přidat měřítko

Přidání nového měřítka pro nějakou oblast hodnocení.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor A je vybraný uzel, který reprezentuje oblast hodnocení

Tok událostí

1. Případ užití začíná, když uživatel zvolí možnost **Přidat měřítko**
2. Systém do dané oblasti vloží nové měřítko hodnocení s názvem „Neznámý“ a skóre „0“, bez vložené barvy.

Výstupní podmínka
Je vytvořeno nové měřítko ve zvolené oblasti hodnocení.

4.2.5.7 Upravit měřítko

Úprava údajů u měřítka hodnocení

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel zvolí libovolný uzel stromu, který reprezentuje měřítko hodnocení
2. Systém zobrazí formulář (viz popis Definovat měřítka)
3. Uživatel zadá údaje do formuláře a stiskne **Odeslat**.
4. Systém uloží změny

4.2.5.8 Odstranit měřítko

Odstranění existujícího měřítka z oblasti hodnocení

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel zvolí **[X]** u libovolného uzlu stromu, který reprezentuje měřítko hodnocení
2. Systém zobrazí potvrzení smazání
3. POKUD Uživatel potvrdí smazaní, POTOM systém měřítko odstraní

4.2.5.9 Specifikovat agregační pravidla

Případ užití popisuje obrazovku, na které se definují agregační pravidla. Existují dva druhy agregačních pravidel. Agregační pravidla mají definované pořadí. Agregační pravidlo je vždy přepis „0“ nebo „0/1“ nebo „1“ pro všechna zdrojová měřítka (každé měřítko má právě jedno z těchto hodnot) na všechna cílová měřítka označených jako „0“ nebo „1“ (každé cílové měřítko má právě jednu z těchto hodnot).

U agregačních pravidel na Praktiku jsou zdrojová a i cílová měřítka všechny z množiny Charakterizovat implementaci praktiky. Tyto agregační pravidla se dají definovat pouze pokud metoda měří charakterizaci implementaci praktik.

U agregačních pravidel na Cíl jsou zdrojová měřítka z Charakterizovat implementaci praktiky a cílová měřítka jsou všechny z množiny Uspokojení cíle. Tyto agregační pravidla se dají definovat pouze pokud metoda měří charakterizaci implementaci praktik a zároveň uspokojení cíle.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel zobrazí agregační pravidla.
2. Systém vypíše všechna agregační pravidla, která existují. Jsou dvě skupiny pravidel (pravidla cíle a praktiky, viz popis)

4.2.5.10 Přidat pravidlo

Přidání agregačního pravidla pro cíl nebo pro praktiku.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí možnost **Přidat pravidlo** u cíle nebo praktiky
2. Systém zobrazí formulář obsahující všechny zdrojová a cílová měřítka. U zdrojových měřítek nabídne volby (0, 0/1, 1) a u cílových (1,0)
3. Uživatel zvolí požadované hodnoty a stiskne **Uložit**.
4. Systém toto pravidlo uloží a přidá na konec existujícího seznamu pravidel u cíle nebo praktiky

Výstupní podmínka
Nové pravidlo je přidáno na konec seznamu.

4.2.5.11 Upravit pravidlo

Úprava stávajícího pravidla. Lze upravit mapování měřítek z množin (0, 0/1, 1) pro zdrojová měřítka a (0,1) pro cílová měřítka.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel u požadované pravidla zvolí **Upravit**
2. Systém ve formuláři pro úpravu pravidla zobrazí zvolené pravidlo
3. Uživatel upraví požadované hodnoty a stiskne **Uložit**.
4. Systém uloží změny

Výstupní podmínka
Pořadí pravidla v seznamu zůstane zachováno.

4.2.5.12 Smazat pravidlo

Odstranění pravidla ze seznamu

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel u měřítka volí možnost **Odstranit**
2. Systém zobrazí potvrzení smazání
3. POKUD Uživatel potvrdí smazání, POTOM systém měřítko odstraní

Výstupní podmínka
Zbývající měřítka pod odstraněným jsou posunuty o jedna nahoru, tak aby byly postupné.

4.2.6 Spravovat modely

4.2.6.1 Zobraz modely

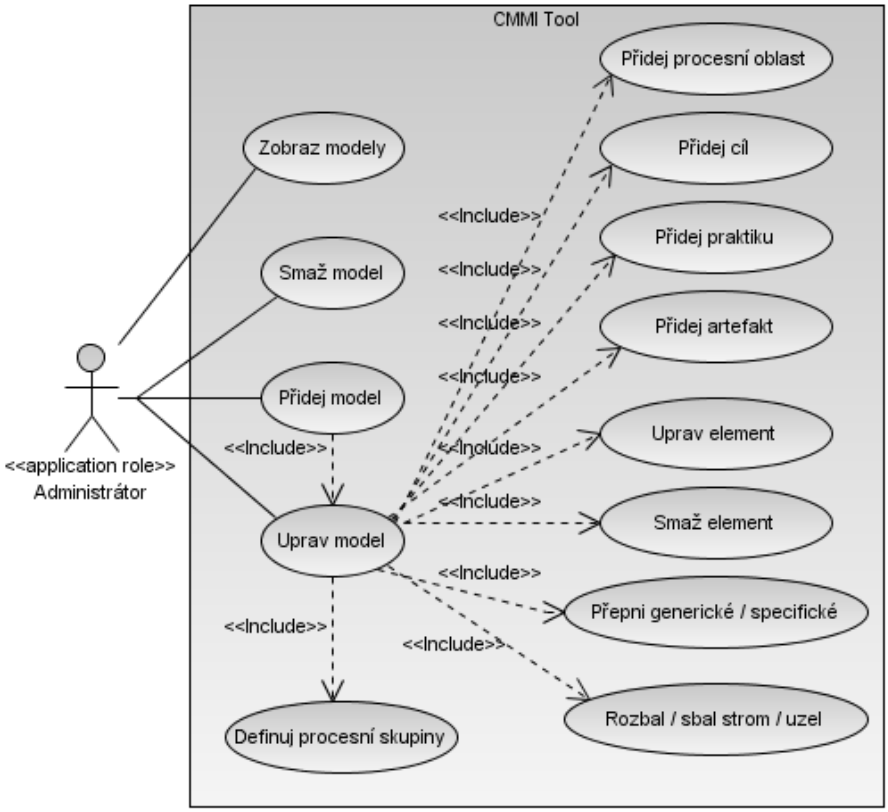
Případ užití popisuje zobrazení všech modelů.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí v administračním panelu **Spravovat modely**
2. Systém zobrazí seznam všech aktuální modelů z databáze v tabulce a umožní řazení dle sloupců (název, akronym, nejvyšší úroveň vyspělosti modelu)
3. Uživatel klikne na sloupce dle kterého chce provést řazení
4. Systém vzestupně / sestupně seřadí tabulku dle zvoleného sloupce.

Výstupní podmínka
Modely nejsou změněny.



Obrázek 4.6: Spravovat modely

4.2.6.2 Přidej model

Vytvoření nového modelu CMMI.

Vstupní podmínka

Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí na obrazovce Spravovat modely volbu **Přidat model**.
2. Systém připraví nový model a zobrazí formulář pro vyplnění údajů
3. ZAHRNOUT <<Uprav model>>

Výstupní podmínka
Nově vytvořený model má unikátní akronym.

4.2.6.3 Uprav model

Tento případ užití popisuje úpravu stávajícího nebo nového modelu CMMI. U každého modelu mimo vlastní model jsou evidovány následující informace:

- akronym (každý model má svůj unikátní akronym)
- jméno (musí být neprázdné)
- nejvyšší úroveň vyspělosti (výběr z 1 - 5)
- popis (volitelný)

Hlavní obrazovka pro úpravu modelu zobrazuje strom modelu. Existují dva módy zobrazení - generický a specifický. Rozdíl je ten, že v generickém nejsou zobrazeny procesní oblasti. Výchozí mód je specifický. Kořenem je název modelu, tento má jako poduzly všechny procesní oblasti. Každá procesní oblast má jako poduzly všechny svoje cíle. Každý cíl má jako poduzly všechny svoje praktiky. Každá praktika má jako svoje poduzly všechny svoje artefakty.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel ve správě modelů volí možnost **Uprav** u stávajícího modelu NEBO je vytvořen model nový.
2. Systém zobrazí formulář o modelu (viz popis)
3. Uživatel vyplní údaje a volí tlačítko **Další**
4. Systém provede validaci údajů (viz popis), POKUD nalezne chyby, tak je zobrazí a pokračuje se bodem 2.
5. ZAHRNOUT <<Definuj procesní skupiny>>
6. Systém zobrazí hlavní obrazovku pro úpravu modelu, která zobrazuje strom (viz popis)
7. KDYŽ Uživatel klikne na **Generický rozměr** nebo Specifický rozměr POTOM ZAHRNOUT <<Přepni generické / specifické>>

8. KDYŽ Uživatel klikne na jakýkoliv element stromu POTOM ZAHRNOUT <<Uprav element>>
9. KDYŽ Uživatel klikne na jakékoliv **[+]**, **[-]**, **Rozbalit vše**, **Sbalit vše** nebo **Přepnout rozbalení** u uzlu POTOM ZAHRNOUT <<Rozbal / sbal strom / uzel>>
10. KDYŽ Uživatel klikne na **Přidej procesní oblast** POTOM ZAHRNOUT <<Přidej procesní oblast>>
11. KDYŽ Uživatel klikne na **Přidej cíl** POTOM ZAHRNOUT <<Přidej cíl>>
12. KDYŽ Uživatel klikne na **Přidej praktiku** POTOM ZAHRNOUT <<Přidej praktiku>>
13. KDYŽ Uživatel klikne na **Přidej artefakt** POTOM ZAHRNOUT <<Přidej artefakt>>

Výstupní podmínka
Uživatel klikl na **Dokončit model** a model byl uložen do databáze.

4.2.6.4 Definuj procesní skupiny

Definice procesních skupin pro model. Každá procesní skupina má svoje jméno.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když je zobrazena obrazovka s definicí procesních skupin.
2. Systém zobrazí všechny Procesní skupiny.
3. KDYŽ Uživatel vyplní jméno skupiny a klikne na **Přidat skupinu** POTOM je tato skupina přidána do seznamu a zobrazena a pokračuje se bodem 2
4. KDYŽ Uživatel u existující procesní skupiny klikne na **Odstranit** POTOM

   4.1 Systém zobrazí potvrzení odstranění skupiny

   4.2 KDYŽ uživatel potvrdí odstranění, POTOM je skupina odstraněna a pokračuje se bodem 2 JINAK se pokračuje bodem 2

Výstupní podmínka
Uživatel zvolil **Další** a procesní skupiny byly uloženy.

4.2.6.5 Přidej procesní oblast

Přidání nové procesní oblasti do modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor A je zvolené specifické zobrazení.

Tok událostí

1. Případ užití začíná, když uživatel volí možnost **Přidat procesní oblast**.
2. Systém zobrazí formulář pro vložení jména a akronymu nové procesní oblasti.
3. Uživatel vyplní formulář a stiskne **Odeslat**.
4. Systém ověří unikátnost akronymu v modelu
5. KDYŽ je akronym unikátní POTOM je nová procesní oblast přidána do modelu
6. JINAK je zobrazeno chybové hlášení

4.2.6.6 Přidej cíl

Přidání nového cíle do modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor A (je zvolené generické zobrazení NEBO je vybrána procesní oblast).

Tok událostí

1. Případ užití začíná, když uživatel volí možnost **Přidat cíl**.
2. Systém zobrazí formulář pro vložení jména a akronymu nového cíle.
3. Uživatel vyplní formulář a stiskne **Odeslat**.
4. Systém ověří unikátnost akronymu v modelu
5. KDYŽ je akronym unikátní POTOM je nový cíl přidána do modelu jako poduzel zvolené procesní oblasti nebo jako přímo generický poduzel kořene modelu.
6. JINAK je zobrazeno chybové hlášení

4.2.6.7 Přidej praktiku

Přidání nové praktiky do modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor A je vybrán cíl.

Tok událostí

1. Případ užití začíná, když uživatel volí možnost **Přidat praktiku**.
2. Systém zobrazí formulář pro vložení jména a akronymu nové praktiky.
3. Uživatel vyplní formulář a stiskne **Odeslat**.
4. Systém ověří unikátnost akronymu v modelu
5. KDYŽ je akronym unikátní POTOM je nová praktika přidána do modelu jako poduzel zvoleného cíle .
6. JINAK je zobrazeno chybové hlášení

4.2.6.8 Přidej artefakt

Přidání nového artefaktu do modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor A je vybrána praktika.

Tok událostí

1. Případ užití začíná, když uživatel volí možnost **Přidat artefakt**.
2. Systém zobrazí formulář pro vložení jména a akronymu nového artefaktu.
3. Uživatel vyplní formulář a stiskne **Odeslat**.
4. Systém ověří unikátnost akronymu v modelu
5. KDYŽ je akronym unikátní POTOM je nový artefakt přidán do modelu jako poduzel zvolené praktiky.
6. JINAK je zobrazeno chybové hlášení

4.2.6.9 Uprav element

Případ užití popisuje úpravu elementu. U každého elementu je evidován unikátní akronym a jméno. Navíc u konkrétních elementů jsou evidovány následující údaje:

Procesní oblast

- Procesní skupina (výběr z definovaných procesních skupin modelu)
- Úroveň vyspělosti procesu (1 - 5)
- Shrnutí (volný text na více řádků)
- Účel (volný text na více řádků)

Cíl

- Shrnutí (volný text na více řádků)

Praktika

- Úroveň vyspělosti praktiky (1 - 5)
- Shrnutí (volný text na více řádků)
- Účel (volný text na více řádků)

Artefakt

- Přímý artefakt (volba ano/ne)

Vstupní podmínka

Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel klikne na jakýkoliv uzel stromu modelu (vyjma kořenového)
2. Systém označí tento element jako vybraný a zobrazí formulář pro úpravu údajů elementu (viz popis)
3. Uživatel upraví údaje a stiskne **Uložit**
4. Systém uloží změny a zobrazí hlášení o uložení

Alternativní tok
Uživatel může přidat element, pokud nejde o artefakt.

4.2.6.10 Smaž element

Případ užití popisuje odstranění elementu z modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel u libovolného elementu ve stromu stiskne tlačítko **[X]**
2. Systém vypíše potvrzení, zda chce uživatel skutečně smazat daný element
3. POKUD uživatel potvrdí smazaní POTOM je daný element, včetně všech poduzlů odstraněn z modelu

4.2.6.11 Přepni generické / specifické

Případ užití popisuje přepnutí pohledu mezi generické a specifické elementy. Generické elementy začínají přímo praktikami hned u kořene uzlu a specifické začínají procesními oblastmi.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. Případ užití začíná, když uživatel volí Generický pohled nebo Specifický pohled.
2. POKUD uživatel zvolí **Generický pohled** POTOM je zobrazen generický pohled a toto tlačítko zmizí a objeví se tlačítko pro přepnutí do specifického pohledu.
3. POKUD uživatel zvolí **Specifický pohled** POTOM je zobrazen specifický pohled a toto tlačítko zmizí a objeví se tlačítko pro přepnutí do generického pohledu.

Výstupní podmínka
Je zobrazen opačný pohled než byl zobrazen doposud.

4.2.6.12 Rozbal / sbal strom / uzel

Rozbalení poduzlu, přepnutí, změna pohledu na strom modelu.

Vstupní podmínka
Uživatel je přihlášen v roli administrátor.

Tok událostí

1. KDYŽ uživatel u nějakého uzlu volí **[-]** POTOM je ikona změněna na **[+]** a podstrom uzlu je sbalen.
2. KDYŽ uživatel u nějakého uzlu volí **[+]** POTOM je ikona změněna na **[-]** a podstrom uzlu je rozbalen.
3. KDYŽ uživatel volí **Rozbalit vše** POTOM je celý strom rozbalen
4. KDYŽ uživatel volí **Sbalit vše** POTOM je celý strom sbalen
5. KDYŽ uživatel volí **Přepnout rozbalení** POTOM uzly které byly sbaleny, jsou rozbaleny a ty, které byly rozbaleny jsou sbaleny.

Výstupní podmínka
Model zůstává nezměněn.

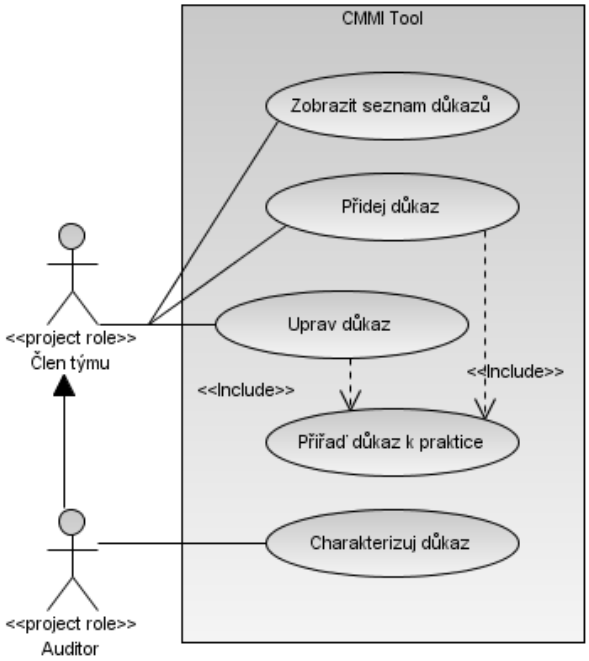
4.2.7 Spravovat důkazy

4.2.7.1 Přidej důkaz

U každého důkazu jsou evidovány následující informace (jde vždy o jednořádkové textové informace, pokud není uvedeno jinak):

- Jméno
- Zdroj
- Popis (víceřádkový text)
- Autor (automaticky vyplněno systémem)
- URI na důkaz
- Typ důkazu (volba právě jednoho z: Rozhovor, Dokument, Pozorování, Hardware, Software, Různé)
- Stav (volba právě jednoho z: Vloženo, Revidováno)

Vyžadovány jsou jméno, typ a stav. Ostatní pole mohou zůstat prázdná.



Obrázek 4.7: Spravovat důkazy

Vstupní podmínka
Uživatel je zařazen v jakékoliv roli na daný projekt.

Tok událostí

1. Případ užití začíná, když uživatel volí ve správě důkazů **Přidat důkaz**.
2. Systém zobrazí formulář (viz popis)
3. Uživatel vloží údaje a stiskne **Další**
4. Systém ověří zadané údaje (viz popis)
5. POKUD jsou vložené údaje správné POTOM ZAHRNOUT <<Přiřaď důkaz k praktice>>
6. JINAK vypiš chybové hlášení o neplatných vstupech a pokračuje se bodem 2.

Výstupní podmínka
Důkaz je uložen do databáze

4.2.7.2 Uprav důkaz

Jedná se o úpravu již existujícího důkazu.

Vstupní podmínka
Uživatel je zařazen v jakékoliv roli na daný projekt.

Tok událostí

1. Případ užití začíná, když uživatel volí ve správě důkazů u konkrétního důkazu **Upravit důkaz**.
2. Systém zobrazí formulář (viz popis u Přidej důkaz) s předvyplněnými hodnotami upraveného důkazu
3. Uživatel vloží údaje a stiskne **Další**
4. Systém ověří zadané údaje (viz popis u Přidej důkaz)
5. POKUD jsou vložené údaje správné POTOM ZAHRNOUT <<Přiřaď důkaz k praktice>>
6. JINAK vypiš chybové hlášení o neplatných vstupech a pokračuje se bodem 2.

Výstupní podmínka
Změny k upravovanému důkazu jsou uloženy do databáze.

4.2.7.3 Přiřaď důkaz k praktice

Každý důkaz může být mapován k *0* až *n* praktikám. Mapování k praktice probíhá přes procesní instance, ke kterým je důkaz mapován.

Vstupní podmínka
Uživatel je zařazen v jakékoliv roli na daný projekt.

Tok událostí

1. Případ užití začíná, když je u důkazu zobrazena obrazovka pro mapování důkazů k praktikám.
2. Systém vypíše strom modelu, kde listem je praktika. Ke každému uzlu s praktikou přidány listy s procesními instancemi se „zaškrtávátky“ a zatrženy jsou právě ty, ke kterým je důkaz momentálně zařazen.

3. Uživatel zaškrtnutím a odškrtnutím zařadí důkaz do požadovaných procesních skupin jednotlivých praktik a stiskne **Uložit**.

4. Systém uloží dané mapování

Alternativní tok

1. Případ užití začíná, když uživatel klikne na **[+]** nebo **[-]** u jakékoliv uzlu ve stromě

2. Systém rozbalí nebo sbalí podstrom daného uzlu

Výstupní podmínka
Důkaz je přiřazen k procesním instancím všech zvolených praktik. Důkaz je zařazen k maximálně *n* praktikám a maximálně *m* procesním instancím, kde *n* je větší nebo rovno *m*.

4.2.7.4 Charakterizuj důkaz

Tento případ užití popisuje charakterizaci důkazu ve vazbě na danou praktiku a procesní instanci a také o charakterizaci skupiny důkazů k dané praktice v mapovaných procesních instancích.

Vstupní podmínka
Uživatel je zařazen v roli Auditor na daný projekt.

Tok událostí

1. Případ užití začíná, když uživatel volí ve správě důkazů **Charakterizuj důkazy**.

2. Systém vypíše strom modelu. Na všech uzlech praktik jsou vypsány všechny procesní instance. U každé procesní instance je možno hodnotit pomocí dvou kritérií s výběrem hodnot:

   a *Adekvátnost důkazů* (anomálie, žádné důkazy, konfliktní důkazy, nedostatečné důkazy, silné důkazy, zatím nerevidováno)

   b *Charakterizace* (měřítka hodnocení jsou určena dle zvolené metodiky, vybírá se ze seznamu měřítek pro praktiky)

   Dále je ke každému uzlu praktiky vypsán seznam všech důkazů připojených přes tuto skupinu procesů k dané praktice. Každý takto vázaný důkaz lze hodnotit dvěma měřítky:

a *Charakteristika důkazu* (slabý, slabý/silný, silný, necharakterizováno)

b *Typ důkazu* (přímý důkaz, nepřímý důkaz, přímé ujištění)

Hodnocení se váže vždy k danému propojení důkazu na danou praktiku přes zvolenou procesní skupinu. Jeden důkaz může být vázán na více praktik.

3. Uživatel zvolí příslušné hodnocení a stiskne **Uložit**.
4. Systém uloží dané hodnocení do databáze.

Alternativní tok

1. Případ užití začíná, když uživatel klikne na **[+]** nebo **[-]** u jakékoliv uzlu ve stromě
2. Systém rozbalí nebo sbalí podstrom daného uzlu

Výstupní podmínka
Změny jsou uloženy do databáze a jsou okamžitě promítnuty napříč celou aplikací a pro všechny uživatele.

4.2.7.5 Zobrazit seznam důkazů

Případ užití popisuje zobrazení seznamu důkazů ke zvolenému projektu

Vstupní podmínka
Uživatel je zařazen v jakékoliv roli na daný projekt.

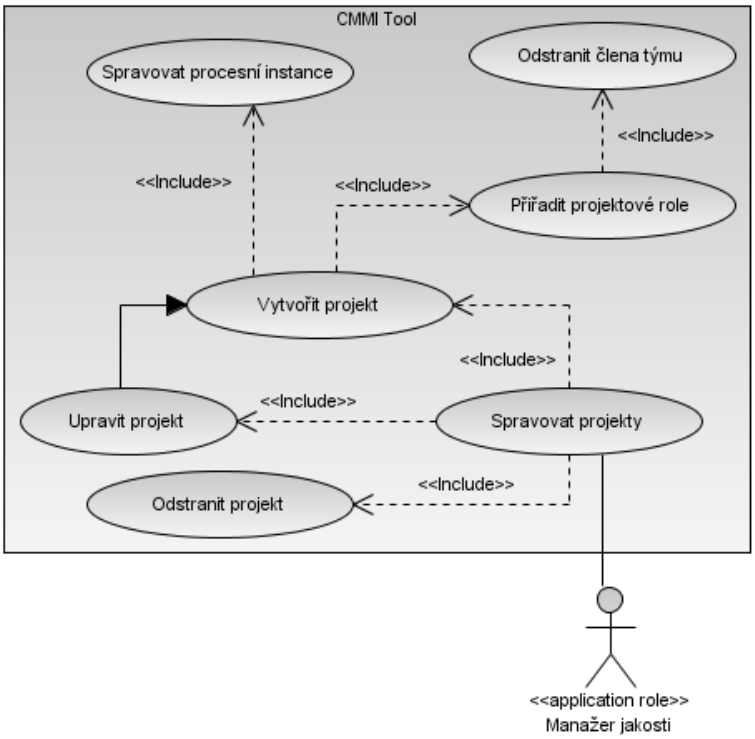
Tok událostí

1. Případ užití začíná, když uživatel volí **Spravovat důkazy** na obrazovce Provést hodnocení organizace.
2. Systém v tabulce vypíše seznam všech důkazů evidovaných v daném projektu, seřazených dle jména. U každého důkazu jsou uvedeny následující údaje:
   - stav
   - informace jestli je důkaz mapován k praktice
   - název
   - kdo důkaz vložil
   - URI odkaz na důkaz

Systém umožní řadit seznam podle všech sloupců

3. Uživatel klikne na sloupec
4. Systém seřadí tabulku podle sloupce sestupně nebo vzestupně podle toho, kolikrát bylo na sloupec klepnuto.

4.2.8 Spravovat projekty



Obrázek 4.8: Spravovat projekty

Případy užití popisují vytváření, čtení, aktualizaci a mazání (CRUD) projektů hodnocení. Každý projekt je vázán na nějakou organizaci. Pro každý projekt jsou uchovávány následující informace, první skupina je zadávána ve formuláři pro úpravu projektu, druhá ve zvláštních případech užití:

- jméno
- akronym (primární klíč)
- zvolená metoda (uživatel může zvolit z existujících metody, výchozí je SCAMPI A 1.2)

- zvolený model (uživatel může vybrat z existujících model, výchozí je CMMI 1.2-Dev)
- cílová úroveň vyspělosti (uživatel může vybrat z úrovní 2 až 5)
- Projektový tým
- Procesní instance

4.2.8.1 Spravovat projekty

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel na hlavní obrazovce volí **Spravovat projekty**
2. Systém vypíše všechny aktivní organizace systému
3. Uživatel zvolí organizaci, pro kterou chce spravovat projekty
4. Systém vypíše formou tabulky všechny projekty z této organizace se základními informacemi a umožní operace Vytvořit nový, upravit stávající a smazat projekt.
5. KDYŽ uživatel zvolí **Smazat** POTOM ZAHRŇ <<Smazat projekt>>
6. KDYŽ uživatel zvolí **Upravit projekt** POTOM ZAHRŇ <<Upravit projekt>>
7. KDYŽ uživatel zvolí **Vytvořit projekt** POTOM ZAHRŇ <<Vytvořit projekt>>

Výstupní podmínka
Všechny provedené změny v projektech jsou okamžitě promítnuty do databáze a jsou platné pro všechny uživatele.

4.2.8.2 Přiřadit projektové role

Přiřazení projektových rolí vybraným uživatelům pro vytvoření projektového týmu. Manažer kvality smí vytvářet a upravovat týmy každého projektu.

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel na přejde na obrazovku **Přiřadit členy týmu**
2. Systém vypíše všechny dosud nepřiřazené uživatele v systému k tomuto projektu a také seznam rolí ve kterých je možno uživatele přiřadit na projekt (Auditor, Člen týmu)
3. Uživatel zvolí uživatele, kterého chce přidat a zvolí týmovou roli, ve které má být uživatel na projekt přidán a stiskne **Přidat**
4. Systém přidá uživatele k členům týmu v požadované roli
5. KDYŽ Uživatel zvolí **Hotovo** POTOM je scénář ukončen
6. JINAK se pokračuje bodem 2

Výstupní podmínka
Přiřazený uživatel smí pracovat s projektem dle zvolené role. Každý uživatel smí být na projektu pouze v jedné roli.

4.2.8.3 Odstranit člena týmu

Manažer kvality smí kohokoliv z projektového týmu odstranit. Tento uživatel pak automaticky ztrácí právo s projektem jakkoliv nakládat.

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel volí u stávajícího člena **Odstranit**
2. Systém se dotáže, zda uživatel chce člena skutečně odstranit
3. KDYŽ uživatel zvolí **Ne** POTOM případ užití končí alternativně, bez smazání uživatele
4. JINAK KDYŽ Uživatel zvolí **Ano** POTOM zvolený člen je odstraněn z týmu v jeho roli.

Výstupní podmínka
Odstraněný uživatel s daným projektem již nesmí nijak nakládat, data, která do projektu zadal, zůstanou zachována.

4.2.8.4 Spravovat procesní instance

Správa instancí organizačních procesů. Jedná se o CRUD pro tyto instance. Každá instance má svoje jméno a popis.

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když se uživatel přejde na obrazovku **Správa procesních instancí**
2. Systém vypíše všechny procesní existující instance a formulář pro vyplnění informací o procesní instanci obsahující jméno a popis
3. KDYŽ uživatel korektně vyplní formulář A stiskne **Přidat**, POTOM je vytvořená nová procesní instance a pokračuje se bodem 2
4. JINAK KDYŽ uživatel u jakékoliv procesní instance zvolí **Upravit** POTOM

   4.1 Systém vyplní formulář informacemi o zvolené procesní instanci

   4.2 Uživatel upraví informace a stiskne **Uložit**.

   4.3 Systém informace uloží a pokračuje se bodem 2
5. JINAK KDYŽ uživatel zvolí u jakékoliv procesní instance (mimo první, která reprezentuje org. procesy) **Smazat** POTOM

   5.1 Systém zobrazí potvrzení operace smazání

   5.2 KDYŽ Uživatel potvrdí volbu, POTOM je instance smazána

   5.3 pokračuje se bodem 2

Výstupní podmínka
Existuje vždy alespoň jedna procesní instance, která je automaticky vytvořena systémem ke každému procesu a nelze ji odstranit, pouze upravit.

4.2.8.5 Vytvořit projekt

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel volí **Vytvořit projekt hodnocení**
2. Systém vypíše formulář pro zadání základních informací o projektu (viz výše)
3. Uživatel vyplní údaje a volí **Další**
4. KDYŽ nejsou všechna povinná pole vyplněna a klíč projektu není unikátní POTOM systém vypíše příslušná chybová hlášení a pokračuje se bodem 2
5. JINAK ZAHRNOUT <<Přiřadit projektové role>>
6. ZAHRNOUT <<Spravovat procesní instance>>
7. Uživatel zvolí **Uložit**
8. Systém uloží projekt do databáze

Výstupní podmínka
Nový projekt je uložen do systému.

4.2.8.6 Upravit projekt

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel volí u stávajícího projektu **Upravit projekt**
2. Systém načte informace o stávajícím projektu z databáze a pak pokračuje formulářem, ZAHRNOUT <<Vytvořit projekt>>

Výstupní podmínka
Stávající projekt je aktualizován v databázi.

4.2.8.7 Odstranit projekt

Vstupní podmínka
Uživatel je přihlášen v roli Manažer kvality.

Tok událostí

1. Případ užití začíná, když uživatel volí u stávajícího projektu **Odstranit projekt**
2. Systém se dotáže, zda uživatel chce projekt skutečně smazat.
3. KDYŽ uživatel zvolí **Ne** POTOM případ užití končí alternativně bez smazání
4. JINAK KDYŽ uživatel volí **Ano**, POTOM je projekt odstraněn ze systému

Výstupní podmínka
Zvolený projekt je odstraněn včetně všech informací na něho vázaných.

4.3 Popis datového modelu

Entitní datový model je rozdělen do několika balíků, které sdružují entitní třídy popisující související části systému. Balík `method` obsahuje entity popisující metodu, `model` pak třídy popisující definici CMMI modelu. Dalším balíkem je `project`, který obsahuje entitní třídy projektu a obsahuje balík `rating`, který obsahuje popis hodnocení projektu. Tyto jsou popsány v následujících podkapitolách.

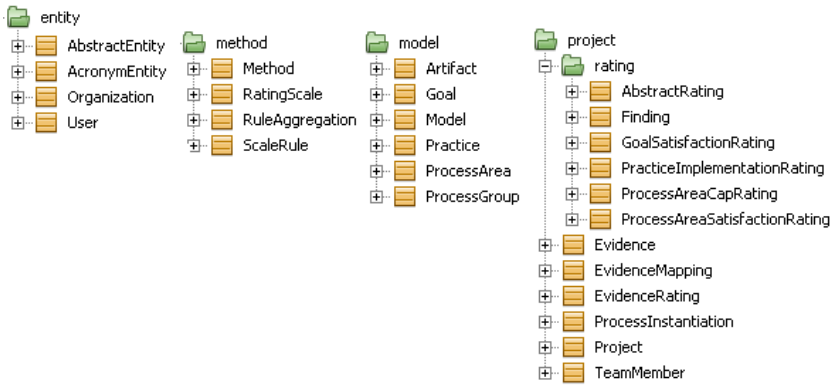
V kořenovém balíku `entity` jsou obecné entitní třídy, jako první je `AbstractEntity`, ze které dědí všechny entity a definuje zda jde o novou nebo již existující entitu. Tato třída implementuje rozhraní `Serializable`. Další třídou je `AcronymEntity`. Tato dědí z abstraktní entity a obsahuje následující atributy společné pro všechny entity, které jsou typu akronym: jméno, akronym a id. Posledními entitami v kořenovém balíku jsou `User` (definující uživatele, implementuje *Spring UserInfo* rozhraní) a `Organization` (definující organizaci).

Všechny entitní třídy jsou *JavaBeans*[10], tj. všechny atributy mají privátní (nebo chráněné) a k nim poskytují veřejné set/get metody. V diagramech jsou pro přehlednost uvedeny pouze atributy (nikoliv operace, které nejsou důležité) a to pouze ty, které mají význam z hlediska popisu funkčnosti. Nejsou například uvedeny atributy, které definují sériové číslo entity používané při serializaci, nemusí být uvedené id entity a podobně.

4.3.1 Balík method

Hlavní třídou je třída `Method`. Obsahuje základní atributy každé metody, jako je jméno, popis a seznam všeho, co se hodnotí. Další třídou je `RatingScale`, která definuje jednotlivá měřítka hodnocení pro vybrané oblasti. Je vždy vázána one-to-many na `Method` pod tím, k čemu se definují měřítka. Například přes `processAreaCapLevel` jsou definovány měřítka pro Schopnost procesních oblastí. Vazba je oboustranná, čili i ze třídy `RatingScale` lze získat informaci, ke které metodě se váže.

Další dvě třídy `RuleAggregation` a `ScaleRule` uchovávají agregační pravidla. Agregační pravidla jsou oboustranně definovány pro cíl a praktiku. Tyto jsou vázána na metodu.



Obrázek 4.9: Entity - rozdělení do balíků

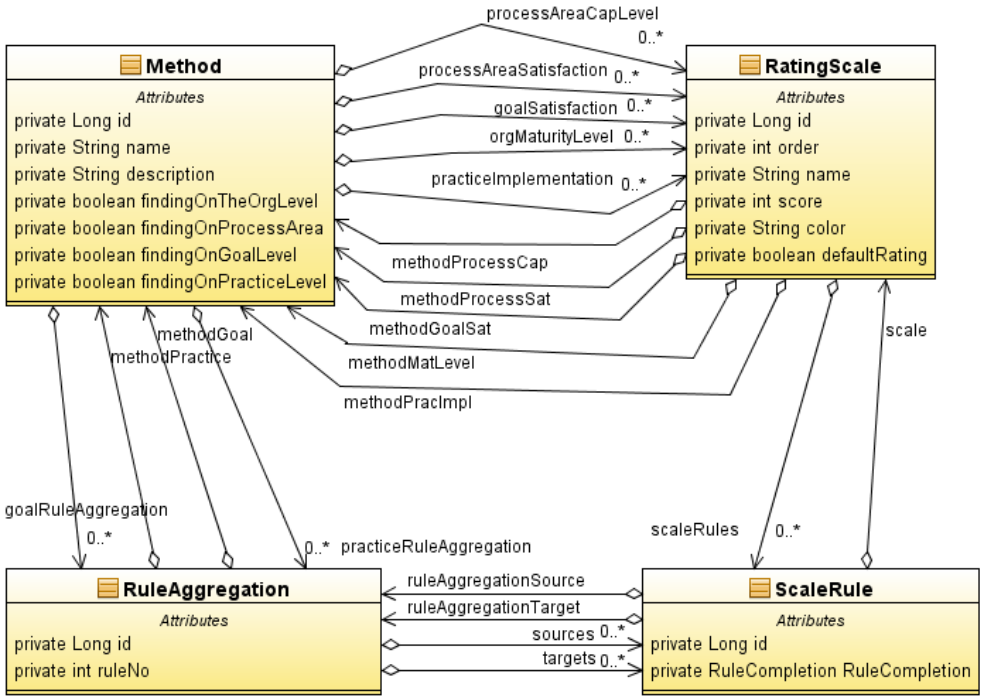
Každé pravidlo vždy určuje kolik musí být daných měřítek a jakého typu (`ruleAggregationSource`) aby nastal přepis na daná měřítka u jiných typů (`ruleAggeragationTarget`). Druh (zda jde o 0, nebo 0/1 nebo 1) je určen třídou `ScaleRule` v atributu `RuleCompletion`.

4.3.2 Balík model

Všechny třídy vyjma `ProcessGroup` rozšiřují abstraktní třídu `AcronymEntity`, takže mají definován akronym a jméno. Třída `ProcessGroup` definuje skupiny procesů, které jsou určeny pro každou metodu zvlášť a může jich být 0 až n. Do těchto skupin mohou pak být řazeny procesní oblasti, popsané třídou `ProcessArea`. Do procesní oblasti spadají cíle, definované třídou `Goal`. Generické cíle (jeden z dvou druhů cíle) však nejsou vázány do procesních oblastí a tak pro generický tak existuje vazba přímo na `Model`. V tomto případě pak vazba na `ProcessArea` není nastavena. `Goal` je vázán právě na jednu rodičovskou entitu a to buď `ProcessArea` nebo `Model`. Dále jsou definovány oboustranné 1,1 a 0:N vazby s následující sémantikou: `Goal` může obsahovat 0 až n `Practice` a `Practice` může obsahovat 0 až n `Artifact`.

4.3.3 Balík project

Základní entitou reprezentující projekt je třída `Project`. Na tuto třídu jsou vázány (ať už přímo nebo nepřímo) všechny entity, které se vztahují k projektu. V tomto modelu a podkapitole nejsou uvedeny třídy z Rating, ty jsou popsány v další podkapitole, kde je třída `Project` znovu uvedena. Na třídu Project se váže 0 až n tříd `TeamMember`, které definují členy týmu v projektu. Jde v podstatě o vazební tabulku mezi projektem a uživatelem s atributem určujícím roli uživatele. Dále je k projektu vázáno 0 až n `Evidence` reprezentující nalezené důkazy. Třída definuje atributy, kterými je důkaz popsán. Další samostatnou entitou jsou procesní instance, které jsou reprezentovány třídou `ProcessInstantiation`. Instancí je 1 až n u každého projektu. Každý důkaz pak může být mapován do několika



Obrázek 4.10: Entity class diagram balíku method

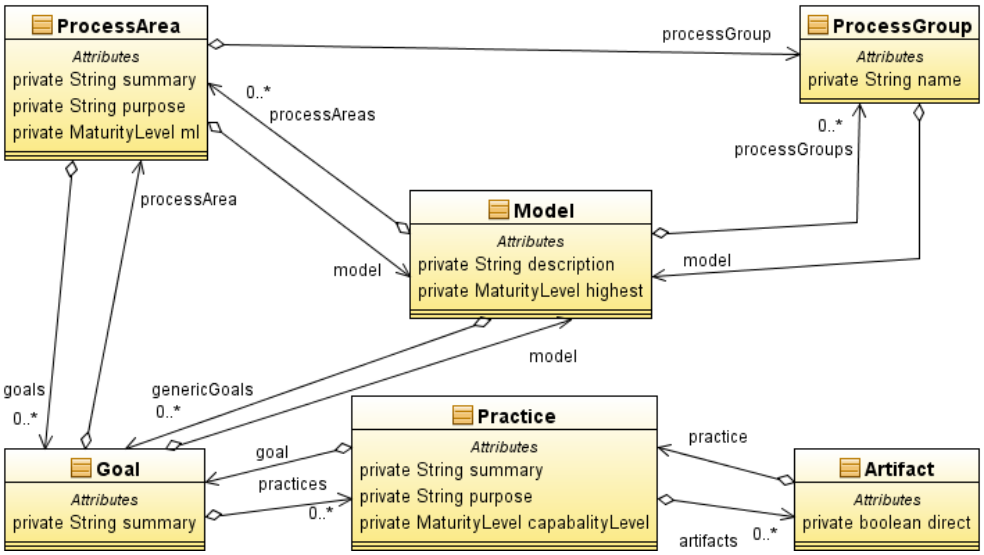
procesních instancí prostřednictvím vazební entity `EvidenceMapping`, která ke každému mapování definuje i některé další atributy. Důkazy ke každé procesní instanci mohou být hodnoceny. Tyto hodnocení je uchováváno v entitě `EvidenceRating`.

4.3.4 Balík project.rating

V tomto balíku je uchováváno celkové hodnocení organizace na všech úrovních v rámci projektu hodnocení. Pro definici hodnocení je použita třída `AbstractRating`, kterou všechny třídy, vyjma `Finding` rozšiřují a provádí vazbu na konkrétní hodnocené entity (např. cíl nebo procesní oblast). Jde o vazební entity mezi projektem (třída `Project` z balíku `project`) a danou entitou modelu s atributem hodnocení. Třída `Finding` je určena pro ukládání různých nálezů k hodnoceným entitám a je ve vazbách nepovinná. Hodnotící třídy mohou mít i další atributy, jak je vidět na schématu.

4.4 Popis implementace

Tato kapitola popisuje provedenou implementaci systému. Zabývá se popisem architektury a zvoleným řešením implementačně zajímavých částí kódu. Při implementaci byla použita řada moderních frameworků, předpokládá se, že je čtenář s těmito seznámen. Základ je popsán v podkapitole frameworky s odkazy k hlubšímu studiu.



Obrázek 4.11: Entity class diagram balíku model

4.4.1 Architektura

Aplikace používá klasický návrhový vzor *Model - View - Controller* (MVC), který poprvé v roce 1979 popsal Trygve Reenskaug a od té doby se používá při návrhu většiny software. Zaručuje oddělení vrstev a žádné zpětné závislosti. Velkou výhodou oddělení vrstev je jednoduché rozšiřování a upravování software se zanášením minima chyb. Tento architektonický vzor (návrh), je tak dokonale známý a používaný, že tato diplomová práce nemá ambice se řadit k nesčetně předchozím, které popisu „kola“ věnují alespoň jednu kapitolu.

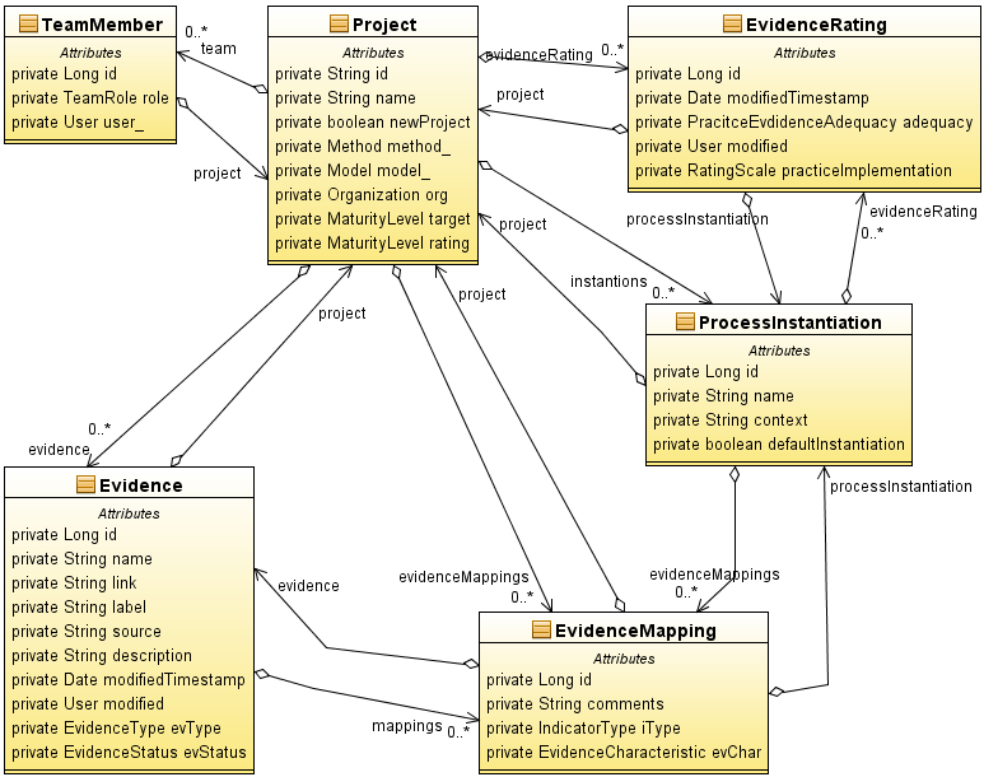
Kapitola se zabývá popisem konkrétní implementace jednotlivých částí MVC, zejména některých jejich zajímavých částí a popisuje použité frameworky.

4.4.1.1 Model

Použité technologie

ORM

V objektovém světě jazyka Java bylo zvoleno ORM řešení. ORM je zkratka z Object-Relationship-Mapping a jde o automatické mapování objektů na databázi. Tato problematika je složitější, než na první pohled vypadá, není vše automatické a i vývojář musí vědět, že za objekty stojí databáze a entitní třídy (přes které se provádí mapování na databázi) je třeba u složitějších aplikací psát s obezřetností. Pro implementaci bylo zvoleno notoricky známé Java Persistence API (JPA), které definováno ve standardu JSR-220. Toto rozhraní je velice jednoduché, avšak dostatečně silné pro implementaci složitých systémů. Používají se anotace místo složitých a nepřehledných XML souborů. Vývoj probíhá velice rychle. Jako



Obrázek 4.12: Entity class diagram balíku project

konkrétní implementace JPA byl zvolen nástroj Hibernate, jakožto jeden z nejstabilnějších a nejrychlejších frameworků spolehlivě implementující JPA.

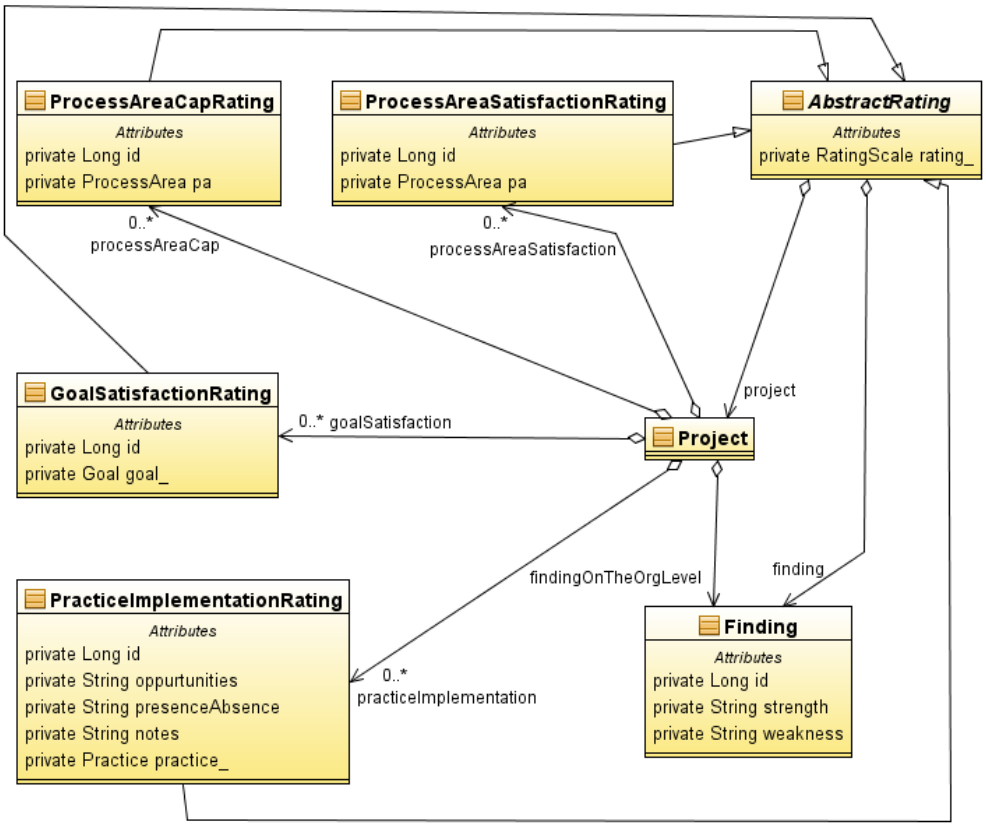
Dependency Injection / AOP

Jako dependecy-injection a AOP nástroj byl zvolen Spring ve verzi 3, který byl v době psaní diplomové práci ve verzi Release Candidate 2, Dne 16.12.2009 byla uvolněna finální verze. Již dříve bylo oznámeno, že pro aplikace, které mají být k dispozici začátkem roku 2010, je doporučeno použít tuto verzi, která oproti 2.5 přináší mnoho zlepšení, hlavně v MVC části. V modelu se tento framework stará o dependecy-injection, o správu AOP a o automatické spravování instancí beanů. Jde o nejrozšířenější a nejstabilnější nástroj pro vývoj enterprise Java aplikací, které poskytují srovnatelný výkon jako EJB 3.0, nabízí podstatně více možností a nevyžadují aplikační server[13], který vždy přináší spoustu komplikací.

Celá konfigurace Spring kontextu je v souboru `applicationContext.xml` a případných odkazovaných souborech z tohoto hlavního konfiguračního souboru.

Zabezpečení

Pro zabezpečení je použit Spring 3 Security, který implementuje standardy zabezpečení a



Obrázek 4.13: Entity class diagram balíku project.rating

umožňuje jednoduché zanesení zabezpečení do kódu. Je též dobře integrován se Springem.

Logování

Pro logování je napříč celou aplikaci použito rozhraní *Apache Commons Logging*. Jako konkrétní implementace bylo zvoleno *java.util.logging* (juli). Úrovně logování lze nastavit v souboru `logging.properties`, dle standardní syntaxe.

Použité návrhové vzory

GenericDao

DAO - Data Access Object, jde o objekt, který zajišťuje přístup k datům, zpravidla obsahuje CRUD metody (viz výše). Je definován pro každou entitu zvlášť. Objet se vytváří proto, aby došlo k zapouzdření přístupu do databáze. V podstatě obsahuje stále ten stejný kód, který by se stále opakoval. Proto je vhodné toto DAO napsat jednou, a pak pomocí generik (známých od Java 5) vytvořit konkrétní instance pro jednotlivé entity. Samozřejmě je toto možné provést i bez generik, ale přicházíme o jednu krásnou vlastnost a to typovou

kontrolu. Jeden ze způsobů, jak takové generické DAO implementovat, popisuje Petr Mellqvist [6]. DAO použité v mojí aplikaci drobně rozšiřuje o obecnou `find` metodu generické DAO popsané v jeho článku. Zde se pokusím nastínit, jak takové DAO vypadá.

DAO je definováno rozhraním Můžete poskytovat několik různých implementací, pro ukládání do různých úložišť. V aplikaci CMMI Tool je k dispozici JPA implementace (využívající Hibernate), není však problém pro optimalizaci výkonů DAO přepsat třeba přímo do Hibernate. Rozhraní vypadá následovně:

```
public interface GenericDao <T, PK extends Serializable> {

    /** Persists new object. */
    T create(T newObject);

    /** Updates entity. object. */
    T update(T transientObject);

    /** Finds entity by primary key.*/
    T read(PK id);

    /** Removes entity with specified id form database.*/
    void delete(PK id);

    /** Return list of all persisted entities.*/
    List<T> findAll();

    /** Executes specified named query with parameters.*/
    List<T> findByNamedQuery(String queryName, Object... parameters);
}
```

K tomuto rozhraní je pak definována implementace a její konkrétní instance je pak definována jak Spring beana, např. DAO pro entitní třídu `Model`:

```
<bean id="modelDao" class="cz.strmik.cmmitool.dao.GenericDaoImpl" autowire=
"byName">
     <constructor-arg> <value>cz.strmik.cmmitool.entity.model.Model</value>
</constructor-arg>
</bean>
```

V celé aplikaci je používán *autowiring* podle jména. Podle jména anotované proměnné Spring zkusí najít správnou beanu, která má stejné jméno. Proto není třeba psát mnoho zbytečných řádků s XML konfigurací, které se jen budou starat o nastavení instancí bean. Použití `modelDao` beany kdekoliv ve Spring kontextu je pak velice jednoduché (předpokládá se, že v konfigurace Spring kontextu je definovaná beana se stejným typem, viz výše):

```
@Autowired
private GenericDao<Model, Long> modelDao;
```

V první specifikaci generika je uvedena třída, pro kterou se DAO vytváří a v druhém je uveden typ primárního klíče.

Na vrstvě DAO jsou definovány následující transakce a entitní manažer

```
<!-- JPA setup -->
<bean id="entityManagerFactory" class="
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="cmmi-toolPU"/>
    <property name="loadTimeWeaver">
        <bean class=
"org.springframework.instrument.classloading.ReflectiveLoadTimeWeaver"/>
    </property>
</bean>
<bean id="transactionManager" class=
"org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
<context:annotation-config/>
<aop:config>
    <aop:pointcut id="daoServiceMethods" expression=
"execution(* cz.strmik.cmmitool.dao.*.* (..))"/>
    <aop:advisor advice-ref="txAdviceDao"
pointcut-ref="daoServiceMethods"/>
</aop:config>
<tx:advice id="txAdviceDao" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="create*" propagation="REQUIRED"/>
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="remove*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="loadUserByUsername" propagation="REQUIRED"/>
        <tx:method name="*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>
```

Entitní manažer se pak získá v implementaci DAO následovně:

```
@PersistenceContext
private EntityManager entityManager;
```

Servisní vrstva

Nad vrstvou DAO je vytvořena servisní vrstva, ve které se provádí veškeré operace (logika) na *backend* aplikace. DAO jsou volány primárně z této vrstvy. Pro zjednodušení mohou být DAO volána i z View vrstvy a proto jsou definovány transakce i na DAO. DAO je nezávislé na servisní i View vrstvě. Pouze servisní vrstva závisí na DAO. Servisní vrstva je uložena v balíku `cz.strmik.cmmitool.service` Na úrovní servisní i DAO vrstvy je

prostřednictvím Spring zajištěno transakční zpracování, transakce jsou propagovány. Předpokládá se, že čtenář je seznámen s tímto pojmem a práce nevysvětluje výhody tohoto řešení. Transakce jsou ve Spring konfiguračním souboru definovány následovně:

```
<aop:config>
    <aop:pointcut id="serviceMethods" expression="execution
(* cz.strmik.cmmitool.service.*.* (..))"/>
    <aop:advisor advice-ref="txAdviceService"
pointcut-ref="serviceMethods"/>
</aop:config>
<tx:advice id="txAdviceService"
transaction-manager="transactionManager">
    <tx:attributes> <tx:method name="*"
propagation="REQUIRED" /> </tx:attributes>
</tx:advice>
```

Servisní třídy jsou vždy definovány rozhraním a implementací. Všechny jsou definované jako „Spring managed beany“ a do aplikace se „injektují“ pomocí „autowiringu“ (automatického vložení). Definují a používají se stejně jako DAO, viz výše.

Aspekty

Aspekty umožňují velice zajímavé činnosti. Můžete napsat metodu, kterou pak pomocí XML souboru nakonfigurujete, aby byla volána před zavoláním nějaké metody, po zavolání nějaké metody, a podobně. Používá se zejména pro logování. V aplikaci CMMI Tool existuje jeden aspekt, který je volán po načtení entity, jelikož standardní JPA anotace z nějakého důvodu nefungovala. Provede se konfigurace, kdy se řekne, za jakých podmínek a kdy má být daný aspekt volán. Nejlépe to je vidět přímo na konkrétním použití. Do konfigurace beanů nastavíme automatické konfiguraci dle aplikací

```
<aop:aspectj-autoproxy/>
<bean id="daoAspect" class="cz.strmik.cmmitool.aspect.DaoAspect" />
```

a samotný aspekt, který je volán po návratu z metody, která vrací objekt v `methodDao` pak vypadá následovně

```
@Aspect
public class DaoAspect {
    ...
    @AfterReturning(
        pointcut="bean(methodDao)",
        returning="retVal")
    public void setupMethodBools(Object retVal) {
    ...
   }
...
}
```

Zabezpečení a Uživatelé

Pro zabezpečení aplikace je použit napříč všemi vrstvami Spring Security [14]. Na vrstvě modelu se stará o abstrakci k přístupu k uživatelům. Pro získávání uživatelů je implementováno zvláštní DAO, objekt nesoucí uživatele implementuje Spring rozhraní `UserInfo`, a tak je zajištěno, že s tímto objektem lze pracovat napříč celou aplikací a lze těchto informací (například, jaké je uživatel role) využívat i v deklarativním zabezpečení jednotlivých částí aplikace a to buď na úrovni kódu nebo i webového rozhraní. Přihlašovaní a odhlašování je řešeno též v kapitolách View a Controller, v tomto místě se zabýváme pouze práci s uživateli. Cele zabezpečení je konfigurováno v souboru `security.xml`, který je zahrnut do hlavní konfigurace kontextu. Aplikaci lze lehce rozšířit o možnost autentizace například oproti LDAP a podobně. S minimem kódování a s využitím vlastností frameworku *Spring 3 Security*.

Hesla se do databáze ukládají hašované pomocí SHA-1, solené uživatelským jménem. Následující část konfigurace ukazuje aktuální nastavení. Zde by bylo možno přidat jiný zdroj – například LDAP – do uzlu *autentication-manager*.

```
<authentication-manager>
    <authentication-provider user-service-ref="userDao">
        <password-encoder hash="sha" >
            <salt-source user-property="username"/>
        </password-encoder>
    </authentication-provider>
</authentication-manager>

<b:bean id="passwordEncoder" class="org.springframework.security.
authentication.encoding.MessageDigestPasswordEncoder">
    <b:constructor-arg value="sha"/>
</b:bean>

<b:bean id="securityContextFacade" class="cz.strmik.cmmitool.
security.SecurityContextHolderFacade" autowire="byName" />
```

`UserDao` rozšiřuje `UserDetailsService` a překrývá metodu *UserDetails loadUserByUsername(String userName)*, která zajišťuje získávání uživatele z databáze. DAO se dále stará o správu uživatelský účtů administrátorem. Z rozhraní `UserDetails` jsou zajímavé zejména metody *List<GrantedAuthority> getAuthorities()*, která vrací seznam autorit, které daný uživatel má. Napříč celým kódem a i ve webové části lze zabezpečit jednotlivé části aplikace na základě faktu, že přihlášený uživatel musí mít nějakou námi definovanou autoritu.

4.4.1.2 View

Pro View vrstvu byla zvolena prověřená a široce známá technologie Java Server Pages (JSP), umožňující generovat dynamické stránky. Samo o sobě by toto bylo těžkopádné a tak pro zpracovávání vstupů a generování výstupů byly krom standardních knihoven značek byly

použity JSTL, Spring knihovny a například pro formuláře pro integraci přímo s *UniForm* formulářemi byla vyvinuta vlastní knihovna značek pojmenovaná `strmik`. Pro vykreslování dynamických prvků, řazení tabulek na klientovi v prohlížeči, generování stromových struktur a AJAX zpracování některých požadavků byla použita javascriptová knihovna *jQuery*. Veškeré konfigurace jsou v souboru `init.jspf`.

Jednotlivé JSP soubory jsou dle významu v různých adresářích (a podadresářích) logicky poskládané tak, aby na základě kontextové cesty bylo možno automaticky generovat drobečkovou navigaci, což implementuje jedna ze značek z vlastní knihovny značek.

Aplikace je internacionalizovaná a lokalizovaná do angličtiny. Pro ukládání textů jsou použity standardní *properties* soubory a díky internacionalizaci je aplikace připravena na lokalizaci do dalších jazyků.

Přihlášení a odhlášení uživatele

Systém umožňuje zapamatovat si přihlášeného uživatele. Pokud uživatel tuto volbu zvolí, pak si systém uživatele pamatuje na základě cookie prohlížeče a tento se ani po vypršení sezení (session) nemusí znovu přihlašovat. Toto je implementováno prostřednictvím *Spring 3 Security* frameworku, aplikace používá vlastní přihlašovací obrazovku definovanou v souboru `login.jsp`.

Část konfigurace zabývající se zamezením přístupu z `security.xml` je uvedena zde:

```
<http auto-config="true">
    <intercept-url pattern="/admin/**" access="ROLE_SUPERVISOR"/>
    <intercept-url pattern="/qmanager/**" access="ROLE_QUALITY_MANAGER"/>
    <intercept-url pattern="/appraisal/**"
access="IS_AUTHENTICATED_REMEMBERED"/>
    <intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/>

    <logout logout-success-url="/"/>
    <remember-me key="cmmiTool"/>

    <form-login login-page="/login.jsp"
authentication-failure-url="/login.jsp?login_error=1"/>
</http>
```

Kořenová složka je dostupná všem, další podložky jen určitém rolím a část zabývající se prováděním hodnocení vyžaduje přihlášeného uživatele.

Knihovna značek

Protože ve View vrstvě v JSP by se často opakoval stejný kód, v rámci vývoje byla vytvořena knihovna značek, která obsahuje sadu značek pro vytváření polí UniForm (viz 4.2.7) formulářů přímo z atributů bean. Knihovna je definována souborem `strmiktags_library.tld` a mimo jednoduchých značek generující formulářová pole (text, výběr z hodnot, ...) obsahuje následující zajímavé značky.

Značka `<breadcrumbs/>` je určena pro automatické generování drobečkové navigace. Tak jak se uživatel postupně zanořuje v aplikaci volbou různých menu, tak se dostává stále hlouběji do aplikace. Tuto strukturu aplikace reflektuje i adresářová struktura aplikace, ve které jsou uloženy JSP soubory. A tak podle cesty (skutečně podle request URI), kde se JSP nachází lze automaticky vygenerovat drobečkovou navigaci. Názvy adresářů se překládají s pomocí *.properties* souboru, takže je možno navigaci jednoduše lokalizovat.

Další značkou je `<tree/>`, která generuje HTML kód pro *jQuery* komponentu *jTree*. Jako vstup bere strom vygenerovaný v komponentě *Controller*. Strom je reprezentován klasickou strukturou, kde každý uzel stromu má list odkazů na podřazené uzly. Značce se předá kořenový uzel a ta na základě této informace vygeneruje strom. Umí i některé pokročilé funkce (jako generování odkazů, generování odkazů ke smazání uzlu, vykreslení nějaké ikonky či barvy uzlu a podobně). Vše je dokumentováno.

Poslední značkou mimo značky generující pole formuláře je značka `<urlencode>`, která pouze zajišťuje kódování URL pomocí metody `URLEncoder.encode()`.

4.4.1.3 Controller

Pro implementaci Controller byl použit Spring 3 MVC. Všechny třídy spadající do této vrstvy aplikace jsou uloženy v balíku `cz.strmik.cmmitool.web.controller`. Jde o třídy se Spring anotací `@Controller`, které vždy dědí z třídy `AbstractController`, která implementuje společné funkce. Pomocí `@Autowired` anotací jsou *injektovány* všechny potřebné beany servisní (případně i DAO) vrstvy. Každý Controller obsluhuje část aplikace (jednu logiku, jednu kontextovou cestu) a všechny operace na ní prováděné. Jsou používány atributy (zejména pro vícekrokové formuláře), které ukládají svoje hodnoty do session. Tyto všechny atributy jsou definované v *package protected* třídě `Attribute`. Zde je ukázka významných částí kódu jednoho Controller s extra komentáři popisující význam.

```
@Controller
// mapování do kontextu
@RequestMapping("/appraisal/evidence")
// atributy uložené v~session
@SessionAttributes({Attribute.PROJECT, Attribute.EVIDENCE})
public class EvidenceController extends AbstractController {

    // seznam všech použitých JSP, které se zobrazují
    private static final String DASHBOARD = "/appraisal/evidence/dashboard";
    // ...

    // servisní vrstva
    @Autowired
    private EvidenceService evidenceService;
    // ...

    // atributy modelu, k~těmto lze přistupovat z~JSP
    @ModelAttribute("types")
```

```
    public Collection<EvidenceType> getTypes() {
        // ...
        return levels;
    }
    // ...

    // výchozí GET požadavek, vracející dashboard
    @RequestMapping(method = RequestMethod.GET, value = "/")
    public String showDashboard(ModelMap model, HttpSession session) {
        // ... příprava dat pro dashborad a poté zobrazení příslušného JSP.
        return DASHBOARD;
    }

    // jakýkoliv add.do v~kontextu obsluhovaném controlleru
    @RequestMapping(value = "/add.do")
    public String addEvidence(ModelMap model) {
    // ...

    // další metody a privátní metody

    // logování
    private static final Log log =
         LogFactory.getLog(EvidenceController.class);
}
```

Další použitou funkcionalitou jsou tzv. „property editory“, umožňující přímo mapovat uživatelský vstup do určitých konkrétních proměnných a také převádět objekty do formulářového pole pro uživatele. Toho se používá zejména ve formulářích, kde je třeba objekt uživateli zobrazit v opravitelné podobě a pak opět uživatelský vstup převést zpět do reprezentace objektu. Všechny jsou registrované v třídě `BindingIntializer` a umožňují přímo v JSP a Controller pracovat s proměnnými, které jsou automaticky (pomocí property-editoru) převedeny na `String` a ze Stringu. Nejlépe to je vidět na následujícím příkladu. Pro každý typ, který automaticky chceme takto mapovat, se vytvoří třída rozšiřující třídu `PropertyEditorSupport`. Spring samozřejmě umí automaticky převádět hodně věcí, například výčtové prvky, avšak pro vlastní objekty je třeba napsat vlastní kód:

```
public class RatingScaleEditor extends PropertyEditorSupport {

    @Override
    public void setAsText(String text) {
       // ...
    }

    @Override
    public String getAsText() {
       // ...
    }
}
```

Další problematikou řešenou v každé aplikaci, která interaguje s uživatelem, je validace uživatelského vstupu. Ta se ve Springu řeší implementací validátoru, který provádí validaci uživatelského vstupu. Tato třída implementuje rozhraní `Validator`. V CMMI Tool existuje třída `AbstractValidator`, která implementuje toto rozhraní a ještě obsahuje nějaké společné řetězce pro více validátorů. Tyto všechny pak rozšiřují abstraktní třídu `AbstractValidator`. S výhodou se dají používat metody z třídy ValidationUtils, které ulehčí práci, zejména z JavaBean třídami.

```
public class AcronymValidator extends AbstractValidator {

    @Override
    public boolean supports(Class<?> clazz) {
        return AcronymEntity.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmpty(errors, "acronym", "field-required");
        ValidationUtils.rejectIfEmpty(errors, "name", "field-required");
    }

}
```

4.4.1.4 Diagram nasazení

Tato podkapitola obsahuje typický diagram nasazení aplikace CMMI Tool. Diagram nasazení je vytvořen dle standardní UML 2.1 notace a je uveden na obrázku *Diagram nasazení CMMI Tool* (4.14). Jde o server-klient řešení, kde serverem je servletový kontejner Tomcat, který dále využívá databázový server pro ukládání dat a LDAP jako zdroj uživatelů. Klientem je zde počítač uživatele, na kterém běží webový prohlížeč a v něm je otevřena aplikace CMMI Tool.

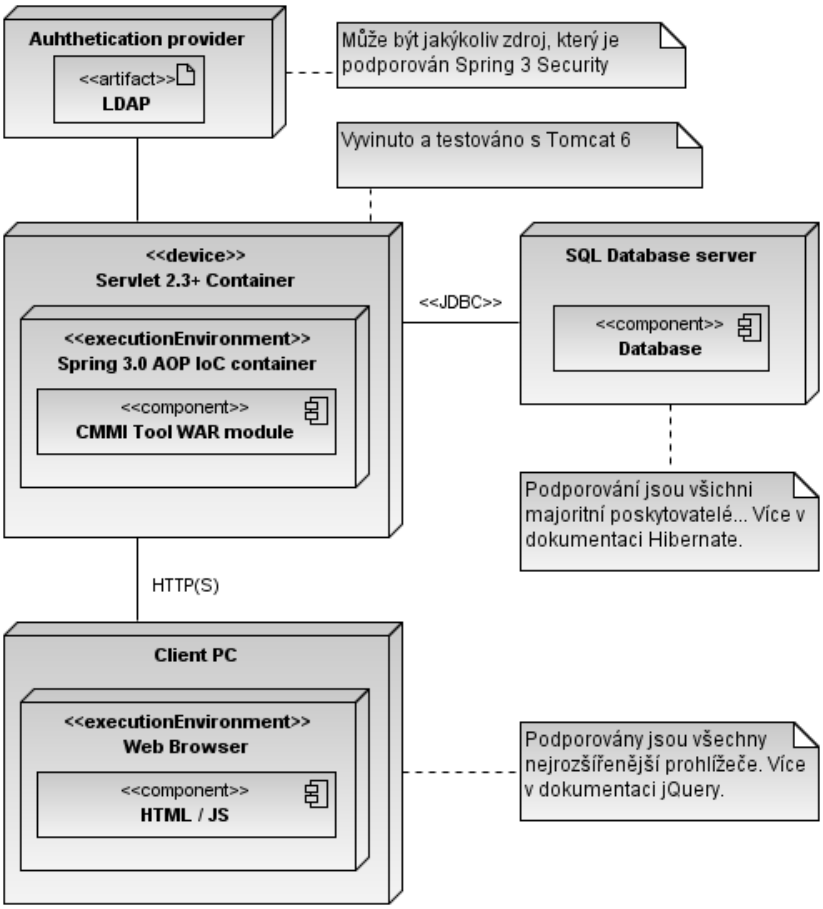
4.4.2 Použité frameworky

Tato kapitola popisuje vlastnosti použitých frameworků. Uvádí důvody, proč byly dané frameworky zvoleny pro implementaci. Nabízí inspiraci pro další vývojáře, kteří by se mohli rozhodnout tyto frameworky taktéž použít. Čtenář, který má zájem o podrobnější vhled je odkázán na použitou literaturu, zejména referenční příručky k jednotlivým frameworkům.

4.4.2.1 Java persistence API

[8] Java persistence API poskytuje POJO[1] model persistence dat pro objektově-relační mapování. Bylo vyvinuto EJB 3.0 expertní skupinou, jako část JSR 220, ale jeho použití není

1. POJO - plain old Java object, starý Java objekt, bez složitých vazeb, konfigurací... Pracující pouze s doménovými problémy a nikoliv se složitou infrastrukturou aplikace.



Obrázek 4.14: Diagram nasazení CMMI Tool

limitováno pro softwarové komponenty EJB. Může být přímo použito webovými aplikacemi a aplikačními klienty, dokonce i mimo Java EE platformu, v Java SE aplikacích. Více v JSR 220[2].

CMMI Tool využívá toho, že JPA může být použito i v Java SE aplikacích a používá jeho rozhraní k definici entitních tříd a metod.

4.4.2.2 Hibernate

[7] Hibernate je výkonná objektově/relační perzistentní a dotazovací služba. Hibernate vás nechá vyvíjet perzistentní třídy plně následující objektově-orientační idiom - včetně asociací, dědičnosti, polymorfismu, kompozice a kolekcí. Nechá vás vyjadřovat dotazy ve vlastní přenositelném rozšířeném SQL jazyku označeném jako HQL, stejně jako v nativním SQL,

2. <http://www.jcp.org/en/jsr/detail?id=220>

nebo s objektově-orientovaným *Criteria API*.

Hibernate ve verzi 3 nabízí vyspělý a v současné době asi nejpoužívanější framework pro ORM v Java. Nabízí jak vlastní sadu anotací, tak i implementaci JPA, která byla zvolena pro aplikaci CMMI Tool. Podporuje většinu databází, nabízí velkou možnost detailní konfigurace keší (first-level, second-level, ...) pro jemné ladění výkonu. Na rozdíl od ostatních frameworků nabízí i přímou kontrolu generovaného SQL.

4.4.2.3 Spring 3 Core

[13] Spring jako platforma umožňuje aplikacím aby byly postavené z POJO. Toto je pravdivé pro programovací model Java SE stejně jako v několika prostředích implementujících částečně nebo úplně Java EE. Spring umožňuje službám, aby byly aplikovány na POJO ne invazivním způsobem.

Mezi klíčové vlastnosti jádra Springu patří:

- IoC (Inversion of Control), DI (Dependency Injection)

  Jedná se o to, že aplikační kód nezajímá, kde se vezme ta která instance třídy nebo zdroj, který aplikace použije. Deklarativně se napíše, že daná třída vyžaduje dané objekty. Spring kontejner zajistí vytvoření a správu instancí potřebných tříd a jejich injektování (vložení) do tříd. Tyto informace jsou nakonfigurované v XML souboru. Díky *Autowired* anotacím se nemusí již ručně nastavovat kam má být který objekt vložen, ale lze to velké míry automatizovat, díky čemuž je autor odstíněn od psaní nudné infrastrukturní konfigurace.

- Validace, data-binding, obalování dat, propertyeditor

  Spring nabízí jednoduché vytváření a automatické používání validátorů, umožňuje automatické vázání (binding) dat pomocí property editorů a obalování dat. Tak aby kód od sebe byl vždy správně oddělen a byl velice jednoduše rozšiřitelný a spravovatelný.

- Spring Expression Language

  Unifikovaný EL lze používat pro konfiguraci, vyhledávání Spring beanů. Velice ulehčuje konfiguraci.

- Aspekty

  Kontejner nabízí použití aspektů, což znamená, že vytvoříte kód a pomocí konfigurace můžete navěsit jeho volání kamkoliv do kódu (např. s využitím Spring EL). Vhodné například pro logování, přidávání další funkcionality, další validace a podobně. Konkrétním příkladem může být metoda, která se volá před vstupem do určitých metod, nebo po jejich výstupu. Spring implementuje *aspect4j* rozhraní i vlastní. Hlavním přínosem této funkcionality je udržení a tvorba jednoduchého a přehledného kódu, zabývající se pouze doménovou oblastí a nikoliv infrastrukturou.

- Testování

  POJO aplikace mohou být jednoduše testovatelné bez Spring kontejneru v *jUnit* nebo *TestNG*. Spring je s těmito frameworky implementován a přináší možnost vytváření tzv. „mock“ objektů, které výrazně urychlí a zjednoduší testování. Jde o objekty, které se chovají podle předdefinovaného vzoru a slouží jako vstupy pro testované třídy, tak abychom testem testovali právě jednu danou třídu a nikoliv jiné třídy. Říká se tomu testování v izolaci. Spring umí „mockovat“ velikou škálu standardních objektů, JNDI počínaje a JPA konče. Má připravené též třídy pro testování aplikací ve Spring MVC. Nabízí jak jednotkové, tak integrační testování.

- Transakce

  Spring poskytuje kompletní subsystém pro správu transakcí a transakční zpracování. Umožňuje řídit JPA transakce a transakce celkově podrobně a jednoduše definovat.

4.4.2.4 Spring 3 MVC

[13] Spring 3 model-view-controller (MVC) framework je navržen okolo DispatcherServlet, který obsluhuje požadavky na zpracovatele (handlers), s konfigurovatelným mapováním zpracovatelů, rozhodování View, lokalizací a tématu stejně jako s podporou pro nahrávání souborů. Výchozí zpracovatel je založen na @Controller a @RequestMapping anotacích, nabízejíc širokou škálu flexibilních zpracovatelských metod. S příchodem Spring 3 anotační mechanizmus @Controller umožňuje vytvoření RESTful webových služeb skrz @PathVaraiable anotací a dalších vlastností.

To všechno vede k velice rychlému vývoji přehledných aplikací. Spring MVC může být navíc integrován snad se všemi používanými frameworky pro View vrstvu, JSP & JSTP počínaje, přes Tiles, Velocity, XSLT, JaperReports apod. Aplikace CMMI Tool využívá pro View vrstvu JSP s JSTL. Framework může být dokonce integrován se stávájícími MVC frameworky, jako je například Struts nebo JSF. V dnešní době nadcházejících portálů Spring MVC prostřednictvím portlet MVC nabízí též podporu Java Portlet API 2.0.

4.4.2.5 Spring 3 Security

[14] Spring Security poskytuje komplexní služby zabezpečení pro aplikace založené na J2EE. Je speciální zaměřen na aplikace vytvořené s použitím Spring Frameworku a využívá spoustu jeho vlastností.

Nabízí zabezpečení (autentizaci a autorizaci) na všech vrstvách aplikace. Nabízí též integraci se všemi běžnými standardy (HTTPS BASIC, LDAP, Kerberos, OpenID, ...), kompletní přehled je v referenční příručce. V aplikaci k těmto konkrétním implementacím programátor přistupuje transparentně. Framework nabízí hojně používanou funkci *zapamatuj si mě*.

Jeho použití je naprosto přímočaré, v podstatě jde o to, vytvořit jeden Spring konfigurační soubor. S pomocí anotací pak lze zabezpečovat třídy, metody a podobně. Framework

vychází z integrace *Acegi*[3] frameworku se Springem. Při implementaci lze používat též Acegi konfigurační soubory a anotace.

4.4.2.6 jQuery

Tento framework se stal hlavním a jediným kandidátem na AJAXové vylepšení uživatelského rozhraní. Není rozšířenějšího, známějšího a nejvíce podporovaného volně šiřitelného frameworku. Existuje spousta doplňujících modulů. Framework umožňuje AJAXové vylepší uživatelské rozhraní bez podrobné znalosti java skriptu. Aplikace ho používá zejména pro zobrazování oken, validaci, zobrazování dynamického stromu a uživatelského řazení tabulek. Podrobné informace lze nalézt na adrese <`http://jquery.com/`>

4.4.2.7 Uniform formuláře

Jde o framework, který nabízí tvorbu unifikovaných formulářů a jejich jednoduchou „customizaci“. Pro tento projekt se rozhodl kdysi i nejrozšířenější portálový projekt Liferay[4]. Pomocí tohoto frameworku jsou vytvářeny přehledné a standardní formuláře, nikoliv prostřednictvím tabulek, ale *div* značek. Mají standardní hlášení o úspěchu, o chyby při validaci a podobně. Podrobnější informace lze nalézt na domovské adrese <`http://www.sprawsm.com/uni-form/community/page/documentation`>. Pro vytváření jednotlivých polí formuláře a nativní integraci se Spring MVC byla vytvořena knihovna značek, které umožňuje automatické generování formulářů. Uniform zajistí stejné a přehledné formuláře napříč celou aplikaci, jednoduše opravitelné a dobře vypadající i bez úprav a znalosti CSS. Vytvořené formuláře splňují všechny požadavky na přístupnost a obsahují moderní HTML kód.

3. <`www.acegisecurity.org`>
4. <`www.liferay.com`>

Kapitola 5

Závěr

Hlavním úkolem této práce bylo analyzovat a implementovat nástroj pro zavádění systému řízení kvality, který by měl mít otevřené a veřejně dostupné zdrojové kódy. A měl by právě pokrýt nedostatky v současné době existujících systémů, hlavně v možnosti centrální databáze a sdíleného přístupu více uživatelů. Současně jedním z hlavních kritérií byl fakt, že aplikace musí být dále rozšiřitelná a upravitelná, protože je jasné, že pojmout tak rozsáhlé téma jako je CMMI a software pro jeho zavádění za dva roky v jednom člověku v rámci jedné diplomové práce je nemožné. Toto bylo přiloženou implementací splněno. Systém splňuje analyzované případy užití. Pro implementaci byly použity rozšířené a známe frameworky, zejména Spring 3, jehož použití zaručuje jednoduchou rozšiřitelnost. Na webu aplikace na <http://code.google.com/p/appraisaltool/> jsou k dispozici jak kompletní zdrojové kódy systému, tak balík pro jednoduchou instalaci a nasazení. V současné době je tu prostor pro další pokračování a rozšiřování aplikace, zejména v oblasti pokročilých výstupních sestav a zpráv o probíhajícím zavádění systému kvality. Tak, aby se ze systému, který umožňuje sběr kategorizaci a přehled důkazů o činnostech potvrzující provádění systému kvality, stal plnohodnotný systém, který by po té byl použitelný i pro profesionální firmy zabývající se zaváděním CMMI.

5.1 Testování

Testování aplikace proběhlo formou uživatelského testování oproti analyzovaným případům užití. Tyto případy užití aplikace plně splňuje a je proto připravená fungovat jako důležitý pomocník při zavádění systému řízení kvality CMMI. Mimo tohoto uživatelského testování jsou v aplikaci připraveny jednotkové testy, které umožňují rozšiřování a úpravu aplikace bez zanesení regresních chyb.

5.2 Přínosy

Používáním nástroje uživatelé dosáhnou hned několika benefitů. Zejména jde o velkou úsporu času i peněz. Hlavní činnosti – evidence, kategorizace a hodnocení důkazů jsou prokryty funkcionalitou aplikace. Hlavní výhodou je centralizovaný a webový přístup k celému systému. To také přináší veškeré výhody tenkého klienta, jako je žádná instalace na počítači uživatele a přístup k informacím odkudkoliv, počínaje počítači a konče třeba mobilními telefony. Nástroj přináší též zpříjemnění a zpřehlednění celé práce při zavádění systému řízení

kvality. Webový klient s využitím posledních de facto standardních webových technologií je komfortním nástrojem.

Systém může být díky použitým technologiím integrován se stávajícími systémy, které jsou již ve společnosti používány, jako je na příklad LDAP. Tím, že je pro ukládání informací využívána výhradně relační databáze, je umožněno jednoduché zálohování uživatelských dat. Dostupné zdrojové kódy aplikace pod licencí GNU GPL 2 na prestižním serveru code.google.com <`http://code.google.com/`> přináší možnost jakékoliv softwarové firmě (i jednotlivcům) si aplikaci upravit, případně doplnit o vlastní funkcionalitu, která je vzápětí uvolněna pro všechny další uživatele. Aplikace byla s tímto úmyslem dekomponována a umožňuje jednoduchou rozšiřitelnost a modifikovatelnost. Popisem implementace se z tohoto důvodu zabývá jedna kapitola.

Aplikace nemá v současné době žádné grafické téma, jelikož grafická práce není mojí doménou. Prezentační vrstva je však navržena tak, že využívá moderní způsoby zobrazování (elementy div, framework jQuery, formuláře tvořené pomocí UniForms), což umožňuje jednoduché přidání uživatelských stylů zkušenějším a nadanějším grafikem. Aplikace je lokalizována do anglického jazyka. Architektura aplikace prezentační vrstvy umožňuje jednoduchou lokalizaci do dalších jazykových verzí.

Příloha A

Obsah přiloženého CD

- Zdrojový text práce v XML ve formátu DocBook
- Kompletní UML 2 diagramy ve formátu PNG
- Barevné obrazovky aplikace ve formátu PNG
- PDF forma práce
- Distribuční soubory implementovaného systému (binární obraz WAR)
- Distribuční soubory implementovaného systému ve formě bundle s Tomcat 6
- Všechny potřebné knihovny pro kompilaci a běh aplikace
- Zdrojové kódy aplikace

Příloha B

Nasazení aplikace

Tento návod popisuje nasazení distribuovaného bundle se servletovým kontejnerem Tomcat 6 a všech potřebných knihoven (Spring 3 + Hibernate), na kterých CMMI Tool závisí. Instalace je zde popsána v krocích.

Požadavky

- Java SE 5.0 JRE nebo vyšší s nakonfigurovanou cestu v systémové proměnné JRE_HOME
- Nástrojem Hibernate podporovaná databáze. Předkonfigurováno pro výchozí konfigurace *PostgreSQL* na localhost.
- Okolo 50 MB místa na disku a volný port číslo 8080

Instalační kroky

1. Rozbalte bundle (všechny následující cesty jsou relativní k rozbalenému adresáři Tomcat z bundle)
2. Pokud používáte jinou databázi, než PostgreSQL, potom musíte do adresáře `/lib` nakopírovat JDBC ovladač pro tuto databázi.
3. Přejděte k `webapps/cmmi-tool/WEB-INF/classes/META-INF/persistence.xml` a otevřete ho v textovém editoru
4. Vyplňte vlastnosti připojení k vaší databázi. Pokud nerozumíte nějaké hodnotě, vyhledejte googlem „persistence.xml“.

   Musíte nastavit následující vlastnosti:

```
<!--  database setup   -->
<property name="hibernate.dialect"
    value="org.hibernate.dialect.PostgreSQLDialect" />

<!--  connection to database  -->
<property name="hibernate.connection.url"
    value="jdbc:postgresql://localhost:5432/cmmi-tool" />
<property name="hibernate.connection.password" value="cmmi" />
<property name="hibernate.connection.username" value="cmmi" />
```

5. Přejděte do složky `/bin`, ujistěte se, že v systémové proměnné JRE_HOME je cesta k vaší JRE a do příkazové řádky zadejte **./startup** nebo **startup.bat** (záleží na Vašem operačním systému)

A instalace je hotova. V případě jakýchkoliv problémů sledujte log `/logs/castalina.out`. CMMI Tool je nyní nasazen na URL <`http://localhost:8080/cmmi-tool/`>

Výchozí jméno a heslo administrátora je *admin/admin*, doporučuje se okamžitě po přihlášení změnit.

Příloha C

Měřítka

Tato příloha definuje XSD gramatiku pro XML soubor s výchozími měřítky pro SCAMPI. Uvádí též příklad platného XML souboru. Tento soubor je součástí distribuce aplikace.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.strmik.cz/cmmi/ratingSchema"
    xmlns:tns="http://xml.strmik.cz/cmmi/ratingSchema"
    elementFormDefault="qualified">
    <xsd:element name="ratings">
     <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
       <xsd:element name="rating">
        <xsd:complexType>
         <xsd:sequence minOccurs="1" maxOccurs="unbounded">
          <xsd:element name="scale">
           <xsd:complexType>
            <xsd:sequence>
             <xsd:element name="name">
              <xsd:complexType>
               <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                 <xsd:attribute name="lang" type="xsd:string" use="required"/>
                </xsd:extension>
               </xsd:simpleContent>
              </xsd:complexType>
             </xsd:element>
             <xsd:element name="color">
              <xsd:complexType>
               <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                 <xsd:attribute name="type" type="xsd:string" use="required"/>
                </xsd:extension>
               </xsd:simpleContent>
              </xsd:complexType>
             </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="score" type="xsd:int" use="required"/>
           </xsd:complexType>
          </xsd:element>
```

```
         </xsd:sequence>
         <xsd:attribute name="id" type="xsd:ID" use="required"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Příklad XML souboru s měřítky hodnocení dle výše uvedené XML gramatiky.

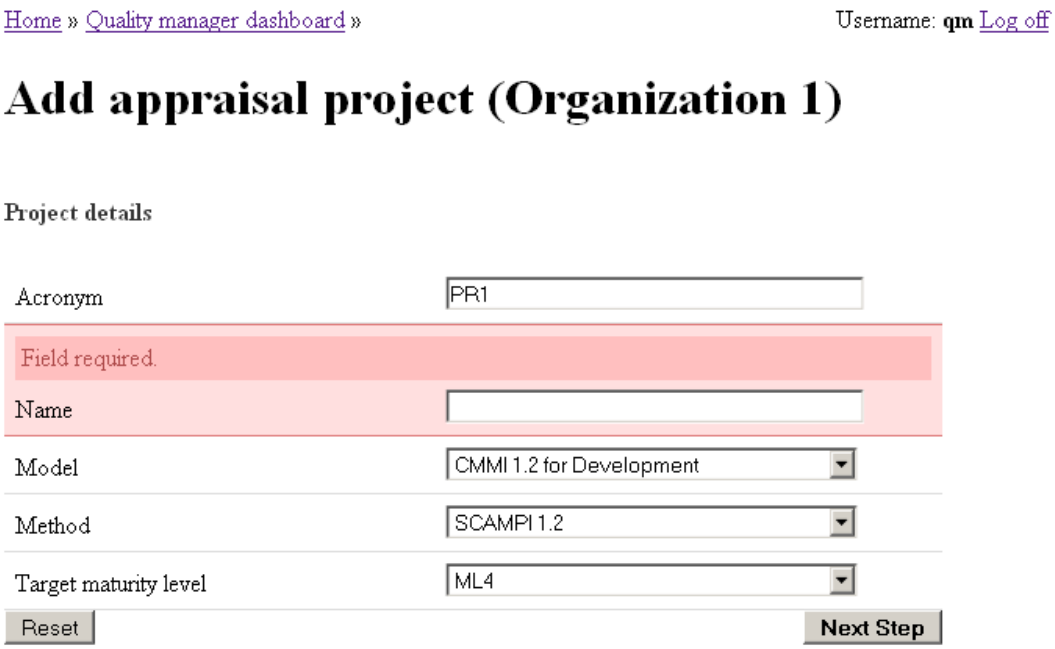
```
<ratings>
    <rating id="orgMaturityLevel">
        <scale score="-1">
            <name lang="en">Not rated</name>
            <color type="html">red</color>
        </scale>
        <scale score="1">
            <name lang="en">ML1 - Initial</name>
            <color type="html">green</color>
        </scale>
        <scale score="2">
            <name lang="en">ML2 - Managed</name>
            <color type="html">green</color>
        </scale>
        <scale score="3">
            <name lang="en">ML3 - Defined</name>
            <color type="html">green</color>
        </scale>
        <scale score="4">
            <name lang="en">ML4 - Quant. Managed</name>
            <color type="html">green</color>
        </scale>
        <scale score="5">
            <name lang="en">ML5 - Optimizing</name>
            <color type="html">green</color>
        </scale>
    </rating>
    <rating id="processAreaCapLevel">
        <scale score="-1">
            <name lang="en">Not rated</name>
            <color type="html">red</color>
        </scale>
        <scale score="0">
            <name lang="en">CL0 - Incomplete</name>
            <color type="html">red</color>
        </scale>
        <scale score="1">
            <name lang="en">CL1 - Performed</name>
```

```
            <color type="html">green</color>
        </scale>
    </rating>
</ratings>
```

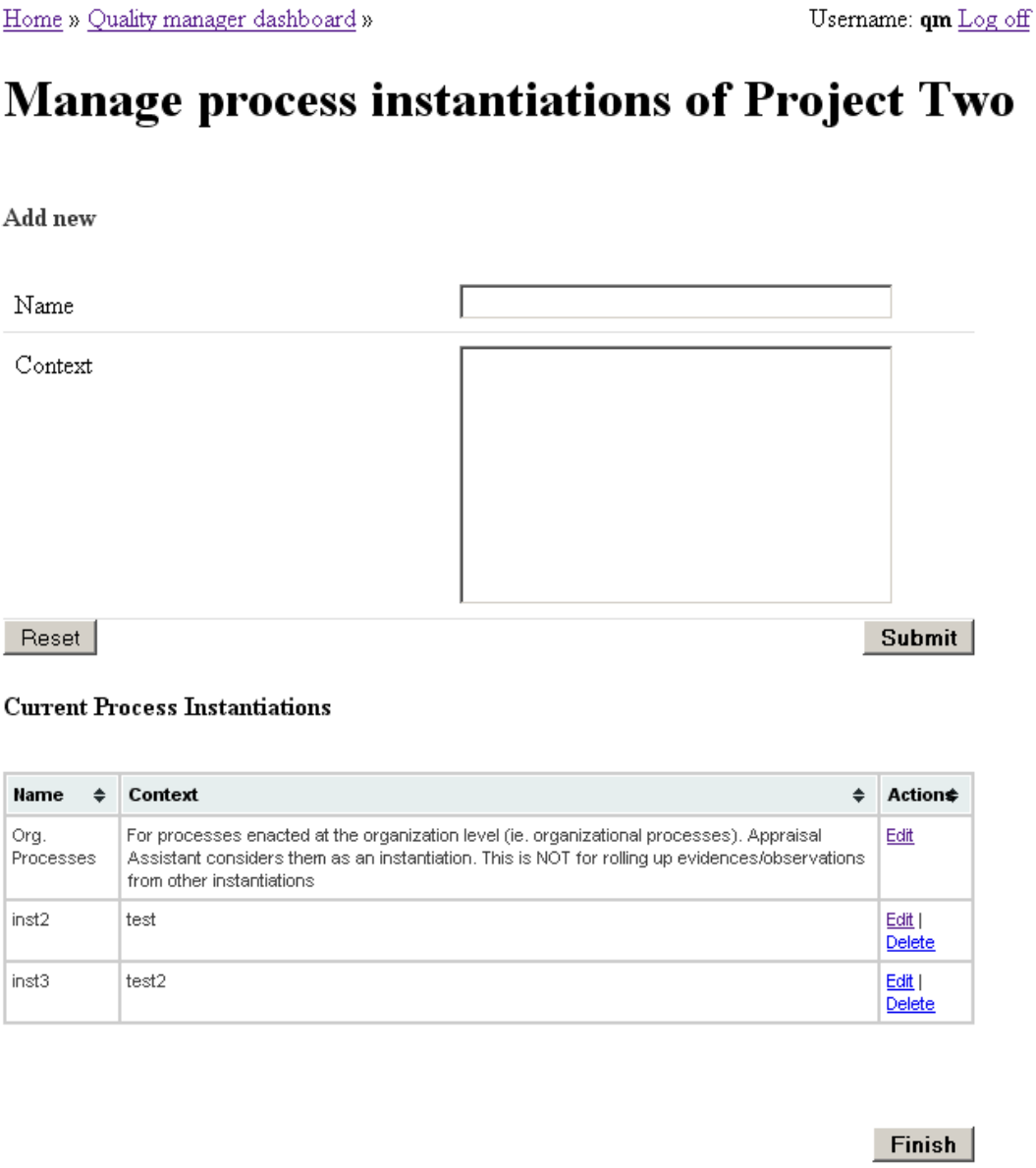
Příloha D

Obrazovky implementace (část I.)

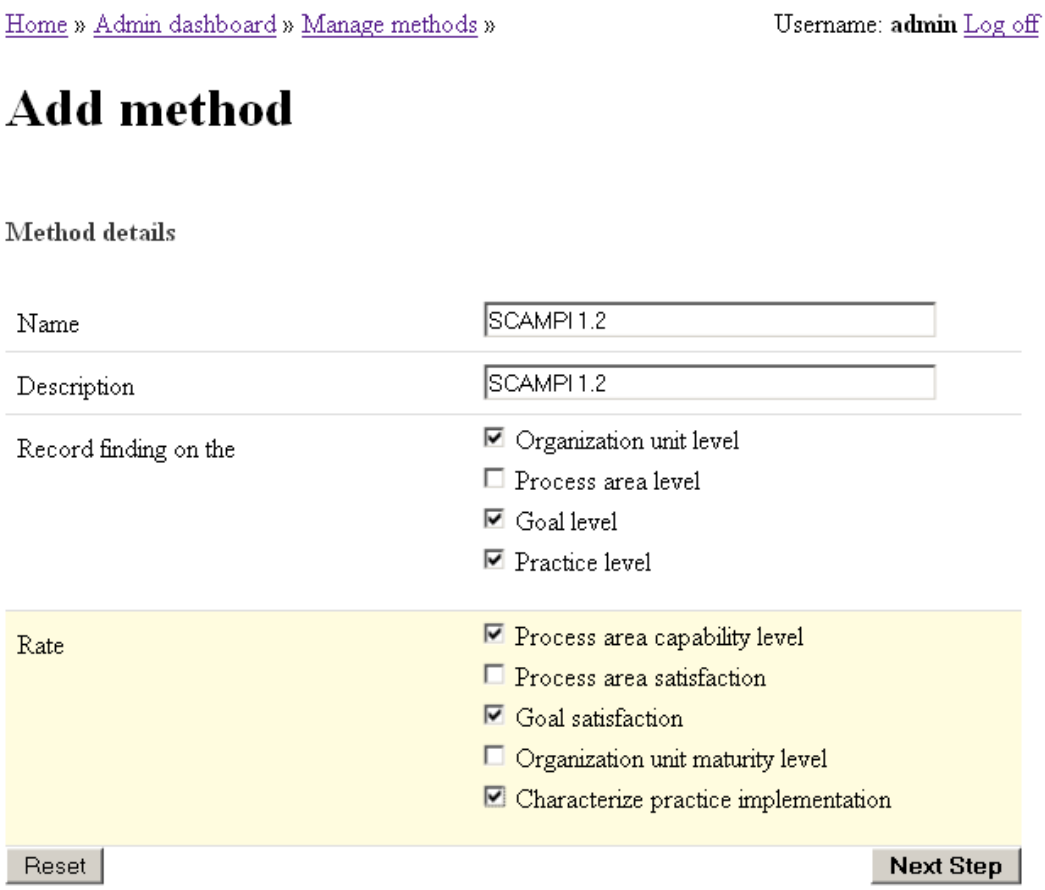
Tato příloha obsahuje vybrané obrazovky implementace.



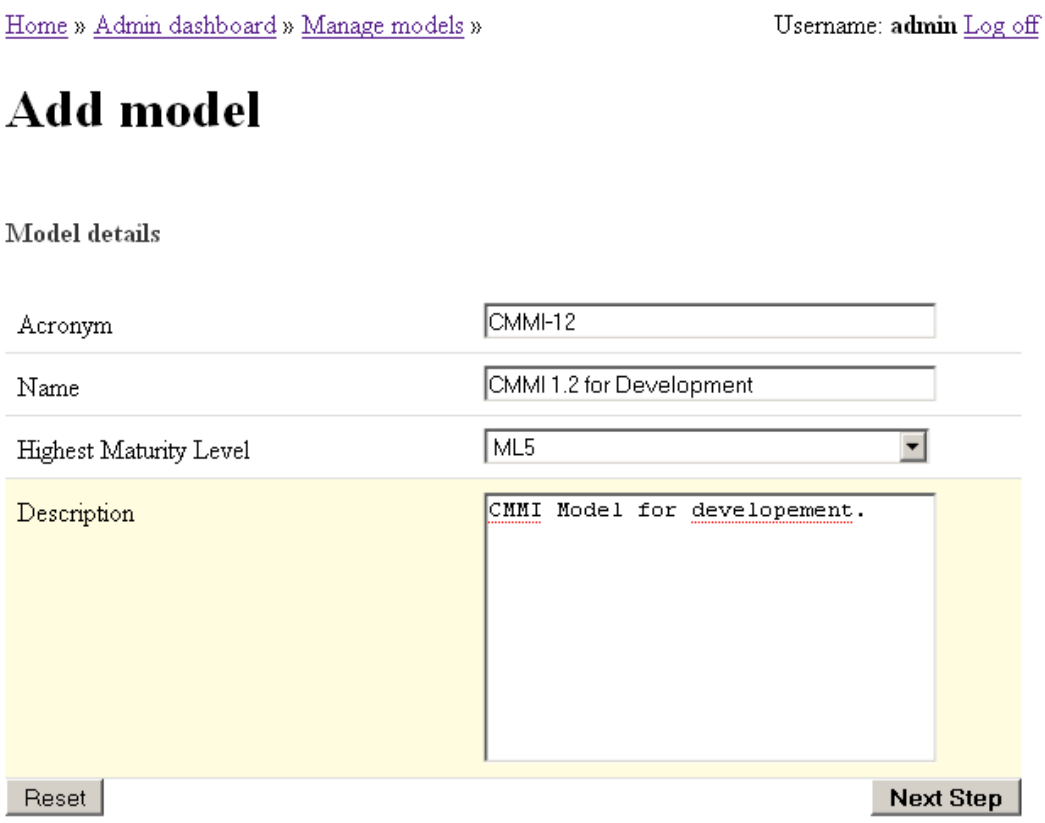
Obrázek D.1: Přidání nového projektu v organizaci 1, ukázka validace



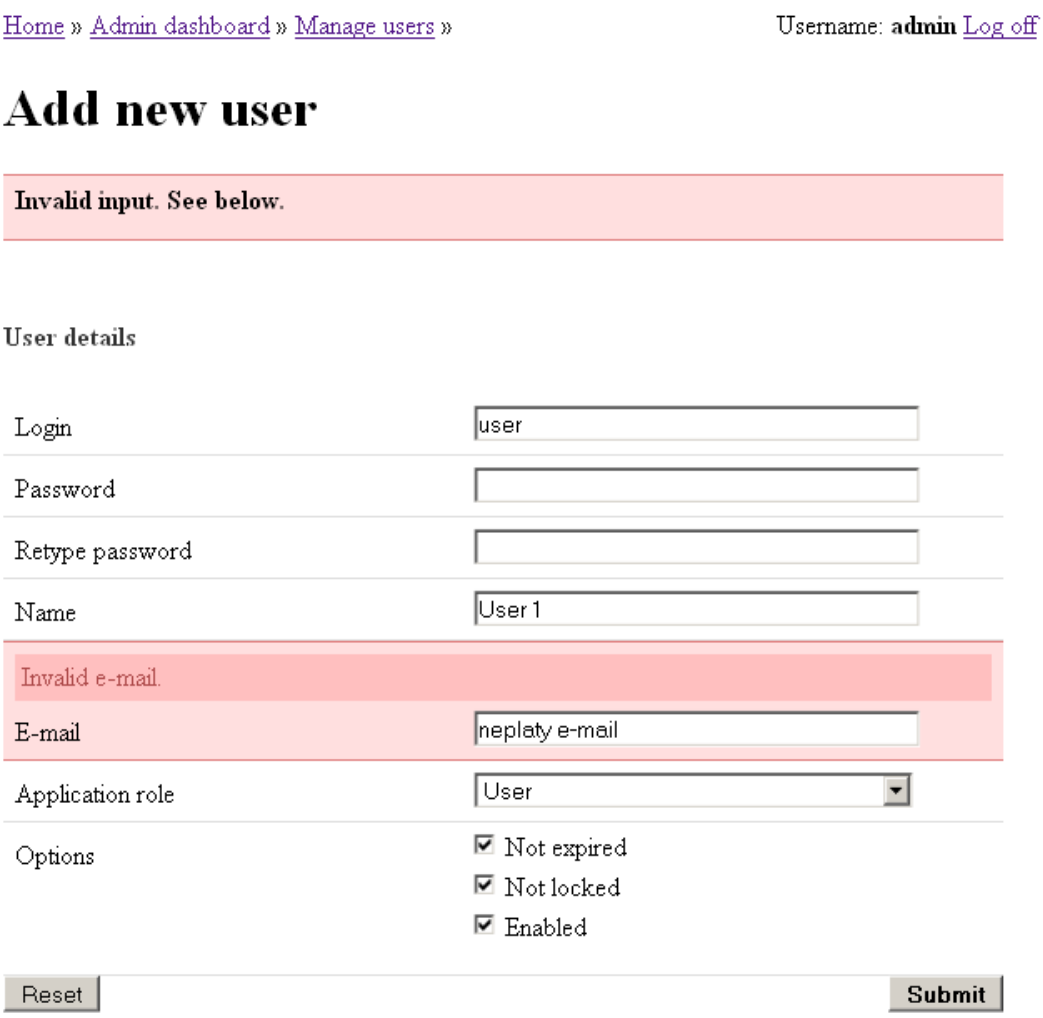
Obrázek D.2: Definice procesních instancí u projektu



Obrázek D.3: Přidání nové metody



Obrázek D.4: Přidání nového modelu



Obrázek D.5: Přidání nového uživatele (ukázka validace)

Home » Admin dashboard » Manage methods » Username: **admin** Log off

Define aggregation rules of method SCAMPI 1.2

Practice rule

Add rule

NC	FI	LI	NY	PI	NI	->	NC	FI	LI	NY	PI	NI
0	0	0	0	0	0	->	0	0	0	0	0	0

Submit

Existing rules

ID	NC	FI	LI	NY	PI	NI	->	NC	FI	LI	NY	PI	NI	Actions
1	0/1	1	0	0	0	0	->	0	1	0	1	0	0	Edit | Delete
2	0	1	1	0/1	0	0	->	0	1	1	1	0	1	Edit | Delete
3	0	0	0/1	0	0	0	->	0	0	0	0	0	0	Edit | Delete

Goal rule

Add rule

NC	FI	LI	NY	PI	NI	->	NS	S	NR	NA
0	0	0	0	0	0	->	0	0	0	0

Submit

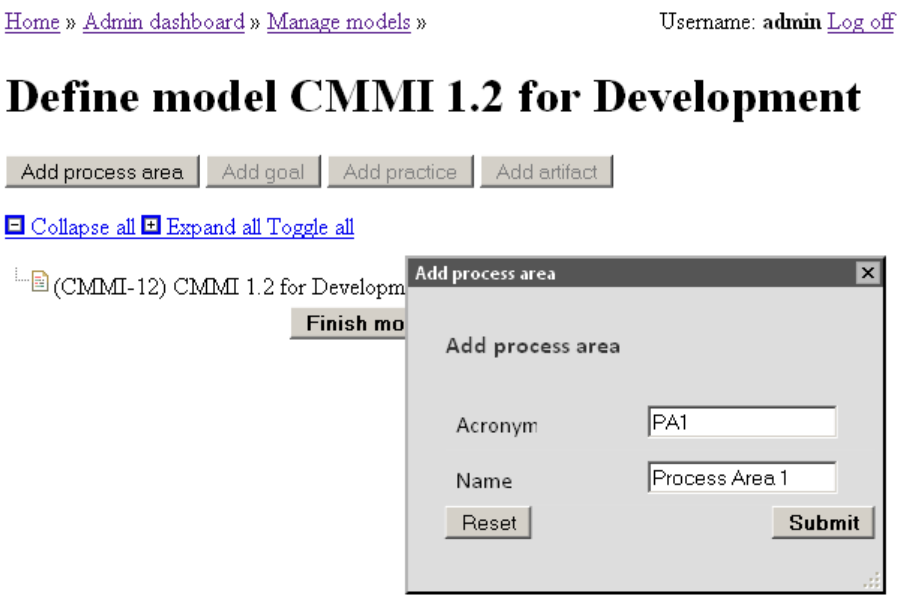
Existing rules

ID	NC	FI	LI	NY	PI	NI	->	NS	S	NR	NA	Actions
1	1	1	0	0	1	0	->	1	0	1	0	Edit | Delete
2	0	0	0	1	1	0/1	->	0	1	0	1	Edit | Delete

Finish method

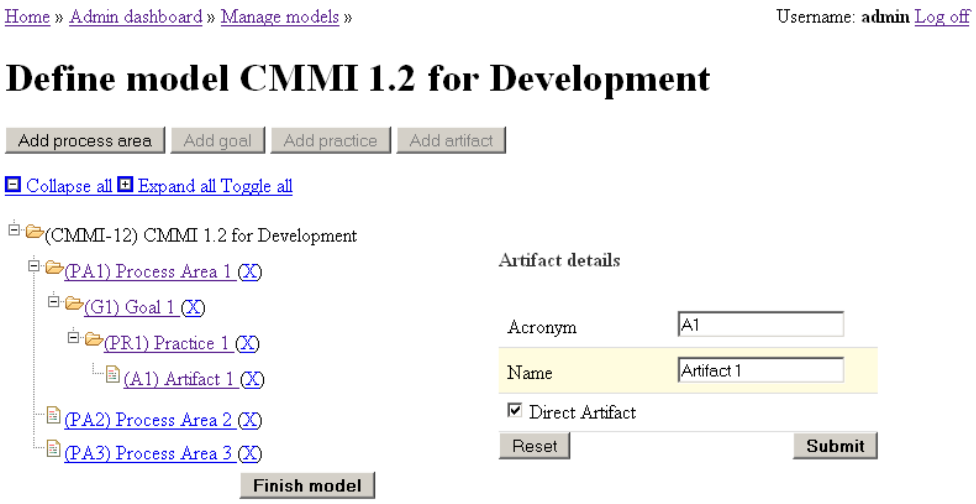
Finish method

Obrázek D.6: Definice agregačních pravidel metody

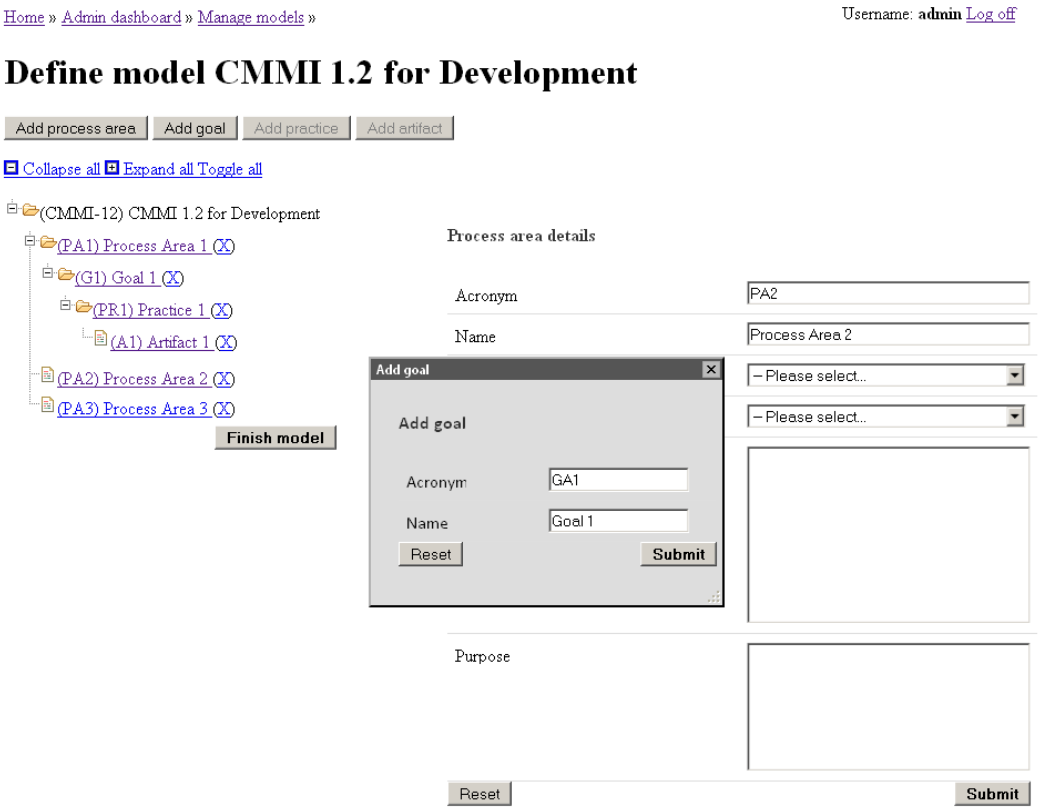


Obrázek D.7: Definice modelu - přidání procesní oblasti

Obrázek D.8: Definice modelu – Definice procesní oblasti

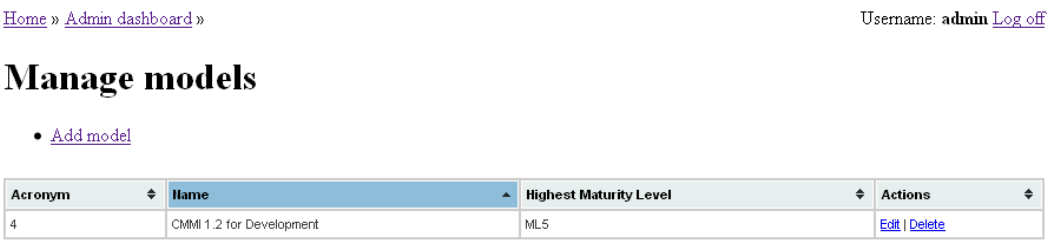


Obrázek D.9: Definice modelu - definice artefaktu



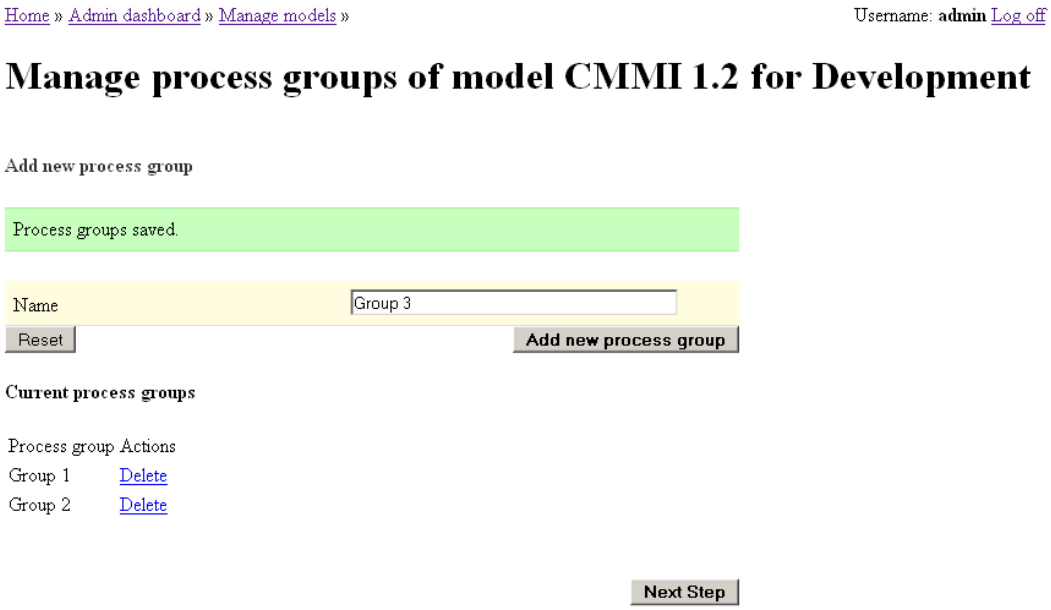
Obrázek D.10: Definice modelu - přidání cíle

Obrázek D.11: Definice metody - definice měřítek

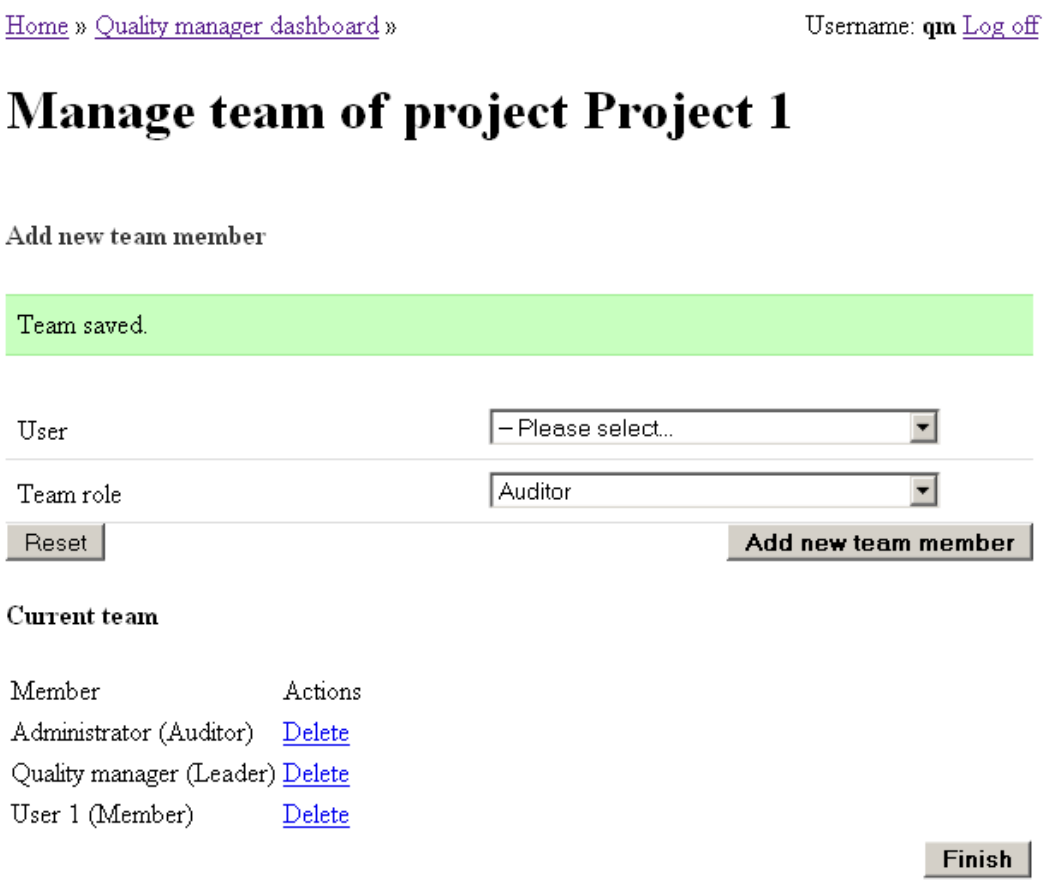


Obrázek D.12: Správa modelů

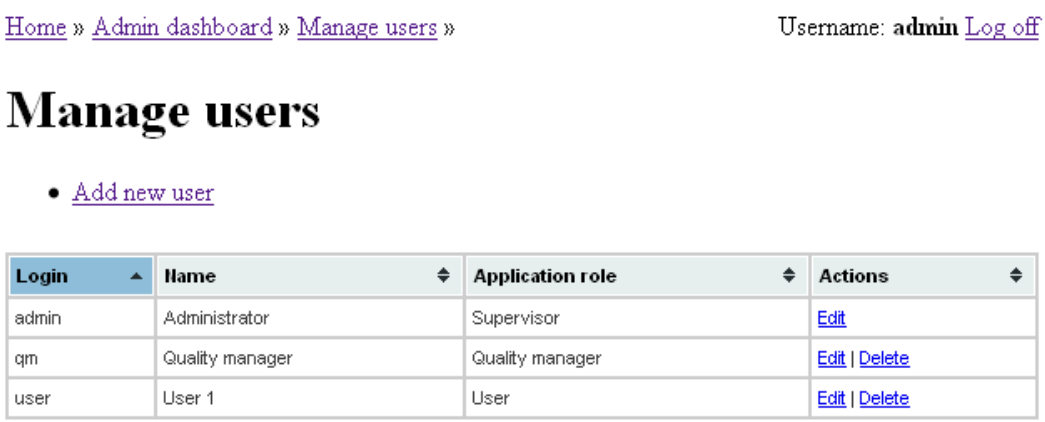
Obrázek D.13: Správa organizací



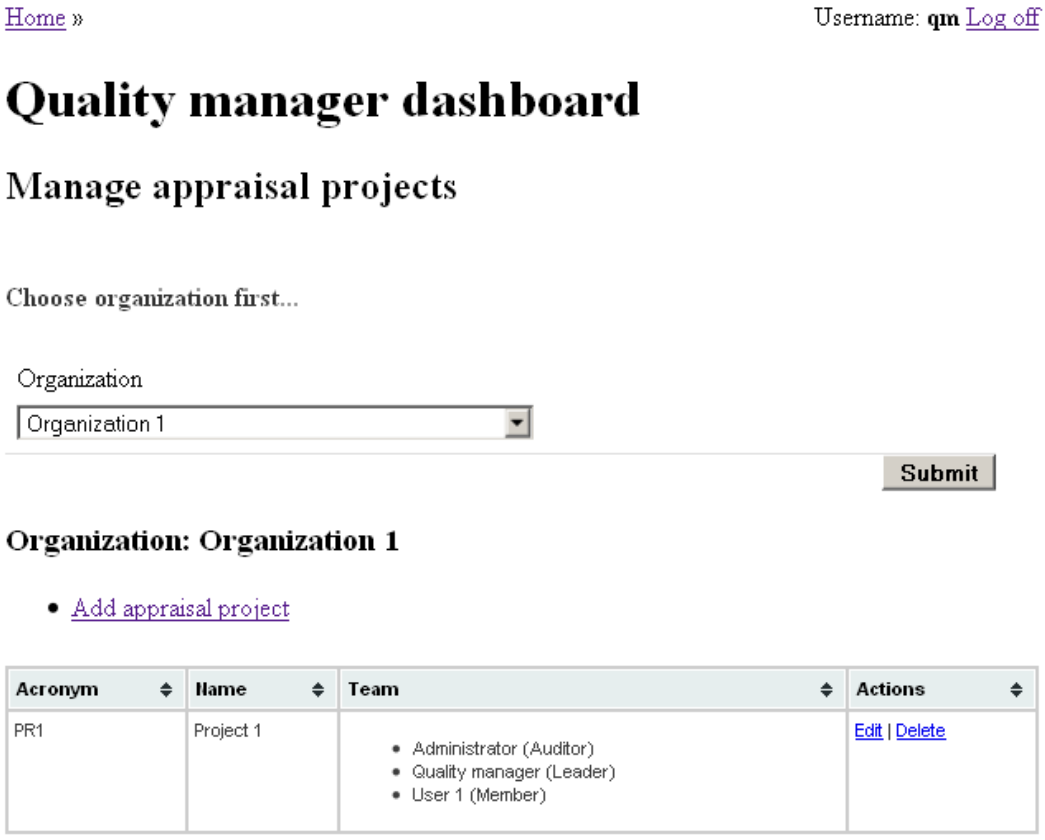
Obrázek D.14: Definice modelu - správa procesních oblastí



Obrázek D.15: Správa projektu - správa týmu



Obrázek D.16: Správa uživatelů

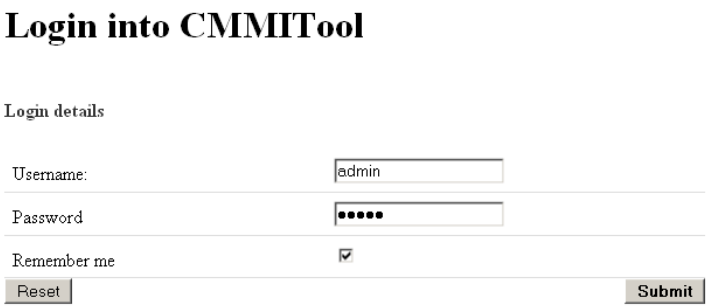


Obrázek D.17: Rozcestník manažera kvality - správa projektů

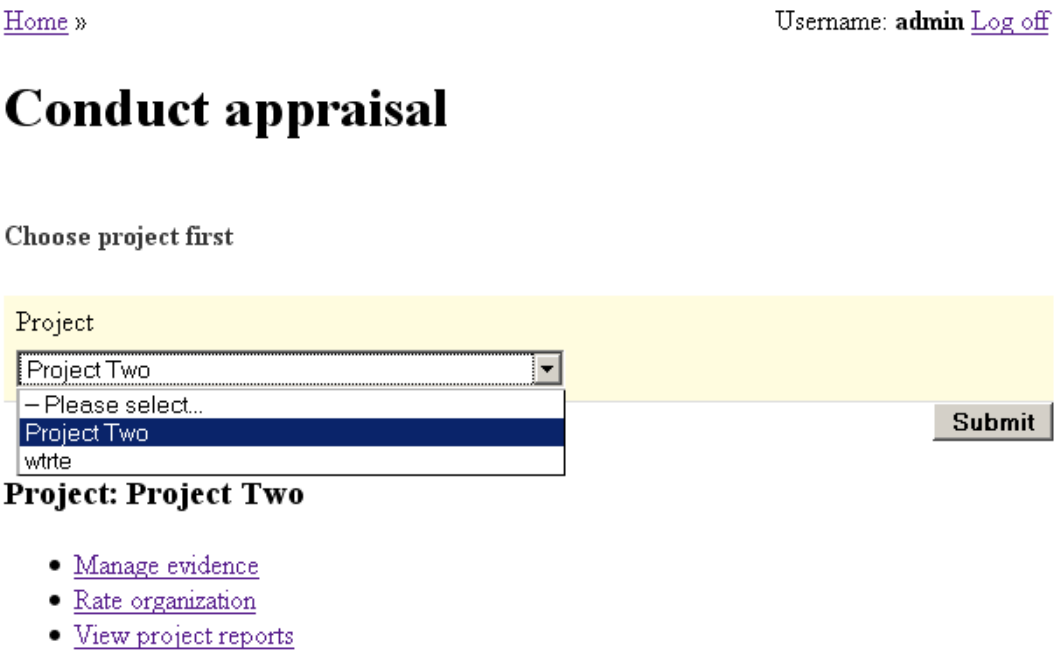
Příloha E

Obrazovky implementace (část II.)

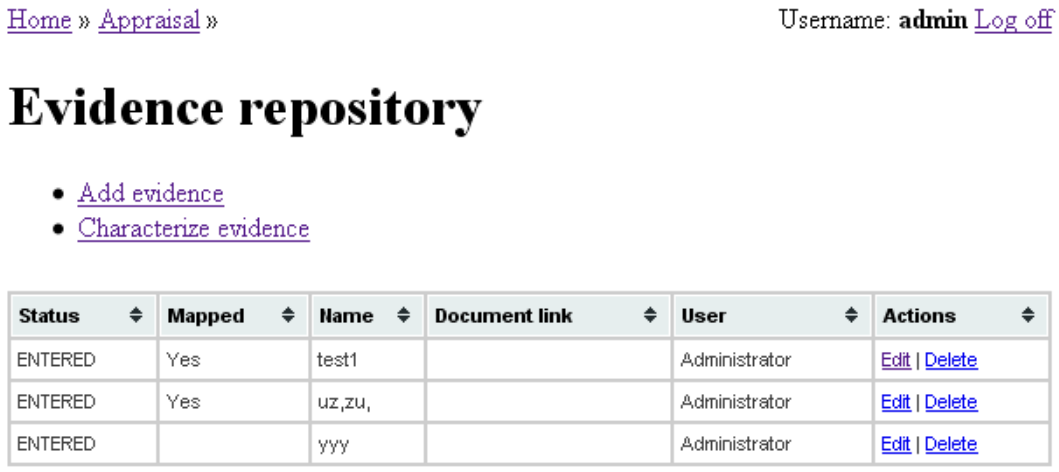
Tato příloha obsahuje vybrané obrazovky implementace.



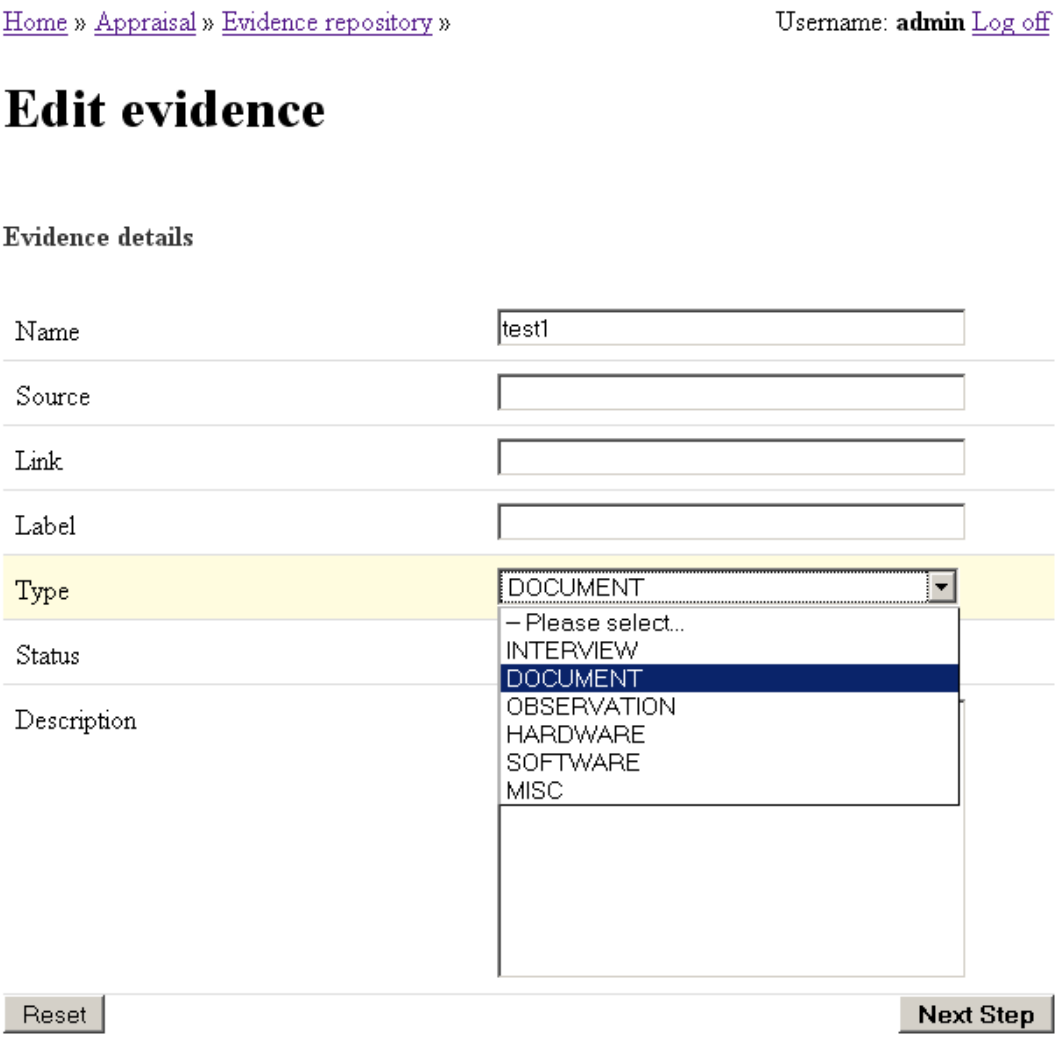
Obrázek E.1: Přihlašovací obrazovka



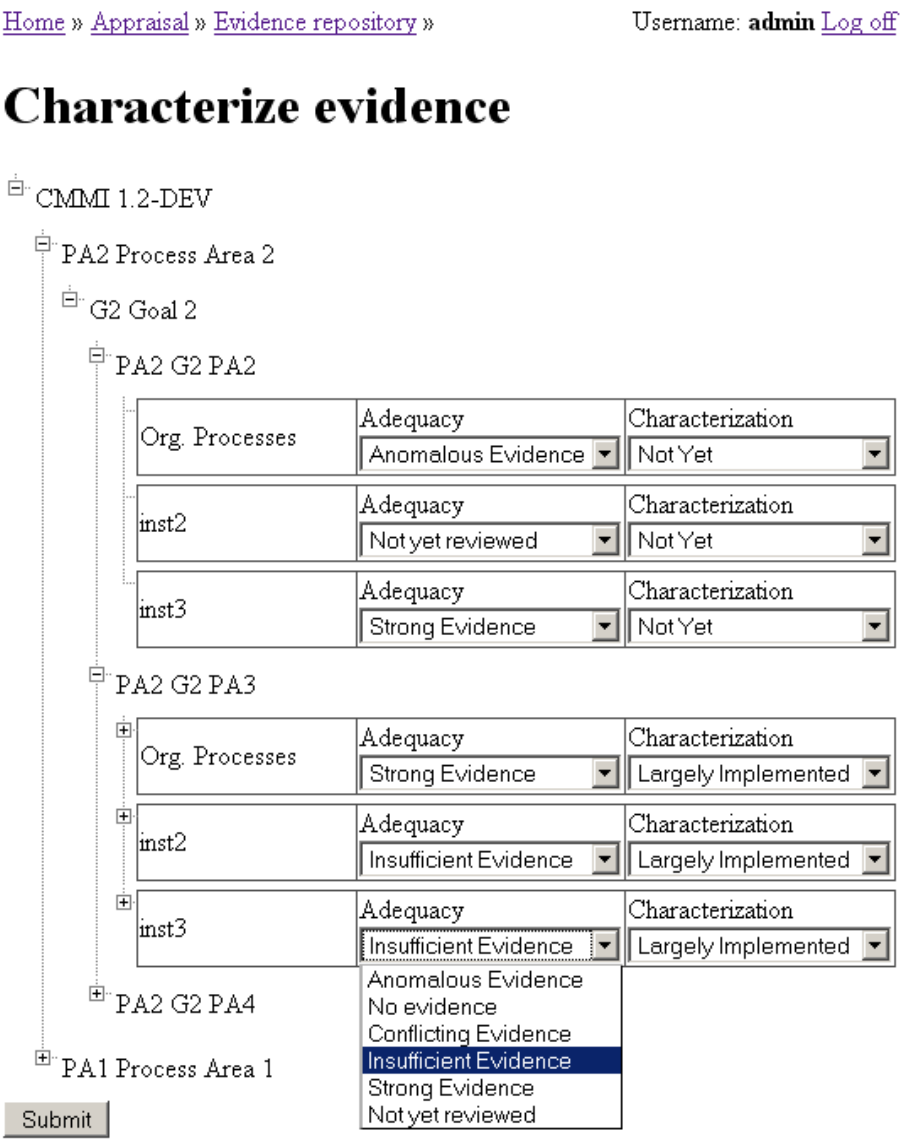
Obrázek E.2: Provedení hodnocení - volba projektu



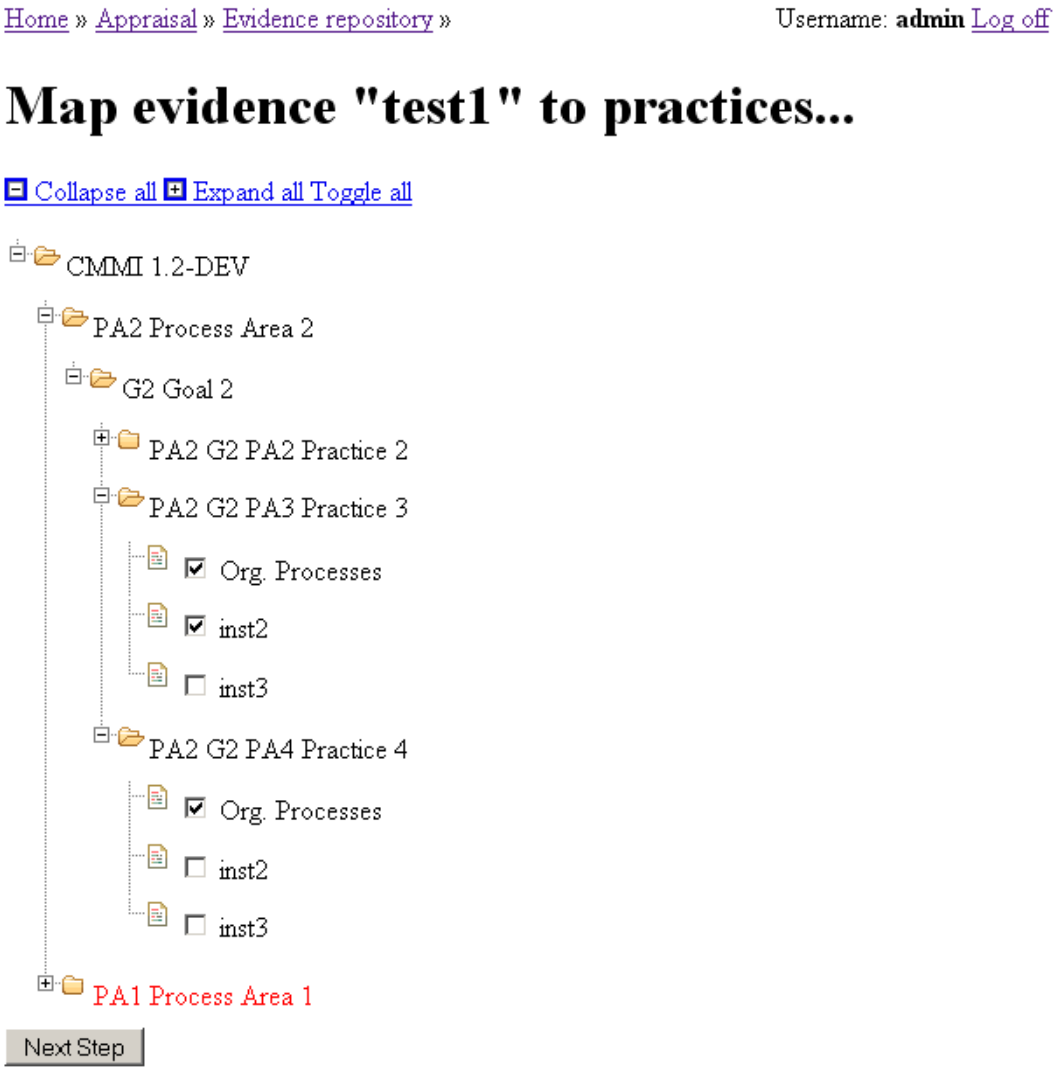
Obrázek E.3: Provedení hodnocení - seznam důkazů



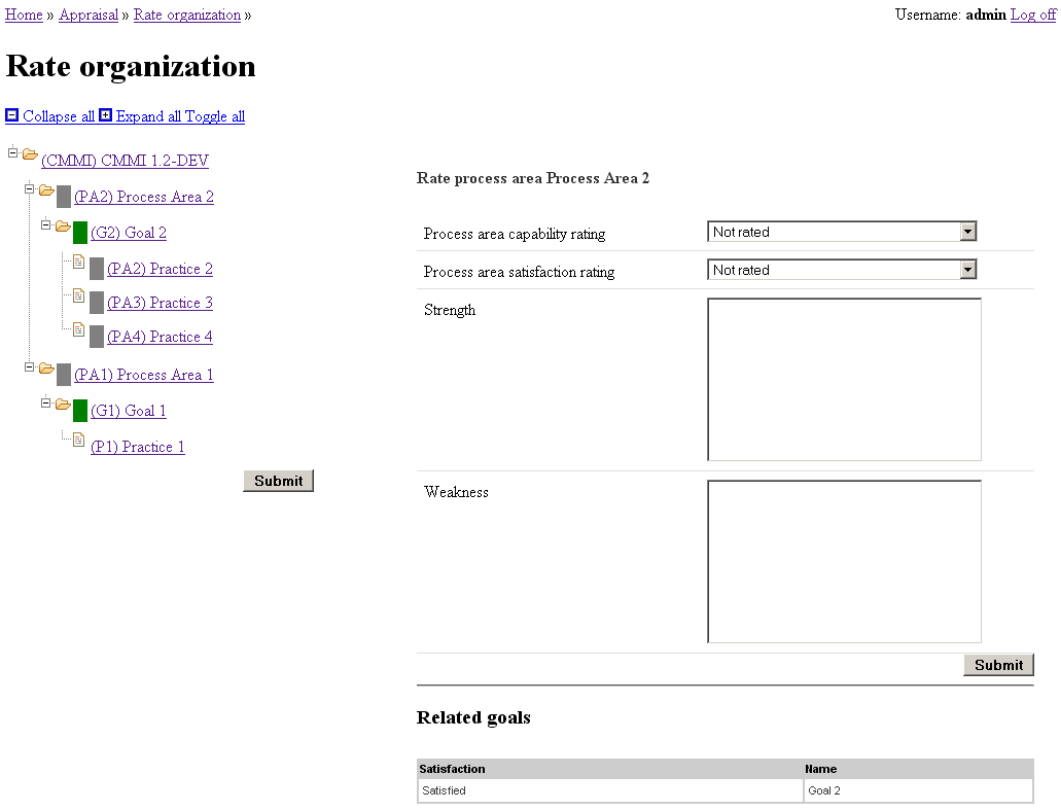
Obrázek E.4: Provedení hodnocení - úprava důkazu



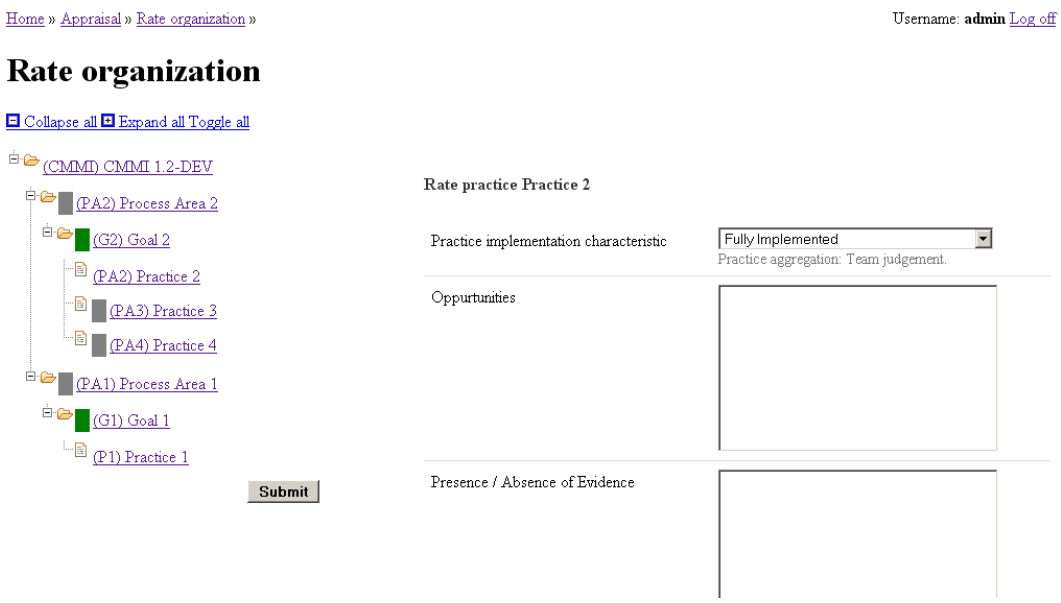
Obrázek E.5: Provedení hodnocení - charakterizace důkazu



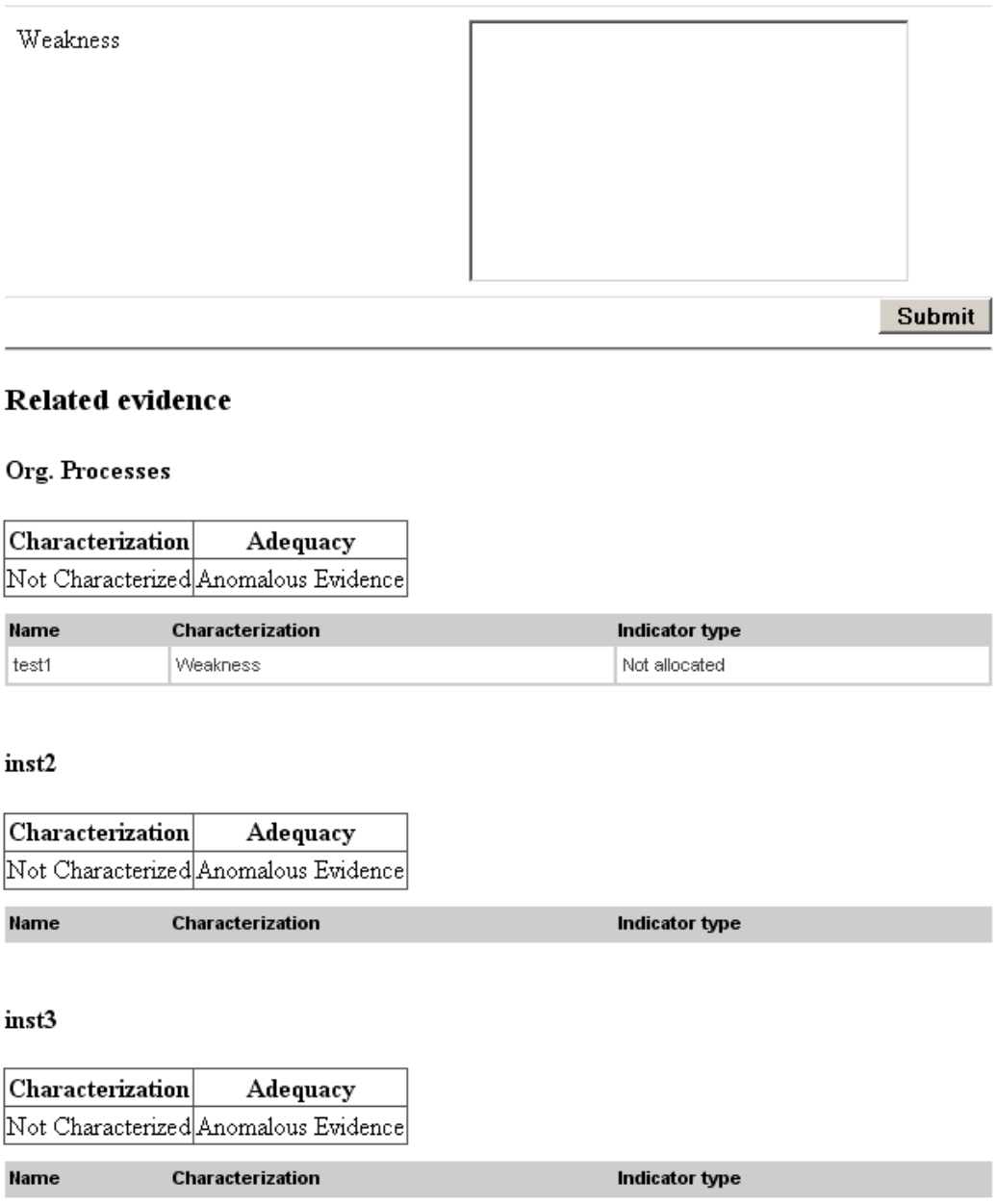
Obrázek E.6: Provedení hodnocení - mapování důkazu na praktiky



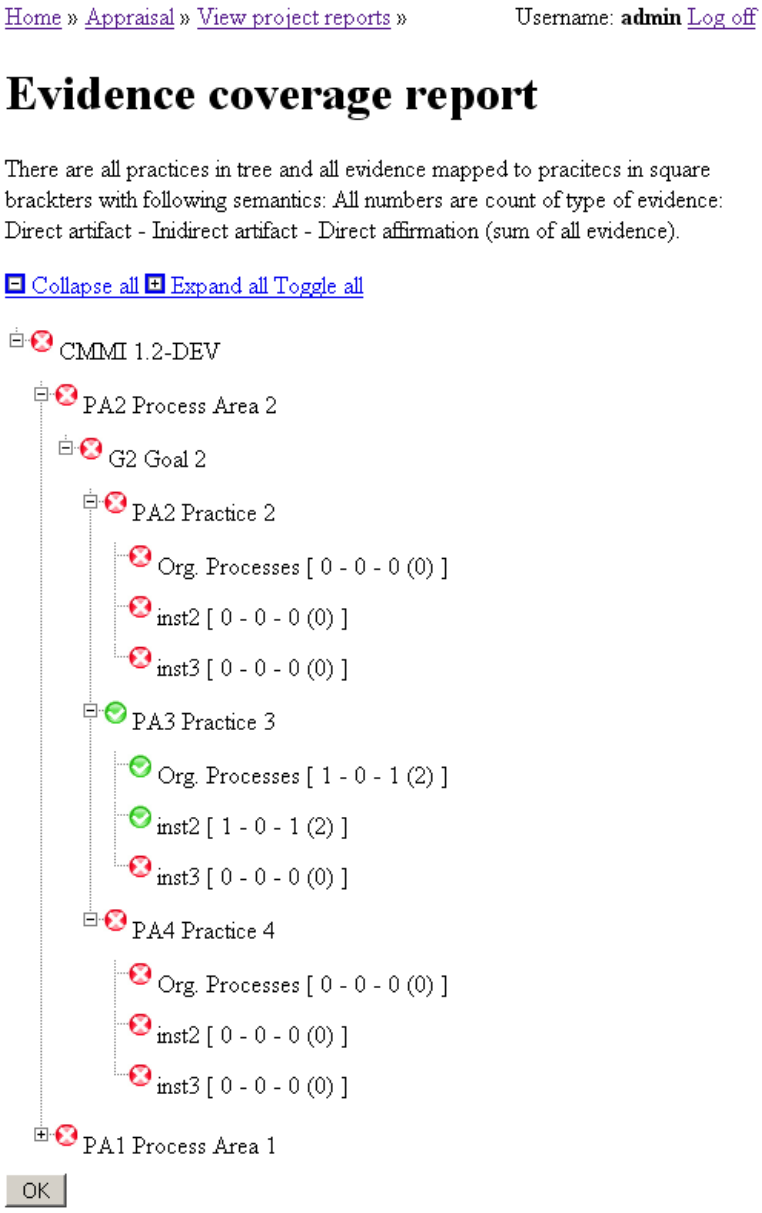
Obrázek E.7: Provedení hodnocení - hodnocení procesní oblasti organizace



Obrázek E.8: Provedení hodnocení - Hodnocení praktiky v cíli (1. část)



Obrázek E.9: Provedení hodnocení - Hodnocení praktiky v cíli (2. část), zobrazení mapovaných a charakterizovaných důkazů



Obrázek E.10: Provedení hodnocení - Zpráva o pokrytí důkazy

Literatura

[1] Ben Alex, Luke Taylor: *Appraisal Assistant Beta – User manual*, Griffith University, 2007, <http://www.sqi.gu.edu.au/AppraisalAssistant/about.html>. 3.1

[2] CMMI Product Team: *CMMI®for Development, Version 1.2*, Carnegie Mellon University., 2006. 2.1.1, 2.1, 2.1.3, 2.2, 2.1

[3] Mike Konrad, Sandy Shrum: *CMMI®: Guidelines for Process Integration and Product Improvement (2nd Edition)*, Addison-Wesley Professional, 2006, 978-0321154965.

[4] Richard Turner: *CMMI®Survival Guide: Just Enough Process Improvement*, Addison-Wesley Professional, 2006, 978-0321422774.

[5] Kent A. Johnson: *Interpreting the CMMI®: A Process Improvement Approach, Second Edition*, AUERBACH, 2008, 978-1420060522.

[6] Per Mellqvist: *Don't repeat the DAO!*, IBM, 2006, <http://www.ibm.com/developerworks/java/library/j-genericdao.html>. 4.4.1.1

[7] Red Hat Middleware: *Hibernate*, Red Hat, Inc., 2009, <https://www.hibernate.org/>. 4.4.2.2

[8] Sun Microsystems, Inc.: *Java Persistence API*, Sun Microsystems, Inc., 2007, <http://java.sun.com/javaee/technologies/persistence.jsp>. 4.4.2.1

[9] Sun Microsystems, Inc.: *JavaDoc 6*, Sun Microsystems, Inc., 2007, <http://java.sun.com/javase/6/docs/api/index.html>.

[10] Bloch, J.: *Java efektivně, 57 zásad softwarového experta*, Grada Publishing, 2002, 80-247-0416-1, <http://www.grada.cz/>. 4.3

[11] Dennis M. Ahern, Jim Armstrong, Aaron Clouse, Jack R. Ferguson, Will Hayes, Kenneth E. Nidiffer: *CMMI®SCAMPI Distilled Appraisals for Process Improvement*, Addison Wesley Professional, 2005, 0-32-122876-6. 2.2, 2.3

[12] Novotis: *Spice Vision - web*, Novotis, 2007, <http://www.novotis.com/spicevision/index.html>. 3.2

[13] Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Alef Arendsen, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abed Rabbo: *Spring 3 Reference Documentation*, Spring, 2009, <http://www.springsource.org/>. 4.4.1.1, 4.4.2.3, 4.4.2.4

[14] Ben Alex and Luke Taylor: *Spring Security: Reference Documentation*, Spring Source, 2009, <http://www.springsource.org/>. 4.4.1.1, 4.4.2.5

[15] Wikipedia: *Capability Maturity Model Integration (Wikipedia page)*, Wikipedia, <http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration>. 1