Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Bakalářská práce

Systém sběru dat pro efektivní regulaci štíhlé výroby

Jiří Havlík

školitel: Mgr. Miloš Prokýšek

České Budějovice 2011

Bibliografické údaje:

Havlík J., 2011: Systém sběru dat pro efektivní regulaci štíhlé výroby.

[Data acquiring system for efficient regulation of lean production. Bc. Thesis, in Czech]–49p. Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace v češtině

Cílem této bakalářské práce je analýza údajů z výrobního procesu. V praktické části jsou uvedeny postupy tvorby softwarové aplikace, umožňující tuto analýzu. Bakalářská práce se zaměřuje na specifika tzv. "štíhlé výroby" ve firmě Bosch, konkrétně ve výrobním závodě Robert Bosch, s.r.o. v Českých Budějovicích. V teoretické části je daná problematika teoreticky zpracována a jsou v ní formulovány předpoklady pro vytvoření softwarové aplikace pro evidenci a analýzu údajů poskytnutých výrobním procesem. V praktické části bakalářské práce jsou popsány postupy použité při tvorbě konkrétní softwarové aplikace.

Klíčová slova: Štíhlá výroba, Analýza, Databáze, Softwarová aplikace

Annotation in English

The main goal of my bachelor work is an analysis of a directly acquired data from a manufacturing process and its practical implementation as a software application. The bachelor work focuses on specifics of "lean manufacturing" in the company Robert Bosch, s.r.o. in Ceske Budejovice. The main goal of theoretical section is to elaborate mentioned subject-matter and to define requirements for creating of an application for recording and consequential analysis of the acquired data. In practical section of the work, there are described all the methods that are used during development of the application.

Keywords: Lean production, Analysis, Database, Software application

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Poděkování

Rád bych na tomto místě poděkoval Mgr. Miloši Prokýškovi za vedení mé práce.

Dále děkuji firmě Robert Bosch, s.r.o. za poskytnutí zajímavého tématu a za svolení k jeho zpracování ve formě této bakalářské práce.

Obsah

1.Úvod

Strategické řízení výrobního podniku je zodpovědná činnost, která se zvláště v dnešní době neobejde bez podrobné analýzy podnikových procesů. Jednou z důležitých disciplin výrobní filozofie nazývané š*tíhlá výroba* je i důsledné sledování a analýza výrobních procesů. Tato bakalářská práce se zaměřuje právě na zmíněnou disciplinu a předkládá návrh na řešení požadavku na evidenci a analýzu získaných údajů. Předložené řešení má podobu softwarové aplikace, která by měla zjednodušit cestu informací od výrobních linek k výrobnímu managementu a zefektivnit tak řízení takto organizované výroby.

1.1. Zaměření

Popis problematiky štíhlé výroby a zdůvodnění potřeby účelové analýzy výrobních dat. Realizace evidenčního a analytického softwarového nástroje pro potřeby štíhlé výroby.

1.2. Motivace

Toto téma jsem si vybral proto, že jsem byl v minulosti s problematikou analýzy výrobních dat úzce konfrontován. Byl jsem požádán o vytvoření jednoduše modifikovatelného evidenčního a analytického softwarového nástroje pro potřeby montážního oddělení firmy Robert Bosch, s.r.o. Nástroj měl být z důvodu požadavků na jednoduchost a kompatibilitu s formátem dalších používaných souvisejících údajů postaven na platformě Microsoft Excel VBA a měl uživateli umožnit snadný zápis a následné vyhodnocování údajů, relevantních pro analýzu efektivity a produktivity montážního procesu. Tato aplikace byla po svém vytvoření následně využita pro další zkoumání nároků na údaje potřebné pro tyto analýzy a v průběhu tohoto zkoumání byla mnohokrát modifikována.

Výsledky empirického zkoumání výstupů testovací aplikace umožnily jednoznačně určit požadavky na vstupní i výstupní údaje. Získané informace poté byly využity ke specifikaci zadání pro vývoj komplexního softwarového řešení. Tato plánovaná aplikace, za předpokladu ověřené dostatečné využitelnosti pro všechna výrobní oddělení, měla nahradit mé stávající testovací VBA řešení a stát se hlavním používaným evidenčním a analytickým nástrojem pro potřeby štíhlé výroby ve firmě Robert Bosch, s.r.o..

Má nabídka na realizaci takové aplikace v rámci mé bakalářské práce tedy není pouze logickým vyústěním událostí, je rovněž jakýmsi završením mých aktivit v této problémové oblasti ve firmě Robert Bosch, s.r.o.

1.3. Představení společnosti Robert Bosch, s.r.o.

V roce 1886 založil německý podnikatel Robert Bosch *Dílnu pro jemnou mechaniku a elektrotechniku*. Na počátku 20. století firma investovala finanční prostředky do výstavby nové továrny ve Stuttgartu, kde zaměstnala několik desítek pracovníků. Tímto krokem přešla od řemeslného výrobního přístupu k průmyslovému. Od té doby firma založila mnoho dalších továren a dceřiných společností po celém světě. K dnešnímu dni koncern Robert Bosch GmbH zaměstnává více než čtvrt milionu pracovníků v 50 zemích. Přibližně dvě pětiny celkového počtu pracovníků je zaměstnáno v německých závodech. Společnost je v současnosti vedoucím mezinárodním dodavatelem technologií a služeb. Velmi významnou oblastí aktivit firmy je vlastní výzkum a vývoj nových výrobků a technologií.

Společnost Robert Bosch GmbH v roce 1988 spolu s dalšími třinácti tehdy významnými firmami založila Evropskou Nadaci pro Management Kvality, EFQM, která se snaží působit na zlepšování konkurenceschopnosti evropských organizací a podporovat trvale „udržitelnou excelenci“.

„V roce 1992 byla založena společnost Robert Bosch s.r.o. v Českých Budějovicích jako společný podnik stuttgartského koncernu Bosch GmbH Stuttgart a Motoru Jikov a.s. Od roku 1995 je jediným vlastníkem společnosti koncern Bosch“ (citováno z příručky *Výrobní systém BOSCH, Metodická příručka pro spolupracovníky*. České Budějovice: Tiskárna Jaroslav Karmášek, 2007. 59 s [8]).

Výrobní program zahrnuje komponenty automobilové techniky. Společnost, která v současné době zaměstnává přibližně 2 tisíce pracovníků, dodává své výrobky všem významným evropským automobilkám. Stejně jako koncern Bosch, firma Robert Bosch s.r.o. věnuje mnoho prostředků do vývoje nových komponent.

Od roku 2000 koncern Bosch aplikuje do svých procesů principy štíhlé výroby. Souhrn všech těchto principů dostal název Bosch Production System, BPS a od roku 2002 je zaváděn rovněž ve výrobním závodu v Českých Budějovicích. V tomto roce vznikl BPS tým, který začal zavádět BPS do výrobních a logistických procesů společnosti.

2.Teoretická část

Primárním cílem a v praxi použitelným výsledkem této bakalářské práce je funkční softwarová aplikace. Úvodem je potřeba definovat základní pojmy a provést podrobnou účelovou deskripci problému. Z toho důvodu má tato teoretická část práce převážně kompilační charakter a klade si za cíl především zmapovat rysy štíhlé výroby a to hlavně v porovnání s odlišnými přístupy k organizaci výroby. Významná část je věnována historii a rozdílům mezi štíhlým a konvenčími přístupy k organizaci výroby. Na konci teoretické části jsou formulovány požadavky na údaje, jejichž pravidelné sledování má význam pro účinné řízení výrobního procesu

2.1. Výrobní proces

Primárním cílem každého podniku je dosahování zisku. V případě výrobního podniku je zisku dosahováno transformací zdrojů (často označovanými manažerskou zkratkou 5M z angl. men, machines, methods, materials a money) na výstupy (výrobky, služby) mechanismem, který nazýváme výrobní proces (obr. 2.1) [14].

Obrázek 2.1: Výrobní proces. Zdroj: ZČU

Výstupy jsou poté nabízeny zákazníkům za dohodnutou prodejní cenu. Ziskem firmy jsou výnosy z prodeje výstupů snížené o náklady vložené do vstupů. Je zřejmé, že jedním ze sekundárních cílů výrobního manažera je minimalizace těchto nákladů při zachování požadovaných rysů procesu a výstupního výrobku na přijatelné úrovni. Štíhlý přístup k tomuto problému je v mnoha směrech progresivní, neboť povyšuje požadavek na zachování stávajícího stavu, na princip neustálého zlepšování.

2.2. Výrobní proces jako systém

Podíváme-li se na výrobní proces z hlediska teorie řízení, můžeme vidět otevřený systém, charakterizovaný fyzickými (materiál, výrobky) a informačními toky [14]. Fyzické toky jsou limitovány vnitřní charakteristikou systému, kde nejdůležitější roli hraje kapacita systému a schopnost vyhovět požadavkům na výstupy, především požadavkům kvalitativním.

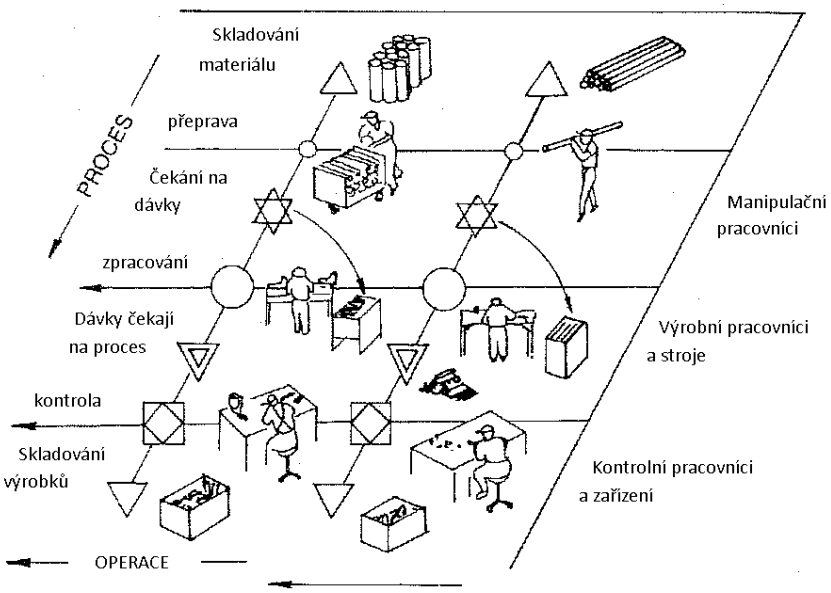
Míra regulovatelnosti fyzických toků do značné míry závisí na vlivech vně systému a měřitelná odezva výrobního systému na regulaci zpravidla zdaleka nebývá okamžitá. Na obecný výrobní proces tedy nelze pohlížet jako na běžný regulovatelný systém. Po provedení dekompozice obecného výrobního procesu navíc zjistíme, že také jednotlivé prvky (lidé, stroje, logistické toky materiálu a výrobků) jsou významně ovlivňovány děním vně systému, takže realizace i samotná definice nástrojů pro jeho regulaci je velmi problematická.

Z pohledu teorie řízení tento systém tedy nevykazuje vysokou míru robustnosti ani flexibility a proto je kvůli zefektivnění jeho regulace žádoucí vyvíjet trvalou snahu o zkrácení reakční doby na minimum. Splnění tohoto předpokladu je podmíněno důsledným a pravidelným sledováním výrobního procesu a správnou interpretací získávaných údajů. Pro usnadnění těchto úkolů jsou v dnešních výrobních podnicích často používány různé evidenční a analytické nástroje. Významnou roli zde zastupuje výpočetní technika se specializovanými informačními, popř. tzv. ERP, softwarovými systémy. Ty mají důležité místo mezi instrumenty pro efektivní řízení informačních toků, které v sobě zahrnují veškeré dokumenty a informace doprovázející toky fyzické. Řízení informačních toků je důležitým úkolem dnešního výrobního manažera.

I přes uvedené limity lze při regulaci výrobního procesu uplatnit některé automatizační principy. Je například možné na tento specifický druh systému pohlížet v souladu s filozofií neustálého zlepšování (v japonštině „Kaizen“) jako na opakující se cyklus činností. Souhrn těchto činností je jádrem konceptu PDCA (z angličtiny, plan-do-check-act, plánuj – proveď – zkontroluj – jednej). Tento koncept, známý také jako demingův nebo shewartův cyklus, vyjadřuje nutnost cyklického sledování chování systému s důrazem na vyhodnocování účinnosti provedených regulujících opatření.

2.3. Výrobní proces jako síť procesů a operací

Představa výrobního systému jako funkční sítě procesů a operací je dalším, zcela odlišným pohledem na diskutovaný problém. Procesy transformují materiál do podoby výsledného výrobku a operace jsou akce, které uskutečňují tyto procesy. Podle Shigea Shinga je pochopení této představy klíčem k efektivnímu zlepšování [1].



Obrázek 2.2: Síť procesů a operací, převzato z [1] a upraveno

Na obrázku (viz Obr. 2.2) lze vidět tok materiálu v čase i prostoru – jeho transformaci ze surového materiálu, přes neopracovaný výrobek až po výsledný produkt. Operace symbolizují práci prováděnou za účelem uskutečnění této transformace – pohyb pracovníků a zařízení v čase i prostoru. Analyzujeme-li proces, zkoumáme tok materiálu nebo výrobků. Naproti tomu při analýze operací se zaměřujeme na práci prováděnou na výrobku. Abychom umožnili fundamentální zlepšování procesu, musíme rozlišit tok práce od toku výrobků a zkoumat je odděleně. Přestože je možné na proces pohlížet jako na sérii operací, bylo by chybou vizualizovat jej pouze v jednou rozměru, tedy jako jednosměrnou linii. Takové zobrazení procesu by mohlo evokovat mylnou domněnku, že zlepšováním dílčích operací lze zefektivnit celkový proces. Naopak, mnohé pokusy o zlepšování samostatných operací prokázaly, že podobná snaha může negativně ovlivnit výkonnost celkového procesu. Shigeo Shingo toto tvrzení ve své knize [1] demonstruje na mnoha příkladech a tato bakalářská práce se v souladu s uvedenou myšlenkou také zabývá zlepšováním výrobního procesu jako celku.

2.4. Historie průmyslové výroby

V této kapitole je při popisu historického vývoje štíhlé výroby čerpáno z knihy: WOMACK, James P. *The Machine That Changed the World : The Story of Lean Production*. [USA] : Harper Perennial, 1991. 323 s. ISBN 978-0060974176 [2].

2.4.1. Řemeslná výroba – Craft production

Ještě na konci 19. století nebyla průmyslová výroba z dnešního pohledu prakticky nijak jednotně organizována a neexistovaly normy unifikující design a funkci výrobků ani výrobní postupy. Tento způsob organizace výroby, který nazýváme řemeslná výroba, měl řadu nevýhod. Řemeslná výroba byla mimo jiné charakterizována vysoce kvalifikovanou pracovní silou. Výrobními závody byly větší dílny zaměstnávající kvalifikované pracovníky, kteří byli často pouze najímáni na provedení dílčích úkonů. Vysoká kvalifikace pracovníků byla dána jejich jistým ztotožněním se s řemeslem. Mnozí z nich trávili u řemesla mnoho let po vyučení a naučili se vnímat ekonomický význam své role ve výrobním procesu. Vyráběli si vlastní stroje a nástroje a měli také úzký kontakt se zákazníkem. Někteří tito nájemní pracovníci byli navíc majiteli menších dílen a vyráběli i díly pro svého najímatele. Tento vztah lze považovat za předstupeň vztahu dodavatel – zákazník, jak jej vnímáme dnes.

Důsledkem tohoto uspořádání byla mimo jiné zmíněná absence standardů, což mělo velmi negativní dopad na kvalitu výrobků. Pokud dílna obdržela zakázku na výrobu většího množství stejných výrobků, nebyla schopna udržet rozměry ve stanovených tolerancích, takže ostatní části celku musely být přizpůsobovány, aby je bylo možno smontovat do finální sestavy. Každý výsledný produkt byl tedy de-facto prototypem. Z ekonomického hlediska ještě významnějším faktem bylo, že náklady na výrobu jednoho kusu neklesaly s objemem vyrobených kusů. Kvůli absenci rozměrových standardů tomu bylo často dokonce naopak, protože eventuální požadavek na dokonale stejný design či funkci více kusů jednoho typu výrobku komplikoval výrobu a stupňoval náklady. Pro řemeslnou výrobu je rovněž typické využití univerzálních strojů a nástrojů pro zpracování materiálu či polotovaru i pro následnou montáž dílců do komplexnějších celků. Jen malé množství používaných nástrojů či zařízení bylo koncipováno specielně pro konkrétní typ výrobku. Další nevýhodu lze vidět i v neschopnosti dílen zabývat se vývojem nových technologií, protože veškerou svou energii a prostředky tyto dílny věnovaly včasnému zpracování svých nízkoobjemových zakázek.

I přesto, že tento systém umožňuje relativně pružně uspokojovat individuální požadavky zákazníků, což lze považovat za jednu z mála jeho výhod, jeho fatální slabinou je právě neschopnost zaručit a udržet stabilní kvalitu výrobků.

2.4.2. Hromadná výroba – Mass production

Trh práce na začátku 20. století negativně vnímal zmíněné nedostatky řemeslné výroby a reagoval shromažďováním individuálně vlastněného kapitálu v tzv. manufakturách. V těchto továrnách zaměřených převážně na rukodělnou výrobu se začalo prosazovat hierarchické uspořádání zaměstnanecké struktury podle pravomocí a zodpovědností zaměstnanců. Manažeři zodpovědní za prosperitu závodu hledali způsoby, jak zefektivnit výrobu, zvýšit produktivitu a získat tak konkurenční výhodu. Ve výrobě se proto hromadně zaváděly stroje, často jednoúčelové, navržené specielně pro konkrétní výrobu.

V roce 1913 se americký průmyslník Henry Ford pokusil zvýšit efektivitu výrobního procesu tím, že zavedl systém, který umožnil montážním pracovníkům zůstat celou směnu na jednom místě. Klíčovým prvkem tohoto systému byl tzv. běžící pás, doručující zpracovávané díly od jednoho stanoviště k druhému ve směru logiky montážního postupu. Tento prvek je dnes laickou veřejností označován za nejvýznamnější přínos hromadné výroby, ve skutečnosti jím ale byl vznik standardů pro volnou a snadnou zaměnitelnost dílů. Důsledky těchto změn byly dalekosáhlé. dodavatelé byly nucení k respektování rozměrových i jiných standardů, takže zákazník-zpracovatel nemusel vynakládat velké množství svých prostředků na zaručení smontovatelnosti s ostatními díly. Ford navíc používal postupy, např. předtvrzené kovové díly či poloautomatické montážní stroje, které umožnily snadněji dodržet specifikace výrobku. Díky tomu se také snížily požadavky na kvalifikaci výrobních pracovníků a firmy začaly ve velkém využívat levnou pracovní sílu přistěhovalců, u kterých nebyly kladeny vysoké nároky na kvalifikaci. Henry Ford totiž uplatňoval princip bezproblémové zaměnitelnosti nejenom pro vyráběné díly, ale i pro zaměstnance. Obsluha strojů na jeho výrobní lince byla velmi snadná, takže zapracování nového dělníka či výměna stávajícího nevyžadovala prakticky žádné dodatečné náklady.Vznikaly také odborné pozice, na kterých byly nároky na kvalifikaci vyšší. Kvalita výroby byla sledována kontrolorem specialistou umístěným na konci linky. Hlubší znalosti, týkající se konkrétní linky byly požadovány také po seřizovačích, kteří zajišťovali přenastavení strojů a nebo nápravu v případě výskytu neplánované události na lince.

Hromadná výroba principielně překračuje mnoho limitů řemeslné výroby, ale ze své podstaty s sebou přináší některé významné nedostatky. Za kvalitu zodpovídal specialista umístěný na konci linky a protože ne všechny kontrolovatelné znaky dílce mohly být sledovány u každého kusu, byly výrobky kontrolovány namátkově. Ve chvíli, kdy byla odhalena neshoda, prošlo linkou často mnoho kusů, které, v lepším případě, musely být opraveny, v tom horším byl ztracen materiál pro jejich výrobu i pracovní doba výrobou strávená. Pokud došlo k závadě stroje, který byl součástí linky, byli ostatní „nevinné“ stroje vyřazeny z provozu také. Pracovní síla mohla být přesunuta na jinou linku, pokud to bylo možné, nebo čekala, až skupina specialistů závadu odstraní.

2.4.3. Štíhlá výroba – Lean production

Ve druhou světovou válkou zdecimovaném Japonsku narazily pokusy o zavedení hromadné výroby v automobilovém průmyslu na několik překážek. Trh byl zaplaven relativně kvalitními a lacinými automobily ze západu, Japonští dělníci nebyli ochotní plnit roli snadno vyměnitelné součástky systému a v ostrovním Japonsku se prakticky nevyskytovali imigranti, kteří ale tvořili skutečné jádro pracovní síly v západních výrobních společnostech. Produktivita Japonského dělníka navíc byla několikanásobně nižší, než u dělníka západního – oproti americkému dokonce devětkrát [3]. Přestože byla vládou zavedena opatření omezující dovoz západních aut do Japonska, což vnitřní trh oživilo, nebyly exportující firmy schopny konkurovat velkým výrobcům za hranicemi země.

V této situaci se Taiichi Ohno, jeden z manažerů automobilky Toyota, rozhodl začít experimentovat s jednotlivými výrobními činiteli za účelem dosažení minimálních nákladů se současným zvyšováním produktivity. Prvním krokem bylo zkrácení času potřebného pro přeseřízení stroje při změně typu výrobku. Toho dosáhl instalací dopravních a přeseřizovacích mechanismů pro nástroje [4]. Během experimentování zjistil, že náklady na výrobu jednoho kusu jsou nižší, pokud sníží i počet dílů v dávkách mezi operacemi a linkami. Potřeba rozsáhlých skladovacích prostor vyrobených dílů, které byly u hromadné výroby běžnou a nutnou záležitostí byla tímto rovněž eliminována. Tento fakt měl pozitivní vliv nejenom na vstupní náklady. Snížení množství kusů na výstupu rovněž znamenalo, že případné jakostní vady byly snáze odhalitelné.

Taiichi Ohno usoudil, že zásadní příčinou nižší produktivity japonského dělníka jsou prostoje, popř. zbytečná práce (Muda, z Japonštiny) a hledal cesty, jak tento nechtěný element

minimalizovat. Zavedl systém tzv. záchranné brzdy, což byla šňůra, za kterou měl pracovník zatáhnout při sebemenším podezření na nesprávnou funkci stroje, kvalitativní neshodu, či v jakékoli jiné neočekávané situaci. Každý pracovník na lince měl tuto možnost či spíše povinnost a každý byl také poučen, jak se zachovat v případě, že za brzdu zatáhne jiný dělník. V případě, že se tak stalo, linka se zastavila a celá skupina pracovníků z konkrétní linky začala závadu či problém odstraňovat. To pochopitelně kladlo vyšší nároky na proškolení pracovníků, než v případě výroby hromadné.

Další změny v personální struktuře následovaly: Byly zrušeny původní pozice mistrů výroby, u kterých byl kladen důraz spíše na manažerské schopnosti, než na technologické znalosti, týkající se konkrétní výroby. Namísto nich byly zavedeny pozice koordinátorů. Koordinátoři byli schopni zastat práci jakéhokoli řadového pracovníka, kdykoli to bylo nutné, měli vyšší zodpovědnost za chod linky a za to byli lépe odměňováni. Ohno také zavedl tzv. princip seniority, neboli odměňování v závislosti na době strávené ve firmě. Tento motivující princip měl příznivý dopad na omezení fluktuace vyškolených pracovníků i na růst jejich individuálních schopností.

Významné změny byly provedeny i v toku hodnot závodem. Byly zavedeny principy tahu (Kanban) a JIT (Just-In-Time), což v praxi znamenalo, že se v adekvátním množství vyráběl pouze ten produkt, který byl právě objednán zákazníkem a na každou linku bylo vnitřním systémem dodáváno pouze takové nezbytné množství materiálu, které bylo požadováno následujícím prvkem procesu.

V návrhu linek, strojů, nástrojů i samotných výrobků se začaly široce uplatňovat tzv. Poka-Yoke (angl. Mistake-Proofing) principy, tedy technologické způsoby, umožňující efektivně předcházet chybám pracovníka [1]. Jeden z dalších důležitých principů JIDOKA (angl. Autonomation), neboli automatizace s lidskou inteligencí, měl zase řešit automatické včasné rozpoznání špatného kusu či události, která by mohla způsobit prostoj. Byla zavedena pravidla, která systému umožnila vhodně zareagovat bezprostředně po události.

Změny, které Taiichi Ohno a jeho spolupracovníci prosadili a realizovali ve snaze o snížení nákladů, se dotkly, jak je vidět, celého hodnotového toku. Tyto inovace byly následně zapracovány do nového výrobního konceptu, který dostal jméno Toyota Production Systém, TPS.

Navzdory pomalému globálnímu ekonomickému růstu po ropné krizi v roce 1974 byly výsledky firmy Toyota Motors příznivé, což přitahovalo pozornost široké průmyslové komunity. Důvodem úspěchu firmy Toyota bylo bezpochyby nekompromisní uplatňování konceptu TPS a principů tahu ve všech částech a fázích výrobního procesu. Manažeři západních průmyslových firem se vzhledem k výsledkům Toyoty pochopitelně pokoušeli včlenit myšlenky TPS do výrobních systémů ve svých závodech. Mnozí z nich ovšem nezaznamenali takové zlepšení ukazatelů výkonnosti procesu, jaké očekávali. Hlavní příčinou nepříznivých výsledků byl často fakt, že metodika TPS nebyla implementována kompletní. Shigeo Shingo ve své knize [1] zdůrazňuje, že k dosažení očekávaných výsledků je nutné implementovat štíhlý koncept do všech fází hodnotového toku.

2.5. Štíhlá výroba ve firmě Robert Bosch České Budějovice

Společnost Robert Bosch, s.r.o. v Českých Budějovicích vyvíjí podobně jako ostatní závody koncernu Bosch intenzivní snahu o dosažení příznivých ekonomických výsledků. Důležitým projevem této snahy je také nekompromisní zavádění principů štíhlého konceptu do všech firemních procesů. Firma pro tento účel vytvořila vlastní výrobní systém, nazývaný Bosch Production System, BPS. BPS je založeno na všeobecně platných pravidlech, která popisují výrobní systém jako celek a jejichž hlavním cílem je minimalizace plýtvání. Pro realizaci principů BPS jsou používány nástroje BPS, jejichž kontinuální uplatňování je předpokladem pro úspěšné zavádění štíhlé výroby.

Jedním z pěti základních prosazovaných principů je princip *Vyvarování se chyb*. Tento princip je založen na myšlence, že preventivní opatření a rychlá reakce při eventuálním výskytu chyby je ekonomicky výhodnější, než následné pracné odhalování. Mezi nástroje tohoto principu patří poka-yoke, TPM (Total Productive Maintanance – metoda údržby, zaměřená především na prevenci) a důsledná vizualizace.

Dalším důležitým principem je *Princip tahu*, který je zde realizován tzv. Kanban kartami. Ty poskytují informace o toku materiálu závodem, zabraňují nadprodukci a také slouží jako zakázka pro výrobu.

Jiný princip, *Flexibilita* vyjadřuje schopnost jednoduchého a rychlého přizpůsobení se aktuálním požadavkům zákazníka. Princip je realizován rychlým přeseřízením, standardizovanou prací a nivelizací skladových zásob.

Nejdůležitějším principem je *Optimalizace celkového procesu*. Pro realizaci tohoto principu je nezbytné důsledné kontinuální sledování procesu. Jedním z nástrojů, který má pomoci zaznamenat a analyzovat údaje z výrobního procesu je i sledovací databázová aplikace, jež je výsledkem této bakalářské práce.

2.6. Cíle a nástroje štíhlé výroby

Definice štíhlé výroby (TOMEK, Gustav. *Nákupní marketing*. Praha : Grada, 1996. 176 s. [5]): „*Štíhlá výroba znamená vyrábět jednoduše v samořízené výrobě. Koncentruje se na snižování nákladů přes nekompromisní úsilí po dosažení perfekcionismu. Ke každému dni ve výrobě patří principy Kaizen, analýza toků a systémy Kanban. Toto úsilí vtahuje do změn všechny pracovníky podniku – od vrcholového managementu až po pracovníky ve výrobě.*“

Štíhlý koncept spočívá především ve schopnosti výrobního systému sledovat vlastní chování a aplikovat účinné metody pro zlepšování měřitelných výstupních parametrů. Tyto parametry musí být jednoznačně a kvantifikovatelně definovány. S ohledem na tuto nutnost je vhodné vysvětlit několik pojmů.

2.6.1. Plýtvání

Plýtvání je běžně používaný výraz obecného jazyka, kterým vyjadřujeme skutečnost, že zdroje potřebné pro vykonávání konkrétní činnosti nejsou využívány efektivně. Intuitivně vnímaným cílem výrobního manažera je plýtvání co nejvíce omezit.

V ekonomicko-hospodářském kontextu zmiňujeme tyto druhy plýtvání [15]:

- Nadprodukce, nadvýroba – Objem výroby je obvykle větší, než množství výsledných produktů požadovaných zákazníkem. Z mnoha důvodů je nutné tuto nadprodukci udržovat v mezích, jejichž opodstatněnost je procesem neustále prověřována. Nadprodukce je největším zdrojem plýtvání a všechny ostatní druhy z ní vyplývají.

- Prostoje – doba, kdy jsou pracovníci, zařízení (výrobní, logistické apod.) nebo výrobek nečinný a během které není k výrobku či službě přidávána žádná hodnota. Může být způsobeno prostoji nebo úzkými hrdly předchozí fáze procesu.

- Doprava – Nadbytečná manipulace s materiálem nebo výrobkem. V užitém kontextu tento pojem vyjadřuje především cestu rozpracovaného výrobku od jedné části výrobního procesu k druhé.

- Opravy, přepracování – Výrobky, které nebyly vyrobeny dle požadovaných specifikací mohou být v některých případech opraveny. Tento druh plýtvání ovlivňuje všechny ostatní druhy plýtvání.

- Nutnost pohybu – Tento druh plýtvání vyjadřuje skutečnost, že pracovník, výrobek či stroj je při práci nucen vykonávat zbytečný pohyb. Tato nutnost bývá vyvolána přítomností fyzických bariér, špatným rozvržením pracovního prostředí nebo nevhodnou fyzickou polohou jednotlivých prvků procesu.

- Skladování – Pokud nevyrovnané tempo jednotlivých fází procesu způsobí, že je v jednom kroku vyrobeno větší množství výrobků, než právě požaduje krok následující, je nutné tyto výrobky uskladnit do doby, než je bude možné zpracovat.

- Zmetky – Zmetky jsou vyrobené kusy, které kvalitativně neodpovídají požadovaným specifikacím a nemohou být nabídnuty zákazníkovi.

2.6.2. Prostoj

Pojmem prostoj označujeme v praxi časový interval, během kterého lidé nebo výrobní zařízení nevyrábí nebo nepřidávají k výrobku žádnou hodnotu. Pro maximalizaci výnosů je tedy žádoucí prostoje minimalizovat. Aby to bylo možné, je potřeba prostoje sledovat a vést spolehlivou a podrobnou evidenci výskytů jednotlivých druhů a jejich příčin. Z těchto požadavků plyne ideální představa, že by tyto události měl zaznamenávat pokud možno automaticky výrobní systém bez nutnosti zásahu člověka. Vzhledem k pestré škále příčin jednotlivých typů prostojů není tato představa jednoduše uskutečnitelná, takže ve většině případů jsou záznamy událostí pořizovány lidskou obsluhou výrobní linky. S ohledem na nestandardnost člověka je stanoveno časové rozlišení záznamového nástroje tak, aby byla zachována jeho podrobnost i vypovídací schopnost. V praxi bývá často zvolena forma jednoduché tabulky a rozlišení stanoveno např. na čtvrthodinové intervaly, ve kterých pracovníci zaznamenávají aktuální stav linky. Tyto záznamy poté bývají na denní bázi sledovány a vizualizovány např. pomocí OEE (Overall Equipment Effectiveness – Celková efektivita zařízení) grafů. Důležitým účelem OEE nástrojů je právě získávání informací potřebných k redukci

množství prostojů. Používají se k analýze zdrojů a typů prostojů a ke sledování využití kapacity jednotlivých prvků výrobního procesu a mohou napomoci nalézt a identifikovat slabé články procesního řetězu.

2.7. Ukazatele ve výrobním procesu.

Bylo řečeno, že pro úspěch regulace výrobního procesu, tedy aby poměr mezi výstupy a vloženými vstupy byl pro nás příznivý, je nezbytné sledovat a evidovat jeho chování. ***Co*** a také ***jak*** lze ale vlastně sledovat? Z důvodu vyhovění požadavku na jednoznačnost a dlouhodobou platnost získaných údajů je žádoucí sledovanou veličinu vhodným způsobem kvantifikovat. K těmto účelům slouží především ukazatele **Produktivity** a **Efektivity** výrobního procesu

Produktivita a efektivita jsou nepřímo měřitelné ukazatele způsobilosti výrobního procesu. Pravidelné a důsledné sledování těchto ukazatelů ve výrobním procesu je základním předpokladem pro jeho účinnou regulaci. Výsledky měření produktivity představují mimo jiné i vhodný zdroj informací pro stanovení výše spravedlivé odměny pracovníků na výrobě se podílejících, takže její sledování ve výrobním provozu má pozitivně motivační účinek.

Další z ukazatelů, efektivita, se od produktivity významně liší. Při sledování efektivity se více zaměřujeme na ztráty, ke kterým došlo vinou prostojů, nekvality, popř. jiných nežádoucích vlivů. Jak bude popsáno v dalším textu, k řešení obou problémů, tedy měření produktivity i efektivity je možné zvolit více různých přístupů. Rozhodnout o tom, který přístup je pro daný typ výroby vhodnější, není jednoduché. Často je zapotřebí konfrontovat nastavené limitující manažerské požadavky s předem obtížně odhadnutelnými výsledky dlouhodobého monitorování. Zvolený přístup poté určí i matematickou metodu, která je použita pro výpočet daného ukazatele. Ve spojení s důslednou analýzou prostojů se jedná o velice komplexní sadu instrumentů pro účinné vyhodnocování způsobilosti a výkonnosti výrobního procesu.

2.7.1. Produktivita

Produktivita je obvykle definována jako vztah mezi kvantitou výstupů a vstupů (výrobního) systému za jednotku času [6],[16]. V užitém kontextu ukazatelů výkonnosti průmyslových jednotek je účelnější pohlížet na produktivitu poněkud odlišným způsobem. V souladu s obecnou definicí vyjadřujeme míru produktivity v procentech, její hodnotu ale

určíme jako podíl mezi reálným a plánovaným objemem výroby (vztah 1.1). Objem výroby je zpravidla ekvivalentní počtu kvalitativně shodných kusů výrobku daného typu.

$$\text{Produktivita}\,[\%] = \frac{\sum n_{\text{reálný}}\,[\text{ks}]}{\sum n_{\text{plánovaný}}\,[\text{ks}]} \tag{1.1}$$

Reálný počet kusů je počet kusů daného typu výrobku, vyrobených za časový interval. Plánovaný počet kusů je definován jako podíl délky výrobního časového intervalu a průměrné doby strávené výrobou jednoho kusu daného typu, vynásobený počtem pracovníků, kteří se na výrobě v měřeném časovém úseku podíleli.

Na tomto místě je vhodné zdůraznit, že pro zachování smyslu měření jsou ve výpočtu zohledněny pouze kvalitativně vyhovující reálné kusy. Alternativní způsob výpočtu, který lépe respektuje specifika konkrétního výrobního provozu či linky, spočívá ve stanovení průměrné délky výroby tzv. referenčního kusu (vztah 1.2).

Konkrétní vztah pro výpočet produktivity práce je součástí duševního vlastnictví zadavatele. Existuje důvodná obava, že by zveřejnění tohoto matematického vzorce mohlo ohrozit postavení firmy vůči její konkurenci, proto jej zde neuvádím.

(1.2)

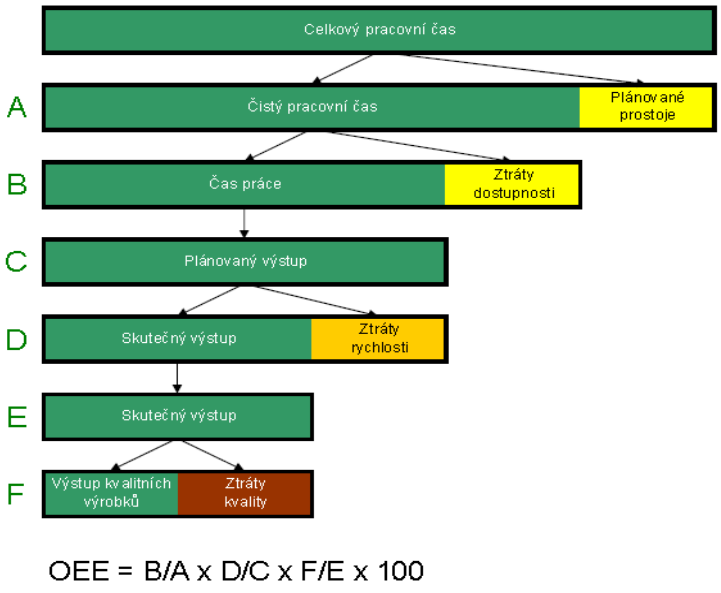
Tento vztah je implementován v analytickém modulu finální aplikace.

2.7.2. Efektivita

Ukazatel efektivity vyjadřuje zpravidla v procentech, do jaké míry je využíván potenciál lidí a zařízení vzhledem k výstupům sledovaného procesu. V případě výrobního procesu může být tato míra využití vyjadřována například v množství kusů vyrobených za časovou jednotku. Důležitou proměnnou ve výpočtu představují zdroje v podobě počtu pracovníků podílejících se na výrobě během sledovaného časového intervalu.

Alternativním přístupem k problému je využití a aplikace metodiky OEE. OEE (Overall Equipment Effectiveness, celková efektivita vybavení) je tzv. KPI (Key Performance Indicator, hlavní ukazatel výkonnosti) výrobního procesu [7]. Je to sada statistických nástrojů vyhodnocujících obvykle více parametrů sledovaného procesu. Počítá se zejména dostupnost (podíl *čistého pracovního času* a *času práce*), výkon (*ideální doba cyklu* dělená podílem čistého *pracovního času* a *počtu kusů*), kvalita (podíl *počtu dobrých kusů* a *počtu všech kusů*)

a OEE, jež je součinem předchozích tří ukazatelů (obr. 2.3). OEE je sada ukazatelů, jejichž sledováním a vhodnou reakcí na výsledky lze předejít nevyrovnanému výrobnímu taktu jednotlivých fází procesu. Díky komplexnímu monitorování a dlouhodobé analýze výsledků lze např. odhalit, která kombinace sortimentu je výhodnější a u které se naopak plýtvá zdroji. Druhy výrob, které lze provádět s variabilním počtem pracovníků mohou být zase podrobeny analýze, jejíž výsledky poskytnou odpověď, jaký počet pracovníků je nejvýhodnější. Vzhledem k tomu, že nástroje sledují vstupy i výstupy, lze získat odpovědi i na velmi komplikované logistické otázky nivelizace dodávek. Tyto údaje lze použít jako podklady pro optimální konfiguraci objemu dodávek od subdodavatelů a zajistit tak maximální plynulost výroby s minimálními náklady.



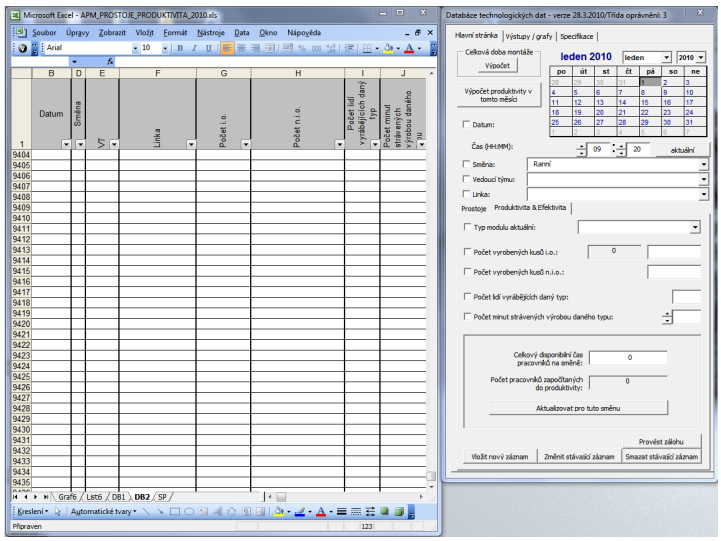
Obrázek 2.3: Metodika OEE, převzato z [7] a upraveno

2.8. Dostupné nástroje pro evidenci a analýzu údajů z výrobního procesu

2.8.1. Univerzální nástroje

V malých firmách lze pro účely evidence výrobních údajů využít možností tabulkových procesorů. V této oblasti existuje řada alternativ, z nichž některé jsou dostupné dokonce zdarma a se zdrojovými kódy. Zástupci této kategorie jsou např. OpenOffice.Org Calc [19] nebo KOffice KSpread [20]. Ty ale většinou nenabízejí pokročilé funkce pro analýzu, popř. nejsou dostatečně kompatibilní s nejrozšířenějším produktem tohoto typu, jímž

je Microsoft Excel z balíku MS Office. Kompatibilita je nedostatečná především u těch úloh, které vyžadují použití maker nebo tzv. automation (funkční propojení s dalšími aplikacemi využívajícími komponentní objektový model, COM). Excel nabízí dostatečnou funkčnost pro nezávislá pracoviště, ale selhává při nasazení do síťového prostředí. Pomocí maker lze řešit požadavky na správu uživatelů a sledování jejich aktivity. Vzhledem k prakticky nulové škálovatelnosti tohoto řešení jej ovšem nelze doporučit pro nasazení do rozsáhlých výrobních provozů.



Obrázek 2.4: Aplikace pro evidenci výrobních dat v MS Excel

Dalším problémem je sdílení dat více uživateli. Vestavěné možnosti sdílení jsou pro tento účel nevhodné, protože při jeho aktivaci nelze použít ochranu sešitu heslem (pomineme-li, že bezpečnost takto uzamčených dat je přinejmenším diskutabilní). Problém lze řešit např. rozdělením aplikace na serverovou a klientské části, přičemž serverová data jsou otevírána a načítána jen ve chvíli, kdy o ně klient požádá (obr. 2.4). Poté je serverová část opět uzavřena a poskytnuta pro použití dalším klientem.

2.8.2. Speciální nástroje

V současné době je k dispozici celá řada komerčních produktů pro řízení výrobních procesů (MES – Manufacturing Execution Systems). Některé z těchto softwarových aplikací jsou součástí podnikových informačních systémů [9]. Příkladem takového systému je **SAP** s aplikací **HYDRA** od firmy MPDV [21]. Vysoká modularita tohoto řešení poskytuje prostor pro konfiguraci systému podle požadavků zákazníka. Nevýhodou je ale vysoká cena a proto

se nehodí pro nasazení tam, kde jsou očekávány časté změny ve struktuře databáze, validačních pravidel a vyhodnocovacích algoritmů.

Některé MES se specializují pouze na několik dílčích činností, nebo poskytují vlastní implementaci OEE, např. **OEE Impact** firmy GEMBA Solutions [22], nebo **Provideam OEE** firmy DTL Systems [23]. Tato řešení jsou ale postavena podle obecné struktury výrobních provozů a často nejsou schopna respektovat specifika a požadavky konkrétního zákazníka. Kromě toho, tyto aplikace nabízejí řadu redundantních funkcí, které zákazník buď vůbec nepotřebuje, nebo mohou být nahrazeny finančně dostupnější alternativou. Určitou výhodou aplikace budované v praktické části této bakalářské práce je právě skutečnost, že tato aplikace je postavena „na míru“ výrobním provozům firmy Robert Bosch, s.r.o.

3.Praktická část

3.1. Cíle praktické části

Cílem praktické části je realizace softwarové aplikace pro evidenci a analýzu výrobních dat pro firmu Robert Bosch, s.r.o. Následující text popisuje postupy a metody použité při realizaci této aplikace. Tento popis poslouží zadavateli jako servisní dokumentace k aplikaci pro případ vzniku nutnosti její změny nebo rozšíření.

3.2. Analýza problému a předpokladů pro implementaci řešení

Firma Robert Bosch, s.r.o. velmi progresivně uplatňuje a realizuje myšlenky štíhlého konceptu v celém českobudějovickém závodu. To mimo jiné znamená, že management firmy vybudoval a neustále vylepšuje strategii pro kontinuální sledování toku hodnot závodem. Jedním z nástrojů, který má pomoci tuto strategii plnit je softwarová aplikace pro evidenci a analýzu relevantních údajů přímo z výrobních linek.

Výrobní provozy ve firmě Robert Bosch, s.r.o. jsou koncipovány v souladu s principem tahu jako hierarchická soustava montážních linek. Celou soustavu lze ilustrovat jako hlavní montážní linii, na kterou jsou v určité fázi, dané logikou postupu montáže, připojovány vedlejší linky zajišťující předmontáž sestav potřebných pro hlavní montáž. Na vedlejší linky mohou být připojeny další montážní linky nižší úrovně atd.

Pro náležité využití zdrojů v průběhu výroby je nezbytné v prvé řadě tok materiálu výrobním provozem neustále sledovat a pravidelně vyhodnocovat výsledky těchto pozorování. Efektivní provádění obou činností může usnadnit optimálně navržený počítačový program.

Aby bylo možné exaktně formulovat zadání pro tvorbu tohoto programu, bylo nutné stanovit okruhy informací, které jsou relevantní pro následnou analýzu. Bylo ovšem nutné zohlednit také uživatelskou náročnost získávání a ručního vkládání většího množství údajů.

Jedním z nejdůležitějších okruhů těchto údajů je **čas** a **místo události**. Událostí je zde rozuměn časový interval ohraničující dobu, kdy se buď vyrábělo (údaj pro produktivitu), nebo

nevyrábělo (údaj pro analýzu prostojů). Místo lze na základě konkrétních nároků na místně rozlišovací schopnost vyhodnocovacího nástroje specifikovat podle úrovně na (seřazeno od nižší úrovně k vyšší): **pracoviště**, **linku**, **oddělení** nebo **závod**. Časové údaje lze rozčlenit na **datum** a **čas**, popř. identifikaci **směny**, na které k události došlo a **délku trvání** události. Dalšími důležitými údaji je identifikace **týmu** účastnícího se události, **typu montovaného výrobku** a v případě prostojů **typu prostoje** a **příčiny**. Abychom mohli vyhodnocovat produktivitu pracovníků, musíme stanovit rovněž způsob záznamu času všech lidí na směně, tzv. **disponibilního času pracovníků na směně.**

3.2.1. Volba platformy, programovacího jazyka, frameworku

Pravděpodobně nejčastěji používaným způsobem záznamu detailů událostí pomocí výpočetní techniky je vkládání každé nové události na konec homogenní datové struktury – v jednodušších aplikacích tuto roli plní serializovatelné generické datové typy fronta či zásobník, méně vhodně také seznam nebo pole. Vzhledem k rozsahu projektu byl v našem případě zvolen záznam událostí do databázových tabulek. Konkrétní databázovou platformou se stal Microsoft SQL Server a to především s ohledem na softwarově technologické konvence platné ve firmě Robert Bosch, s.r.o., přičemž marginálním požadavkem byla rovněž uživatelská dostupnost uložených dat. Databáze má být vytvořena s maximálním důrazem na konzistentnost dat, především pak referenční integrity. To mimo jiné znamená, že vnitřní funkce serveru pohlídají jedinečnost údajů, u kterých je tato požadována a naopak zabrání odstranění údaje, který je propojen s údajem uloženým v jiné tabulce [10].

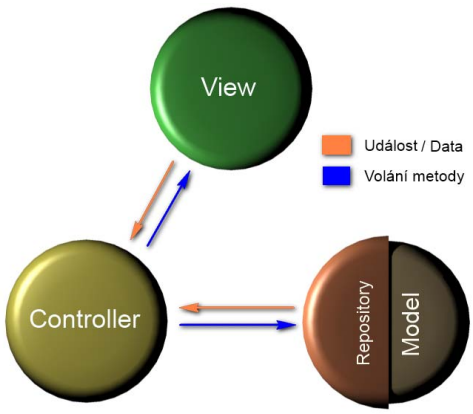
Platforma pro uživatelské prostředí a celou aplikační část byla volena především s ohledem na požadavky uživatelů. Důležitým kritériem pro volbu aplikační platformy je ovšem také robustnost, uživatelská přístupnost a použitelnost výsledku ve stávajícím prostředí. Klíčovým požadavkem při tvorbě prezentační vrstvy aplikace se stala uživatelská přívětivost a dostupnost dat.

Všechny uvedené požadavky splňuje typ aplikace programované pro použití ve webovém prohlížeči. Jako vývojová platforma byl tedy, opět s ohledem na zavedené konvence platné ve firmě Robert Bosch, s.r.o., vybrán ASP.NET. Pro zvýšení robustnosti výsledku byl použit návrhový vzor MVC (Model View Controller) implementovaný ve stejnojmenném frameworku pro ASP.NET. Tento framework ze své podstaty zajišťuje

dostatečnou izolaci a nezávislost datové, logické a prezentační vrstvy. Programátor navíc obdrží dodatečný benefit v podobě podpory pro vývoj automatizovaných jednotkových testů. Tyto testy jsou jedním z nástrojů tzv. extrémního programování.

Dalším krokem byl výběr vhodného typu datové vrstvy. Hlavně kvůli přehlednosti a úspornosti zápisu v posledních verzích .NET frameworku byla zvolena technologie LINQ to SQL (Language Integrated Query to SQL). Zápis databázových dotazů v LINQ je totiž mimořádně intuitivní a programátorsky úsporný. S výhodou lze v LINQ používat také tzv. lambda expressions, jež jsou novým rysem jazyků .NET frameworku (od verze 2.0). Tyto důvody vedly k volbě jazyka C# jako hlavního jazyka pro tvorbu cílové aplikace.

Aby bylo usnadněno případné rozšiřování aplikace v budoucnu, byla do projektu vložena třída `VyrobaRepository` podle návrhového vzoru The Repository design Pattern [11]. Tato třída ještě více odděluje uživatele od datového úložiště a činí aplikaci de-facto nezávislou na typu datového modelu (obr. 3.1). Lze tedy s minimálními úpravami kódu použít jiný druh úložiště nebo jiné řešení datové vrstvy (např. „Entity Framework“ namísto LINQ).



Obrázek 3.1: Symbolické znázornění interakce prvků návrhového vzoru MVC s třídou Repository

3.2.2. Požadavky zadavatele - konkretizace zadání

Aby výsledná aplikace mohla být nasazena do ostrého provozu, zadavatel požaduje splnění několika požadavků a omezení. Tyto požadavky lze rozdělit do tří hledisek (tab. 1).

Hledisko	Č.	Požadavek
datové	1	Datová část musí obsahovat všechna pole potřebná pro uživatelskou identifikaci záznamu
	2	Data musí být uchovávána spolehlivě, jejich integrita nesmí být ohrožena ani uživatelem ani neočekávaným chováním vlastních funkcí.
aplikační	3	Aplikace bude použita ve víceuživatelském prostředí uživatelů více úrovní (rolí). Tyto úrovně je nutné respektovat především při autorizaci přístupu k jednotlivým funkcím.
Uživatelské	4	S aplikací budou pracovat lidé z výrobních provozů, proto je nutné poměrně velký důraz při vývoji klást na uživatelskou přívětivost a ergonomii uživatelského prostředí.
	5	Ze stejného důvodu, jako v bodu 4 je nutné nastavit vkládací pole tak, aby uživatel mohl vložit pouze takovou hodnotu, která je v dané situaci relevantní.

Tab.1: seznam hledisek požadavků zadavatele na výslednou aplikaci.

3.3. Návrh Aplikace

3.3.1. Použitá obsahová a typografická konvence

Pokud není uvedeno jinak, všechny příklady a výpisy uvedené v příloze pracují s tabulkou výrobních linek. Práce s ostatními tabulkami je analogická. Kompletní výčet tabulek, polí a jejich závislostí je duševním vlastnictvím zadavatele.

Kurzívou jsou zvýrazněny názvy prvků, které se neváží přímo ke zdrojovému kódu, např. jména složek.

`Neproporcionálním písmem` jsou uvedena všechna jména objektů, typů, metod a proměnných vyskytujících se ve zdrojovém kódu aplikace.

3.3.2. Databáze

Na tomto místě položíme datové základy aplikace, na kterých budeme dále stavět. Vzhledem ke stávajícímu serverovému prostředí ve firmě Robert Bosch, s.r.o. je databáze budována na platformě MS SQL Server.

Databáze sestává z několika tabulek, které jsou označeny prefixem identifikujícím jejich účel:

- *aspnet* – tabulky vytvořené pomocí nástroje *aspnet_regsql.exe*. Tyto tabulky slouží pro administraci aplikace a kromě tabulek nástroj vytvoří také několik uložených procedur a pohledů. Vzhledem k tomu, že tabulky jsou generovány volně dostupným nástrojem, dále jim není věnována pozornost.

- *adm* – tyto tabulky rozšiřují generované možnosti administrace, aby lépe vyhovovaly specifickým firemním požadavkům na správu uživatelů.

  Jednou z nejdůležitějších tabulek tohoto typu je *adm_depts* – detaily oddělení. Téměř všechny uživatelské tabulky se na tuto tabulku odkazují. To umožňuje vyhovět požadavku č. 3 (viz tab. 1) tak, že uživatel pracuje vždy jen s údaji relevantními pro jeho oddělení.

- *spec* – tabulky obsahující jednoznačně indexovaná data použitá v hlavních tabulkách událostí. Jedním z hlavních účelů těchto tabulek je uložení údajů, které jsou při editaci událostí nabídnuty uživateli v rozbalovacím poli, tzv. *DropDownListu*. (Tímto způsobem řešení interakce uživatele je splněn požadavek č. 4 a částečně i požadavek č. 5 z tab. 1)

- *pt* – tímto prefixem jsou uvozeny hlavní tabulky uchovávající údaje o události výroby nebo prostoje.

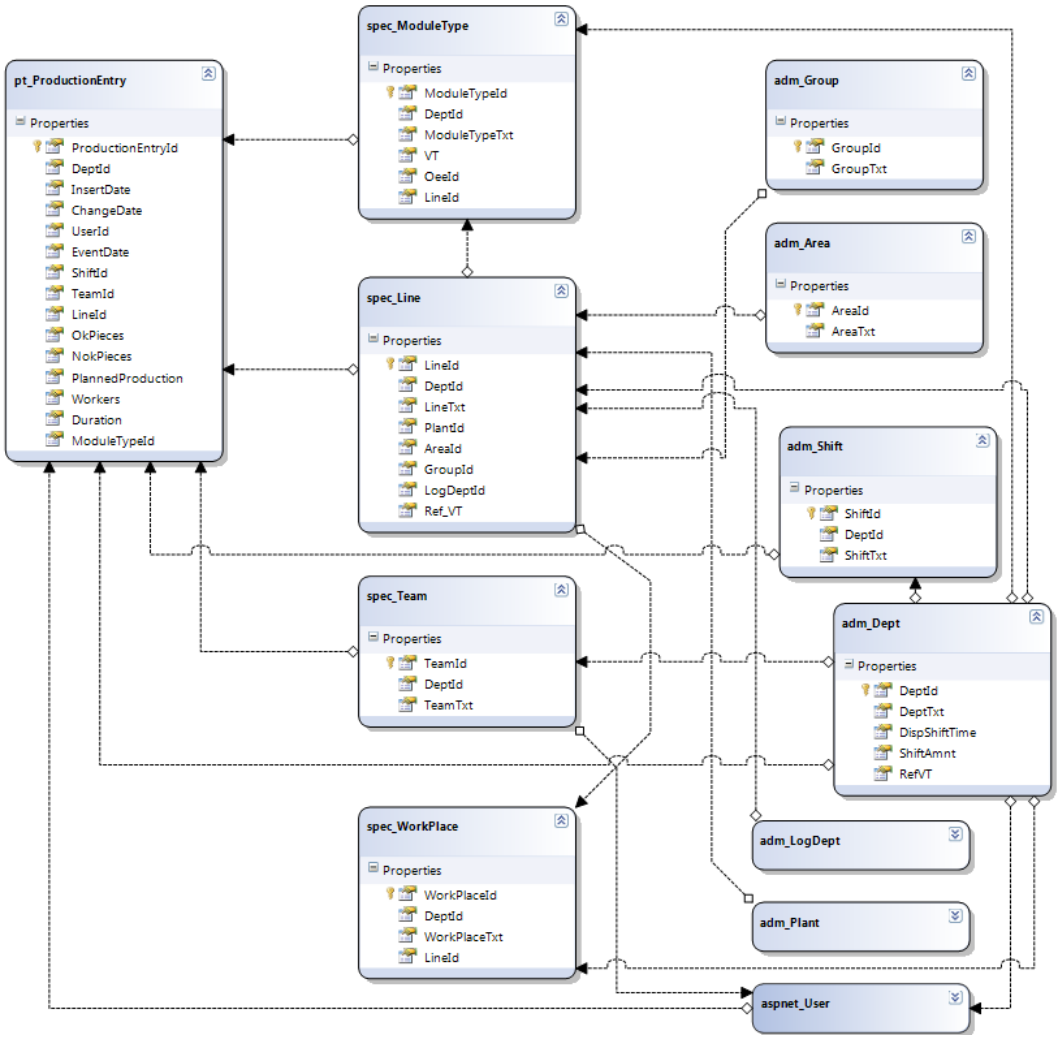
První sloupec téměř každé (výjimka je popsána níže) vytvořené tabulky je primárním klíčem, použitým pro indexování a jednoznačnou identifikaci záznamu. Toto pole je typu *int* nebo *bigint* a má status *Identity = true* s hodnotou *Identity increment = 1*.

V poli primárního klíče je tedy ukládána numerická hodnota jednoznačně identifikující záznam, na který může referencovat jiná tabulka cizím klíčem. Cizí klíč má vždy jméno totožné se jménem primárního klíče v odkazované tabulce.

Jedinou výjimkou je tabulka pro disponibilní čas pracovníků na směně. U této tabulky vznikl logický požadavek na jednoznačně rozlišitelnou hodnotu času pro každou jedinečnou kombinaci data, směny a místní identifikaci události. Tato kombinace ve zmíněné tabulce tvoří referencovatelný primární klíč.

Všechny databázové tabulky tvoří hierarchickou strukturu (její část, týkající se výroby, viz obr. 3.2), na jejímž dně jsou tabulky pro záznam událostí o výrobě nebo prostoji (tabulky

pt_ProductionEntries a pt_DownTimeEntries). Nad těmito událostními tabulkami jsou vystaveny tabulky pro uložení specifikací (tabulky s prefixem *spec_*) a tabulky pro uložení informací umožňujících administraci aplikace (prefix *aspnet_* a prefix *adm_*).



Obrázek 3.2: Část datového modelu. Zobrazena je část databáze, týkající se událostí relevantních pro výpočet produktivity.

Po vytvoření základní struktury databáze (tabulky s prefixem *pt_, spec_, adm_*) k ní připojíme úložiště pro pokročilou správu uživatelů. Toto provedeme pomocí příkazu *aspnet_regsql.exe*, spuštěného z příkazové řádky Windows.

Posledním krokem před generováním aplikační části datové vrstvy je definice tzv. pohledů. Pohledy (Views) jsou vlastně virtuální tabulky, tvořené často komplikovaným SQL dotazem, které neuchovávají skutečná data, ale slouží pro jejich jednodušší a často i rychlejší získávání z databázového serveru. Jednodušší proto, že logika dotazu je vložena již do databáze a aplikace se stará pouze o další zpracování odpovědi, např. zobrazování v tabulce, tzv.

Gridu. Výhoda rychlosti se projevuje především u virtualizovaných serverů, jejichž výkonová kvóta je často pouhým zlomkem celkového výkonu hardwaru, na kterém je server spuštěn. Databázový server bývá totiž výkonnější, než server webový a je tedy účelné ponechat část výkonově náročných úkolů na něm. Dalším důvodem použití pohledů je skutečnost, že aplikace využívá pro zobrazení souhrnných výpisů služeb Open Source vývojového balíku firmy Telerik, *Telerik Extensions for ASP.NET MVC* [17]. Pokud bychom se vyhnuli použití pohledů, čelili bychom později problémům při implementaci technologie AJAX (Asynchronous Javascript And XML) pro vázání dat na prvek *Telerik MVC Grid*.

Abychom vyloučili chybu při zpracování dotazu, jehož výsledkem by byly některé položky prázdné, je vhodné při definici SQL pohledů použít tzv. vnějšího spojení [10]. V našem případě LEFT OUTER JOIN (typická definice pohledu je uvedena ve výpisu 3.1)

3.3.3. Model

Další důležitou částí funkčního celku webové aplikace je projekt v ASP.NET MVC. Aplikace je vytvářena jako projekt programu Microsoft Visual Web Developer 2010 Express (dále pouze VWD). Jméno projektu je VyrobaMVC. Po založení projektu se ve složce vytvoří také několik podsložek, jejichž význam je patrný z jejich názvu. Zde je výčet těch nejdůležitějších:

- *Content* – složka pro uložení definic kaskádových stylů, obrázků a dalšího obsahu, který je využíván při renderování výsledných webových stránek.

- *Controllers* – složka, ve které se nacházejí soubory obsahující definice tříd jmenného prostoru `VyrobaMVC.Controllers`. Tyto třídy komunikují s modelem a s pomocí šablon uložených ve složce *Views* generují výstup, který uživatel obdrží v podobě webové stránky, JSON objektu, nebo třeba textového souboru, nabídnutého uživateli ke stažení.

- *Models* – Zde se nacházejí třídy jmenného prostoru `VyrobaMVC.Models` . Členem tohoto prostoru jsou především třídy vytvořené designérem LINQ to SQL, obsažené v souboru *VYROBA.DBML*. Dále jsou zde uloženy třídy pro validaci vkládaných dat a již zmíněná třída `VyrobaRepository`.

- *Scripts* – v této složce jsou, jak již název napovídá, uloženy skripty jazyka JavaScript. Dále jsou zde uloženy soubory JavaScriptové knihovny *JQuery*, která poskytuje prvky s funkčností, simulující chování GUI desktopových aplikací.

- *Views* – Složka Views obsahuje podsložky pro uložení souborů ASPX (popř. ASCX, vysvětleno dále). Ty slouží jako XML šablony pro vygenerování výsledného HTML souboru poslaného do webového prohlížeče klienta.

Aplikace s databází přímo komunikuje prostřednictvím souboru částečných tříd (partial classes – nový rys jazyka C# 2.0) generovaných designérem LINQ to SQL. Tento soubor, nazvaný *Vyroba.DBML* je uložen ve složce Models a jeho obsah je členem jmenného prostoru `VyrobaMVC.Models`. Třídy obsahují všechno potřebné pro řízení databázových tabulek a podporu pro objektovou reprezentaci dat včetně vzájemných závislostí jednotlivých objektů, definovaných v databázi.

Třída `VyrobaRepository`, uložená v souboru *VyrobaRepository.cs* je další třídou, jež je definována ve jmenném prostoru `VyrobaMVC.Models`. Úkolem této třídy je zapouzdřit funkcionalitu pro vytváření, prohlížení, změnu a smazání objektu v databázi, tedy tzv. metody CRUD (anglický akronym tvořený počátečními písmeny sloves: Create, Read, Update, Delete). Návratovou hodnotou členských metod vracejících více objektů je generická kolekce objektů `IQueryable<T>`, kde šablona `T` vyjadřuje datový typ člena kolekce. Argumentem funkcí, které vracejí pouze jeden objekt z tabulky, je zpravidla celé číslo. Toto číslo identifikuje vrácený objekt podle primárního klíče tabulky. Výjimkou z tohoto pravidla je metoda `GetUserId()`, která vrací ID datového typu `Guid`, identifikující přihlášeného uživatele. Příklady CRUD metod s výjimkou editace stávajících objektů zobrazují výpisy 3.2a až 3.2d v příloze. Změny stávajících objektů jsou řešeny pomocí LINQ a metody `UpdateModel()`, která je členem abstraktní třídy `Controller`, definované ve jmenném prostoru `System.Web.MVC`. Jakékoliv změny v databázi jsou vždy potvrzeny metodou `Save()` (Výpis 3.2e, příloha).

3.3.4. Controllers

Další logickou vrstvou návrhového vzoru MVC je vrstva Controller, jejímž hlavním úkolem je zpracování vstupu uživatele. Každému objektovému typu projektu VyrobaMVC je přidělen vlastní controller. V každém controlleru je pak sada tzv. akčních metod, které

přijímají požadavky klienta HTTP GET, popř. POST. Akční metody, které přijímají od klienta podnět pro zobrazení interaktivního formuláře používají metodu GET. Naproti tomu akční metody, jejichž úkolem je zpracovat data z formuláře a reagovat např. voláním členské metody třídy `VyrobaRepository`, používají HTTP metodu POST. Speciálním případem akčních metod je metoda `_AjaxBinding()` (viz dále), která využívá technologie AJAX. Soubory controllerů jsou uloženy ve složce *Controllers*.

3.3.5. Views

Tato vrstva je tvořena XML šablonami, soubory s příponou ASPX (ASCX), které obsahují definice výsledných HTML dokumentů posílaných do prohlížeče klienta. Soubory jsou umístěny do podsložky nazvané stejně jako příslušný controller. Tyto podsložky se nacházejí ve složce *Views*. Jednotlivé šablony obsahují předlohy webových stránek. Ve složkách nazvaných podle tabulek projektu VyrobaMVC jsou uloženy šeblony pro CRUD. Na rozdíl od statického HTML je kód těchto souborů doplněn speciálními značkami, které umožňují modifikovat vzhled výsledné webové stránky na straně serveru, podobně jako v případě klasického konceptu ASP.NET WebForms. Na rozdíl od WebForms, Views návrhového vzoru ASP.NET MVC standardně neobsahují pole ViewState ani PostBack. Z toho důvodu tedy nelze použít komponenty, které jsou na této funkčnosti založeny [12], např. komponentu GridView, která uživateli zprostředkuje veškerou interakci s databází, včetně CRUD.

Vzhledem k požadavkům zadavatele na uživatelský komfort a pokročilé možnosti práce s databází bylo pro vyřešení tohoto problému vybráno rozšíření od firmy Telerik. Softwarový balík *Telerik Extensions for ASP.NET MVC* poskytuje ideální řešení pro zobrazení výčtu z tabulek včetně uživatelsky definovatelného třídění, řazení, stránkování, seskupování a filtrování výstupů. Kromě toho lze při vázání dat k této komponentě s výhodou využít technologie AJAX.

3.3.6. Helpers

Složka Helpers sdružuje několik různorodých pomocných tříd zjednodušujících a zčitelňujících zdrojový kód ostatních prvků projektu. Typičtí představitelé této skupiny jsou:

- `ControllerHelpers` – třída, vyskytující se ve jmenném prostoru `VyrobaMVC.Controllers`, jejímž úkolem je validace uživatelem vloženého

obsahu ovládacích prvků obsažených ve webovém formuláři (Výpis 3.3, příloha). Členská metoda `GetRuleViolations` je volána po každém odeslání vyplněného formuláře na server (Výpis 3.15, příloha)

- `CSV` – třída (Výpis 3.4, příloha) je definována ve jmenném prostoru `VyrobaMVC.Helpers`. Tato třída zapouzdřuje metodu `Export2CSV()`, která vytváří soubor typu CSV a posílá jej klientovi jako HTTP odpověď. Tento textový soubor lze dále zpracovávat např. v programu MS Excel. Součástí bakalářské práce je doplněk *ChartWiz.XLA*, jenž upravuje ovládání nejpoužívanějších analytických nástrojů Excelu pro použití na výstupech aplikace VyrobaMVC.

- `HtmlHelperExtensions` – Statická třída `HtmlHelperExtensions` je také definována ve jmenném prostoru `VyrobaMVC.Helpers` a mimo jiné zapouzdřuje metody pro zobrazení a ovládání *JQuery* komponenty *#datepicker*.

3.3.7. Komunikace vrstev konceptu MVC

Vrstva Model se vztahuje k doménové logice aplikace a je obvykle zastoupena databází a datovou vrstvou. Tu v aplikaci VyrobaMVC tvoří třída `VyrobaRepository` a třídy vygenerované pomocí nástroje LINQ to SQL.

Vrstva View je tvořena XML šablonami, popisujícími vzhled výsledné HTML stránky. Zprostředkovává aplikaci vstup uživatele a pracuje s vrstvou Model v tom smyslu, že zobrazuje data uložená v databázi. Obě tyto vrstvy by v souladu s filozofií architektury MVC měly být odděleny od uživatele [12]. Vstup uživatele je předáván prostřednictvím webové stránky a síťového protokolu odpovídající akční metodě některé z tříd vrstvy Controller ke zpracování. Akční metody mohou přistupovat k Modelu a při ukončení mohou, v závislosti na typu své návratové hodnoty, např. vrátit nový obsah stránky, či její části.

V klasickém typu ASP.NET aplikace, tzv. WebForms aplikaci, jsou URL mapovány přímo na ASPX soubory obsahující obsah webové stránky. Součástí těchto souborů je tzv. „Code behind“, jehož úkolem je zpracování událostí. Přístup k databázi je řešen buď vhodnou datovou vrstvou přístupnou z „Code behind“, nebo namapováním přímo na .NET komponentu. Návrhový vzor MVC používá odlišný přístup, ve kterém jsou URL vázány na akční metody Controllerů. Za toto namapování je zodpovědná .NET komponenta *Routing Engine (Routing Handler)* poskytující mnoho možností nastavení URL pro aplikaci *(Routing Rules)*.

Nastavení pravidel routování je uloženo ve složce aplikace v souboru *Global.asax*. V tomto souboru je předdefinováno několik jednoduchých pravidel, se kterými pracuje i aplikace VyrobaMVC.

- Prvním nepovinným parametrem za URL adresou aplikace je jméno třídy controlleru. Pokud není zadán, je použita jeho implicitní hodnota „Home“ a dotaz je vyřízen zobrazením domovské stránky aplikace.

- Druhým nepovinným parametrem je jméno akční metody. Typickým využitím tohoto parametru je specifikace stránky, umožňující uživateli vložit či změnit nastavení pro některou akci CRUD. Pokud tento parametr není vložen, použije se hodnota „Index“. Aplikace VyrobaMVC vrátí výpis tabulky odpovídající prvnímu parametru.

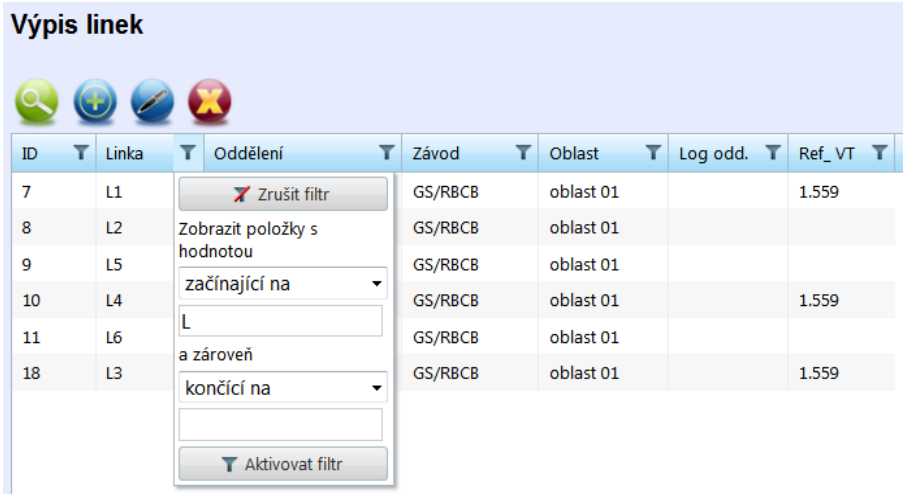
- Třetím parametrem je číslo, identifikující položku tabulky specifikované prvním parametrem. Pokud je druhým parametrem požadováno zobrazení (zpracování) konkrétního záznamu v databázi, je tento parametr povinný.

Třídy controllerů obsahují několik akčních metod, jejichž úkolem je zprostředkovat data mezi prohlížečem klienta a serverem. Akční metody použité v projektu VyrobaMVC poskytují dva hlavní způsoby předání informace ve směru od prohlížeče na server. První způsob, který využívá metodu HTTP GET je realizován za použití textu vloženého do URL. Návratovou hodnotou těchto metod je `ViewResult`, tedy obsah, definující vzhled výsledné HTML stránky podle šablon uložených ve složce *Views*. Druhý způsob, tedy metoda HTTP POST, je využit, pokud klient vrací data vyplněného formuláře. V tomto případě data nejsou vložena do URL, ale poslána skrytě v odpovědi HTTP (Výpis 3.5, příloha).

Předpokladem pro správnou funkci controllerů je přístup k modelu, proto jsou v nich instancovány objekty tříd `VyrobaRepository` a `VyrobaDataContext` (Výpis 3.6, příloha).

Jednotlivé metody CRUD jsou řešeny tak, aby se ovládání aplikace příliš nelišilo od ovládání běžných desktopových aplikací. GET metoda `Index()`, která reaguje na URL požadavek na zobrazení výpisu všech položek, nemá žádný vstupní argument a ve `ViewResult` vrací `IQueryable<T>` (Výpis 3.7, příloha).

Šablony *Index.aspx* jsou „strongly-typed“, což znamená, že pracují s daty konkrétního objektového typu. Tělo tvoří inicializace objektu *Grid* rozšíření *Telerik* (Výpis 3.8) [17]. Do HTML stránky je renderován *Grid* pro zobrazení a zpracování seznamů (obrázek 3.3).



Obrázek 3.3: Šablona Index.aspx s *Gridem* zobrazená ve webovém prohlížeči. Nad *Gridem* jsou zobrazeny ikony pro CRUD

Součástí šablony *Index.aspx* je skript obsahující funkci `onRowSelected()` (Výpis 3.9, příloha). Tato funkce je spuštěna v reakci na událost výběru položky v seznamu a nastavuje javascriptovou proměnnou `srow`. Pokud uživatel klikne na jednu ze čtyř ikon nad *Gridem*, spustí funkci `Popup`(), která otevírá nové okno prohlížeče a volá příslušnou akční CRUD metodu s parametrem této proměnné.

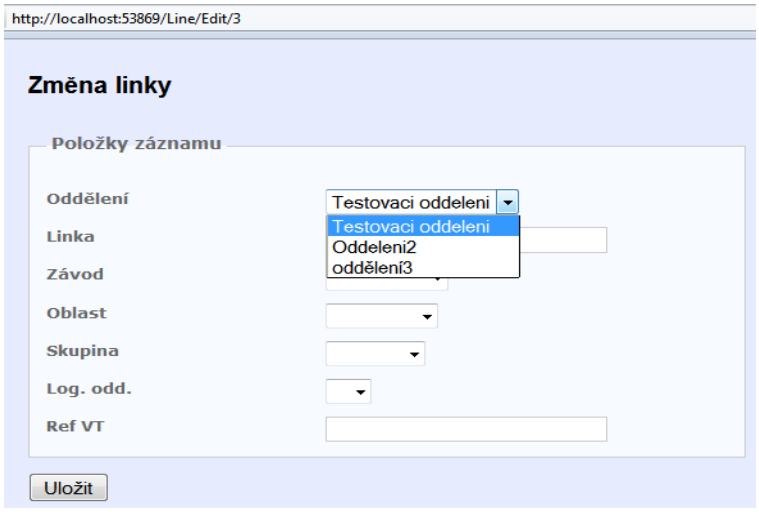
První z těchto metod je metoda `Details(int id)`, jejíž celočíselný parametr určuje položku, která má být načtena z databázové tabulky a zobrazena v tzv. Pop-Up okně se všemi detaily. V některých controllerech je do `ViewResult` této akční metody vloženo pole struktury `ViewData[]` pojmenovaného podle položky tabulky (popř. pohledu), jejíž hodnota se má zobrazit (Výpis 3.10, příloha). Tělo šablony Details.aspx je tvořeno definicí sady polí a jejich propojení s daty (Výpis 3.11, příloha).

Další akční metoda, `Edit(int id)` generuje data zobrazované položky podobně jako metoda `Details(int id)` s tím rozdílem, že parametrem vráceného `ViewResult` není šablona *Details.aspx*, ale *Edit.aspx* (Výpis 3.12, příloha).

Šablona *Edit.aspx* stejně jako *Create.aspx* obsahuje pouze odkaz na částečnou šablonu *Editform.ascx* (Výpis 3.13, příloha)

Částečná šablona *Editform.ascx* je stejně jako šablona *Details.aspx* tvořena definicí sady polí. Avšak v tomto případě jsou data objektu ViewModel vázána na ovládací prvky (Výpis 3.14, příloha).

Ovládací prvky `Html.DropDownList()` jsou naplněny relevantními daty odpovídající tabulky předanými přes pole `ViewData[]` (obr. 3.4).



Obrázek 3.4: Ukázka šablony Edit.aspx s DropDownListem naplněným daty.

Reakcí na požadavek POST odeslaný částečnou šablonou *Editform.ascx* je spuštění odpovídající POST akční metody. Pokud uložení změn do databáze proběhne v pořádku, je tato skutečnost uživateli oznámena hláškou v okně, které se po několika sekundách automaticky zavře (Výpis 3.15, příloha).

Pokud při ukládání změněných dat naopak dojde k chybě (Typickou zachycenou chybou je potenciální nebezpečí porušení referenční integrity), jsou jednotlivá `ViewData[]` naplněna původními hodnotami (v případě POST metody Create() jsou vyprázdněna). Kromě toho je vygenerována validační informace (Výpis 3.16, příloha), která je zobrazena pomocí metody `ValidationMessageFor()` volané v šabloně.

Poslední CRUD akční metodou je `Delete()`, jejíž GET verze slouží pro zobrazení detailu o záznamu, spolu s dotazem, zda záznam odstranit z databáze. Její kód se výrazně neliší od kódu akční metody `Details()`.

POST verze se pokusí vybraný záznam odstranit z databáze. V případě neúspěchu je zobrazena odpovídající chybová hláška (Výpis 3.17, příloha).

Ve všech controllerech, které zobrazují *Grid*, se nachází ještě jedna metoda. Akční metoda `_AjaxBinding()` v případě potřeby posílá data ve formě JSON [18]. Tato komunikace je zcela v režii komponenty *Telerik Grid* (Výpis 3.18, příloha).

Díky AJAX vazbě je možné data v *Gridu* filtrovat, řadit apod. bez nutnosti obnovení celé webové stránky, protože metoda `_AjaxBinding()` vrací pouze samotná data (Výpis 3.19, příloha).

3.3.8. Výstupy a grafy

Grafickou reprezentaci numerických dat je vhodné použít v takových aplikacích, kde nás zajímá trend zkoumané veličiny, popř. tam, kde potřebujeme odhalit dílčí anomální hodnotu. Tato vizualizace je v aplikaci VyrobaMVC řešena dvěma způsoby:

- Generování grafu přímo do webové stránky – tento způsob je použit pro zobrazení grafu produktivity týmů pracovníků.
- Generování souboru CSV a jeho následné zobrazení ve formě kontingenčních grafů v aplikaci MS Excel – pro vizualizaci dat událostí.

Všechny výstupy pro analýzu nasbíraných dat jsou řešeny členskými metodami třídy `ChartController`.

Produktivita

Produktivita je vypočítána s pomocí LINQ dotazu, zohledňujícím skutečnost, že ne všichni pracovníci, které chceme započítat do produktivity, se přímo účastnili dílčích událostí výroby. K evidenci hodnoty disponibilního času všech pracovníků je použita databázová tabulka uchovávající tuto hodnotu pro každou unikátní kombinaci data, směny a oddělení, popř. týmu. Členská neakční metoda `cProd()` vypočítá produktivitu pracovníků od začátku měsíce do data určeného vstupním parametrem metody (Výpis 3.20, příloha).

Graf vložený do HTML stránky

Pro generování obsahu grafu je použita metoda `CreateSeries()`, která volá metodu `cProd()` pro každý datum z uživatelem definovaného intervalu (Výpis 3.22, příloha). Struktura `chartsettings` obsahuje informace o začátku a konci časového

intervalu, příslušnosti přihlášeného uživatele k oddělení a údaje o vzhledu požadovaného grafu. Struktura (třída) `point` je členem komponenty *Chart*, obsahující informace o poloze jednotlivých bodů a nastavením její vlastnosti `AxisLabel` lze rovněž nastavit popisek dat na vodorovné ose

Pro vložení grafu do výsledné webové stránky využívá aplikace VyrobaMVC služeb komponenty *Chart (System.Web.DataVisualisation.dll)*. Přestože je tato komponenta primárně určena pro WebForms aplikace, kterým poskytuje komfortní možnosti nastavení mapování, je možné ji prakticky bez omezení použít i v aplikačním návrhovém vzoru MVC. Jedním ze způsobů, jakým lze vložit dynamicky generovaný graf do webové stránky je generovat graf do souboru obrázku prostřednictvím akční metody controlleru (Výpis 3.23, příloha). vytvořený soubor je následně uložen a jeho adresa poskytnuta webové stránce ve formě dynamického odkazu (Výpis 3.21, zobrazený výsledek viz obr. 3.5).

Obrázek 3.5: Ukázka stránky s grafem pro výstup produktivity. (Zde zobrazená data jsou vygenerována generátorem náhodných čísel.)

Výstup do CSV souboru

Testovací aplikace, vytvořená v programu MS Excel, využívá unikátní rys tohoto tabulkového procesoru, jímž jsou kontingenční tabulky a grafy [13]. Tato funkce je díky své flexibilitě vhodná k experimentování s nasbíranými numerickými daty, proto byla možnost exportu do tohoto programu vestavěna i do aplikace VyrobaMVC. Jako souborový formát pro export byl vybrán formát CSV, především pro jednoduchost implementace exportu.

CSV je souborový formát, který je tvořen hodnotami oddělenými čárkami nebo středníky. Typ oddělovače je závislý na jazykové mutaci konkrétní aplikace, neboť v některých jazycích je čárka používána v roli desetinného oddělovače. Z toho důvodu má metoda `Export2CSV()` ve třídě `CSV` vstupní parametr `separator`, jímž lze oddělovač explicitně určit (Výpis 3.4). Dalším parametrem této metody je opět výsledek LINQ dotazu, který je metodě předán ve formě `IQueryable<T>` (Výpis 3.24).

3.3.9. Doplněk XLA

Graf vložený do webové stránky řeší mnoho požadavků na přehlednou vizualizaci dat, ale v některých situacích lze považovat za nedostatek jeho statickou povahu. Metoda `CreateSeries()` vytváří data grafu pomocí dotazu do databáze, takže zdrojový kód controlleru musí obsahovat potřebné informace pro formulaci databázového dotazu. Aplikace VyrobaMVC používá pro získávání dat z databáze konstrukce LINQ, které jsou součástí kódu a nelze je tedy měnit za běhu programu. Jedním z řešení požadavku na variabilitu těchto grafických výstupů je poskytnout uživateli možnost zapsat databázový dotaz přímo v jazyku SQL, ale z hlediska bezpečnosti není tato metoda příliš vhodná.

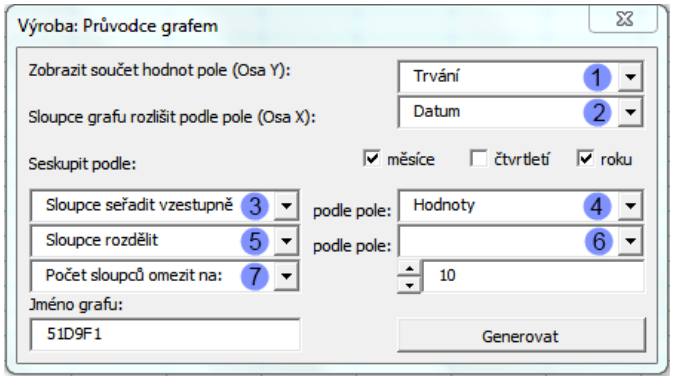
Kontingenční tabulky a grafy programu MS Excel jsou velice silným a pružným nástrojem pro přehlednou vizualizaci dat a poskytují uživateli dostatečnou volnost nastavení. Široké možnosti jejich počátečního nastavení mohou být ale komplikací pro uživatele, kteří tyto nástroje používají zřídka či vůbec. Jeden z mnoha způsobů, kterým lze uvedený problém řešit, je vytvoření funkční nadstavby ve formě jednoduchého průvodce zpřístupňujícího pouze ty funkce, které jsou pro danou situaci užitečné. Tento průvodce může být vložen do sešitu jako makro vytvořené v jazyku VBA, nebo jej lze připojit k Excelu ve formě doplňku XLA nebo XLL. Vzhledem k tomu, že data exportovaná z aplikace VyrobaMVC mají formát obyčejného textového souboru, nemohou obsahovat prakticky žádnou funkčnost použitelnou programem MS Excel. Byla tedy zvolena varianta průvodce vloženého do doplňku.

Účelem průvodce je poskytnout uživateli pouze skutečně nezbytné možnosti nastavení pro vygenerování kontingenčního grafu. Uživatel je tedy veden jemu blízkou logikou k vytvoření grafu, který může dále upravovat.

Prvním krokem tvorby tohoto doplňku je vytvoření okna formuláře (obr 2.7). V dalším kroku je třeba nastavit událostní procedury ovládacích prvků tak, aby se změna jejich hodnoty projevila na vzhledu a konfiguraci prvků ostatních. Tímto nastavením je řešeno splnění

požadavku č.5 z tab. č.1. Do událostní procedury tlačítka Generovat je umístěno volání procedury `ChartWiz()` (Výpis 3.25), která vygeneruje požadovanou kontingenční tabulku i graf.

Použití doplňku je intuitivní. Po provedeném exportu vybraných dat z aplikace VyrobaMVC je otevřen program MS Excel (pokud je k této aplikaci asociována přípona CSV) a obsah exportovaného souboru je vložen do nového prázdného sešitu. Po iniciaci doplňku kliknutím na připojené tlačítko či stisknutím definované kombinace kláves se otevře formulář (obr 3.6), do kterého jsou načteny vstupní hodnoty podle typu exportovaných dat. Uživatel je poté veden k nastavení formuláře podle předdefinovaných pravidel.



Obrázek 3.6: Formulář průvodce kontingenčním grafem exportovaných dat.

Po kliknutí na tlačítko Generovat je spuštěna procedura `ChartWiz()`, jejímž úkolem je samotné vygenerování kontingenční tabulky a vytvoření grafu z této tabulky (Výpis 3.25). Důsledkem tohoto postupu je fakt, že při odstranění kontingenční tabulky dojde ke zrušení závislosti grafu na tabulce a všechny aktuálně zobrazované údaje se uloží na pevno do grafu. (užitečné pro uchování aktuálních výsledků).

Kontingenční grafy umožňují velmi rychle a pohodlně nastavit vzhled a formát grafu. Poskytují prakticky stejné možnosti jako agregační, filtrační a další varianty dotazu `SELECT` jazyka SQL. Umožňují sumarizovat data způsobem, kterého není možné jinak dosáhnout [13].

Doplněk v praxi zjednodušuje inicializaci vstupních parametrů zejména pro tyto druhy kontingenčních grafů:

- Graf všech prostojů s hodnotami součtu jejich trvání ve sledovaném období.
- Graf technických prostojů dělených nebo seskupených podle libovolného kritéria (např. směna, pracoviště).
- Graf součtů trvání jednotlivých prostojů podle linek. Hodnoty seskupeny podle měsíce či roku.
- Graf součtu trvání logistických prostojů. Seřazený a seskupený podle libovolného kritéria.
- Všechny uvedené výstupy lze seřadit od vyšších hodnot k nižším a poté omezit jejich počet, čímž velmi jednoduše docílíme grafického znázornění paretovy analýzy. Tato analýza vychází ze statistického předpokladu, že většina nepříznivých jevů je způsobena menšinou vstupních podmínek.

Doplněk je přizpůsobený také k použití na exportovaných výstupech záznamů o produktivitě, takže lze získat např. grafickou reprezentaci vyrobeného počtu kusů určitého typu výrobku v různých časových obdobích.

4.Závěr

Zpracováním teoretické a praktické části došlo ke splnění cílů vytýčených v úvodu práce. V teoretické části byl představen fenomén štíhlé výroby a stručně popsány důvody, proč je tento způsob organizace výrobního procesu v široké míře zaváděn v dnešních výrobních podnicích. Byl zdůrazněn význam zpětné vazby v tak komplikovaném systému, jakým výrobní proces bezesporu je a následně byly vyjmenovány měřitelné ukazatele, které jsou obvykle za tímto účelem sledovány výrobním managementem. V závěru teoretické části byly zmíněny stávající softwarové nástroje umožňující realizovat tuto zpětnou vazbu.

V praktické části byla nejprve odůvodněna potřeba softwarové aplikace, která by umožnila efektivně provádět analýzy popsané v teoretické části. Poté byly popsány postupy a metody tvorby databázové aplikace a doplňku programu MS Excel. Metodický popis tvorby aplikace musel být omezen pouze na její reprezentativní část, týkající se pouze jedné z devatenácti databázových tabulek. Důvodem tohoto omezení jsou obavy vedení firmy z prozrazení firemního tajemství veřejnosti a hrozby ztráty jejího postavení vůči konkurenci. Rozhodl jsem se tedy pro komplexnější popis nejzajímavějších částí architektury MVC a jejich vzájemné interakce. Vzhledem k uvedenému omezení vznikl popis tvorby univerzální databázové aplikace s několika specifickými rysy, jakými jsou např. výpočet produktivity práce pomocí konstrukcí LINQ, vkládání grafu do webové stránky či export dat do tabulkového procesoru a jejich následné zpracování speciálním doplňkem.

Téma by si jistě zasloužilo širší zpracování především v otázce rozmanitosti typů vyhodnocovaných údajů. Přesto je možno považovat cíle praktické části bakalářské práce větším dílem za splněné, neboť v jejím rámci vznikla funkční softwarová aplikace, poskytující výrobnímu managementu požadované údaje pro efektivní řízení výrobního procesu.

4.1. Souhrn bibliografických citací

[1] SHINGO, Shigeo. *A Study of the Toyota Production System: From an Industrial Engineering Viewpoint*. [USA] : Productivity Press, 1989. 304 s. ISBN 978-0915299171.

[2] WOMACK, James P. *The Machine That Changed the World : The Story of Lean Production*. [USA] : Harper Perennial, 1991. 323 s. ISBN 978-0060974176.

[3] JIRÁNEK, J. Štíhlá výroba. Praha: Grada Publishing, 1998, 224 s

[4] OHNO, Taiichi. *Toyota Production System: Beyond Large-Scale Production*. Tokyo : Productivity Press, 88. 152 s.

[5] TOMEK, Gustav. *Nákupní marketing*. Praha : Grada, 1996. 176 s.

[6] KHAN, Jamal. *Gaining Productivity*. [USA] : Arawak publications, 2008. 539 s. ISBN 978-9768189936.

[7] *OEE for Operators: Overall Equipment Effectiveness*. [USA] : Productivity Press, 1999. 96 s.

[8] *Výrobní systém BOSCH, Metodická příručka pro spolupracovníky*. České Budějovice : Tiskárna Jaroslav Karmášek, 2007. 59 s.

[9] KLETTI, Jürgen. *Manufacturing Execution System - MES*. [USA] : Springer, 2007. 284 s. ISBN 978-3642080647.

[10] MOLINARO, Anthony. *SQL Kuchařka programátora*. Brno : Computer Press, 2009. 574 s. ISBN 978-80-251-2617-2.

[11] CONERY, Rob, et al. *Professional ASP.NET MVC 1.0*. [USA] : Wrox, 2009. 456 s.

[12] PALERMO, Jeffrey, et al. *ASP.NET MVC 2 in Action*. [USA] : Manning Publications, 2010. 450 s. ISBN 978-1935182795.

[13] WALKENBACH, John. *Microsoft Excel 2003 programování ve VBA*. [USA] : Computer Press, 2006. 865 s.

[14] *Produkcni systemy* [online]. 2006 [cit. 2011-04-20]. Produkcni_systemy. Dostupné z WWW: <http://www.kip.zcu.cz/kursy/svt/eb/prum_eng/produkcni_systemy.html>.

[15] *Lean manufacturing* [online]. 2009 [cit. 2011-04-20]. Lean manufacturing. Dostupné z WWW: <http://www.leanmanufacture.net>.

[16] *TeachMeFInance.com* [online]. 2005 [cit. 2011-04-20]. Definition of productivity. Z WWW: <http://www.teachmefinance.com/Financial_Terms/productivity.htmlroducts/aspnet-mvc.aspx>

[17] *Telerik* [online]. 2011 [cit. 2011-04-20]. ASP.NET MVC Components. Dostupné z WWW: <http://www.telerik.com/products/aspnet-mvc.aspx>

[18] *JSON* [online]. 2010 [cit. 2011-04-20]. Úvod do JSON. Dostupné z WWW: <http://json.org/json-cz.htm>.

[19] *OpenOffice.org* [online]. 2011-04-08 [cit. 2011-04-20]. OpenOffice.org. Dostupné z WWW: <http://www.openoffice.org/>.

[20] *MPDV Microlab GmbH* [online]. 2010 [cit. 2011-04-20]. MPDV Microlab GmbH. Dostupné z WWW: <http://www2.mpdv.de/en/>.

[21] *Gemba Solutions* [online]. 2011 [cit. 2011-04-20]. Gemba Solutions. Dostupné z WWW: <http://www.gembasolutions.co.uk>.

[22] *Provideam* [online]. 2011 [cit. 2011-04-20]. Provideam. Dostupné z WWW: <http://www.provideam.com/>.

4.2. Seznam obrázků

4.3. Slovníček méně známých pojmů a zkratek:

AJAX - Asynchronous JavaScript and XML, technologie vývoje webových aplikací, která umožňuje aktualizaci webové stránky bez nutnosti obnovy jejího kompletního obsahu

Autonomation – Koncept automatizace, vyvinutý ve firmě Toyota, který je založen na vizualizaci, Poka-Yoke a dalších principech, umožňujících provádět činnost s minimální chybovostí.

CRUD – Create, Read, Update, Delete – Souhrh čtyř funkcí zajišťujících uživatelskou manipulaci s daty databázové aplikace

ERP – Enterprise Resource Planning, souhrnné označení aplikací pro správu a plánování podnikových zdrojů.

Grid – Mřížka, u softwarových aplikací označuje vizuální prvek, zobrazující sadu databázových údajů.

GUI - Graphical User Interface, grafické rozhraní umožňující interakci softwarové aplikace s uživatelem

Jidoka – viz **autonomation**

JIT – Eliminace časových ztrát

JSON – platformově nezávislý formát zápisu dat organizovaných v polích, určený pro jejich přenos

Kaizen - systematické zkoumání procesu tvorby hodnot zaměřené na trvalé zlepšovaní produktů, postupů a pracovních podmínek.

LINQ - Language Integrated Query, integrovaný jazyk pro dotazování do SQL, XML a jiných druhů databází. Dostupný v C# od verze 3.0

MES – Manufacturing Execution System, počítačový systém, umožňující evidenci a analýzu výrobních a souvisejících procesů

Muda – Japonský termín pro ztráty, plýtvání, neefektivní činnost

MVC – Model View Controller, softwarová architektura, rozdělující datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní.

OEE - Overall equipment effectiveness, souhrn ukazatelů pro účinné sledování výrobního procesu

Poka-Yoke – japonský termín pro zabránění vzniku neúmyslných chyb

TPM -Total productive maintenance – techniky kontinuální údržby strojního vybavení

4.4. Přílohy

Výpis 3.1, definice typického pohledu. Pohled pro výpis všech linek s dostupnými detaily. [SQL]

```
SELECT          dbo.spec_Lines.LineId, dbo.spec_Lines.LineTxt, dbo.adm_Depts.DeptTxt, dbo.adm_Plants.PlantTxt,
dbo.adm_Areas.AreaTxt, dbo.adm_Groups.GroupTxt,
                dbo.adm_LogDepts.LogDeptTxt, dbo.spec_Lines.Ref_VT
FROM        dbo.spec_Lines LEFT OUTER JOIN
                dbo.adm_Depts ON dbo.spec_Lines.DeptId = dbo.adm_Depts.DeptId LEFT OUTER JOIN
                dbo.adm_Plants ON dbo.spec_Lines.PlantId = dbo.adm_Plants.PlantId LEFT OUTER JOIN
                dbo.adm_Areas ON dbo.spec_Lines.AreaId = dbo.adm_Areas.AreaId LEFT OUTER JOIN
                dbo.adm_Groups ON dbo.spec_Lines.GroupId = dbo.adm_Groups.GroupId LEFT OUTER JOIN
                dbo.adm_LogDepts ON dbo.spec_Lines.LogDeptId = dbo.adm_LogDepts.LogDeptId
```

Výpis 3.2a: Metoda `AddLine()` ve třídě `VyrobaRepository` pro přidání výrobní linky do tabulky *spec_Line*. [C#]

```
public void AddLine(spec_Line line)
{
    db.spec_Lines.InsertOnSubmit(line);
}
```

Výpis 3.2b: Metoda `FindAllLines_vw()` pro vyhledání všech linek v pohledu *vw_spec_lines_general*.[C#]

```
public IQueryable<vw_spec_lines_general> FindAllLines_vw()
{
    return db.vw_spec_lines_generals;
}
```

Výpis 3.2c: Metoda `GetLine()` pro vyhledání konkrétní linky v tabulce *spec_Line*. [C#]

```
public spec_Line GetLine(int id)
{
    return db.spec_Lines.SingleOrDefault(d => d.LineId == id);
}
```

Výpis 3.2d: Metoda `DelLine()` pro odstranění výrobní linky z tabulky *spec_Line*. [C#]

```
public void DelLine(spec_Line linka)
{
    db.spec_Lines.DeleteOnSubmit(linka);
}
```

Výpis 3.2e: Metoda `Save()` pro uložení všech provedených změn v databázi. [C#]

```
public void Save()
{
    db.SubmitChanges();
}
```

Výpis 3.3: Třída ControllerHelpers. [C#]

```
    public static class ControllerHelpers
  {
    public      static      void      AddRuleViolations(this      ModelStateDictionary      modelState,
IEnumerable<RuleViolation> errors)
    {
      foreach (RuleViolation issue in errors)
      {
        modelState.AddModelError(issue.PropertyName, issue.ErrorMessage);
      }
    }
  }
```

Výpis 3.4: Metoda Export2CSV pro vytvoření CSV souboru. [C#]

```
        public static void Export2CSV<T>(this IQueryable<T> list, string filename, string[] exclude,
string separator)
        {
            StringWriter sw = new StringWriter();
            bool headerPrinted = false;
            foreach (var obj in list)
            {
                Type type = obj.GetType();
                PropertyInfo[] properties = type.GetProperties();
                string s = String.Empty;
                if (!headerPrinted)
                {
                    foreach (PropertyInfo propertyInfo in properties) //název sloupce z databáze
                    {
                        if (exclude.Count(x => x.Contains(propertyInfo.Name)) > 0)
                            continue;
                        if (s.Length > 0)
                            s = String.Join(separator, new string[] { s, propertyInfo.Name });
                        else
                            s = propertyInfo.Name;
                    }
                    sw.WriteLine(s);
                    headerPrinted = true;
                }
                s = String.Empty;
                for (int idx = 0; idx < properties.Length; idx++)
                {
                    if (exclude.Count(x => x.Contains(properties[idx].Name)) > 0)
                        continue;
                    var value = properties[idx].GetValue(obj, null);
                    var formattedValue = value == null ? String.Empty : value.ToString();
                    DateTime datetime;
                    if (value != null)
                    {
                        if (DateTime.TryParse(formattedValue, out datetime))
                            formattedValue = datetime.ToShortDateString();
                        if (value.GetType() == typeof(string))
                        {
                            formattedValue = SysUtil.StringEncodingConvert(formattedValue, "iso-8859-
5", "iso-8859-2");//odstranit diakritiku
                            formattedValue = "\"" + formattedValue + "\"";
```

```
                    }
                }
                if (s.Length > 0)
                    s = String.Join(separator, new string[] { s, formattedValue });
                else
                    s = formattedValue;
            }
            sw.WriteLine(s);
        }
        HttpResponse response = System.Web.HttpContext.Current.Response;
        response.AddHeader("Content-Disposition", "attachment; filename=" + filename);
        response.ContentType = "application/ms-excel";
        response.ContentEncoding = System.Text.Encoding.GetEncoding("utf-8");
        response.Write(sw);
        response.End();
    }
```

Výpis 3.5: POST požadavek obsahující data vyplněného formuláře editace výrobní linky s přednastavenými hodnotami. Tento požadavek je skrytý v odpovědi HTTP protokolu. [protokol HTTP]

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 79
DeptId=1&LineTxt=JmenoLinky+++++++++++++++&PlantId=1&AreaId=1&GroupId=1&Ref_VT=
```

Výpis 3.6: Inicializace – zpřístupnění datové vrstvy uvnitř controlleru. [C#]

```
        VyrobaDataContext model = new VyrobaDataContext();
        VyrobaRepository repository = new VyrobaRepository();
```

Výpis 3.7: Akční metoda Index(), controller LineController. [C#]

```
        public ActionResult Index()
        {
            IQueryable<vw_spec_lines_general> lines = repository.FindAllLines_vw();
            return View(lines);
        }
```

Výpis 3.8: Fragment šablony *Index.aspx* zobrazující seznam linek. Inicializace `Grid`. [XML / C#]

```
    <%=  Html.Telerik().Grid(Model)
        .Name("Grid")
        .DataBinding(dataBinding => dataBinding
          .Ajax()
            .Select("_AjaxBinding", "Line", new {id = (string)ViewData["id"]})
            )
        .ClientEvents(events => events.OnRowSelected("onRowSelect"))
        .RowAction(row =>
        {
            row.Selected = row.DataItem.LineId.Equals(ViewData["id"]);
        })
        .DataKeys(k => k.Add(o => o.LineId))
        .Columns(c =>
        {
            c.Bound(o => o.LineId).ReadOnly().Title("ID");
            c.Bound(o => o.LineTxt).Title("Linka");
            c.Bound(o => o.DeptTxt).Title("Oddělení");
            c.Bound(o => o.PlantTxt).Title("Závod");
            c.Bound(o => o.AreaTxt).Title("Oblast");
            c.Bound(o => o.LogDeptTxt).Title("Log odd.");
```

```
            c.Bound(o => o.Ref_VT);
        })
        .Resizable(resizing => resizing.Columns(true))
        .Scrollable(s => s.Height(500))
        .Pageable(p => p.PageSize(100))
        .Sortable()
        .Selectable()
        .Filterable()
    %>
```

Výpis 3.9: funkce obsluhující kliknutí myši na položce seznamu a otevírající nové okno prohlížeče. [XML / JavaScript]

```
    <script type="text/javascript">
        var srow;
        function onRowSelect(e) {
            $(e.row).addClass('t-state-selected');
            srow = e.row.cells[0].innerHTML.toString();
        }
        function Popup(pageURL, title, w, h) {
            var left = (screen.width / 2) - (w / 2) - 100;
            var top = (screen.height / 2) - (h / 2) - 100;
            var tmp = (pageURL.substring(pageURL.length - 6)).toLowerCase();
            var s;
            if (tmp == "create") s = ''; else s = '/' + srow;
            if (s == '/undefined' || (srow == '' && tmp != "create")) return;
            var targetWin = window.open(pageURL + s, title, 'toolbar=no, location=no, directories=no,
status=no, menubar=no, scrollbars=yes, resizable=no, copyhistory=no, width=' + w + ', height=' + h +
', top=' + top + ', left=' + left);
        }
    </script>
```

Výpis 3.10: Akční metoda Details, zobrazující detaily výrobní linky. Některá pole jsou vynechána. [C#]

```
        public ActionResult Details(int id)
        {
            spec_Line line = repository.GetLine(id);
            if (line == null)
                { ViewData["msg"] = "Ojekt nenalezen"; return View("RedMessage"); }
            else
            {
                ViewData["DeptId"] = new SelectList(repository.FindAllDepts(), "DeptId", "DeptTxt",
repository.GetLine(id).DeptId);
                ViewData["PlantId"]    =    new    SelectList(repository.FindAllPlants(),    "PlantId",
"PlantTxt", repository.GetLine(id).PlantId);
                ViewData["AreaId"] = new SelectList(repository.FindAllAreas(), "AreaId",
.
.
.
                return View("Details", line);
            }
        }
```

Výpis 3.11: Fragment šablony *Details.aspx* s položkou `Model.IsValid`. [XML]

```
    <fieldset style="background-color: #F7FBFF">
        <legend>Položky záznamu</legend>
        <div class="row"><span class="label">Platné:
        </span><span class="formw"><%: Model.IsValid %></span></div>
.
.
.
    </fieldset>
```

Výpis 3.12: Fragment akční metody `Edit(int id)`. [C#]

```
            }
            return View("Edit",line);
        }
```

.Výpis 3.13: Hlavní část těla šablon *Edit.aspx* a *Create.aspx*. [XML / C#]

```
            <% Html.RenderPartial("EditForm"); %>
```

Výpis 3.14: Fragment typického formuláře v částečné šabloně *Editform.ascx*. [XML / C#]

```
    <% using (Html.BeginForm()) {%>
        <%: Html.ValidationSummary(true) %>

        <fieldset style="background-color: #F7FBFF">
            <legend>Položky záznamu</legend>
            <div class="row"><span class="label">
                Oddělení
            </span><span class="formw">
                <%: Html.DropDownList("DeptId")%>
                <%: Html.ValidationMessageFor(model => model.DeptId) %>
            </span></div>
            <div class="row"><span class="label">
                Linka
            </span><span class="formw">
                <%: Html.TextBoxFor(model => model.LineTxt) %>
                <%: Html.ValidationMessageFor(model => model.LineTxt) %>
            </span></div>
.
.
.
         </fieldset>
            <input type="submit" value="Uložit" />
    <% } %>
```

Výpis 3.15: V případě úspěšně provedené akce se PopUp okno zmenší, zobrazí „zelenou hlášku“ a po několika sekundách zmizí. Pokud akce skončí chybou, je analogicky zobrazena „červená hláška“. [XML / JavaScript]

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <script type ="text/javascript">
        function onload() {
            window.resizeTo(400, 200);
            setTimeout("autoclose()", 3000);
        }
        function autoclose() {
            window.close();
        }
    </script>
    <h2 class = "greenmessage"><% =ViewData["msg"] %></h2>
</asp:Content>
```

Výpis 3.16: POST metoda `Edit()` pro uložení změněné linky. Některá pole jsou vynechána. [C#]

```
        [HttpPost]
        public ActionResult Edit(int id, FormCollection collection)
        {
            spec_Line line = repository.GetLine(id);
            try
            {
                UpdateModel(line);
                repository.Save();
                ViewData["msg"] = "Změny uloženy do databáze";
                return View("GreenMessage");
            }
            catch
            {
                foreach (var issue in line.GetRuleViolations())
                {
                    ModelState.AddModelError(issue.PropertyName, issue.ErrorMessage);
                }
                ViewData["DeptId"] = new SelectList(repository.FindAllDepts(), "DeptId",
"DeptTxt", repository.GetLine(id).DeptId);
.
.
                return View("edit", line);
            }
        }
```

Výpis 3.17: POST verze metody `Delete()`. [C#]

```
        [HttpPost]
        public ActionResult Delete(int id, string confirmButton)
        {
            spec_Line line = repository.GetLine(id);
            if (line == null)
                { ViewData["msg"] = "Ojekt nenalezen"; return View("RedMessage"); }
            try
            {
                repository.DelLine(line);
                repository.Save();
                { ViewData["msg"] = "Objekt smazán"; return View("GreenMessage"); }
            }
            catch
```

```
            {
                ViewData["msg"] = "Objekt nelze smazat, protože je referencován jinde";
                return View("RedMessage");
            }
        }
```

Výpis 3.18: AJAX metoda pro *Telerik Grid*. [C#]

```
        [GridAction]
        public ActionResult _AjaxBinding()
        {
            IQueryable<vw_spec_lines_general> lines = repository.FindAllLines_vw();
            return View(new GridModel(lines));
        }
```

Výpis 3.19: Obsah objektu JSON, vráceného metodou `_AjaxBinding()`. [HTTP protokol]

```
{"data":[{"LineId":1,"LineTxt":"L1","DeptTxt":"Testovaci_oddeleni","PlantTxt":"GS/R
BCB","AreaTxt":"oblast01","GroupTxt":"Group1","LogDeptTxt":null,"Ref_VT":null},…
```

Výpis 3.20: Výpočet produktivity pomocí LINQ dotazu v metodě *cProd()*. [C#]

```
        [NonAction]
        private double cProd(DateTime dat, int? dpt, int? tm)
        {
            double dpt_dtps = repository.GetDept((int)dpt).DispShiftTime / 60.0;
            double pcs_tm = (double)(from d0 in model.pt_ProductionEntries
                                     where (d0.EventDate.Day <= dat.Day)
                                     && (d0.EventDate.Month == dat.Month)
                                     && (d0.EventDate.Year == dat.Year)
                                     && (d0.TeamId == tm)
                                     select  d0.OkPieces  /  (/*  Algoritmus  vypoctu  je  dusevnim
vlastnictvim zadavatele* /).Sum();
            double wrk_tm = (double)(from d0 in model.pt_ProductionEntries
                                     where (d0.EventDate.Day <= dat.Day)
                                     && (d0.EventDate.Month == dat.Month)
                                     && (d0.EventDate.Year == dat.Year)
                                     && (d0.TeamId == tm)
                                     select (from d1 in model.spec_NMs
                                             where d0.EventDate == d1.Date
                                                 && d0.ShiftId == d1.ShiftId
                                                 && d1.TeamId == tm
                                             select d1.NmVal).Sum()).Distinct().Sum();
            return (double)(/* Algoritmus vypoctu je dusevnim vlastnictvim zadavatele* /);
        }
```

Výpis 3.21: Vložení grafu do HTML stránky v šabloně *ProdChart.aspx*. [XML / C#]

```
        <div id="chart" style="background-color: #E7EBFE">
            <p>
                <img src="/Chart/CreateChart?chartType=
<%=System.Web.UI.DataVisualization.Charting.SeriesChartType.Line %>" alt="" />
            </p>
        </div>
```

Výpis 3.22: Generování obsahu grafu produktivity. [C#]

```
        [NonAction]
        public Series CreateSeries(spec_Team team)
        {
            Series seriesDetail = new Series();
            seriesDetail.Name = team.TeamTxt;
.
.
            DataPoint point;
            DateTime counter = chartsettings._from;
            while (counter <= chartsettings._to)
            {
                point = new DataPoint();
                point.AxisLabel = counter.Date.ToString();
                point.YValues = new double[] { double.Parse(cProd(counter, team.DeptId,
team.TeamId).ToString()) }; // dosazování vypočítaných hodnot produktivity
                seriesDetail.Points.Add(point);
                counter = counter.AddDays(1);
            }
            seriesDetail.ChartArea = "Result Chart";

            return seriesDetail;
        }
```

Výpis 3.23: Generování, uložení a odeslání obrázku s grafem. [C#]

```
        public FileResult CreateChart()
        {
            Chart chart = new Chart();
.
.
            chart.Titles.Add(CreateTitle("Produktivita"));
            chart.Legends.Add(CreateLegend());
            IEnumerable<spec_Team> teams = (from d in model.spec_Teams
                                            where d.DeptId == chartsettings._dpt
                                            select d); // seznam všech týmů v oddělení
            foreach (spec_Team team in teams)
            {
                chart.Series.Add(CreateSeries(team));
            }
            chart.ChartAreas.Add(CreateChartArea());
            MemoryStream ms = new MemoryStream();
            chart.SaveImage(ms);
            return File(ms.GetBuffer(), @"image/png"); // vlož obrázek do stránky
        }
```

Výpis 3.24: POST verze metody `CSV()`, která je zodpovědná za přípravu dat pro export do formátu CSV. [C#]

```
    [HttpPost]
    public void CSV(ChartSettings cs)
    {
        chartsettings = cs;
        ViewData["DeptId"] = new SelectList(repository.FindAllDepts(), "DeptId", "DeptTxt", cs._dpt);
        IQueryable<vw_pt_downtimeentries_main> CsvOutput = from d in model.vw_pt_downtimeentries_mains
                                                           where (d.EventDate >= cs._from)
                                                                && (d.EventDate <= cs._to)
                                                                && (d.DeptId == cs._dpt)
                                                           orderby (d.EventDate)
```

```
                                                    select d;
        string[] excl = new string[] {/* Zde jsou uvedeny výjimky sloupců, které chceme vynechat */};
        Helpers.CSV.ExportToCSV<vw_pt_downtimeentries_main>(CsvOutput, "output.csv", excl, ";");
        return;
    }
```

Výpis 3.25: Programový kód modulu doplňku generujícího kontingenční tabulku a graf. Kód se na objekt formuláře odkazuje jeho jménem „PivotSettings“. Jednotlivé ovládací prvky jsou pojmenovány písmeny identifikujícími jejich typ (tb = TextBox, cb = CheckBox, ddl = DropDownList) a číslem značícím pořadí ve formuláři (viz obr. 3.6) [VBA]

```
Option Explicit
Sub ChartWiz()
    On Error Resume Next
    Dim Name As String
    PivotSettings.Show
    If PivotSettings.doit.Value = True Then'skryté - Pokud uživatel klikl "Generovat", hodnota je True
        Name = PivotSettings.tb2.Text
        ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:= _
            Range("A1").CurrentRegion.Address).CreatePivotTable TableDestination:="", TableName:= _
            Name, DefaultVersion:=xlPivotTableVersion10
        ActiveSheet.PivotTableWizard TableDestination:=ActiveSheet.Cells(3, 1)
        If PivotSettings.ddl5.ListIndex > 0 Then
            ActiveSheet.PivotTables(Name).AddFields RowFields:= _
                PivotSettings.ddl2.Value, ColumnFields:=PivotSettings.ddl6.Value
        Else
            ActiveSheet.PivotTables(Name).AddFields RowFields:=PivotSettings.ddl2.Value
        End If
        With ActiveSheet.PivotTables(Name).PivotFields(PivotSettings.ddl1.Value)
            .Orientation = xlDataField
            .Caption = "Hodnoty"
            .Function = xlSum
        End With
        With PivotSettings
            If (.cb1.Visible = True) And (.cb1.Value Or .cb2.Value Or .cb3.Value) Then _
                ActiveSheet.PivotTables(Name).PivotFields(.ddl2.Value).LabelRange.Group _
                    Start:=True, End:=True, By:=1, _
                    Periods:=Array(False, False, False, False, .cb1.Value, .cb2.Value, .cb3.Value)
' podmínka řídí seskupení podle měsíce, čtvrtletí, roku datumového pole v ddl2
            If .ddl3.ListIndex > 0 Then
                ActiveSheet.PivotTables(Name).PivotFields(.ddl2.Value) _
                    .AutoSort .ddl3.ListIndex, .ddl4.Value
                If .ddl7.ListIndex > 0 Then ActiveSheet.PivotTables(Name).PivotFields(.ddl2.Value) _
                    .AutoShow xlAutomatic, xlTop, .sb1.Value, .ddl4.Value
            End If
        End With
        ActiveSheet.Cells.Select
            ActiveSheet.Cells(3, 1).Select
        Charts.Add: ActiveChart.Location Where:=xlLocationAsNewSheet
        Selection = Nothing
    End If
End Sub
```