Jan Zmatlík

Semestrální práce předmětu Manažerská psychologie

Rozbor knihy Brian Tracy: Snězte tu žábu

22.1.2021

Proč zrovna Sněz tu žábu?

Je pro to hned několik důvodů. V první řadě mě zaujaly recenze, dle kterých se kniha v první řadě věnuje efektivitě, time managementu, plánování a dalšími užitečnými tématy, na něž je v dnešním světě výzkumu a vývoje kladen velký důraz, především kvůli maximalizaci zisku. Připusťme si, nikdo vás neplatí čistě z dobroty srdce. Musí za vámi být vidět výsledky.

Dalším důvodem byla diskuze s kolegy ze zaměstnání i spolužáky ze školy. Nu a v neposlední řadě je důvod čistě pragmatický – v omezeném časovém okně to byla jedna z mála knih, kterou se mi podařilo zajistit.

Předesílám že později diskutované názvy kapitol a některých termínů nemusí dle knihy stoprocentně odpovídat – kniha v původním jazyce, tedy angličtině, se mi popravdě sháněla mnohem snáz.

Čím se kniha zabývá

Už podle předmluvy je zřejmé, že jde o sadu praktických doporučení, které mají za cíl jediné: naučit se s omezenou časovou investicí zacházet tak dobře a efektivně, jak jen lze. V podstatě jde o snahu maximalizovat výsledek. Co se týče konzumace žab, je zřejmé že je to metafora. Žábou si představme nějakou úlohu, kterou musíte vyřešit. Je však potřeba si předtím vybrat tu správnou, jejíž splnění přinese největší užitek, pokud je příliš velká pro jednoho tak jak ji vykuchat, rozsekat, rozmělnit, naporcovat, rozemlít na kaši a seskládat do stravitelných kusů do kterých se postupně pustíme. Mám samozřejmě na mysli tu úlohu, nikoliv nešťastného plaza.

Kniha je rozdělena do 21 kapitol a vzhledem k tomu že ve firmě, kde jsem zaměstnán, je velká podpora těchto technik – a sám jsem na několika školení time managementu byl – se pokusím kapitoly doplnit konkrétními pozorováními a vlastními zkušenostmi.

Pusťme se do toho.

Kapitola 1 – cíle a priority.

V první řadě autor doporučuje jasně si určit priority, cíle, čeho chceme v příštích několika let dosáhnout. Poté teprve je možno určit kroky, které k těmto cílům povedou. V podstatě to vede na problematiku roadmap, jen s tím rozdílem že nesestavujeme roadmapu konkrétního produktu ale naší vlastní.

Dále doporučuje si ty cíle skutečně sepsat na papír, protože pokud je člověk neustále bude mít na očích, neustále si bude připomínat čeho že vlastně chce dosáhnout. K tomu je samozřejmě potřeba přidat termíny – nutno však realistické, a tady dle mých vlastních zkušeností velice často je problém. Obvykle lidé neumí odhadovat, nebo neví jak na to. Autor se to pokouší alespoň nastřelit pokusem o sepsání všech činností, které je potřeba k uskutečnění těchto cílů a seřadit si je podle návazností a důležitostí. Tady z vlastních zkušeností vím že poměrně dobře funguje Kanban, jehož základy položila firma Toyota a kde původně sloužil k organizaci štíhlé výroby. Základy SCRUM a ostatních frameworků agilního vývoje softwaru fungují na úplně stejném principu, je potřeba si důkladně rozmyslet a naplánovat co bude potřeba udělat pro hladký průběh projektu, je potřeba vědět v jakém stavu daná úloha je, kdo se tím zabývá – a v případě zvyšující se priority lze vizuálně posunout na tabuli lísteček nahoru.

Vizualizace je v tomto ohledu dosti podceňovaná, nicméně ačkoliv to na první pohled může odporovat logice, funguje to.

Nu a pak je potřeba se do toho pustit a vytrvat.

Kapitola 2 – naplánuj si den s předstihem

Tady se ukazují neskutečné rozdíly mezi manažerem či prodejcem versus vývojář. Jestliže průměrný čas exekuce úlohy manažera či prodejce mohou být hodiny až dny, u vývojáře to jsou obvykle dny až týdny – u některých úloh dokonce měsíce. Ten task už nejde dále rozdrobit, zvlášť pokud je potřeba vymyslet např. architekturu softwaru – tam je potřeba mít neustále v hlavě celý obraz a dokud to není hotovo, tak to jde velice, velice těžko uspíšit – a spíše nejde než jde. Zažil jsem manažerská rozhodnutí, která právě pokusem uspíšit návrh pomocí oddelegování části úloh na druhého architekta skončila ještě větším zpožděním dodávky. Než architekt A vysvětlil B oč jde, koncept jaký má vymyšlený, B řešení rozporoval, A musel vymýšlet obhajobu...nejenže se to protáhlo řádově o týdny, ještě tam vznikly nedomyšlené části – které se pak projevily jako technický dluh, který ztěžoval další práci.

Závěrem – nerozporuji autorovo doporučení si den naplánovat s předstihem aby se pak v tom dni neztrácel čas rozmýšlením co dál. Jen doplňuji, že pro některé aplikace je den příliš krátký úsek a mnohem vhodnější je např. inkrement dvou týdnů, v agilních metodách vývoje mnohdy označovaný jako sprint. Dlouhodobé plánování, e.g. měsíční, kvartální, dává smysl zachovat – z backlogu připravených úloh, které jsou nějakým způsobem rozmyšlené, se pak úlohy tahají do dalšího sprintu. Opět, moje zkušenosti si s autorovými doporučeními nerozporují – jen se to nesmí brát jako dogma ale je potřeba to přizpůsobit okolnostem.

Kapitola 3 – 80/20 alias Paretovo pravidlo

Paretovo pravidlo je mě důvěrně známý koncept. Dle mých zkušeností opravdu nemusí být produkt/komponenta/API/jakýkoliv kus kódu vyleštěný do zrcadla a skutečně nějakých 20% dobře napsaných funkcí je nejpoužívanějších a nejvíce recyklovaných.

Nicméně, pokud se Paretovo pravidlo použije špatně – jakýmkoliv způsobem – je zde prostor pro vznik tzv. technického dluhu, něčeho co se v marketingu/managementu jen obtížně chápe. A ono to skutečně má význam dluhu – včetně úroků.

Pokud píšu nějakou komponentu, o které a priori vím že bude jedna z nejvyužívanějších, jakékoliv nedostatky mi mohou být vytýkány, velice pravděpodobně se na ně přijde vcelku brzy, ale pokud nebudou bránit užití, dojde k nasazení komponenty tak, jak je. Měsíce, roky to může bez problému fungovat. Pak přijde nový požadavek na komponentu a tady je kámen úrazu. Pokud jsem skutečně správně identifikoval těch nejužitečnějších 20% a trefil jsem se, ten kdo to bude upravovat to bude mít snadné. Ale pokud ne, to nejmenší co hrozí je oprava mých nedostatků. Čas, který jsem ušetřil hned, někdo bude muset splatit později. I s úroky. A klidně to mohu být já za dva měsíce kdy si z toho nepamatuji vůbec nic a začínám znovu včetně shánění informací.

Opět, nezatracuji Paretovo pravidlo, má svůj význam a spousta věcí je za 20% času ‚good enough for rock‘n‘roll‘ – jen tvrdím: opatrně s tím.

Kapitola 4 – následky

Smysl této kapitoly je v podstatě naučit se prioritizovat a vybírat si tak úlohy s nejvyšší přidanou hodnotou. Každou úlohu si můžeme rozdělit na tabuli podle **priority** a **urgence**. Opět zmiňuji Kanban, nicméně jsou i jiné metody jak tyto dvě metriky vizualizovat.

Pointa je, že úlohy s vysokou prioritou by měly mít nejzávažnější následky pokud se nesplní včas. Jak ubíhá čas, prioritním úlohám se zvyšuje **urgence**. Ideálně by člověk měl pracovat na úlohách prioritních ale nikoliv urgentních, pak je provede pořádně – a v žádném případě na několika úlohách naráz. To je přímá cesta do pekel. Prioritní a současně urgentní úlohy by se měly vyskytovat jen v případě průšvihů – protože vhodným plánováním a dekompozicí problému lze obvykle urgenci snížit, případně rozprostřít na více lidí.

Urgentní, ale neprioritní úlohou si můžeme představit např. tuctové čtení firemní korespondence – je to první věc kterou provedu po příchodu do kanceláře ale tím urgence končí – pokud však z korespondence není zřejmé že se vyskytla úloha s vysokou prioritou a urgencí současně.

K této problematice mám ještě jeden postřeh, a to je trvalý stav práce na urgentních úlohách a s tím související neustálé přepínání úloh či přerušení. Práce vývojáře je principiálně práce s informacemi. Vývojář musí v hlavě držet v jeden okamžik spoustu věcí a tato činnost je velice citlivá na jakékoliv vyrušení. Je vcelku běžné dva, tři dny bádat nad problémem, přemýšlet, hrát si s tím, zkoušet – a finální výsledek pak napsat během půl hodiny. Je velice těžké si v takovém režimu pod sebou zamlátit do skály skobu ke které se pak snadno jde vrátit. Jde to samozřejmě, ale rozhodně to není snadná záležitost a i já se to po pěti letech vývoje v Pythonu stále učím. Je pak zřejmé, že úloha, která by zabrala několik dní čistého času, se při trvalém přerušování a přepínání úloh snadno může vyšplhat v celkovém čistém čase na týdny. Fakt, že tento pracovní režim je navíc z dlouhodobého hlediska extrémně vyčerpávající, snad ani není potřeba zdůrazňovat.

K myšlence ‚zamlátit si do skály skobu‘ ještě jeden postřeh. Je to činnost, s níž je spojena nějaká režie, jalová práce chcete-li. Drtivá většina úspěšných vývojářů pracuje s nějakým nástrojem na správu verzí. První instinkt by zněl dělat verze tak často jak lze, ovšem zaprvé to stojí čas, zadruhé pak je problematické ve verzích se zorientovat a i kdybych pak chtěl nějaké příspěvky do kódu sloučit dohromady aby se v tom zbytek týmu smysluplně vyznal, ten úklid zase stojí čas a nějakou režii. A najít kompromis mezi opatrným postupem který mi zaručí že se mám v případě problémů kam vrátit, oproti přímé práci a riziku vyrušení a nutnosti začít znovu je velice těžká úloha, která v podstatě nemá optimální řešení.

Suma sumárum: v každý okamžik mít přehled o stavu, prioritě a urgenci úloh, hlídat si že se nebude prioritním úlohám zvyšovat urgence a ze všech sil se snažit vyhnout práci na více úlohách naráz.

Kapitola 5 – kreativní prokrastinace

Tato kapitola se trochu prolíná s předchozí – opět se zabývá problematikou určování úloh s co nejvyšší přidanou hodnotou.

Tady si nicméně opět dovolím trochu rozporovat doporučení autora dělat pouze ty důležité úlohy a ty méně důležité odsunout/nedělat. Opět zmíním technický dluh. Je to něco, co se velice, velice těžko vysvětluje vedení že je potřeba likvidovat, sem tam se to ale přeci jen povede. Jde o to, že některé úlohy jsou obzvlášť vyčerpávající a kdybych měl kupříkladu stále jen odbavovat takové úlohy, nutnost služeb psychiatrické kliniky bude můj nejmenší problém.

Dle mých zkušeností je více než vhodné občas nějakou takovouto úlohu proložit něčím méně důležitým kvůli psychickému odpočinku. Stále je to přidaná hodnota, dá vám to šanci nabrat dech, navíc technický dluh je obvykle dobře popsán a je zjevné co je potřeba udělat a zejména proč. Psychická zátěž a hlavně urgence jsou obvykle relativně nízké.

Uvědomuji si, že zabíhám do tématu Paretova pravidla i prioritizace – je však důležité uvědomit si, že priority vývojářům obvykle určuje někdo, jehož role se nazývá vlastník produktu. Ten rozhoduje **CO** bude vývojář dělat. **JAK** by ideálně mělo být na vývojáři.

Kapitola 6 – Metoda ABCDE

Opět, jedna z metod která má za úkol pomoci s rozhodováním priorit/urgencí. Obsah kapitoly jsem v podstatě probral jako součást předchozích rozborů.

Kapitola 7 – Cíl na klíčové oblasti/dovednosti/události

Je důležité si uvědomit, že vás firma platí nejen za to že máte vhodnou sadu dovedností – ale zejména proto, že jich budete správně užívat, obzvlášť v situacích, které si o to přímo říkají, protože pokud ne, tak nepřinášíte za váš plat firmou očekávaný užitek. V každém případě musí být zřejmé a jasné mezi vámi a vaším nadřízeným: CO se od vás chce, zda jste schopni to dodat a zda to opravdu dodáváte. Ať už je to cokoliv.

Je naprosto v pořádku některou z klíčových dovedností pro pozici nezvládat na první dobrou – pokud to transparentně vykomunikujete a s nadřízeným si dohodnete např. že tyto činnosti necháte na někom kdo je zvládá a/nebo ještě lépe, dohodnete si výcvik v těchto dovednostech.

Kupříkladu já jsem začínal jako SW tester, teď jsem v podstatě plnohodnotný Python vývojář. V době nástupu jsem o Pythonu věděl akorát že existuje. Dnes zaškoluji další Pythonisty a jsem jeden ze zodpovědných za revizi nových příspěvků do kódu.

Autor zmiňuje, že je důležité neustále si klást otázku ‚kdybych si měl vybrat jednu dovednost, která zvýší mojí přidanou hodnotu, co by to bylo?‘ a s tímto se mohu plně ztotožnit.

Kapitola 8 – pravidlo 3 a dost

Nu dobrá, tohle bude trošku složitější. Autor tvrdí: jednoznačně si určete tři nejdůležitější povinnosti a pak se soustřeďte na tyhle protože právě ty dělají 90% vaší přidané hodnoty. Zbytek musí jít na někoho odlifrovat. Hezká myšlenka na papíře, nicméně předpokládá-a to poměrně striktně-že vaše práce jde snadno delegovat protože máte v zásobě někoho kdo je schopen se o tyto povinnosti kvalifikovaně a v rozumném čase postarat a nebo zácvik takového člověka netrvá nějakou významnou dobu. Ve vývoji rozhodně není tento předpoklad a priori splněn a to užití tohoto pravidla činí obtížným.

Troufnu si tvrdit, že čas pro zácvik na práci na projektu bude řádově menší než čas pro zácvik vývojáře, přičemž nechci v žádném případě znevažovat přidanou hodnotu projektových manažerů. Alespoň ne těch schopných. V extrémních případech se bavíme o řádu let a i během této doby je potřeba dělat rozhodnutí typu ‚Mám to nechat udělat nováčka co se s tím bude patlat dva týdny, architekt mu to na review dvakrát shodí ať to přepracuje a architektovi to review sebere dva dny času nebo to má udělat architekt co to bude mít za pět dní hotovo?‘ přičemž syrový čas, strávený architektem není zdaleka jediná metrika, kterou je nutno použít. Je potřeba zohlednit i zácvik a skrze něj budování zastupitelnosti.

Příběh, který autor v kapitole zmínil, skutečně u vývojářů nejde použít tak jak je. Ano, je potřeba povinnosti postupně delegovat, ano, je potřeba uvolnit si ruce skutečně na ty nejdůležitější povinnosti, s tím plně souhlasím. Ale ve vývoji tento proces skutečně může trvat roky. A přitom ani moc nezáleží na tom, že vývojář zná perfektně jazyk. Nemůžete dělat architekta když neznáte platformu.

Time management je zas věc, která má potenciál pomoci – ale skutečně existuje dobrý důvod proč vývojářům nechat pružnou pracovní dobu. Jde o to, že pokud je vývojář mentálně v té poslední fázi vývoje, to jest: má v hlavě jak my říkáme ‚nachroustáno' vše co potřebuje a dopsat kód je otázka málo jednotek hodin či desítek minut, pokud se tohle přeruší koncem pracovní doby, zítra se začíná od začátku a na dodávce se v extrému může přidat celý den. Pak je opravdu na zváženou, jestli těch pár desítek minu nestrávit hned a pak příští den odejít dřív. Ano, souhlasím že je potřeba hlídat si v rozumné míře počet hodin, které se vývojem stráví a nepřenášet si práci domů. Jen tvrdím že pravidla kolem time managementu nejsou tak černobílá jak se autorovi zdají být.

A co se týče rovnováhy mezi pracovním a osobním životem, nu...občas to jde opravdu těžko roztrhnout. Z vlastních zkušeností vím, že klidně přemýšlím o dalším skriptu/objektu/komponentě třeba u mytí nádobí. Nebo při řízení auta na cestě domů. Python pro mě není jen způsob obživy ale i záliba. A to obrovská. A vím, že lidí tohoto typu je v mém okolí víc, než by se dalo čekat. Pak autorova tvrzení ‚nikdy nezapomínej proč dřeš tak tvrdě' a ‚čas který strávíš v práci má mít kvalitu vs čas který strávíš pro sebe, své zájmy a své blízké má mít kvantitu' vlastně tak trochu postrádají smysl když práce je svým způsobem zábava za kterou nás shodou okolností i platí.

Kapitola 9 – Než začneš, pořádně se připrav

Ano i ne. Je vždy lepší mít k dispozici všechny podklady, veškerou dokumentaci, všechen potřebný hardware než začnete svojí práci. Krutá realita je že ne vždy tomu tak skutečně bude. A v případě, že je úkol natolik abstraktní že nevíte jak ho pořádně uchopit, nepodceňujte tzv. ‚průzkum bojem'. Pustíte se do toho po hlavě, rýpnete do bahna, něco obzvlášť hnusného vyplave a vy se s tím pak budete muset nějak vypořádat. Pořád je ale tohle lepší než uvíznout na mrtvém bodě a jen pálit čas.

Co se týče tématu ‚skoro hotové', s tím mohu jedině souhlasit. Udělat ‚první výkop' co je ‚good enough for rock'n'roll', pak ‚vzít za kormidlo a srovnat kurz' je naprosto validní přístup k problému v mnoha případech. Viz. průzkum bojem. Ono to ale má i další důsledky, na které je potřeba být o dost opatrnější. V případech kdy je nutno po takovýchto technikách sáhnout, je dobré udělat funkční prototyp, nebo ‚proof of concept' chcete-li, a poté na základě tohoto průzkumu vystavět koncepční návrh a řešení problému. Opět je tu nemalé riziko vzniku technického dluhu a zdůrazňuji že je opravdu velice tenká linie mezi něčím co stojí za to používat dál a opravdu prototypem, sloužícím pouze k demonstraci řešení a nikoliv jeho finální podobě.

Nu a co se týče pracovního prostředí a myšlenky ‚pokud máte na stole jen to co potřebujete ke splnění úkolu tak se budete více soustředit'...popravdě, tady záleží vývojář od vývojáře. Mnohem důležitější než perfektní organizace materiálů a hardwaru podle mého názoru je zda se vývojář ve svém prostředí cítí komfortně. Pak teprve z něj padají výsledky. Osobně si myslím že občasné klubo drátů, šroubovák nebo konektor, jež lze po příliš dotěrném kolegovi hodit, nejsou na škodu.

Kapitola 10 – jeden barel za druhým

Už citát na začátku kapitoly naznačuje že se s tímto pravidlem stoprocentně ztotožním a další čtení mi opravdu dalo za pravdu. Celá myšlenka této kapitoly je nepřepínat síly na zdolání větší úlohy ale spíše nasadit pevné a vytrvalé tempo. Nejde o rychlost ale skutečně o vytrvalost. Jeden z mých kolegů sepsal své životní zkušenosti jako seznam tipů, triků a postřehů a jeden z nich, který je relevantní k této kapitole, zní ‚metoda Čínských skoků na vlastní nebezpečí'.

Z mých zkušeností, pokud je problém příliš složitý, je potřeba to vedení s předstihem dát vědět a pak se to pokusit rozdrobit na menší kousky, které lze plánovat, delegovat, ideálně vyřešit nezávisle na původním problému či zadání. Pokud se to podaří nadrobit, je mnohem snazší udržet trvalé tempo vývoje – na základě čehož lze tempo týmu měřit a je tak možno si s vedením nastavit i na poměrně dlouhé časové úseky realistická očekávání → méně stresu, lépe se daří vyhnout se prioritním a současně urgentním úlohám, lépe lze plánovat kapacity vývojářů a jejich obsazenost. Spousta benefitů pro všechny zúčastněné a přitom žádná negativa.

Pokud tento přístup selže, vždy je možno aplikovat ‚průzkum bojem' a na základě výsledků pak navrhnout řešení které je možno rozdrobit, delegovat a plánovat, viz. Komentář ke kapitole 4.

Kapitola 11 – vylepšuj své klíčové dovednosti

Jednoznačně ano, mám s tím své zkušenosti a plně korespondují s myšlenkou této kapitoly. Popravdě, pro vývojáře je to z nátury jejich profese v podstatě absolutní a nepostradatelná nutnost. Sám mohu porovnat kód od vývojáře, který věnuje obrovskou hromadu samostudiu jazyka i jeho aplikace versus kód od vývojáře, který zamrzl na mrtvém bodě nebo tomu věnuje jen malé úsilí. Druhé kategorie je naštěstí jak šafránu, alespoň v mém okolí.

Dovolím si nicméně trošku rozporovat autorovo tvrzení, že každá dovednost se jde naučit. Bohužel, není tomu do důsledku úplně tak. Je potřeba mít na dovednost alespoň nějaké předpoklady, pojmenujme to třeba rozvíjitelným talentem. Vezměte průměrného člověka a pokuste se ho naučit dovednost, kterou nikdy předtím neměl/neznal/nepokusil se o to. Každému člověku bude trvat jinou dobu se to naučit. Průměrně to mohou být např. týdny, pro někoho to bude představovat ale dny, pro někoho měsíce, roky...a takový člověk nebude mít reálně několik let času k dispozici, aby se např. naučil řídit nákladní automobil, couvat s přívěsem, pájet a svařovat...nebo naučit se programovat.

Tohle je bohužel krutá realita, kterou jsem vypozoroval už před nějakým časem. Jde kohokoliv naučit cokoliv tak aby toho využil v zaměstnání? Pravděpodobně ano. Bude ta snaha, práce a čas vynaložený k naučení se té dovednosti adekvátní vzhledem k přínosu pro zaměstnavatele pro každého člověka co se o to pokusí? Jednoznačně ne. I kdyby se přesto takový člověk o to pokusil, dokáže se to naučit v takovém čase aby to pro firmu bylo benefitem? Krajně nepravděpodobné. Někteří lidé na některé dovednosti prostě nemají hlavu a je důležité si to uvědomit a znát své limity.

Dejme tomu však, že dotyčný má pro klíčové dovednosti vhodné předpoklady. Autor zmiňuje jako vhodné metody sebevzdělávání četbu relevantní literatury, semináře, případně audioknihy. Pro programování jsou audioknihy v podstatě nepoužitelné. Semináře a workshopy mohou být vhodné pro úplné začátečníky, ale jakmile se dostanete přes určitou úroveň znalostí, jsou téměř k ničemu. Co se týče literatury...osobně jsem se mnohem více naučil z příkladů na internetu než četbou knih, mimo jiné i z důvodu že lze snadno provést ‚CTRL+cizí → CTRL+vlastní → spustit a kouknout kde to umře když tam je chyba', od jisté doby hledáním v automaticky generované dokumentaci.

Kapitola 12 – využij svých nevšedních dovedností

Zcela nepochybně. Autor doporučuje průběžně se sám sebe ptát ve smyslu ‚Co je to, co umím naprosto skvěle, co nikdo další ne, čímž jsem současně velmi cenný pro zaměstnavatele?‘. Bezprostředně s tím souvisí i tvrzení, že člověka nejvíc na práci baví zejména to, co mu jde – a toto tvrzení koresponduje s mojí úvahou nad kapitolou 8.

Bohužel to vede na problematiku zastupitelnosti. Firma ze své vlastní pozice by neměla úplně záviset na jedné osobě, co je schopna jako jediná provést nějakou sadu úloh. Pro korporát je to přímá cesta do pekel.

Tady jde proti sobě myšlenka dát jasně najevo svoje unikátní dovednosti které firma musí finančně ocenit a snaha firmy, budující si udržitelnost v této oblasti a nezávislost na jedné konkrétní osobě, která může snadno přeběhnout ke konkurenci a patřičné know-how z firmy zmizí.

Kapitola 13 – poznej limity

Tohle je obzvlášť důležité a současně také velice, velice obtížné. Nejlépe vysvětlím příkladem. Dejme tomu že jsem doteď psal pouze uživatelské rozhraní. Najednou mi bude přidělen úkol dopsat něco do podpůrné knihovny a projektový manažer chce dopředu vědět kolik času mi to zabere. V tuto chvíli, pokud si neuvědomím že mi chybí zkušenosti v této problematice, mohu snadno naodhadnout menší objem práce než tam skutečně je a naslibovat nerealistické náklady/termíny. Je potřeba si uvědomit že ty zkušenosti nemám a transparentně dát najevo že ten časový odhad je pouze hodně indikativní a pokud chce přesnější, bude se muset zeptat někoho kvalifikovanějšího.

Obecně je potřeba dostat se alespoň do stavu, který popisuje Sokratés: vím že nic nevím. Vědět, kde je hranice toho co vím a toho co nevím právě zabraňuje těmto problémům. Samozřejmě jakmile tohle vím, dá se s tím něco dělat.

Nemusí přitom jít o limity dané znalostmi dané technologie nebo problematiky. Dobré návyky, time management, organizační schopnosti jsou také oblastí, kde – pokud je zde nedostatek – vzniká úzké hrdlo.

Limitujícím faktorem nemusí ale být moje vlastní neznalost. V korporátní firmě – která by neměla záviset na konkrétních osobách – zaručuje fungování interní směrnice nebo proces. V procesu mohou být vyžadovány činnosti, které tam mohou být z historických důvodů ale vlastně nejsou vůbec potřeba. Pro vývoj mohu dát perfektní příklad: drtivá většina překladačů kódu vytváří na disku obrovskou hromadu souborů. A každý tento vytvořený soubor musí antivirus otevřít a otestovat na škodlivost a to čas, strávený překladem, značně natahuje – v extrémních případech i o desítky minut. A pokud se pokouším dejme tomu opravit chybu iterativně(změním kus kódu → překlad → opakuj), u velkých projektů i třeba jen vypnutí překladu kódu který se nezměnil a použít už přeloženou komponentu může představovat zvýšení počtu iterací za den na více než dvojnásobek, čímž produktivita drasticky vzroste.

Autor dále tvrdí že i zde platí Paretovo pravidlo – 80% limitací je interních, ať už v podobě osobních limitů nebo organizačních problémů. Ze zkušenosti se pro vývoj spíše přikláním k té druhé variantě – drtivá většina identifikovaných problémů je technického charakteru.

S tím souvisí další nepříjemnost: i pokud se v organizaci podaří najít problém který něco brzdí a drhne to tam, pokud je procesního charakteru, je potřeba přesvědčit vedení a dohodnout řešení – a

bohužel co člověk, to vlastní názor a neexistuje optimální řešení, kdežto u technických problémů většinou existuje objektivně správné řešení, dané obvykle použitou technologií, architekturou a podobnými faktory, snadno může vzniknout více variant, lze pak provést optimalizační úlohu která vybere na základě zadaných kritérií to optimální, čímž se dohady a handrkování značně redukují.

Kapitola 14 – natlač se do toho

Jeden z mých kolegů na toto téma podotkl ‚citron taky musíš zmáčknout aby z něj něco vyteklo', další kontroval ‚no jo ale pak už to nebude citron'. Obojí tvrzení mají něco do sebe.

Pakliže ten nátlak je interní, sám sebe dokážu namotivovat k tomu abych se do úloh pustil a neodešel dokud není hotovo, abych dělal něco extra aby bylo hotovo, to samo o sobě funguje poměrně dobře – i v tomto mám zkušenosti. Koneckonců…je skvělý pocit dotáhnout do cíle něco, o čem ostatní byli přesvědčení že nejde, že se nedá stihnout, že na to nejsou prostředky, etcetera, etcetera, a dokázat jim že nemají pravdu. Že jste někdo, kdo – jakmile si něco usmyslí – tak neodejde bez výsledku.

Bohužel stejně tak i zkušenosti s tím, že nátlak je externí. Pokud je dost velký, není to bez následků. Pozor také na syndrom vyhoření. Co se týče práce pod nátlakem, byl jsem v situaci, kdy jsme řešili obrovský průšvih v podstatě měsíc v kuse, obvykle jsme s dalším kolegou nestrávili v kanceláři méně než 10 hodin denně a extrém byl týden téměř 14 hodin denně v kuse. Náš psychický stav se snadno dal označit za velmi labilní a toto je příklad, jak by to v žádném případě nemělo vypadat. Ano, chybu jsme opravili, ale následky v podobě nespavosti jsme si sebou nesli pěkných pár měsíců. Že jakákoliv motivace vzala za své snad ani není třeba dodávat.

Kapitola 15 – Maximalizuj svůj výkon

Ne však vyčerpáním se do bezvědomí.

Mozek je orgán, jež spotřebovává u lidí jednu z největších částí kyslíku i živin, což znamená že v čím lepším fyzickém stavu jsme, o to více jsme ze sebe schopni vydat. To samé platí o odpočinku a pravidelném spánku, stejně tak sportu.

Ač jako člověk zvyklý pít někdy až extrémní množství kávy, tady musím autorovi dát za pravdu. V nedávné době jsem dospěl k rozhodnutí změnit své stravovací návyky poté, co jsem stoupl na váhu a ručička přelezla velice nepříjemnou cifru. Je to sice pouze několik týdnů od této změny(novoroční předsevzetí), ale efekt je už znát.

Autor dále radí najít si denní(či noční) dobu, kdy je mozek nejaktivnější a tam zanést řešení těch nejsložitějších úloh, stejně tak vyvarovat se příliš dlouhé pracovní době. Sám jsem spíše noční tvor a obvykle nejvíce práce udělám v pátek mezi pátou odpoledne a desátou večer. Důvod je prostý: všichni jsou doma a já mám konečně klid na nerušenou činnost. Ovšem s dodatkem o počtu hodin denně strávených prací, s tím podle vlastních zkušeností souhlasím.

Kapitola 16 – namotivuj se

Tohle bude složité. Vývojáři jsou...zvláštní sorta lidí. Polovina z nich se přímo vyžívá v tom, že jsou od zbytku společnosti nějak odlišní, jiní, nenormální, divní, co nejvíc odlišní od čehokoliv co se považuje za normální. Druhá polovina se naopak snaží ze všech sil vypadat jako normální lidé a bojí se jakéhokoliv nařčení že patří do té první skupiny.

U testerů je tato odlišnost ještě umocněna. Tester se vyžívá ve zkáze a destrukci. Náš obvyklý adrenalinový sport je najít obrovskou chybu těsně před vydáním k zákazníkovi a pak jen sledovat jak kolem toho všichni zděšeně pobíhají. A čím větší průšvih odhalíme, o to větší zvrácené uspokojení z toho máme.

Jak se namotivovat do akce je tedy obzvláště složitá úloha a popravdě, techniky doporučované autorem jsem zkusil – a bez jakéhokoliv úspěchu. Můj mindset zkrátka funguje jinak.

Uvedu příklad: autor doporučuje stále si v případě složité situace sám sobě opakovat ‚To zvládnu! Tohle prostě zvládnu!‘ dokud tomu sám nezačnu věřit. Mé bakalářské studium bylo vcelku problematické a podařilo se mi ho úspěšně zakončit až na druhý pokus – a jedna z hlavní motivací dotáhnout to do konce bylo zjištění, že se kolegové začali vsázet o stravenky jestli to zvládnu nebo vzdám. Co jsem si opakoval? ‚Přece těm idiotům neudělám radost.‘ Dlužno podotknout že stále máme dobré vztahy i s těmi kdo vsázeli proti mně.

Kapitola 17 – technologie – žrout času

Souhlasím. Sám už jsem několikrát revidoval a optimalizoval obvyklé komunikační kanály. Složitější trošku je, že počítač je pro mě zdroj obživy tak poněkud nejde provést autorovo doporučení nezapínat žádnou techniku dokud si nerozmyslím pořádně co budu dělat a kdy, ale jinak s tím plně souhlasím.

Pokud mám meeting, je potřeba abych se mu plně věnoval. Pokud u toho dělám něco jiného, šance je že mi uteče něco důležitého. Na druhou stranu: je naprosto v pořádku zvednout se a odejít, pokud už moje přítomnost nemá přidanou hodnotu a jen bych pálil čas.

Co se týče e-mailů, tady je problematika poněkud složitější. Na jednu stranu ano, je potřeba číst jen ty důležité a pravdou je, že z automatizovaných systémů odchází hromada automaticky generovaných e-mailů a drtivou většinu z nich je vcelku bezpečné odignorovat. Pokud někdo chce abych mu schválil příspěvek do kódu, ozve se mi sám. Zažil jsem nicméně i situace, kdy ten dotyčný věděl velice dobře že bych mu to hodil na hlavu a vrátil k přepracování a zkusil si vyžádat review od někoho jiného, pak se občas hodí věnovat pozornost i těmto automaticky generovaným e-mailům. Filtrování podle předmětu funguje obvykle poměrně dobře dle mých zkušeností. Pravdou ovšem je, že jsem schopný text číst poměrně rychle a tak letmý pohled ani na 50 e-mailů nepředstavuje pro mě nijak velkou časovou investici.

Dává ale smysl vyhradit si čas na čtení e-mailů a poté je pustit z hlavy. Opět, souvisí s přerušováním a jeho negativními vlivy na produktivitu.

Co se týče kontaktu s ostatními/s okolnostmi/se světovým děním, pandemie ukázala že v tomto ohledu má autor skutečně pravdu. Jestli to je důležité, dostane se to k vám jinak. Dotyčný se připomene. Události se budou debatovat u odpolední odpočinkové kávy. Byly dny, kdy jsem se natolik zahloubal do práce že jsem se po třech dnech systematického pokroku musel ptát co je vlastně za den – a stejně mi neuniklo nic z těch opravdu podstatných věcí.

Kapitola 18 – na plátky a na kostičky.

Tato kapitola je opět jednou, se kterou plně souhlasím a ztotožňuji se s ní. Na principech, popsaných v této kapitole funguje vlastně většina metodik agilního vývoje softwaru.

Rozdělit problém na menší samostatné úlohy má i ještě další výhody: krom toho že pokud se dohodne společná část(interface nebo API) je možno tyto dílčí úlohy delegovat na více vývojářů, tak u menších a jasně vymezených úloh se mnohem snáz odhadne jejich časová náročnost, což je jednoznačný benefit i pro řízení projektu.

Nátlak na uzavírání úloh je opět přítomný ve většině agilních metodik. Tým musí cítit ten nátlak dodržet závazek jak v obsahu práce, tak i v termínu. Jedině pak mohou tyto techniky fungovat – a ukazuje se že tomu tak skutečně je.

Kapitola 19 – vytvoř si větší časové bloky

Neexistuje nic horšího než každou druhou hodinu dvacetiminutový meeting. Vážně. Slyšel jsem dost vývojářů i architektů mrmlat přesně na toto. Všichni tlačí vedení a projektový management aby všechny meetingy organizovali pokud možno v jeden den a zbytek týdne nechali čas rezervovaný na vývoj tak aby vývojáři nebyli přerušování v práci. Důvody jsem zmiňoval v rozboru kapitol 4 a 8, nevidím důvod se k tomu vracet. Stejně tak to platí i pro odbavování e-mailů – stanovte si jeden, dva bloky denně a konec. Nepřepínat na to v průběhu aktivního času.

Sdílený kalendář a jeho důsledné vyplňování je velice jednoduchým a přitom velmi účinným nástrojem, jak jednoznačně ostatním dát najevo kdy nemají a nebo naopak kdy mohou rušit.

Kupříkladu já mám dopoledne vyhrazený čas na podporu ostatních – během té doby pomáhám ostatním řešit jejich úlohy, schvaluji či zamítám příspěvky do kódu, jakmile je po obědě, mají smůlu protože už mám vyhrazený čas na svoje vlastní úlohy.

Kapitola 20 – vytvoř si pocit urgence

Kdybych měl zmínit z této kapitoly jednu věc, je to koncept tzv. ‚flow‘. Termín velice těžko přeložitelný do češtiny a navíc většinou nepřekládaný a užit jako terminus technicus, nicméně pokusím se. ‚Odsýpá to.‘ Stav mysli, kdy je člověk natolik ponořený a zahrabaný do činnosti, kdy má v hlavě ‚nachroustáno‘ vše potřebné, kdy každou vteřinu je za ním vidět smysluplný výsledek. Pro vývojáře extrémně důležitý pracovní režim lidského mozku.

Flow je věnována řada publikací, protože jakkoliv je tento stav mysli žádoucí a prospěšný, o to těžší se do tohoto stavu se dostat. Asi nejjednodušší způsob dle mých zkušeností je vypnout notifikace od e-mail klienta, od skypu, teamsů a podobných ‚firemních kecálků‘, nasadit sluchátka, vím že někteří kolegové notně využívají i obyčejné špunty do uší, ti s vytříbenějším vkusem zainvestovali do sluchátek s aktivním potlačením okolního šumu, ujistit se že je poblíž káva či jiná oblíbená tekutina – a pustit se do práce. Po čase aktivního soustředění se to zvrhne a už to jede.

Předpokladem nicméně je: přestat žvanit o tom co všechno dnes potřebuji udělat a namísto toho se do těch úloh pustit a začít je odbavovat.

Kapitola 21 – levou zadní

Celá pointa této kapitoly je neodbíhat od úlohy dokud skutečně není hotovo. Některé statistiky, které autor uvádí, bych možná i pro oblast vývoje rozporoval – ale v tom horším směru. Skutečně existuje velice dobrý důvod, proč je od (schopného) vedení tlak na vývojáře aby nedělali na dvou úlohách naráz, a naopak aby se soustředili na jednu konkrétní věc, dopsali ji, vyzkoušeli, zaintegrovali do zbytku projektu a pak teprv, když mají všechno tohle hotovo, sáhli po další úloze.

Co se týče přepínání úloh, dlouhodobě je to opravdu přímá cesta do pekel, nicméně detailněji jsem to rozebíral v kapitole 4 a nechci se k tomu už vracet.

K tomu drobná poznámka z vlastních zkušeností: aby to dobře fungovalo, je nutné velice dobře specifikovat kritéria, kdy je úloha považovaná za opravdu hotovou.

V podstatě veškeré mé zkušenosti s vývojem poměrně dobře korespondují s doporučeními v této kapitole.

Závěrem

Poměrně velkou hromadu doporučení z této knihy jsem znal už z dřívějška a popravdě bylo vcelku zajímavé srovnání s těmito tipy, triky a postřehy s odstupem času, kdy jsem měl možnosti tyto ‚ideální' stavy porovnat s realitou.

Co se týče rozsahu, jak moc tato doporučení aplikovat: užít hlavu. Obzvlášť ve vývoji to platí dvojnásob. Nic není dogma, pokud pravidlo úplně nevyhovuje, nejde aplikovat, nepřináší výsledky, je možné ho modifikovat.

Velké množství kapitol, které se zdánlivě věnují samostatným doporučením ale mají obrovský překryv. Každá kapitola vlastně svým způsobem jen objasňuje jistý úhel pohledu. Důležitý je celý obraz.

Tlak na práci jen na jedné úloze, nenechávat se vyrušovat, umět si správně vybrat úlohu s nejvyšší urgencí a nejvyšší přidanou hodnotou, dělat na ní dokud není hotovo a pak sáhnout po něčem dalším, využít u toho svoje unikátní schopnosti a dovednosti, umět udělat dekompozici problému, vyřešit jen to nejdůležitější a zbytek hodit na nějakého nováčka co se tím současně zaučí – vždyť tohle je vlastně výstup z tuctové a ničím vyjímečné optimalizační úlohy, která jen potřebuje vstupy a nějakou úvahu.