ZA DVA DNY TO NAHODÍME

O úsudku, odpovědnosti a práci Technical Delivery Leada

Valdemar Závadský

O obálce

Na obálce je náš pes Rex. Nejvíc rysů má po border kolii: je chytrý, pracovitý a pořád v pohybu. Když jsme vybírali obálku, došlo mi, že to není náhoda: dobrý Technical Delivery Lead potřebuje skoro totéž co dobrý ovčácký pes.

Výdrž a tah na branku. Bystrost. Schopnost uhlídat velké stádo, aniž by ho honil. A hlavně tu zvláštní radost ze hry, která u border kolie znamená pracovat do padnutí, a u delivery leada chuť řešit složité úkoly, i když se zrovna nikdo nedívá.

Takže až v knize narazíte na kapitoly o stádech, obdélnících a hlídání celku, vzpomeňte si na Rexe. On to má v povaze. My ostatní se to musíme učit.

Jen jednu věc po Rexovi nekopírujte: vydrží běhat od rána do večera a je přesvědčený, že to zvládne každý. Nezvládne. O tom je kapitola 10.

Obsah

Předmluva aneb kniha za pět minut

Tato kniha je o roli Technical Delivery Lead (**TDL**). Je to člověk, který *odpovídá za dodání softwaru* zákazníkovi v dohodnutém čase a rozpočtu, včetně toho, že systém půjde provozovat. Musí rozumět technice i vedení projektu, aby dokázal posoudit problém, rozhodnout, co s ním, a nést za výsledek odpovědnost.

Je určena lidem, kteří do této role vstupují nebo v ní už stojí, a stejně tak těm, kdo s nimi pracují: projektovým manažerům, produktovým vlastníkům, architektům a vedení, které se rozhoduje, zda takovou roli vůbec zřídit. Nenajdete v ní metodiku ani postup krok za krokem. Kniha nejede podle žádného rámce, **TDL** funguje uvnitř všech a proč tomu tak je, vysvětluje krátký text na jejím konci. Má pomoct poznat, na co se v projektu ptát, které signály nepodceňovat a jaké problémy se opakují bez ohledu na technologii. Když se díky ní *na příštím statusu zeptáte* na něco, co by vás dřív nenapadlo, splnila účel.

AI výrazně zrychlila psaní kódu. Pořád ale někdo musí *rozumět celému řešení* a posoudit, jestli ho tým dokáže dodat. Špatný odhad nebo přehlédnutá závislost nezmizí tím, že se rychleji programuje. Jen na chybném rozhodnutí stihne vzniknout víc práce, než si ho někdo všimne.

Dobrý **TDL** *odhalí problém včas*, kdy se s ním ještě dá něco dělat. Ověřuje, že reporty odpovídají tomu, co aplikace skutečně dělá, a že odhad lze s daným týmem stihnout. A hlídá, aby projekt nebyl závislý na jednom člověku.

K tomu potřebuje předem dohodnuté *pravomoci a jasné mantinely*. Musí také vědět, za kým jít, když sám rozhodnout nemůže. A na druhé straně musí být někdo, kdo je ochotný slyšet i nepříjemné věci, dokud je čas je řešit.

A jedno přiznání hned na začátku: tempo, ve kterém **TDL** sám pracuje, není tempo pro tým. Sprint je jeho volba, rytmus patří týmu. Kapitola 10 je o tom, jak to nesmíchat.

Jak se pozná, že to celé funguje? Je jasné, kdo rozhoduje, a nečeká se na něj týdny. Status projektu odpovídá běžícímu produktu, každé riziko někdo řeší v dohodnutém termínu, projekt *přežije dovolenou kohokoli*, včetně samotného **TDL**, a tým se po něm hlásí na další.

Tolik kniha za pět minut. Ta delší verze se čte nejlépe od začátku, protože *kapitoly na sebe navazují*. Kdo má málo času, může začít kapitolami 2 (proč odhad není proveditelnost), 5 (proč zelený status nestačí) a 8 (jak mluvit o riziku, aby se dalo rozhodovat) a shrnutími na konci kapitol.

Na úvod

U některých softwarových projektů je od začátku *celkem jasné*, *jak je dodat*. Produkt je srozumitelný, tým sehraný, prostředí stabilní a je jasné, kdo o čem rozhoduje. V takové situaci není potřeba kolem dodávky stavět zvláštní roli. Stačí dobré produktové vedení, schopný technický tým a disciplína dokončovat práci.

TDL začíná být důležitý tam, kde to takhle jednoduše nefunguje. Obchodní závazek vznikl dříve než podrobná analýza. Systém má několik vrstev a každou dodává jiná skupina. Část výsledku závisí na zákaznickém prostředí, které ještě neexistuje. Termín je pevný, rozpočet omezený a technologie se mění rychleji než organizační návyky. Každý jednotlivý tým může pracovat dobře, a přesto se *celek* nehýbe.

Z PRAXE

Dva týdny před velkou marketingovou premiérou nového platebního zařízení dorazilo od výrobce čtyřicet testovacích kusů. Měsíce příprav, pozvaní zákazníci, rezervované prostory. Třicet pět zařízení selhalo hned po vybalení z krabice. Z pěti zbylých během tří dnů odešla další tři. Se dvěma funkčními kusy se premiéra nedala odehrát a musela se o půl roku odložit. Přitom až do chvíle, kdy se otevřely krabice, vypadal plán na papíře v pořádku: úkoly se plnily, termíny seděly, všichni pracovali naplno. Jen se nikdo včas nezeptal, co přesně už umí ta věc, která má za čtrnáct dní ležet na stole před zákazníky.

Odtud je i název knihy. „Za dva dny to nahodíme“ je nejčastěji pronášená věta v dějinách softwaru a málokdy trvá dva dny. Každý ji slyšel a každý ji někdy řekl, autor této knihy vícekrát, než by rád přiznal. Není to lež. Je to

upřímný optimismus člověka, který vidí svůj kus a nevidí celek. Tato kniha je o tom, kdo má vidět celek a co má dělat ve chvíli, kdy tu větu slyší.

Právě o takových situacích je tato kniha. Není to metodika a nesnaží se z role udělat soubor příkazů. **TDL** potřebuje poznat, *co projekt právě brzdí*, co ještě nevíme a kdo o tom může rozhodnout. Stejná praktika může jeden projekt zachránit a jiný zatížit. Malý emulátor může odstranit měsíce čekání, nebo zakrýt rozdíly vůči reálnému systému. Zkušený zásah může tým odblokovat, ale tým si na něj také může zvyknout a příště už jen čekat na pomoc. Agresivní odhad může být přesnější než opatrný součet, nebo může být jen obchodním přáním převlečeným za seniorní úsudek.

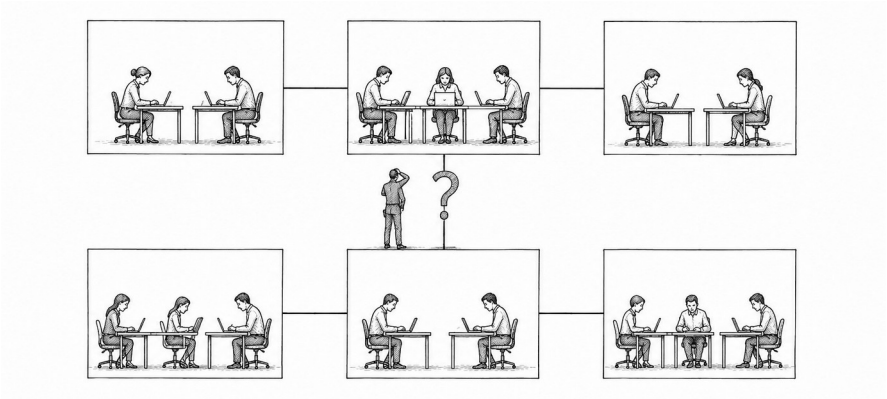
Příběhy v knize jsou *anonymizované a mírně upravené*, aby je nešlo přiřadit ke konkrétnímu zákazníkovi. Jejich podstata je skutečná. Pocházejí z AI produktů, podnikových integrací, regulovaných systémů, cloudu, provozně náročného softwaru a z vývoje platebního zařízení, kde se software potkává s hardwarem, certifikací a výrobní linkou na druhém konci světa. Najdete je v blocích „Z praxe“. Na několika místech text doplňují i ukázky konkrétních dokumentů.

Kniha se věnuje *dvěma tématům*. Prvním je nástup AI, který mění způsob vývoje softwaru. Druhým je pragmatická dodávka: přesvědčení, že software se musí dodávat s rozumnou ekonomikou a že k tomu je potřeba role, jako je **TDL**. Témata se prolínají, ale nevznikla spolu. Pragmatického delivery leada potřebovaly projekty dávno před AI a AI by měnila vývoj i bez něj. S AI je ta potřeba ještě viditelnější.

1. Život mezi obdélníky

Na architektonickém diagramu vypadá software přehledně. Na jedné straně je uživatelské rozhraní, za ním aplikační vrstva, databáze, integrační služby a v moderním produktu také AI workflow. *Každý obdélník má vlastní tým* a každý tým vlastní backlog. Když se na projekt díváme tímto způsobem, zdá se, že stačí rozdělit práci, přidělit odpovědnosti a sledovat postup.

Jenže složitý projekt se často zasekne *na čárách mezi obdélníky*, za které žádný tým neodpovídá. Frontend čeká na API kontrakt (tedy na dohodnutou technickou podobu rozhraní, ne na dokument od právníků), backend na datový model, AI tým na reprezentativní data, infrastruktura na rozhodnutí o topologii a celý projekt na zákaznické prostředí, ke kterému nikdo nemá přístup. Lokální práce pokračuje, tikety se zavírají a status může zůstávat zelený. Zákazník přesto nemá nic, co by mohl použít.



Z PRAXE

Nejnázorněji je to vidět na projektech, kde obdélníky nejsou jen týmy, ale celé firmy. U zmíněného platebního zařízení vypadala mapa projektu takto: výrobce hardwaru v Asii, specializovaná firma na bezkontaktní čtečky karet, německá firma na zabezpečení operačního systému, dodavatel bezpečnostního modulu pro zadávání PIN, certifikační autority karetních asociací, návrhářské studio pro vzhled zařízení. A uprostřed toho softwarový tým, který měl všechno spojit do fungujícího celku. Každá z těch organizací dělala svou část dobře. Ale otázky typu „kdo zajistí, že dotyková vrstva displeje nebude rušit anténu pro bezkontaktní platby“ nepatřily nikomu. Nebyly v žádném obdélníku. Ležely přesně na čáře mezi dvěma z nich. Projekt se nakonec neposouval podle toho, jak rychle pracovaly jednotlivé firmy, ale podle toho, jak rychle se dařilo nacházet a řešit právě tyto otázky bez adresáta.

Právě spoje mezi komponentami, týmy a prostředími jsou práce pro **TDL**. Musí dát dohromady, co zákazník potřebuje, kolik na to je času a peněz a s jakými lidmi to lze postavit. K tomu patří technický návrh, ověření, nasazení i provoz. *Všechno musí dohromady dávat smysl.*

Jak se tedy liší od projektového manažera a architekta? Klasický projektový manažer řídí plán, komunikaci a formální stav projektu. Když ale nastane technický problém, musí si pro jeho posouzení dojít k týmu, protože technice do hloubky nerozumí. Architekt naopak vlastní technický návrh do hloubky, ale obvykle nenese termín, rozpočet ani vztah se zákazníkem. ***TDL** obě tyhle práce spojuje*, a k tomu kus analytika, vývojáře a provozáka. Neznamená to, že projektového manažera či architekta nahrazuje: ve větším projektu s nimi stojí vedle sebe a dělí si práci. Znamená to, že když se rozhoduje o celku, je u toho člověk, který rozumí tomu, jak řešení funguje i co bude rozhodnutí znamenat pro zákazníka a rozpočet, a proto může rozhodovat rychle a nést za to odpovědnost. Proč kniha o projektovém manažerovi dál mluví tak málo, vysvětluje samostatný text na jejím konci: „Kam se poděl projektový manažer?“

Pro úplnost ještě zbytek obsazení, protože kniha se dál soustředí na jednu postavu a ostatní se v ní míhají jen v příbězích. Na straně zákazníka stojí obvykle jeho projektový manažer a člověk, který vlastní produkt a jeho priority, ať se mu říká produktový vlastník, garant nebo věcný správce. Na straně dodavatele je vedle projektového manažera a **TDL** obchodník, který zakázku přinesl a stará se o vztah se zákazníkem, architekt, analytici, kteří překládají potřebu do zadání, vývojové týmy s vlastními technickými leady, testeři a lidé z provozu a podpory, kterým celek nakonec zůstane. Většina těchto rolí má jasně dané, co vlastní. **TDL** vlastní *to, co zbylo mezi nimi*.

Odpovědnost za celek přitom neznamená vlastnictví každého detailu. Kdyby se **TDL** snažil být nejlepším specialistou ve všech vrstvách, rychle by se stal nejdražším *úzkým hrdlem* projektu, místem, kde se hromadí čekající

rozhodnutí, a jediným člověkem v projektu, který nesmí onemocnět. Potřebuje se orientovat ve všech částech řešení a chápat, proč se dělají právě tak. Na podrobnosti má specialisty. Rozdíl je v tom, že někdo musí vidět, zda jejich lokálně správná rozhodnutí dávají dohromady systém, který lze včas postavit, ověřit, nasadit a provozovat.

TDL musí umět *jít do detailu* a pak se zase podívat na celý projekt. Z příliš velké výšky vypadají všechny problémy jako barevné položky v reportu. Příliš hluboko v jedné komponentě zase zmizí čekání a závislosti všude kolem. **TDL** proto sestupuje do detailu cíleně. Otevře logy, spustí prostředí, projde kód nebo vytvoří prototyp tehdy, když potřebuje rozhodnout nebo ověřit rozpor. Jakmile ví, v čem je problém a kdo ho převezme, musí se vrátit k celku.

Největší pastí role je, že technická kompetence přináší *okamžitou odměnu*. Vyřešit problém vlastníma rukama je konkrétní, měřitelné a často rychlé. Vysvětlit souvislosti a nechat druhého člověka rozhodnout a převzít odpovědnost je pomalejší a méně viditelné. Přesto je právě tato druhá práce dlouhodobě hodnotnější. Zdravý projekt není ten, ve kterém má **TDL** odpověď na všechno. Je to projekt, který se dokáže rozhodovat a pokračovat i bez jeho nepřetržité přítomnosti.

TDL proto odstraňuje překážky, vysvětluje souvislosti a domlouvá mantinely, ve kterých se *lidé mohou rozhodovat sami*. Pokud jeho důležitost měříme počtem věcí, které bez něj nejdou, děláme něco špatně.

NEŽ ZAČNETE NOVOU KAPITOLU

*– **Nejdražší problémy projektu** obvykle nebydlí v žádném týmu. Kdo u vás vlastní čáry mezi obdélníky?*

*– **Odpovědnost za celek** není vlastnictví každého detailu. Jděte do detailu, když je to potřeba, a pak se vraťte k celku.*

*– **Zdravý projekt** se pozná podle toho, že se umí rozhodovat i bez nepřetržité přítomnosti jednoho člověka.*

*– **Otázka na závěr:** kolik rozhodnutí ve vašem projektu právě teď čeká na jediného člověka?*

2. Číslo, které nešlo dodat

Odhad softwaru má zvláštní schopnost vypadat přesněji, než ve skutečnosti je. Čím podrobnější tabulku vytvoříme, tím snáze zapomeneme, že její buňky obsahují předpoklady, lokální rezervy a různou míru porozumění. Rozdělení systému na formuláře, služby, integrační metody a technické úkoly může být užitečné. Součet člověkodnů však ještě není plán dodávky.

Odhad není proveditelnost

> **Z PRAXE**
>
> Regulovaný AI systém pro státní organizaci měl v krátkém čase zpracovávat složitá data a vytvářet z nich strukturované dokumenty podle daného pracovního postupu. Zadání nepopisovalo všechny datové struktury ani přesné chování integrací na interní systémy zadavatele, mimo jiné na rozsáhlou spisovou službu, kterou si odhadující tým neuměl představit. Odhad připravovali schopní lidé, ale bez hluboké znalosti domény, a každou neznámou proto přirozeně ocenili opatrně. Výsledek: přibližně 1 600 člověkodnů.
>
> Jenže systém se měl dodat za devadesát kalendářních dnů od podpisu smlouvy. Devadesát kalendářních dnů je zhruba pětašedesát pracovních. Kdo si to vydělí, zjistí, že by od prvního dne muselo naplno pracovat kolem pětadvaceti lidí: okamžitě k dispozici, vybavených společným kontextem, s dokonale rozdělenou prací a bezbolestnou integrací na konci. Takový projekt v realitě neexistuje: pětadvacet lidí prostě nemůže od prvního dne pracovat na jedné hromadě. Překáželi by si a většinu času by spotřebovala domluva místo práce. První týden by navíc padl na

> hledání dostatečně velké zasedačky. (Proč to tak je, rozebírá podrobně kapitola 4.) Tabulka byla poctivá, jen se podle ní nedalo stavět.
>
> Do odhadu nakonec vstoupila doménová znalost, konkrétně deset let práce architekta a vývojáře přímo v prostředí daného zadavatele. Ta ukázala, že integrační práce je ve skutečnosti menší, než jak ji ocenili lidé, pro které byla neznámou. Klíčové ale bylo něco jiného: smlouva připouštěla, že pokud zákazník do dne akceptace nepřipraví svá prostředí, proběhne akceptace proti emulátorům, zjednodušeným softwarovým náhradám, které se navenek chovají jako jeho systémy. Tím se předefinovalo, co znamená první „hotovo“ (ne produkční nasazení, ale smluvní akceptace), a riziko nepřipravenosti nesl smluvně zákazník. Dodávka se přetvarovala pro tým kolem deseti lidí, první fáze mířila k akceptovatelnému výsledku a následná fáze k reálné integraci. Systém prošel akceptací čtrnáct dní před termínem, s deseti lidmi.
>
> Ani tady ale nešlo všechno hladce. Testovací strategie se nastavovala pozdě, slibovaná testovací data nedorazila v množství ani kvalitě, o jaké se mluvilo na začátku, a jedno mezitýmové rozhraní se zaseklo na několik týdnů. K datům i k rozhraní se kniha vrátí v kapitole 6.

Revize takového odhadu nespočívá v tom, že seniorní člověk škrtne polovinu práce. Musí projít, *z čeho odhad vznikl*. Kolik opatrnosti se započítalo opakovaně? Které integrace jsou skutečně neznámé a které jen neznámé konkrétnímu odhadci? Co přesně znamená první závazek? Musí být systém v devadesátý den v produkci, nebo má projít smluvní akceptací? Je zákaznická strana připravena na reálné propojení? A pokud není, kdo ponese riziko ověřování proti emulátorům?

Tenhle příběh neznamená, že zkušený člověk může odhad prostě seškrtat. Nižší odhad vycházel z *konkrétních změn* v tom, jak se systém dodá. Byla k

dispozici doménová znalost, předefinovalo se první ověření hotového výsledku, použily se emulátory a riziko nepřipravenosti zákazníka bylo smluvně rozdělené. Bez těchto podmínek by stejné číslo nebylo odvážné, ale neodpovědné.

To je podstata rozdílu mezi odhadem a proveditelností. Odhad se ptá, kolik práce vidíme. Proveditelnost se ptá, zda existuje *reálná cesta*, jak ji udělat s konkrétními lidmi, závislostmi, prostředími a rozhodovací rychlostí. **TDL** potřebuje obě odpovědi. Pokud se výrazně rozcházejí, nestačí vybrat příjemnější číslo. Je potřeba znovu promyslet rozsah, pořadí práce, složení týmu nebo to, k čemu se smluvně zavazujeme.

Podcenit se dá i model dodávky

Podcenit se přitom nedá jen množství práce. Podcenit se dá i samotný model toho, co vlastně kupujeme a co budeme muset vyrobit sami.

> **Z PRAXE**
>
> Platební zařízení mělo podle původního plánu vzniknout jednoduše: partner v Asii vyvine a vyrobí hardware, dodá k němu funkční operační systém, a domácí tým jen doplní platební aplikace a vše sestaví. Premiéra pro zákazníky za půl roku, certifikace do roka, pak výroba. Plán se rozpadl během prvních dvou měsíců. Ukázalo se, že partner zvyklý vyrábět spotřební elektroniku nedokáže splnit bezpečnostní požadavky platebního průmyslu. Ne vlastní vinou, prostě to nikdy nedělal. Z nákupu hotového produktu se stal vlastní výzkum a vývoj: bylo nutné převzít zdrojové kódy operačního systému, psát vlastní ovladače, přizvat specialisty na zabezpečení, na bezkontaktní čtečky, na bezpečnostní moduly, a všechny je koordinovat. Projekt tím nenarostl o desítky

> procent, ale zhruba na trojnásobek. Původní odhad nebyl špatně spočítaný. Byl spočítaný pro jiný projekt, než který se ve skutečnosti konal. Zařízení se nakonec na trh dostalo, vyrobilo se jich přes dvě stě tisíc a o pár let později se podobná zařízení objevila u konkurence v celém odvětví. Ale rok zpoždění nevznikl špatnou prací. Vznikl špatným předpokladem v den, kdy se podepisoval plán.
>
> Aby bylo jasné, co „vlastní výzkum“ znamenalo v praxi: Android, který přišel od výrobce, měl obyčejné linuxové jádro, jaké mají spotřební tablety. Pro platební zařízení to nestačilo. Německá firma, která jinak staví zabezpečené systémy pro tamní tajné služby a armádu, do něj podle našeho návrhu naportovala plnohodnotný SELinux, ochranu, kterou běžný Android dostal až o několik generací později. Doplnila kontrolu podpisu každého aplikačního balíčku přímo v jádře a zabezpečený start: hardwarový bezpečnostní modul nejdřív ověří operační systém a teprve pak ho pustí. Zadávání PIN se vyřešilo tak, že se dotyky na displeji v tu chvíli k Androidu vůbec nedostanou. Čte je jen bezpečnostní modul, který je zašifruje a pošle bance. Návrh byl náš, implementace jejich, a z řešení je dnes patent platný v Evropě, USA, Číně, Kanadě i Mexiku. Nic z toho nebylo v původním plánu, protože v původním plánu jsme kupovali hotový operační systém. Mimochodem, je to i přesná ukázka místa, kde žádné „dost dobré“ neexistuje: bezpečnost je první položka na seznamu věcí, na kterých kapitola 3 zakazuje šetřit.

Jak o odhadu mluvit, aby se dalo rozhodovat

Stejně důležité je umět odhad komunikovat. Jedno číslo vytváří dojem jistoty a následně spor o to, zda bylo „správné“. U složité zakázky jsou užitečnější scénáře: optimistický, střední a pesimistický, případně varianta

založená na důsledném využití AI ve vývoji. *Každý scénář musí mít předpoklady*. Obchod pak nevybírá magické číslo, ale míru rizika, kterou je ochoten unést. Musí přitom vědět, co se stane, když některý předpoklad nevyjde.

V soutěži s jednou závaznou cenou (a to je většina veřejných zakázek) se scénáře do nabídky nepřeklopí celé. Slouží uvnitř: střední scénář určuje základ ceny a rozdíl proti pesimistickému se vědomě rozděluje na tři místa. Část jde *do ceny jako rezerva* u skutečně neznámých částí. Část do smlouvy: podmínky součinnosti, termíny zákaznických dodávek, definice akceptace. A část do rozsahu: jednodušší, ale dodatelná varianta funkcí tam, kde zadání připouští výklad. A když manévr s akceptací proti emulátorům není k dispozici (což je častější případ), platí stejný princip v tvrdší podobě: riziko nepřipravenosti zákazníka se musí přeložit buď do smluvní součinnosti s termíny, nebo do ceny. Pokud se riziko nedá pokrýt ani smlouvou, ani cenou, je potřeba říct, že zakázka za těchto podmínek nevychází.

Zkušenost může odhad radikálně snížit u technologií a vzorců, které tým opakovaně zvládl a skutečně je má pod kontrolou. Stejná zkušenost ale klame, když srovnává jen názvy. Známá technologie neznamená *známý kontext*.

> **Z PRAXE**
>
> Přihlašování uživatelů přes standard OAuth je práce, kterou zkušený tým odhadne na den či dva, viděl ji stokrát. Jenže v projektu pro státní správu totéž přihlašování zajišťovala vládní přihlašovací služba: prostředí musí zprovoznit lidé na straně státu, mají své lhůty, své odchylky od

> standardu a svou komunikační režii. Samotné programování skutečně trvalo den. Dodání fungujícího přihlašování trvalo desetinásobek. Kdyby se do nabídky napsal jeden člověkoden, protože „OAuth přece známe“, vznikl by závazek, který nejde splnit. Zkušenost srovnávala název technologie, ne situaci, ve které se dodává.

Používejte selský rozum. U práce, kterou tým opakovaně dělal a má ji pod kontrolou, lze díky zkušenosti, automatizaci a AI odhad výrazně snížit. U cizích závislostí, neznámých dat a nové technologie je potřeba být opatrný. Přidejte rezervu, včas technologii vyzkoušejte (takovému experimentu se říká spike, podrobněji v kapitole 7) nebo si dohodněte, kdy se musí plán změnit.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> – ***Odhad říká, kolik práce vidíme.*** *Proveditelnost říká, zda k ní existuje reálná cesta. Potřebujete obě odpovědi.*
>
> – ***Když se odhad a proveditelnost rozcházejí, nevybírá se příjemnější číslo.*** *Znovu se promyslí rozsah, pořadí práce, tým nebo smlouva.*
>
> – *Ptejte se,* ***co přesně znamená první „hotovo“****: produkce, akceptace, nebo demo? Každé z nich je jiný projekt.*
>
> – ***Známá technologie neznamená známý kontext.*** *Pozor na odhady, které srovnávají jen názvy.*
>
> – ***Nejnebezpečnější podcenění*** *nebývá v číslech, ale v předpokladu o modelu dodávky: co kupujeme hotové a co budeme muset vyrobit sami?*
>
> – ***Otázka na závěr:*** *kdyby váš aktuální odhad dostal test proveditelnosti (s konkrétními lidmi a daty v kalendáři), obstál by?*

3. „Dost dobré“ není sprosté slovo

Technické týmy mají přirozenou potřebu dělat věci dobře. Jenže ne vždy se jejich představa vejde *do času a rozpočtu projektu*. Řešení se navrhuje tak, aby uspokojilo každý technický pohled, přestože zákazník potřebuje mnohem jednodušší výsledek. Dokonalost jedné funkce pak spotřebuje prostor, který chybí na integraci, testování nebo provozní připravenost celku.

Slovo „vyhovující“ (nebo chcete-li „dost dobré“) zní v technickém prostředí podezřele: málokdo si chce napsat na vizitku „autor vyhovujících řešení“. Může evokovat obcházení kvality nebo dodání nejlevnější varianty, která právě projde akceptací. Takový výklad je chybný. Vyhovující řešení musí být bezpečné, spolehlivé v dohodnutém rozsahu a provozovatelné. Nemusí však být nejkrásnější, nejobecnější ani nejvíce automatizované.

> **Z PRAXE**
>
> V zadání stálo: uživatel potřebuje vyhledávat v seznamu obchodních případů. Jedna cesta vedla k dokonale navrženému vyhledávacímu prvku s napovídáním, animacemi a univerzální komponentou, kterou by šlo použít v celé aplikaci, tedy k práci na týdny. Druhá cesta vedla ke standardnímu rozbalovacímu seznamu s vyhledávacím políčkem, hotovému za čtvrt hodiny. Obě varianty splnily uživatelský cíl: najít obchodní případ. Zadání ani akceptační kritéria dokonalejší prvek nevyžadovaly. Týdny navíc by zákazníkovi s hledáním nepomohly a chyběly by jinde.

Ještě výraznější je rozdíl u AI funkcí, kde se pracnost variant často neliší o desítky procent, ale o řády.

> **Z PRAXE**
>
> Zadání znělo: uživatel potřebuje do konverzace s AI doplnit související předpisy. Ambiciózní řešení by nechalo systém samostatně odvodit, které předpisy k případu patří, vyhledat je, stáhnout, vyhodnotit jejich relevanci a vložit je do kontextu: technicky zajímavé, ale drahé na vývoj, na provoz i na ověřování, že to celé funguje spolehlivě. Jednodušší řešení nechalo uživatele předpisy vybrat a nahrát ručně. Oba přístupy splnily větu ze zadání. Jednodušší řešení bylo hotové za dny. Ambiciózní varianta by spotřebovala významnou část rozpočtu celého projektu. Pokud automatický výběr předpisů nebyl akceptačním kritériem ani podstatný pro uživatele, nebyl důvod ho v první fázi stavět.

Rozhodnutí o přiměřené kvalitě potřebuje hranice. Není možné šetřit na integritě dat, auditovatelnosti regulovaného procesu, ochraně tajemství, řízení oprávnění nebo schopnosti obnovy tam, kde výpadek způsobí významnou škodu. Lze však šetřit na komfortu, eleganci, obecnosti a automatizaci, jejichž absence nezničí zákaznický výsledek. Záleží na tom, *která vlastnost je pro daný systém opravdu důležitá*.

Jednu hranici je ale potřeba vyslovit zvlášť, protože se s „dost dobrým“ plete nejčastěji: *základní uživatelská přívětivost*. Systém musí zůstat pro uživatele jednoduchý a intuitivní. To není elegance ani vizuální dokonalost, to je součást zákaznického výsledku. Škrtat se naopak dá výbava, které si nikdo nevšimne: vychytávky, jejichž přidaná hodnota v praxi nefunguje, animace, obecnost pro budoucí použití, které možná nikdy nepřijde. Nesmí se škrtnout to, že se věc dá bez přemýšlení použít.

TDL musí s týmem probrat, *co se vyplatí udělat pořádně hned*, protože to později ušetří práci nebo problémy v provozu, a co je jen něčí oblíbené řešení. Někdy jednoduché řešení vytvoří dluh, který se později mnohonásobně vrátí. Jindy složitá platforma nikdy nezíská druhého uživatele. Rozhodující otázka není „je to čisté?“, ale „co nám to později ušetří a co naopak zkomplikuje?“

> **Z PRAXE**
>
> Systém pro přenos telefonních čísel mezi operátory běžel na integrační platformě, krásné a nesmírně složité. Bylo potřeba rychle přidat do procesu odbočku: každý povel pro ústřednu se měl navíc zkopírovat jako textový soubor do sdíleného adresáře na jiném serveru. Platforma na to neměla hotový konektor. Vznikla tedy funkce o pěti řádcích ve skriptovacím jazyce, zabalila se jako komponenta a zapojila do procesu. Za dvacet minut nasazeno, fungovalo to bez chyby a šlo o pár souborů denně. Pak přišel kolega, který trpěl perfekcionismem. Je to nemoc, která nositele nebolí, bolí jen okolí. Čtyři dny, včetně celého víkendu, to přepisoval do „správné“ technologie a zapojoval do automatizované pipeline. Nikdo to nechtěl. Rychlejší to nebylo. Bylo to složitější, technicky čistší a kromě něj tomu už skoro nikdo nerozuměl, což se v některých týmech omylem počítá za přednost. Pět řádků, kterým rozuměl každý, nahradilo řešení, kterému rozuměl jeden. To není zvýšení kvality. To je dluh v hezčím obalu.

Perfekcionismus má cenu, kterou zaplatí někdo jiný a později. Otázky, které ho drží na uzdě, jsou obyčejné: *kolikrát denně to běží*, *kdo to bude udržovat*, *až autor odejde*, *a co se stane*, *když to selže*. Pět řádků, které selžou jednou za rok a každý je opraví za minutu, bývají kvalitnější řešení než konstrukce, která neselže nikdy, ale nikdo si na ni netroufne sáhnout.

Týmu je potřeba *vysvětlit*, *proč* se má něco udělat jednodušeji. Samotný příkaz nestačí. Když lidé vědí, že termín je pevný, integrace ještě není ověřená a musí zbýt čas na skutečná data, rozumějí tomu, proč teď nemohou měsíc vylepšovat jednu funkci. Mohou pak sami hledat řešení, které se do těch podmínek vejde.

U každého zjednodušení musí být jasné, *co se vypouští*, proč je to bezpečné a co tím bude potřeba řešit později. Pokud vzniká technický dluh, tým má vědět, kdy se k němu vrátí. Jinak se pod „dost dobré“ může schovat cokoli.

NEŽ ZAČNETE NOVOU KAPITOLU

– ***Vyhovující řešení*** *musí být bezpečné, spolehlivé a provozovatelné. Nemusí být nejkrásnější ani nejobecnější.*

– ***Na čem se šetřit nesmí:*** *integrita dat, auditovatelnost, tajemství, oprávnění, schopnost obnovy a základní použitelnost pro uživatele.* ***Na čem se šetřit smí:*** *komfort, elegance, obecnost, automatizace.*

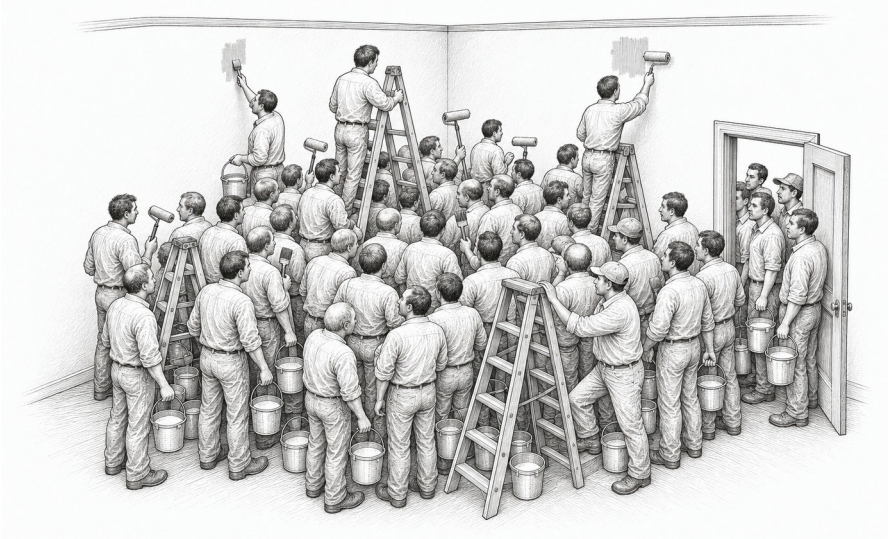
– ***Rozhodující otázka nezní „je to čisté?“, ale „co nám to později ušetří a co naopak zkomplikuje?“***

– ***Vysvětlete týmu, proč je potřeba řešení zjednodušit.*** *Když zná důvod, může sám navrhnout, kde ubrat.*

– ***Technicky čistší není automaticky lepší.*** *Řešení, kterému rozumí jen autor, je dluh v hezčím obalu.*

– ***Otázka na závěr:*** *která funkce ve vašem projektu právě teď dostává dokonalost, kterou nikdo neobjednal?*

4. Tým není násobilka



Když termín hoří, nejviditelnější reakcí bývá přidat lidi. Je to intuitivní, protože větší kapacita by měla znamenat více dokončené práce. Software však není hromada stejných krabic, které lze rozdělit mezi libovolný počet rukou. Nový člověk se musí *nejdřív zorientovat v projektu*, pochopit dosavadní rozhodnutí a domluvit se s ostatními, kam může sáhnout.

Představte si to na malířích. Pokud jeden malíř vymaluje pokoj za hodinu, 3 600 malířů jej nevymaluje za sekundu. *Do místnosti se nevejdou*, budou si překážet a většinu energie spotřebují na koordinaci. Stará projektová moudrost říká totéž stručněji: devět maminek neporodí dítě za měsíc. U softwaru nejsou fyzické stěny tak viditelné, ale existují: jeden datový model, společné prostředí, kapacita na review (vzájemnou kontrolu

práce), jediný produktový rozhodovatel a množství kontextu, který musí lidé sdílet.

> **Z PRAXE**
>
> Nasazení hotového AI asistenta nad firemními směrnicemi u konkrétního zákazníka vypadalo jako drobnost: dva týdny práce pro jednoho člověka, vždyť jde „jen o implementaci“. Po dvou týdnech ale práce hotová nebyla: zákazník měl vlastní požadavky, které znamenaly změny v produktu, a s tím původní plán nepočítal. Vývojář byl mezitím očekáván na velkém projektu, a tak nastoupil náhradník: převzal neúplný kontext, znovu objevoval důvody předchozích rozhodnutí, kladl otázky, na které už jednou padla odpověď. Když se situace zopakovala podruhé, projekt se protahoval a prodražoval, ne kvůli obtížné technologii, ale kvůli opakované ztrátě kontinuity.

> **Z PRAXE**
>
> Na jiné podobné zakázce se zkusil opak. K dvěma lidem, kteří práci téměř dokončili, se přidali další dva na „urychlení“. Výsledek přišel později, než kdyby původní dvojice dokončila práci sama: onboarding, vysvětlování domény a rozšířená review spotřebovaly víc, než nová kapacita přinesla.

Lidé se střídají nebo přibývají, všichni mají co dělat, ale *hotových věcí*, které může zákazník použít, nepřibývá.

Jako základ se osvědčuje *malý tým, který se umí rozhodovat sám*. V moderním prostředí mohou zkušení full-stack lidé s podporou AI zvládnout víc částí řešení než dříve. To neznamená, že zmizely odpovědnosti za UX, QA, infrastrukturu, bezpečnost nebo data. Znamená to, že na každou z nich nemusí být v každém týmu samostatný člověk na

plný úvazek. Tři až pět lidí s jasným vlastnictvím a přístupem ke specialistům často dokáže sdílet kontext rychleji než velká sestava rolí.

Jakmile pracovní celek dlouhodobě roste nad přibližně sedm lidí, začíná být důležitější jeho *tvar* než další kapacita. Objevují se delší rozhodnutí, komunikační režie, překryvy a přehazování odpovědnosti. Odpovědí není jeden větší tým s více ceremoniemi, ale rozdělení na samostatnější oblasti. Mohou sledovat business schopnosti, komponenty nebo technické vrstvy. Správný řez je ten, který minimalizuje každodenní koordinaci a umožňuje brzkou integraci. Jak to dopadne, když tým nabobtná na deset lidí, ukazuje příběh v kapitole 10.

Větší program se pak podobá soustavě malých jednotek. Každá má svůj cíl a prostor rozhodovat, ale sdílí API kontrakty, tempo a eskalační cestu. Nad několika buňkami může působit koordinační role, která udržuje rytmus, sbírá blokace a vrací týmům rozhodnutí. Dobře to vystihuje vojenská analogie. Buňky jsou družstva: malé jednotky, které znají svůj úsek terénu a rozhodují na místě, protože rozkaz shora by dorazil pozdě. Koordinátor je seržant. Nevymýšlí strategii, ale žije mezi družstvy: ví, kdo je právě zablokovaný, drží tempo pochodu a hlídá, aby si jednotky nevlezly navzájem do palebné linie. **TDL** je poručík: odpovídá za to, že celá operace míří na správný kopec, překládá záměr velení do úkolů srozumitelných v terénu a nese důsledky, když se plán potká s realitou. Dobrý poručík nediriguje každý krok svých družstev, ale nikdy není tak daleko od fronty, aby se o problému dozvěděl až z hlášení. Musí zajistit, že koordinační vrstva nezakryje realitu a že závažný problém má krátkou cestu k člověku s pravomocí.

> **Z PRAXE**
>
> Stejný vzorec se opakuje i o úroveň výš, kde už jednotkami nejsou týmy, ale celé domény. Vývoj platebního zařízení řídil čtyřčlenný řídicí výbor: programový manažer, technický ředitel, šéf vývoje hardwaru a šéf vývoje softwaru. Každý z nich měl pod sebou vlastní soustavu týmů a dodavatelů (hardware svou, software svou) a výbor neřešil jednotlivé úkoly, ale spoje: co blokuje koho, které rozhodnutí komu patří a kdy se musí obě větve potkat v jednom funkčním zařízení. Struktura malých autonomních celků s jasnou eskalační cestou se tedy neláme s velikostí projektu. Jen se opakuje na další úrovni.

AI tento princip neruší. Zrychluje produkci kódu a umožňuje jednomu člověku koordinovat více paralelních pokusů. Tím ale často přesouvá úzké místo do review, integrace a produktového porozumění. Velký objem změn může vytvořit větší rozpracovanost rychleji než tradiční tým. Při plánování tedy nestačí vynásobit počet lidí počtem agentů. Je potřeba vědět, *kolik změn tým dokáže* přečíst, pochopit, spojit a ověřit.

Vývojáři pro to mají pojem *technický dluh*: zkratky a odklady v kódu, které dnes ušetří hodinu, ale později stojí dny. Dokud se nesplatí, nabíhají u nich úroky. Jak se tedy pozná, že tým s AI vyrábí takový dluh rychleji, než ho stíhá splácet? Dva signály jsou vidět i bez technického vzdělání. První: roste fronta práce čekající na kontrolu. Změny vznikají rychleji, než je kdokoli stíhá přečíst, pochopit a schválit, takže hotové čeká na hotové. Druhý: na ukázkách je vidět stále více rozpracovaného a stále stejně málo funkčního celku.

Je tedy potřeba *vyhradit čas na kontrolu* a posílat do společného prostředí menší dávky změn, které se dají průběžně ověřit. Postup se pak měří

podle toho, co funguje v celé aplikaci. Tým s AI je jako širší dálnice. Pokud ale na jejím konci zůstala stejná mýtná brána, vyrobili jsme delší kolonu, ne rychlejší cestu.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> *– **Přidání lidí do zpožděného projektu jej obvykle ještě zpomalí.** Nejdřív se ptejte, na čem práce stojí, než začnete shánět další lidi.*
>
> *– **Zdravé jádro jsou tři až pět lidí** s jasným vlastnictvím. Nad sedm lidí přemýšlejte o rozdělení, ne o ceremoniích.*
>
> *– **Struktura buněk, koordinátorů a krátkých eskalačních cest** se opakuje na každé úrovni, od týmu po program.*
>
> *– **AI násobí produkci kódu, ne kapacitu dodávky.** Úzké místo se přesouvá do review, integrace a porozumění.*
>
> *– **Otázka na závěr:** kdyby váš projekt zítra dostal pět lidí navíc, zrychlil by se, a čím byste to doložili?*

5. Status je zelený. A funguje to?

Ze statusu se dozvíte, jak tým popisuje stav projektu. Pořád si ale musíte ověřit, *co aplikace skutečně dělá*. Projektová nástěnka s úkoly (board), procenta dokončení a milníkové prezentace jsou užitečné, protože umožňují koordinaci. Problém nastává ve chvíli, kdy se z nich stane hlavní měřítko zdraví. Uzavřený tiket potvrzuje, že někdo považuje práci za hotovou. Neříká, zda celek dělá to, co zákazník potřebuje.

> **Z PRAXE**
>
> V jednom AI produktu pro zpracování složitých případů byly úkoly uzavřené a jednotlivé komponenty hlášené jako připravené. Při obyčejném průchodu z pohledu businessu však systém vracel výstupy, které byly na první lidský pohled zjevně špatné: vygenerované dokumenty nedávaly smysl, odvozené závěry neodpovídaly vstupním datům. Nešlo o okrajové statistické odchylky, které by odhalil až specialista. Stačil selský rozum. Jak mohl tým úkoly zavřít? Testoval na malých umělých datech, která si sám vytvořil, a nikdy nezkusil do systému vložit rozsáhlejší data s logickou strukturou odpovídající realitě. Komponenty prošly vlastními testy, ale celá aplikace dávala špatné výsledky.

Tým si vytvořil testovací data podle toho, jak zadání sám pochopil. Na nich mu testy prošly, jenže na skutečných datech by chybu viděl hned. **TDL** nemusí sám připravovat kompletní testovací strategii, ale musí si dostatečně brzy *aplikaci vyzkoušet* a podívat se, jestli výsledek dává smysl člověku, pro kterého se staví.

Takovému projití produktu říkejme *průchod*: člověk si sedne k běžícímu systému a projde jím konkrétní scénář očima uživatele, od zadání po výsledek. Pravidelný průchod není formální test ani demonstrace před vedením. **TDL** si při něm ověřuje, co přibylo, co se rozbilo a jestli aplikace stále odpovídá zadání. U delšího projektu je rozumným minimem týdenní frekvence, v rychlé nebo kritické fázi klidně denní či obdenní. Chybu je potřeba chytit dřív, než na ní tým postaví další části řešení.

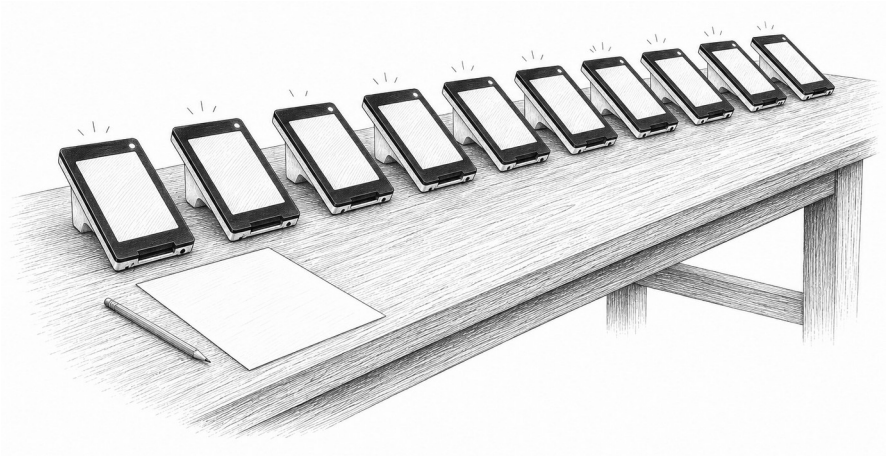
Malá sada scénářů bývá účinnější než nepravidelný velký test. První scénář jednoduše prokáže, že systém žije. Několik dalších projde běžné business cesty. Poslední záměrně zvolí komplikaci, neobvyklá data nebo známou past. Dva až pět průchodů nepokryje celý produkt, ale ukáže, *jestli přibývá fungující práce* a nerozbilo se něco, co už chodilo.

První kontrola při neurčitém pocitu, že projekt „hapruje“, míří na vývojové nebo testovací prostředí. Je dostupné? Běží? Obsahuje aktuální verze, nebo na něm od jara běží sestavení, které si už nikdo nepamatuje? Lze v něm udělat základní cestu? *Když společné prostředí nefunguje*, nikdo na něm neověří, že jednotlivé části spolupracují. Každému může jeho kus lokálně běžet a status pak vypadá lépe než celá aplikace.

Další signály se skrývají v jazyce. „Ještě si držím commit“ (hotová práce zatím leží jen v počítači autora). „Za dva dny to celé nahodíme.“ „Už je to skoro hotové.“ Každá z těchto vět může být jednou pravdivá. (Věta z názvu této knihy má své přirozené místo právě tady a stále platí, že málokdy trvá dva dny.) Pokud se však tyto věty opakují a běžící produkt se stále neposouvá, tým kličkuje kolem čísel. Zvlášť v první čtvrtině projektu je to vážné varování. Zpoždění zachycené takto brzy obvykle neznamená, že

chybí jen několik dní. Znamená, že tým ještě nemá pod kontrolou cestu k integraci.

Realitu doplňují technická data. Nečekaný nárůst chybovosti, propad výkonu nebo opakující se zákaznické nejasnosti mohou ukázat problém, který nástěnka nevidí. Jeden signál sám o sobě nemusí nic dokazovat. Je potřeba *porovnat to, co tým říká*, s běžící aplikací, logy a tím, co hlásí zákazník.



Z PRAXE

Nejdražší verzi „zeleného statusu“ zažil projekt platebního zařízení. Vraťme se k příběhu z úvodu. Plán říkal: za dva týdny proběhne marketingová premiéra (už jednou posunutá po krizi popsané v kapitole 2), testovací kusy jsou objednané, software připravený, všechny dílčí úkoly splněné. Nikdo ale nedržel v ruce skutečné zařízení, které vydrží zapnuté celý den. Když krabice od výrobce konečně dorazily, z padesáti objednaných kusů dorazilo čtyřicet, z nich třicet pět selhalo hned po vybalení a zbytek umíral během dní. Premiéra se rušila na poslední chvíli

> a odložila o půl roku, v okamžiku, kdy už do ní byly nality peníze i reputace. Zpětně bylo snadné vidět, že signály existovaly dávno: výrobce odkládal odeslání vzorků, hlášení o výtěžnosti výroby byla vyhýbavá a nikdo z projektu neměl v ruce fungující kus poslední verze. Zelený status stál na tvrzeních. Skutečné ověření by stálo na jediné otázce: „ukažte mi deset zařízení, která běží týden.“

Někdy produkt přestane umět něco, co už jednou fungovalo.

> **Z PRAXE**
>
> V jednom systému bylo zpracování naskenovaných PDF pomocí OCR (strojového čtení textu z obrázků) ověřeno ve chvíli, kdy byla komponenta dokončena. Fungovalo, tak proč to zkoušet znovu. O několik týdnů později běžná aktualizace softwarových knihoven změnila kompatibilitu a OCR se tiše rozbilo. Nikdo si toho nevšiml, protože tato kritická schopnost nebyla součástí automatického regresního testu a ruční průchod ji dlouho neobsahoval. Přišlo se na to náhodou, při komplexním průchodu po pěti týdnech.

To, že něco jednou fungovalo, *neznamená*, *že to funguje pořád*.

Proto je heslo „důvěřuj, ale prověřuj“ užitečné jen tehdy, když se správně chápe. Nejde o nedůvěru k lidem ani o mikromanagement. Tým popisuje realitu ze svého místa. **TDL** si proto musí věci *ověřovat i z jiných zdrojů* a ideálně si je vyzkoušet sám. Jinak snadno převezme stejné předpoklady, na kterých staví tým.

Aby tým vnímal průchody stejně (jako kontakt s realitou, ne jako kontrolu lidí), pomáhá *domluvit se na nich hned na začátku*. Mluví se o nich od prvního dne, mají stálý a předvídatelný rytmus, nálezy se vracejí týmu jako

společný materiál k řešení, nikoli jako výtka. **TDL** u nich přiznává i vlastní omyly a slepé uličky. Kontrolou se průchody stávají teprve tehdy, když se objeví bez ohlášení a jejich výsledky se začnou používat proti lidem. To je rozdíl, který tým pozná neomylně.

NEŽ ZAČNETE NOVOU KAPITOLU

*– **Report je tvrzení.** Běžící produkt, reprezentativní data a provozní signály jsou důkaz.*

*– **Držte si dva až pět stálých průchodů** systémem: jeden „žije to?“, několik běžných cest, jeden záměrně ošklivý.*

*– **Rozbité společné prostředí není drobnost.** Každému může jeho část lokálně běžet, i když celek nefunguje.*

*– Věty **„za dva dny to nahodíme“ a „už je to skoro hotové“** jsou vážné varování, když se opakují a v aplikaci se nic nemění.*

*– **Co jednou fungovalo, nemusí fungovat stále.** Důležité funkce je potřeba zkoušet opakovaně.*

*– **Otázka na závěr:** kdy jste naposledy sami prošli produkt očima zákazníka, a co jste u toho našli?*

6. Autonomie ano, neviditelnost ne

Silné týmy nepotřebují, aby jim někdo předepisoval každý krok. Potřebují rozumět cíli, kvalitativním hranicím a tomu, kdy musí eskalovat. Někdy se ale pod autonomií schovává předpoklad, *že si zkušené týmy složité věci mezi sebou nějak vyřeší*.

Z PRAXE

V jednom vícevrstvém AI produktu dostal tým AI backendu odpovědnost definovat rozhraní vůči aplikační vrstvě. Předpokládalo se, že zkušené skupiny se dohodnou, vždyť to byli senioři a dohoda o rozhraní je běžná práce. Tři až čtyři týdny přicházely na pravidelných schůzkách uklidňující odpovědi, že se na tom pracuje. Teprve hlubší kontrola přímo v kódu odhalila, že použitelný návrh prakticky neexistuje. Aplikační tým mezitím čekal a stavěl na provizoriích. AI tým pracoval na vlastních úkolech a vůbec nevnímal, že blokuje celek. Problém nebyl v lenosti ani v chybějící inteligenci. Kritický výsledek neměl dostatečně viditelnou podobu, termín a společné vlastnictví.

Náprava stojí za dovyprávění, protože byla nepříjemnou částí příběhu. Nešlo o výtku na poradě, šlo o změnu pravidel. Oba týmy se sešly nad jedním konkrétním průchodem: tady vzniká požadavek, tudy má protéct, tohle má přijít zpátky. Návrh rozhraní pak vznikl během jednoho týdne prací dvojice složené z obou týmů a ztracené týdny se dohnaly zjednodušením méně důležité části rozsahu. A do projektu přibylo pravidlo, které už zůstalo: každý kritický mezitýmový výsledek musí mít viditelnou podobu (konkrétní věc, kterou lze otevřít a ukázat) a k tomu termín a jméno člověka, který za něj ručí. Bez ohledu na to, jak zkušené týmy na obou stranách sedí. Mimochodem, jde o tentýž

> devadesátidenní projekt z kapitoly 2: i dodávka, která nakonec stihla termín s předstihem, měla svá zaseknutí.

Tady byl chybný už předpoklad, že se zkušené týmy samy domluví. AI tým se v integraci tolik nevyznal a při své práci si nevšiml, že ostatní čekají na něj. Není potřeba kvůli tomu kontrolovat každou drobnost. U rozhraní, na kterém stojí další práce, ale musí být brzy vidět *konkrétní návrh* a obě strany si ho musí vyzkoušet.

U rozhraní to může být návrh API kontraktu, příklad konkrétní zprávy, contract test (automatická kontrola, že se obě strany rozhraní stále shodují) nebo jednoduchý průchod přes obě části systému. Otázka „jste domluveni?“ získá příliš snadno optimistickou odpověď. Otázka „ukažme si dnes jeden skutečný průchod“ vede ke *konkrétní ukázce*, nad kterou se dá mluvit. Návrh zůstává týmům, **TDL** si ověří, že se podle něj mohou obě strany pustit do práce.

Podobně se pracuje s riziky.

> **Z PRAXE**
>
> Ve stejném projektu se za malé riziko považovala dostupnost anonymizovaných testovacích dat. Zákazník ujišťoval, že dat má dost, takže se předpokládalo, že je dodá. Ve skutečnosti dorazily řádově jednotky testovacích sad, a část z nich v tak nízké kvalitě, že se na nich nedalo smysluplně pracovat. Bez reprezentativních dat nebylo možné spolehlivě vyhodnotit chování systému, tvorbu dokumentů ani rozhodovací logiku. Riziko označené na začátku jako minoritní začalo určovat kvalitu celého výsledku. Selhání přitom nespočívalo jen v původním hodnocení. Chyběla důsledná opakovaná eskalace: kdyby se

> počet, kvalita a termín dodání dat od začátku pravidelně vracely na společná jednání se zákazníkem, mohl zákazník včas přesměrovat kapacitu svých lidí, nebo riziko vědomě přijmout. Takto se o něm jen vědělo.

Riziko bez vlastníka a data dalšího rozhodnutí není řízené. Je *pouze zapsané*.

Když má tým opustit oblíbené řešení, věnovat víc času testům nebo přednostně dodělat API kontrakt, potřebuje vědět proč. Nestačí mu oznámit změnu úkolů. Je potřeba *vysvětlit, kdo na tom čeká* a co se stane, když se to neudělá. A potom si ověřit, že změna skutečně pomohla.

Autonomie má také hranici pravomoci. **TDL** může dostat mantinely pro technické a kapacitní rozhodování, ale nesmí sám přijmout významné finanční, právní nebo bezpečnostní riziko, pokud k tomu nemá mandát. Organizace proto potřebuje krátkou eskalační cestu, jakýsi *červený telefon*. Uvnitř hranic se rozhoduje rychle. Při jejich překročení se stejně rychle ví, kdo nese důsledky.

Mandát přitom nevzniká sám od sebe. Stojí za to si jej při vstupu do projektu vyjednat explicitně: co **TDL** rozhoduje sám (technologické volby v dohodnutém rámci, pořadí práce, nasazení kapacity do určité výše), co jen doporučuje a kdo rozhoduje zbytek, a kam přesně vede červený telefon. Deset minut takového rozhovoru na začátku ušetří týdny nejasností uprostřed. Je to nejlevnějších deset minut celého projektu.

S mandátem souvisí i jedna hranice, kterou je dobré vyslovit natvrdo. **TDL** má právo (a povinnost) vetovat závazek, který není proveditelný: říct, že

jej v daných hranicích nelze odpovědně splnit, a doložit proč. Nemá však veto nad obchodní strategií. Firma může vědomě přijmout ztrátovou zakázku kvůli referenci nebo vstupu na nový trh. To je legitimní rozhodnutí vlastníka, ne selhání. Úkolem **TDL** je zajistit, aby se takové rozhodnutí dělalo s pravdivým obrazem nákladů a rizik, ne aby je dělal sám.

NEŽ ZAČNETE NOVOU KAPITOLU

– ***U kritického rozhraní chtějte včas vidět návrh a společný průchod.*** *Na otázku „jste domluveni?“ se příliš snadno odpoví ano.*

– ***Riziko je řízené teprve tehdy,*** *když má vlastníka, srozumitelně popsaný dopad a datum dalšího rozhodnutí.*

– ***Eskalovat opakovaně není otravování.*** *Někdy je potřeba připomenout riziko několikrát, aby se opravdu řešilo.*

– ***Tým musí vědět, co může rozhodnout sám*** *a za kým jít, když jeho pravomoc nestačí.*

– ***Otázka na závěr:*** *které kritické riziko vašeho projektu má dnes vlastníka a datum dalšího rozhodnutí, a které jen záznam v tabulce?*

7. Nejdřív lešení, potom fasáda

Na začátku projektu je velké pokušení ukázat viditelnou funkcionalitu. Je pochopitelné: zákazník chce vidět pokrok a tým chce mít pocit, že staví produkt. Přesto může být nejhodnotnějším prvním výstupem něco, co koncový uživatel nikdy neuvidí. Reprodukovatelné prostředí, pipeline (automatická cesta, kterou se změny kódu sestavují, testují a nasazují), emulátor nebo mezitýmový API kontrakt mohou vypadat jako zdržení, ale *tým díky nim nebude později čekat*.

Tomuto zázemí říkejme backstage, zákulisí (název si půjčujeme od stejnojmenného open source projektu firmy Spotify). Jako v divadle: divák do zákulisí nevidí, ale bez něj se představení hraje špatně. Myšlenka je jednoduchá: nový člověk dokáže získat kód, spustit celou potřebnou sestavu systému, nahradit svou komponentu lokální verzí a projít základní scénář *bez osobní asistence autora*. Pokud každý vývojář skládá prostředí ručně, dělá se stejná práce pořád dokola. Navíc má každý trochu jinou sestavu a chyba konfigurace se snadno vydává za chybu produktu.

> **Z PRAXE**
>
> V praxi taková backstage vypadá třeba takto: jedním příkazem se na vývojářském počítači spustí celé řešení: databáze, frontend, aplikační backend, AI backend i brána k jazykovým modelům. Tester si pustí všechno a může kdykoli klikat celou aplikací. Frontendový vývojář si vypne jen předpřipravený frontend, nahradí ho svou rozpracovanou verzí a zbytek systému mu běží dál. Backendový vývojář udělá totéž se svou částí a přes frontend si okamžitě ověří, že vše spolupracuje. Nováček nepotřebuje dva dny asistence autora. Potřebuje návod na

> jednu obrazovku. A autor nemusí potřetí odpovídat na otázky, na které odpovídal minulý měsíc. Zní to jako samozřejmost. Na překvapivě mnoha projektech to samozřejmost není a každý člen týmu si žije ve vlastním, ručně poskládaném světě, který se s ostatními potká až při integraci. Tedy pozdě.

Taková investice však není automaticky správná. U několikaměsíčního projektu s následným supportem, střídáním lidí nebo produktovým charakterem se pravděpodobně vrátí mnohokrát. U jednorázové práce do přibližně tří měsíců ve dvou lidech může být plnohodnotná backstage luxusem. Záleží na tom, kolikrát se prostředí použije, kolik lidí na projekt přijde a *kolik času se ušetří* oproti jeho tvorbě a údržbě.

Stejné je to s emulátory. Externí systém může být dostupný pozdě, pouze v interní síti nebo jen prostřednictvím dokumentace. Čekat na reálné propojení znamená posunout největší nejistotu na konec. Malý emulátor relevantních metod, datových stavů a chyb dovolí týmům postupovat nezávisle a včas si vyzkoušet, *jak má propojení fungovat*. Emulátor zákaznického ERP systému s deseti metodami rozhraní se dnes s podporou AI dá postavit za dny, a přesto ušetří týdny čekání. Jeho stavba má vedlejší přínos: kdo emulátor staví, musí zákaznickému systému skutečně porozumět, a tím rozumí i celku.

Emulátor ale bude umět jen to, co do něj někdo naprogramuje. Pokud vždy odpovídá rychle, má čistá data a zná jen šťastnou cestu, vyrobí *falešnou důvěru*. Musí zachytit latenci, chyby, neúplnost a omezení, která jsou pro výsledek podstatná. A před UAT (uživatelským akceptačním testováním) musí být průběžně porovnáván se skutečným rozhraním.

Zvláštní pozornost vyžaduje akceptace proti emulátorům. Tak proběhl i devadesátidenní projekt z kapitoly 2. Může být naprosto legitimní, pokud zákaznická strana nebude připravena a smlouva tuto možnost výslovně připouští. Taková akceptace potvrzuje chování vlastní dodávky v dohodnutém modelu. Nepotvrzuje produkční připravenost reálné integrace. Následná integrační a stabilizační fáze proto není drobný úklid, ale jiný druh ověření.

> **Z PRAXE**
>
> Že „lešení“ není jen vývojářské pohodlí, ukázala výroba platebního zařízení. Původní úvaha zněla: software je nezávislý na výrobě, vyrobí se hardware a software se do něj nahraje. Realita byla opačná. Pro výrobní linku bylo nutné vyvinout speciální verzi operačního systému a testovacích aplikací, kterými každé jednotlivé zařízení na lince prošlo kontrolou všech periferií, a teprve pak smělo k zákazníkovi. A při výrobě se do každého kusu musel bezpečně nahrát ostrý software včetně bezpečnostních klíčů. Software se tak ukázal být součástí výrobní linky stejně jako lisovací formy: neviditelné lešení, bez kterého by se tisíce kusů týdně nedaly vyrobit. Nikdo z koncových zákazníků tuhle práci nikdy neviděl. Bez ní by neexistoval produkt.

Technický spike (krátký, časově ohraničený experiment) má podobný účel. U nové technologie ověřuje to, *co by při neúspěchu způsobilo největší problém* dříve, než na ní začne záviset celý plán. Dobrý spike má otázku, časový limit, kritérium a záložní cestu. V komerční nabídce může být spojen s rozhodovací branou. U pevně vysoutěžené zakázky lze neznámou část ocenit konzervativněji a část rezervy použít na ověření nové varianty. Pokud se přínos nepotvrdí, pokračuje se ověřenou technologií.

U experimentu musí být předem jasné, *kdy skončí a co se udělá, když nevyjde*. Jinak může tým zkoumat novou technologii tak dlouho, až nestihne dodat. U backstage zase musí být někdo, kdo ji opravdu používá. **TDL** proto neobhajuje práci na zázemí tím, že „se to tak má dělat“. Musí umět říct, které opakované čekání odstraní, kolika lidem pomůže a jak brzy se přínos projeví.

NEŽ ZAČNETE NOVOU KAPITOLU

– ***Nejhodnotnější první výstup projektu často nikdy neuvidí koncový uživatel:*** *prostředí, pipeline, emulátor, kontrakt.*

– ***U backstage počítejte, kolikrát se použije*** *a kolik času ušetří oproti tomu, co stojí její tvorba a údržba.*

– ***Emulátor umí jen to, co do něj naprogramujete.*** *Musí umět i latenci, chyby a špinavá data, jinak vyrábí falešnou důvěru.*

– ***Dobrý spike má otázku, časový limit, kritérium a záložní cestu.*** *Musíte vědět, co uděláte, když nevyjde.*

– ***Otázka na závěr:*** *jak dlouho by u vás novému člověku trvalo spustit celý systém bez cizí pomoci?*

8. Riziko, kterému rozumí i obchod

Technické riziko je potřeba vysvětlit tak, *aby mu druhá strana rozuměla*. Obchodník, zákazník nebo vedení nepotřebují slyšet dlouhou obhajobu složitosti integrace. Potřebují vědět, co se může stát, proč, jaký bude dopad, jaké existují varianty a do kdy je nutné rozhodnout.

U zákaznické závislosti nestačí věta, že „dodání dat je rizikové“. Užitečný popis říká, jaká data jsou potřeba, *v jakém množství a kvalitě*, který tým je má připravit, jakou alokaci to vyžaduje a k jakému datu. Teprve potom lze s konkrétním kontaktem ověřit skutečnou kapacitu. Zákazník může svým lidem změnit priority, riziko vědomě přijmout nebo přizpůsobit projekt. Obecné varování nic takového neumožní.

Stejná zásada platí uvnitř dodavatele. Když tým v první čtvrtině zaostává za očekávaným průchodem a nemá běžící produkt, jde o technický i finanční signál. Odhad a rozpočet nesmějí žít odděleně od produktu. Reforecast, tedy přepočet výhledu, nemá čekat, až se vyčerpá rezerva. Má začít ve chvíli, kdy se změnil předpoklad, rychlost integrace nebo dostupnost zákaznického vstupu.

Rozpočet přitom není jen hlídání čerpání. Už volba optimistického či pesimistického scénáře je rozhodnutím o riziku. Neznámá technologie se může přeložit do rezervy na spike a fallback, nepřipravenost zákazníka do smluvní podmínky, přehnaná varianta funkce do jednoduššího rozsahu. Je to týž mechanismus rozdělení rezervy mezi cenu, smlouvu a rozsah, který popsala kapitola 2. Technická volba vždy někam *přesouvá cenu a odpovědnost*.

Někdy je potřeba vysvětlit i to, *proč se musí zastavit práce* na věcech, které si všichni přejí.

> **Z PRAXE**
>
> V nejkritičtější fázi vývoje platebního zařízení se sešly dva neslučitelné tlaky. Zařízení muselo projít bezpečnostní certifikací karetních asociací. Bez ní neexistoval produkt, jen drahý prototyp. A současně přicházely od marketingu a obchodu stále nové požadavky: změnit umístění čtečky, upravit chování pro uživatele, přidat funkce, které si žádali první zákazníci. Každý jednotlivý požadavek byl rozumný. Dohromady znamenaly, že se certifikace nikdy nedokončí, protože každá změna vracela část ověřování na začátek. Projektové vedení nakonec udělalo rozhodnutí, které nikoho nenadchlo: na šest měsíců zmrazit veškeré nové požadavky a soustředit všechny síly výhradně na certifikaci. Obchodně to znělo jako zastavení pokroku. Ve skutečnosti to byla nejrychlejší cesta k prodejnému produktu. Právě tak se to muselo vedení i obchodu vysvětlit: ne „technici chtějí klid“, ale „každá další změna odsouvá den, od kterého smíme zařízení prodávat“. Certifikace prošla. Kdyby se tehdy nezmrazil rozsah, pravděpodobně by neprošla nikdy.

Někdy už se do požadované ceny nebo termínu vejít nedá. Rozsah se zjednodušil, pořadí práce i tým se promyslely, a pořád chybí desítky procent. Další snížení už znamená *slíbit něco, co nedodáme*. V tu chvíli musí **TDL** říct, kde je hranice a co by se muselo změnit, aby projekt šel dodat. Pokud existuje menší proveditelná varianta, má ji nabídnout.

Jiná otázka nastává, když už projekt běží a ekonomicky se zhoršuje. Mechanické pravidlo „překročení rozpočtu znamená stop“ je příliš jednoduché. Ukončení může přinést smluvní sankce, reputační škodu

nebo zavřít cestu k významné návaznosti. Pokračování může být ještě dražší a pouze oddalovat nevyhnutelné. Rozhodnutí musí zahrnout *náklady dokončení*, *cenu ukončení*, strategické souvislosti a možnost přetvarování. **TDL** dodává podklady o tom, co zbývá udělat, kolik to bude stát a jaká rizika zůstávají. Významnou obchodní ztrátu musí schválit člověk, který k tomu má pravomoc.

Při eskalaci stačí držet jednoduchou osnovu: *co se stalo*, *co to znamená*, jaké jsou varianty, co se doporučuje, kdy je nutné rozhodnout a jaké zbytkové riziko zůstane. Bez variant druhé straně jen předáme problém. A když přijdeme pozdě, nemusí už být z čeho vybírat.

> **UKÁZKA: ESKALACE NA JEDNU OBRAZOVKU**
>
> Co se stalo: Ze třiceti anonymizovaných testovacích sad dohodnutých pro ověření systému jsme k dnešnímu dni obdrželi čtyři. Dvě z nich jsou nepoužitelné kvůli kvalitě.
>
> Co to znamená: Bez reprezentativních dat nedokážeme do akceptace ověřit chování systému na reálných strukturách. Roste riziko chyb, které se projeví až v provozu, a s ním riziko reklamací po nasazení.
>
> Varianty: (1) Váš tým dodá dvacet sad do 15. května. Plán akceptace platí beze změny. (2) Kapacita se na vaší straně nenajde. Ověříme na rozšířených syntetických datech a prodloužíme stabilizační fázi po nasazení o tři týdny. (3) Riziko vědomě přijmete a plán se nemění.
>
> Doporučení: Varianta 1. Nabízíme pomoc našich lidí s anonymizací, aby se dodání zrychlilo.
>
> Rozhodnout do: 10. května. Po tomto datu už změna plánu ovlivní termín akceptace.

> Zbytkové riziko: I s dvaceti sadami zůstává riziko okrajových případů. Pokryje je zesílený dohled v prvních týdnech provozu.

Šest krátkých odstavců. Příjemce nemusí rozumět softwaru, *aby dokázal rozhodnout*, a přesně to je účel. Vejde se to na jednu obrazovku, a přesto je to obvykle nejtěžší text, který **TDL** za měsíc napíše.

Zákazník potřebuje *o problému vědět včas* a rozumět tomu, co s ním lze udělat. Nepříjemný rozhovor se dá odkládat, ale tím se problém nezmenší. Jen ubude času na řešení.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> – ***U rizika popište dopad a možnosti řešení.*** *Domluvte, kdo o něm rozhodne a do kdy.*
>
> – ***Odhad se neaktualizuje, až když dojde rezerva,*** *ale jakmile se změní předpoklad.*
>
> – ***Někdy je největší službou obchodu říct „teď půl roku nic nového“.*** *Musí to ale být řečeno jazykem obchodu: kdy díky tomu bude možné produkt prodávat.*
>
> – ***Při eskalaci navrhněte možnosti řešení.*** *Přijďte dřív, než už nebude z čeho vybírat.*
>
> – ***Nepříjemný rozhovor neodkládejte.*** *Problém se tím nezmenší, jen ubude času na řešení.*
>
> – ***Otázka na závěr:*** *o kterém riziku vašeho projektu zákazník zatím neslyšel jazykem, kterému rozumí?*

9. Být blízko, ale nestát v cestě

TDL musí být technicky důvěryhodný. Nemůže řídit zásadní rozhodnutí pouze přeposíláním názorů specialistů. Potřebuje rozumět datům, integracím, identitě, prostředím, bezpečnostním mechanismům, nasazení i provozu natolik, aby dokázal odhalit nekonzistenci a posoudit důsledek. U AI produktu navíc potřebuje chápat práci s kontextem, evaluace, nedeterministické chování a cenu modelových volání.

Tato šíře ale svádí k hrdinství. Když tým neví, jak postavit kritickou část, zkušený člověk ji může za několik dní vytvořit. Vhodný zásah často opravdu existuje: referenční implementace, sdílený API kontrakt, diagnostický nástroj, emulátor nebo spike může odblokovat více skupin. Rozhodující je, *co po zásahu zůstane*.

Pokud zůstane komponenta, které rozumí jen autor, problém se pouze přesunul. **TDL** bude později potřebný pro každé review, produkční chybu a změnu. Když se objeví problém jinde, nebude na něj mít čas. Proto je potřeba od začátku vědět, *komu se práce předá a kdy*. Ten člověk se na ní má podílet, projít kód spolu s **TDL** a mít k dispozici testy i dokumentaci. A platí to i pro **TDL** samotného: jeho vlastní odchod z projektu je zásah, po kterém něco zůstane. Kapitola 10 ukazuje, co se stane, když se kalibrace role nestihne předat.

> **Z PRAXE**
>
> Osvědčuje se jednoduché pravidlo: **TDL** si vědomě vybírá práci, u které *smí zítra přestat*. Emulátor zákaznického systému, sdílená komponenta, diagnostický nástroj, příprava prostředí. To všechno jsou věci, které

> pomáhají celku, ale snesou den odkladu, kdyby bylo potřeba řešit důležitější problém. Co si **TDL** nikdy nebere, je komponenta na kritické cestě s tvrdým termínem. V okamžiku, kdy by ji vlastnil, přestal by si moci vybrat: hasit požár projektu, nebo dodržet vlastní termín? Obě volby by byly špatně. **TDL** musí mít možnost vlastní práci odložit, když ho projekt potřebuje jinde.

To neznamená, že **TDL** nemá *vlastní závazky*. Ručí za milníky dodávky jako celku, za pravdivost odhadů a jejich průběžných aktualizací, za to, že rizika mají vlastníky, a za to, že problémy vyplouvají včas, ne až když bolí. U vývojáře lze zkontrolovat kód. **TDL** se pozná podle toho, zda projekt došel tam, kam podle jeho slov dojít měl.

Nejužitečnější zásahy nejsou nutně ty, při kterých vznikne nejvíce kódu. Někdy stačí přesně popsat API kontrakt, spojit dva lidi nad jedním běžícím příkladem nebo najít chybu v prostředí, která vytvářela falešný technický spor. Když *problém řeší dva lidé spolu*, naučí se z toho obvykle víc, než když zkušenější úkol sám převezme a dokončí.

Stejně důležité je vědět, kdy do detailu nevstupovat. Programování může být únikem před obtížnou zákaznickou komunikací, nejasným vlastnictvím nebo rozhodnutím o rozsahu. **TDL** může na konci dne ukázat konkrétní commit, zatímco nejdražší blokace projektu zůstala nedotčená. To, že **TDL** celý den programoval, *ještě neznamená, že projektu pomohl* tam, kde bylo potřeba.

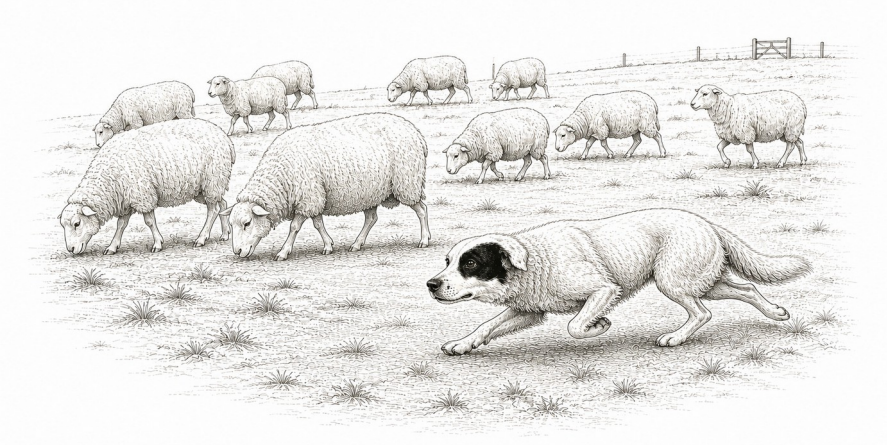
Být vnořený do projektu tedy neznamená kontrolovat každého člověka. Znamená rozumět, proč se práce pohybuje nebo stojí. **TDL** sleduje běžící produkt, kritické API kontrakty, prostředí, rozhodnutí a zákaznické

závislosti. Detail otevírá tam, kde se rozchází tvrzení s realitou. Zbytek nechává vlastníkům.

Tato hranice je i základem rozvoje dalších leadů. Funguje tu obyčejné, léty prověřené pravidlo postupného delegování (žádný vynález této role): první obtížnou situaci řešíme společně, ve druhé vede nový vlastník a zkušenější člověk poskytuje oporu, ve třetí se kontroluje už jen výsledek a způsob uvažování. Nový lead se má naučit *rozhodovat sám*, vysvětlit, z čeho vychází, a změnit názor, když zjistí, že předpoklad neplatí.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> – ***Rozhodující u technického zásahu není, co vznikne, ale co po něm zůstane.*** *Předem domluvte, komu a kdy práci předáte.*
>
> – ***Berte si práci, u které smíte zítra přestat.*** *Nikdy komponentu na kritické cestě s tvrdým termínem.*
>
> – ***To, že jste celý den programovali,*** *ještě neznamená, že jste řešili nejdůležitější problém projektu.*
>
> – ***Rozvoj nástupců:*** *poprvé společně, podruhé s oporou, potřetí jen kontrola výsledku a způsobu uvažování.*
>
> – ***Otázka na závěr:*** *kterou svou práci byste dnes nemohli odložit, kdyby projekt začal hořet jinde?*

10. Tempo, které tým unese



Tahle kapitola v první verzi knihy nebyla. Vznikla z reakcí lidí, kteří se mnou na projektech pracují a text četli. Říkali věci, které do sebe na první pohled nezapadají: že vedení je příliš rychlé a chaotické a nestíhají ho. Že je inspirativní a nakažlivé, a ještě nikdy je práce tak nebavila. Že s podporou AI nedokážou dělat víc než šest hodin denně. A že by v knize chtěli vidět projekt, který dopadl dobře a nikdo u toho nevyhořel. Všechny ty věty zazněly o stejných projektech, často ve stejném týmu. A všechny jsou pravdivé. Je to také jediná kapitola, kterou píšu v první osobě, protože se týká víc mě než role.

Začnu přiznáním. Když se řeší problém, přepínám do režimu, kterému říkám sprint: tři až čtyři týdny, deset až čtrnáct hodin denně, víkendy nevyjímaje. Mně to vyhovuje, umím to a mám z toho radost. Je to ta radost ze hry, o které byla řeč u Rexe na obálce. Jenže Rex vydrží běhat

celý den a většina lidí v týmu ne. Nemají a nemusí. Ovčácký pes, který by po stádu chtěl svoje tempo, by ho rozehnal.

Sprint je můj, rytmus je týmu

Z toho plyne první pravidlo: *sprint je můj, rytmus je týmu*. Tempo, které si **TDL** zvolí sám pro sebe, je jeho osobní volba a nikdy se nesmí mlčky stát normou pro ostatní. To znamená něco velmi konkrétního. Práce, kterou **TDL** udělá v noci nebo o víkendu, se má týmu ráno projevit konkrétně (vyřešené rozhodnutí, odstraněná překážka, připravené prostředí), a ne jako čtyřicet zpráv a přehozené priority. Rozhodnutí se dávkují do denního rytmu, priority se nemění uprostřed dne, a když má tým pracovat o víkendu, je to výjimka, o které se rozhodne nahlas a předem, ne důsledek toho, že **TDL** o víkendu pracoval. Seržant z kapitoly 4 drží tempo pochodu. Nemá se z něj stát člověk, který ho každou hodinu mění.

Tohle mi nejde samo. Sprint svádí k tomu přenášet vlastní tempo na okolí. Člověk si to obvykle neuvědomí, dokud mu to někdo neřekne. Právě proto stojí za to nechat si od týmu říkat, jak vedení vnímá. Dvě protichůdné odpovědi („chaos, nestíhám“ a „konečně to má tah“) nejsou rozpor v týmu. Jsou to dva různí lidé, kteří potřebují dvě různé věci: ten první stabilní rytmus a jasný úsek terénu, ten druhý prostor přidat, když sám chce. Oba jsou dobří. *Chyba je chtít po obou totéž.*

Šest hodin s AI

Druhý poznatek se týká AI a byl pro mě překvapivý. Někteří vývojáři říkají, že s podporou AI nedokážou pracovat déle než šest hodin denně, že je to vyčerpává víc než klasické programování. Zpočátku jsem to považoval za

výmluvu. Není to výmluva. *Práce s AI je skoro bez oddechu*: člověk nepřetržitě čte cizí kód, rozhoduje, opravuje směr a hlídá, zda výsledek dává smysl. Chybí v ní mechanické úseky, u kterých si hlava odpočine: rutinní části, které dřív tvořily půlku dne, teď píše model. Kapitola 4 říká, že úzké místo se s AI přesunulo do review a porozumění. A právě tahle práce člověka hodně vyčerpává.

Prakticky to znamená plánovat s *pěti až šesti hodinami hloubkové práce* denně, ne s osmi, a střídat ji s jiným druhem činnosti: návrhem, testováním, rozhovorem se zákazníkem, průchodem produktem. A sledovat signály přetížení, které jsou vidět zvenku: review se zkracuje na „schválit a jet dál“, do kódu proklouzávají chyby, které by odpočatý člověk viděl, a lidé přestávají klást otázky. Když tyhle signály přehlížíme, přibývají chyby, které budeme později opravovat. Je to další zdroj dluhu z kapitoly 4.

Když se kalibrace nepředá

> **Z PRAXE**
>
> Jeden projekt mě naučil, jak to dopadá, když se kalibrace nepředá. Uprostřed analýzy jsem z něj musel ze dne na den odejít: vyšší moc, která mě na devět měsíců vyřadila z práce úplně. Nedalo se s tím nic dělat a nedalo se ani nic předat, stalo se to v jednom okamžiku. Projekt převzal jiný tým s novým vedením. Zadání znali, kontext měli. Co s sebou neměli, byla kalibrace „dost dobrého“ z kapitoly 3: ta žila v mé hlavě a v několika rozhodnutích, která jsem ještě nestihl udělat nahlas.
>
> Tým narostl na deset lidí: čtyři na frontendu, dva na aplikačním backendu, čtyři na AI části. Úkoly se řešily pečlivě a sofistikovaně, často

> pečlivěji, než zadání chtělo. Projekt se stihl. Rozpočet přetekl zhruba o polovinu (o stovky člověkodnů) a lidé z něj odcházeli vyčerpaní. U některých exponovaných rolí šlo o částečné vyhoření, ze kterého se budou nějakou dobu dostávat. Jejich stížnosti byly oprávněné. Jen příčinou nebyla lenost ani špatná práce. Příčinou byl tvar týmu (deset lidí je nad hranicí, o které mluví kapitola 4) a ztracená kalibrace toho, co je dost dobré. Větší tým v tomhle případě znamenal víc koordinace, víc review a víc rozpracovaného najednou. K vyčerpání tak přispěl. Pro srovnání: projekt platebního zařízení přetekl ještě víc, z roku na tři, jenže tam byl příčinou špatný předpoklad o tom, co se vlastně dodává (kapitola 2). Tady šlo o velikost týmu a chybějící shodu na tom, co už je dost dobré.
>
> A poučení pro mě: pravidlo z kapitoly 9, že je potřeba myslet na předání práce, platí i pro **TDL** samotného. Můj odchod nebyl volba, ale na věci to nic nemění: kalibrace role, která existuje jen v jedné hlavě, je riziko projektu stejně jako komponenta, které rozumí jen autor. Od té doby se snažím rozhodnutí o rozsahu a o „dost dobrém“ říkat nahlas a brzy, zapisovat je a mít vedle sebe člověka, který by je uměl zopakovat. Další nečekaný odchod naplánovat neumím, ale tohle ovlivnit můžu.

Rok a půl pod tlakem

Sprint trvá týdny. Některé projekty ale trvají roky a tlak v nich nepolevuje, a to je jiná disciplína než sprint. U platebního zařízení, o kterém byla řeč v kapitolách 2 a 8, jsme po roce a půl práce museli v softwarové divizi zabrzdit i uvnitř týmu. Lidé pracovali naplno, jenže *dosavadní tempo už nikdo neunesl*: termíny vymyšlené před mnoha měsíci u tabule v zahraniční centrále, požadavky měněné za běhu a na poslední chvíli, stále nepodepsaná smlouva, kvůli které si zákazník mohl diktovat další a další

změny, chronický nedostatek zařízení pro vývoj i testování, a k tomu politika mezi týmy a hledání viníků.

Softwarová divize byla přitom sama soustavou týmů: platforma operačního systému, výrobní tým (testovací verze systému pro fabriku a nástroje pro bezpečné nahrávání klíčů), aplikační tým, testovací týmy zvlášť pro systém a pro aplikace, analytická jednotka a support. Nad tím programové vedení, vedle toho hardwarová větev s vlastními dodavateli čteček, antén i samotného zařízení. Přesně to vícestupňové řízení, které kapitola 4 popisuje jako opakování téže struktury o úroveň výš. Jenže *po roce a půl už byli lidé vyčerpaní*.

> **Z PRAXE**
>
> Udělal jsem tehdy dvě věci, které spolu souvisely víc, než se na první pohled zdálo. První byl projev k celému týmu. Začal jsem ho úryvkem z novoročního projevu Václava Havla z ledna 1990, toho, ve kterém říká: „Předpokládám, že jste mne nenavrhli do tohoto úřadu proto, abych vám i já lhal. Naše země nevzkvétá.“ Ten projev na mě tehdy působil jako něco z jiného vesmíru: naprosto neuvěřitelně a zároveň nesmírně motivačně. Sáhl jsem po něm záměrně. Mluvil jsem k týmu zhruba čtyřiadvacet let poté a velká část lidí v sále byla na sametovou revoluci příliš mladá. Těžko mohli cítit sílu okamžiku, kdy hlava státu ve svém prvním projevu řekne lidem pravdu místo útěchy. Do žádného úřadu mě samozřejmě nikdo nenavrhl ani nevolil, bral jsem to jako metaforu. A pak jsem řekl, že ani já týmu nehodlám lhát. Že situace na projektu není jednoduchá. Že přání manažerů jsou otcem mnoha termínů, které se nedají splnit. Že náš vlastní optimismus nás opakovaně zklamal a že řada podmínek, ve kterých pracujeme, je prostě špatně. Teprve potom přišlo, co se povedlo: software na zařízení skutečně fungoval a poslední

> dodávka odešla stabilní, se známými chybami, ale v dohodnutém termínu. A co se mění: hledání viníků končí, politiku si beru na sebe já, každý tým zavede denní stand-up, a kdo vidí, že se něco dá dělat lépe, udělá to a nečeká na povolení.
>
> Druhá věc byla reorganizace, oznámená v tomtéž projevu. Termíny, plány a všechna důležitá komunikace navenek přešly na jednu kolegyni: jeden hlas a jedno místo, kam se chodí ptát. Já jsem si vzal politiku vůči vedení a zákazníkovi a k tomu nejkritičtější část: tým operačního systému. A padla jediná priorita, které se muselo přizpůsobit úplně všechno, včetně struktury testovacích a aplikačních týmů: bezpečnostní certifikace, bez níž zákazník smlouvu nepodepíše. Je to totéž rozhodnutí, o kterém mluví kapitola 8. Tady je vidět jeho druhá, lidská strana.

Zpětně vidím, proč to fungovalo. Tým už slyšel povzbuzování mnohokrát. Teď potřeboval vědět, že vedení vidí stejné problémy, že certifikace má přednost před vším ostatním a že je jasné, *kdo o čem rozhoduje*. Tím ubylo dohadování a lidé se mohli soustředit na práci. I pochvala musela být konkrétní. Poslední dodávka skutečně odešla stabilní a včas, to si každý mohl ověřit. Bylo se o co opřít, když jsme mluvili o tom, co bude dál.

Jak vypadá úspěch bez vyhoření

Lidé po přečtení první verze chtěli vidět projekt, který dopadl dobře a nikdo u toho nevyhořel. *Nemám takový příklad*. I devadesátidenní projekt z kapitoly 2, který skončil akceptací s předstihem, stál některé exponované role víc, než bylo zdrávo, a bylo by nepoctivé to zamlčet. Umím ale říct, podle čeho se takový projekt pozná. A je to seznam, na který se dá dívat během projektu, ne až po něm.

Sprint je soustředěný v **TDL** a v lidech, kteří se do něj přihlásili sami, ne rozprostřený mlčky po celém týmu. Tým má stabilní denní rytmus a víkend je jeho, pokud nepadne výslovné a zdůvodněné rozhodnutí jinak. Plánuje se s reálnou kapacitou hloubkové práce, ne s osmi hodinami na papíře. Hypercare má předem daný konec. A po projektu platí nejjednodušší metrika, jakou znám: *lidé se hlásí na další projekt s vámi*. Kdo si po dodávce potřebuje od **TDL** na půl roku odpočinout, sice dodal, ale o způsobu vedení tím řekl něco, co žádný zelený status nepřebije.

A ještě jedno přiznání, aby bylo jasné, odkud to všechno pramení. Mě tyhle projekty baví. Ty těžké, které nejdou podle plánu a kde se každý týden řeší problém, se kterým nikdo nepočítal. Platební zařízení, o kterém tu píšu jako o projektu s přetečeným rozpočtem, zmrazeným rozsahem a unaveným týmem, byl nejhezčí projekt, jaký jsem v životě dělal. Věta, která se připisuje Johnu Adamsovi, má pro mě obrovskou sílu: *každý problém je převlečená příležitost*. Jen je potřeba vědět, že to je věta pro **TDL**, ne požadavek na tým. Radost z těžkých věcí se nedá nařídit. Může se přenést na ostatní, když ji na vás vidí a současně vědí, že je nenutíte běžet vaším tempem.

Tahle kapitola není omluva za tempo. Rychlost je v této práci často jediná cesta k termínu a nehodlám předstírat, že to jde jinak. Je to připomínka, komu tempo patří. **TDL** si smí zvolit svoje. Týmu má nastavit rytmus, který unese, a pak ho držet i ve chvíli, kdy sám běží.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> *– **Sprint je můj, rytmus je týmu.** Tempo, které si **TDL** zvolí sám, se nesmí mlčky stát normou pro ostatní.*

> *– **Noční práce TDL má týmu ráno pomoct pokračovat.** Nemá po ní zůstat čtyřicet zpráv a přehozené priority.*
>
> *– **Šest hodin hloubkové práce s AI není lenost.** Je to poctivá kapacita, se kterou se má plánovat.*
>
> *– **Větší tým může k vyčerpání přispět.** V popsaném projektu přibylo koordinace, review i rozpracované práce.*
>
> *– **Unavenému týmu samotné povzbuzení nestačí.** Potřebuje vědět, co má přednost a kdo o čem rozhoduje. Chvalte konkrétní práci.*
>
> *– Na předání musí myslet i **TDL**. Kalibrace „dost dobrého“, která žije v jedné hlavě, je riziko projektu.*
>
> *– **Otázka na závěr:** kdo z vašeho týmu pracoval minulý víkend, a kdo o tom rozhodl?*

11. Projekt nekončí akceptací

Dodávka může splnit smluvní milník, a přesto *nebýt připravena na provoz*. UAT proběhne na omezených datech, nasazení do provozu vyžaduje několik ručních kroků a produkční problém dokáže diagnostikovat jen autor. Pokud se projekt v tomto okamžiku prohlásí za hotový, řada rozhodnutí, která měla padnout ve vývoji, se jen odsunula do provozu.

Zkušenost s provozem mění architekturu už na začátku. Vedle otázky, zda systém udělá správnou věc, se objevuje otázka, *jak selže*, jak se problém pozná, kdo jej uvidí a jak se služba obnoví. Ne každý produkt potřebuje extrémní dostupnost nebo složitý multi-cloud. Každý však potřebuje úroveň diagnostiky a obnovy odpovídající dopadu svého výpadku.

> **Z PRAXE**
>
> Kdo někdy osobně provozoval systém v režimu 24/7, navrhuje jinak. Zkušenost je prostá: dohledová služba dokáže maximálně zjistit, že něco neběží, a zkusit restart. Všechno ostatní (diagnóza, oprava, nahození) končí u autora. Kdo si tedy do systému zabuduje součást, která občas sama od sebe spadne a potřebuje ruční zásah, kupuje si vlastní noční telefonáty. Z této zkušenosti roste návyk navrhovat systémy jako samozotavitelné: služba, která selže, se sama restartuje, ucpaná fronta se sama vyprázdní a začne znovu a rozbitý stav se dá obnovit bez archeologie. Není to estetika. Je to čistá *ekonomika vlastního spánku*, a mimochodem i ekonomika zákazníkova provozu.

Samozotavení má ale hranici, a právě za ní se pozná kvalitní návrh. Některé situace se technicky vyřešit nedají: dvě strany si nerozumí,

odpověď se ztratila, nikdo neví, v jakém stavu věc skončila. Špatný návrh v tu chvíli spadne nebo zavolá vývojáře, obvykle v noci a obvykle tomu, kdo má zítra dovolenou. Dobrý návrh počítá s tím, že se to stane, a připraví pro ten okamžik cestu, která nevede přes softwarový tým: *opakování*, *bezpečné ignorování*, *nebo předání člověku, který na to má proces a pravomoc*. Kapitola 3 říká, že na schopnosti obnovy se šetřit nesmí. Tady je vidět, co to znamená.

Z PRAXE

Platební terminál pošle bance data z karty a částku a čeká na odpověď: schváleno, nebo zamítnuto. Jednou za čas odpověď nedorazí. Spojení se přeruší a nikdo nemůže zaručit, že zpráva došla, ani že nedošla. Terminál v tu chvíli neví, zda zákazník zaplatil, a žádný kód to nezjistí. Řešení má dvě části. Nejdřív se terminál třikrát pokusí transakci stornovat. Když se storno povede, nic se nestalo a zákazník platí znovu. Když se nepovede ani napotřetí, terminál vytiskne zvláštní účtenku pro obsluhu: transakce je v neznámém stavu, zavolejte na podporu. Tam se podívají z druhé strany, zda platba proběhla, a pokud ano, zruší ji. Technicky neřešitelný problém dostal procesní řešení na jeden telefonát a vývojář se o něm nikdy nedozví, což je u produkčního incidentu nejlepší možný stav vývojářova vědomí.

U samoobslužného automatu, kde účtenku nikdo nepřečte, se totéž řeší jinak. Nejdřív se částka jen zablokuje, pak se vydá zboží a teprve potom se blokace dokončí. Dokončení se dá bezpečně opakovat donekonečna, protože se odkazuje na původní blokaci. Když se ztratí odpověď hned na začátku, automat zboží nevydá a pokusí se blokaci zrušit. Když se ani to nepovede, zůstane zákazníkovi v nejhorším případě na účtu blokovaná

částka, která po několika dnech zmizí sama. Ošklivé? Trochu. Ale nikdo nikam nevolá a nikdo nic ručně neopravuje.

Z PRAXE

Podobně to fungovalo v zázemí platebního systému, kde se v noci konsolidovaly dávky transakcí s protistranami. Servery byly dva, práci dělal vždy jen jeden. Přes společnou databázi se domlouvaly, kdo je zrovna vedoucí, a když vedoucí odpadl, druhý to během minuty převzal a pokračoval tam, kde první skončil. Konsolidace měla popsané všechny stavy, ve kterých může skončit, a když některá dávka nesouhlasila, systém založil tiket pro podporu s přesným popisem, co s protistranou dořešit ručně. Žádné noční restarty, žádné zásahy do databáze, žádný vývojář u telefonu. Systém se v noci nudil, a to je u systému kompliment. Tohle je ta kvalita, na které se nešetří: ne krása kódu, ale to, že systém své běžné poruchy zvládne bez lidí a ty výjimečné předá lidem, kteří na ně mají proces.

Reprodukovatelný release (vydání nové verze, které lze kdykoli zopakovat se stejným výsledkem) k tomu patří také. Musí být známé verze všech částí, konfigurace a cesta, kterou se kandidát dostal do prostředí. Kritické toky mají být ověřené, zbytková rizika pojmenovaná a přijatá správným člověkem. Když rollback (návrat na předchozí verzi) není bezpečný, musí existovat roll-forward (rychlá oprava směrem vpřed), manuální kontinuita nebo jiná konkrétní strategie zotavení.

V prvních dnech po spuštění se rychle ukáže, co při návrhu a testování uniklo. Proto má smysl hypercare, období zesílené péče krátce po spuštění. Vývoj, infrastruktura a podpora spolu rychle řeší *problémy z produkce* a ověřují opravy. Co se opakuje, vrací se do vývoje, aby se změnil

návrh nebo způsob práce. V téhle době má být aktivní i **TDL**. Když část problémů sám vyřeší, lépe rozumí tomu, jak se aplikace v provozu chová a co je potřeba opravit.

> **Z PRAXE**
>
> V praxi se osvědčuje, když první dva až tři týdny po spuštění věnuje část své kapacity produkci sám **TDL**: má přístup do produkčního prostředí stejně jako lidé z infrastruktury, umí se podívat do logů, umí vydat opravu a nejkritičtější zákaznické tikety dokáže v případě nouze odbavit vlastníma rukama. Nedělá to proto, aby nahradil support. Dělá to proto, že zákazník v nejcitlivějším období vidí okamžitou reakci, a **TDL** současně na vlastní oči vidí, jak se vyrobené řešení skutečně chová: co padá, co uživatele mate, které problémy se opakují. Tahle zkušenost se pak vrací do produktu. Takový „babysitting“ má ale mít předem jasný konec. Jinak se z něj stane trvalá závislost na jednom člověku.

Před ukončením hypercare je potřeba ověřit, že *provoz už zvládne běžná podpora*. Kritické toky jsou stabilní, incidenty mají známé vlastníky, support umí klasifikovat a eskalovat, provoz má přístupy a diagnostiku a běžné vydání verze nezávisí na jednom člověku. Pokud **TDL** zůstává nejrychlejší cestou pro každý tiket, předání neproběhlo.

Provozní realita také někdy vyžaduje řešení, která na papíře vypadají podivně, a přesto jsou správně.

Z PRAXE

U zařízení v terénu se čas od času projevila chyba přímo v hardwaru: řadič dotykové obrazovky se za určitých okolností tiše odpojil a přestal hlásit doteky uživatele. Jenže tisíce kusů už byly vyrobené a rozmístěné u zákazníků. Hardware se stáhnout a předělat nedal. Řešením byla softwarová „hlídka“: služba, která stav řadiče průběžně kontrolovala, a když se zachoval podezřele, potichu jej resetovala dřív, než si uživatel čehokoli všiml. Puristé mohou namítat, že se maskuje hardwarová chyba. Jistě. Ale produkt v terénu fungoval, zákazníci platili kartami a oprava hardwaru počkala na další výrobní generaci. Provozní odpovědnost někdy znamená přesně tohle: netrvat na čistém řešení, které není k dispozici, a postavit spolehlivé řešení z toho, co k dispozici je.

Stejná logika platí pro akceptaci proti emulátorům z kapitoly 7. Může být správným smluvním výsledkem, ale připravenost na provoz se ověří až se skutečnými daty, přihlašováním, odezvou a oprávněními. Musí být také jasné, kdo provoz převezme. V projektu se proto musí rozlišovat, *co prošlo akceptací a co je připravené ke spuštění*.

NEŽ ZAČNETE NOVOU KAPITOLU

*– **Akceptace potvrzuje splnění smluvního milníku.** Připravenost na provoz je potřeba ověřit zvlášť.*

*– **Navrhujte, jako byste to sami provozovali.** Každá součást, která potřebuje ruční zásah, je něčí budoucí noční telefonát.*

*– **Během hypercare se problémy z provozu řeší přímo s vývojem.** Než skončí, ověřte, že provoz zvládne běžná podpora.*

*– Pokud je **TDL** i po předání **nejrychlejší cestou pro každý tiket,** předání neproběhlo.*

*– **Co se technicky vyřešit nedá, ať má procesní řešení:** opakování, bezpečné ignorování, nebo tiket pro člověka s pravomocí. Ne telefonát vývojáři.*

*– **Otázka na závěr:** co by se stalo, kdyby produkt v noci spadl a jeho autor byl na dovolené?*

12. Úsudek, který se dá předat

Seniorní člověk přijde do projektu a má pocit, že něco nesedí. Už podobnou situaci zažil, i když si hned nevybaví kde. Takový pocit může být užitečný, ale sám o sobě se nedá předat dál. Je potřeba říct, čeho si člověk všiml, co mu to připomíná a *jak si ověří, jestli se nemýlí*.

Věta „tohle mi nevychází“ je dobrý začátek. Následují otázky. Připomíná situace minulý problém s paralelizací, nebo jen používá stejnou technologii? Je rozdíl v datech, vlastnictví, prostředí, regulaci či rozhodovací rychlosti? Co je fakt a co předpoklad? *Co se dá rychle vyzkoušet*, abychom ověřili největší neznámou? Co by změnilo názor?

Kalibrace vyžaduje také *paměť na omyly*. Pokud se v minulosti problém projevil opakovaným „za dva dny“, nefunkčním společným prostředím a čekáním na API kontrakt, stejná kombinace si příště zaslouží pozornost. Není moudré předpokládat, že tentokrát to bude jiné jen proto, že projekt má nový název. Zároveň není správné z jednoho příběhu vytvořit zákon. Když se opakují stejné signály, je potřeba se na věc podívat. Ještě to neznamená, že má stejnou příčinu.

> **Z PRAXE**
>
> Jeden z nejúčinnějších nástrojů předávání úsudku je překvapivě prostý: veřejně vyprávět vlastní chyby. Když se po letech prezentoval příběh platebního zařízení, nejcennější částí nebyl popis úspěchu (dvě stě tisíc vyrobených kusů), ale očíslovaný seznam omylů. Mysleli jsme, že kupujeme hotový produkt. Ve skutečnosti jsme dělali výzkum. Mysleli jsme, že dostaneme stabilní operační systém. Museli jsme ho přestavět. Mysleli jsme, že zadávání PIN je komodita, kterou někdo prodává v

> krabici. Byla to jedna z nejtěžších částí projektu a skončila patentem. Každý z těch omylů byl v době svého vzniku rozumným předpokladem zkušených lidí. Právě proto má takový seznam větší učící hodnotu než jakýkoli seznam úspěchů: neučí, co si myslet, ale kde si dávat pozor na vlastní jistotu. Tým, ve kterém si vedoucí lidé dovolí takhle mluvit o svých omylech, se z nich naučí víc než tým, kde se chyby tiše zametají.

Každá taková zkušenost má ale svoje meze. Malé týmy se snáz domluví, velký program přesto potřebuje mnoho lidí rozdělených do menších celků. Emulátor ušetří čekání, ale neověří za nás skutečný zákaznický systém. Stejně je potřeba přemýšlet o AI, provozním zázemí i vlastních zásazích do kódu. To, že něco pomohlo minule, nestačí. Je potřeba posoudit, jestli jsou pro to podmínky i tady.

Předávání role nezačíná seznamem technologií. Kandidát musí rozumět tomu, co zákazník potřebuje a *jak to má v systému fungovat*, přiznat nejistotu, nabídnout varianty a zůstat v kontaktu s produktem. Musí dokázat vstoupit do detailu a zase jej opustit. A především nesmí stavět vlastní hodnotu na nenahraditelnosti.

Jestli se to daří, je vidět na projektu. Lidé vědí, kdo o čem rozhoduje, a nečekají na odpověď týdny. Týmy dokážou spustit prostředí bez autora. Nástěnka úkolů odpovídá běžícímu produktu. Riziko je přeložené do dopadu a přijímá je správný člověk. Po incidentu se kromě opravy vyřeší i to, jak problém příště poznat nebo mu předejít a kdo se o něj postará. **TDL** pak může řešit další věci a *nemusí osobně hlídat každou minulou chybu*.

Úspěšný Technical Delivery Lead tedy nemusí být nejviditelnější člověk na projektu. Podstatné je, že se dodává, problémy se řeší včas a tým rozumí

tomu, proč padla jednotlivá rozhodnutí. A že když **TDL** odjede na dovolenou, *práce i provoz pokračují*.

> **NEŽ ZAČNETE NOVOU KAPITOLU**
>
> – ***U intuice vysvětlete, čeho jste si všimli,*** *jakou zkušenost vám to připomíná a jak si ji ověříte.*
>
> – ***Stejné signály nemusí mít stejnou příčinu.*** *Ověřte, jestli minulá zkušenost platí i v tomto projektu.*
>
> – ***Seznam vlastních omylů má větší učící hodnotu než seznam úspěchů.*** *Vyprávějte ho nahlas.*
>
> – *Hodnota **TDL** se nepozná podle toho, kolik věcí musí udělat osobně, ale **kolik dobrých rozhodnutí projekt udělá bez něj**.*
>
> – ***Otázka na závěr:*** *kdy jste naposledy nahlas vyprávěli vlastní omyl, a co se z něj tým naučil?*

Kam se poděl projektový manažer?

Pozorný čtenář si někde kolem čtvrté kapitoly položí otázku, na kterou kniha zatím neodpověděla: kde je v tom všem projektový manažer? A kde jsou zavedené způsoby řízení vývoje, podle kterých se software přece dělá: vodopád s milníky, Scrum, kanban, SAFe? Kniha o nich skoro nemluví. Je tedy potřeba vysvětlit, *jak do popsaných projektů patří*.

Na projektech, o kterých kniha vypráví, projektový manažer je, a to prakticky vždy. Dělá práci, bez které by se projekt rozpadl stejně jistě jako bez fungujícího kódu: projektový plán a jeho údržbu, hlídání termínů, koordinaci mezi týmy a dodavateli, formální stav projektu, akceptační kritéria (co přesně, kdy a v jaké podobě se předává) a komunikaci se zákazníkem včetně papírování, které k veřejným zakázkám patří. Jednání se zákazníkem probíhá téměř vždy ve stejné sestavě: na straně dodavatele projektový manažer a **TDL**, na straně zákazníka jeho projektový manažer. Je to spousta práce a je velmi potřebná. Jen neřeší vlastní dodávku softwaru. Tou se zabývá tato kniha, a proto v ní projektový manažer stojí vedle hlavní postavy, ne uprostřed. Tahle kniha se soustředí na *technickou odpovědnost za dodání*.

Je to rozdělení práce. Projektový manažer hlídá smluvní stránku projektu: rozsah, plán, termíny, akceptaci, peníze a vztah se zákazníkem. **TDL** odpovídá za technickou stránku: proveditelnost, běžící systém a technické riziko. Obě odpovědnosti se potkávají například u statusu. Projektový manažer ho reportuje. **TDL** za něj ručí, že odpovídá běžícímu produktu, a ne nástěnce. Kapitola 4 přirovnává koordinační roli mezi buňkami k seržantovi, který drží tempo pochodu a ví, kdo je právě zablokovaný. Na

většině projektů tu roli drží právě projektový manažer. A pokud metafora někoho svádí k hodnostem, ať ji čte jinak: seržant a poručík nejsou v tom obrazu nadřízený a podřízený, jsou to dvě různé odpovědnosti, které se navzájem kryjí.

> **Z PRAXE**
>
> Nejlepší ukázka té dělby je v kapitole 10, jen tam není tak pojmenovaná. Kolegyně, na kterou po roce a půl projektu platebního zařízení přešly termíny, plány a všechna důležitá komunikace navenek, byla projektová manažerka. Od té chvíle měl projekt jeden hlas směrem ven a jedno místo, kam se chodí ptát, a na **TDL** zůstala politika vůči vedení a nejkritičtější tým. Nebyla to degradace ani povýšení. Bylo to rozdělení dvou prací, které do té doby dělal jeden člověk, a proto obě trpěly. Zpětně se ukázalo, že to bylo nejdůležitější organizační rozhodnutí celého projektu, a přitom se netýkalo ani řádku kódu.

Někdy dělá **TDL** obě role sám, typicky na menším projektu, kde se samostatný projektový manažer nevyplatí. Jde to, ale má to předvídatelnou cenu: *trpí ta půlka, která je tišší*. Plán, akceptační kritéria a zákazník se ozvou sami, protože mají termíny a jména. Proveditelnost, průchod produktem a technické riziko se neozvou, dokud není pozdě. Kdo dělá obě role, musí si na tu tichou půlku vyhradit čas násilím, třeba pevným průchodem produktem každý týden, jinak ho pohltí kalendář.

Kdo roli hledá pod jiným jménem, najde ji nejspíš u velkých technologických firem jako Technical Program Manager: člověk, který vede dodávku přes několik týmů, drží závislosti a rizika a rozumí technice natolik, aby ho nikdo neopil rohlíkem, ale nevlastní kód. V konzultačních firmách se totéž jmenuje Delivery Lead nebo Technical Delivery Manager.

Blízko je i Technical Product Manager, ale ten sedí na druhé straně stolu: rozhoduje, co a proč se má postavit, a v příbězích této knihy to bývá role zákazníka. **TDL** odpovídá za to, zda se to dá postavit a zda to skutečně dojde do cíle. *Názvy i konkrétní rozdělení práce se liší podle firmy.*

A metodiky? Nejdřív drobnost, na které jejich autoři trvají: vodopád je model životního cyklu, Scrum a SAFe jsou rámce, ne metodiky, a kanban je metoda, kterou přiložíte na to, co už děláte. Pro tuto knihu na tom rozlišení nezáleží, a tak jim tady kniha, s omluvou puristům, dál říká metodiky. Kniha není proti žádné z nich a **TDL** funguje uvnitř všech. Organizují práci: kdo co dělá, v jakém rytmu a s jakými obřady. To je užitečné. Nic z toho, o čem je tahle kniha, ale žádná z nich nezaručuje. Ani jedna nezajistí, že odhad je proveditelný, že status odpovídá běžícímu produktu nebo že riziko slyšel zákazník jazykem, kterému rozumí. *Každá metodika má svůj vlastní zelený status.* Sprint review s ukázkou na umělých datech je stejné divadlo jako splněný milník na Ganttově diagramu, liší se jen kostýmy.

U zakázek s pevným termínem a pevnou cenou máme navenek pevné fáze a milníky, protože je vyžaduje smlouva. Uvnitř tečou úkoly kanbanem, pořadí práce určují rizika (nejdřív spike a integrace, které mohou plán zbořit, teprve potom pohodlné části) a pravidelné statusy nejsou denní, ale několikrát týdně nad běžícím produktem. Kdo hledá jméno, nejblíž je tomu OpenUP, odlehčený Unified Process: iterace uvnitř, fáze a milníky navenek, architektura a rizika řídí pořadí práce. Podstatné je *průběžně ověřovat*, jestli se tímhle způsobem práce opravdu blížíme k dodání.

Doslov: proč tento text vznikl

Tu knihu jsem chtěl napsat *ze tří důvodů*.

První důvod je prostý: tyhle zkušenosti jsem sbíral pětatřicet let, od informačních systémů státní správy přes telekomunikace a platební zařízení až po dnešní AI produkty. Za tu dobu se vyměnily všechny technologie, většinou několikrát. Pořád se ale opakovaly stejné problémy. Odhady, které vypadají přesněji, než jsou. Statusy, které svítí zeleně, zatímco produkt nefunguje. Rizika zapsaná, ale neřízená. Hrdinové, kteří pomohli a stali se závislostí. *Chtěl bych, aby si každý nemusel projít vším znovu.* Vlastní zkušenost kniha nenahradí, ale může pomoct poznat, kam se dívat a na co se ptát.

Druhý důvod je AI. Jeden člověk s ní dnes zvládne napsat množství kódu, na které dřív býval potřeba tým. Někdo ale pořád musí *rozumět zadání*, ověřit výsledek a poznat, že se projekt začíná ubírat špatným směrem. Z mojí zkušenosti je právě tahle práce čím dál důležitější.

Třetí důvod je praktický. V naší firmě potřebujeme víc lidí, kteří tuhle roli zvládnou. Nestačí jim dát název pozice a seznam technologií, které mají umět. Potřebují vědět, za co budou odpovídat, *co mají sledovat* a podle čeho se rozhodovat. Proto jsem se pokusil popsat konkrétní zkušenosti, včetně toho, co se nepovedlo. A proto jsou na konci kapitol otázky, které si mohou položit nad vlastním projektem.

Je také fér přiznat, co tato role stojí. Nároky, které na ni kniha klade, mohou poskládané za sebou vypadat jako práce bez vypnutí: průchody,

eskalace, hypercare, neustálá dostupnost tam, kde je největší tření. Ve skutečnosti pro roli platí totéž, co pro projekt: *rytmus je udržitelnější než hrdinství*. Průchod produktem zabere hodiny týdně, ne dny. Hypercare má předem daný konec. A **TDL**, který si nebere komponenty s tvrdými termíny, má paradoxně klidnější kalendář než vývojář pod tlakem sprintu: může si část práce odložit a jít řešit to, co projekt právě potřebuje. Tempo, které si v nárazu dovolí sám, přitom nesmí být normou pro tým. Kapitola 10 to říká natvrdo. Jeden člověk utáhne jednu složitou dodávku, nebo dvě klidnější. Víc znamená, že přestává vidět, a tím přestává plnit svou základní funkci. A co role dává: málokterá práce v oboru nabízí rychlejší učení, viditelnější výsledky celku a větší vliv na to, jak rostou lidé okolo. Kdo ale v práci hledá především dlouhé soustředěné programování, bude v této roli nešťastný, a je lepší to vědět předem než po roce.

S touhle knihou klidně nesouhlaste. Porovnávejte ji s vlastními projekty a přemýšlejte, co by u vás pomohlo a co by naopak nefungovalo. Jestli díky ní dřív poznáte problém nebo se lépe rozhodnete, mělo smysl ji napsat. A pokud bych měl vybrat jednu věc na konec: řekněte si, *co opravdu víte*, co jen předpokládáte a jak si to ověříte. Od toho se dá začít.

Příloha: prvních deset dní v běžícím projektu

Kniha záměrně nepopisuje postupy krok za krokem. Jedna výjimka se ale osvědčuje: checklist pro chvíli, kdy **TDL** vstupuje do projektu, nového nebo rozběhnutého po někom jiném. Není to návod, co dělat celý projekt. Je to způsob, jak si během deseti dní poskládat *pravdivý obraz*. Body jdou po sobě tak, aby se daly zapamatovat: jejich první písmena skládají ***ZAČNE TO ZAS***, což je zároveň nejpoctivější shrnutí toho, jak nástup do běžícího projektu vypadá. Odkazy vedou do kapitol, kde je téma rozebráno.

– ***Z**ávazky a smlouva*. Co je přesně slíbeno, co znamená „hotovo“, jak vypadá akceptace, jaké jsou milníky a sankce. Zvlášť hledat, co se slíbilo dřív, než se vědělo. (kapitoly 2 a 11)

– ***A**utorita a mandát*. Co smím rozhodnout sám, co jen doporučuji a kudy vede červený telefon. Pokud to nikdo neřekl, vyjednat to teď. (kapitola 6)

– ***Č**ekající rozhodnutí*. Co na koho čeká a jak dlouho. Nejstarší čekající rozhodnutí je první kandidát na zásah. (kapitoly 1 a 9)

– ***N**achystaná prostředí*. Běží společné vývojové a testovací prostředí? Dokážu si systém spustit sám, bez asistence? Rozbité prostředí je první diagnóza. (kapitoly 5 a 7)

– **Eskalace nanečisto.** Na konci prvních deseti dní sepsat stav projektu podle osnovy eskalace z kapitoly 8 (co se děje, co to znamená, varianty, doporučení, termíny), i kdyby se nikam neposílala. Co neumíte popsat, potřebujete ještě zjistit.

– ***T**ým a jeho tvar*. Kdo co vlastní, kde se vlastnictví překrývá nebo chybí, kdo je přetížený a jak velké jsou buňky. (kapitola 4)

– Tempo týmu. Kdo pracuje přesčas a proč, s kolika hodinami hloubkové práce denně se reálně plánuje a kdo rozhoduje o víkendech. Sprint **TDL** nesmí být mlčky rytmem týmu. (kapitola 10)

– ***O**bdélníky a čáry*. Komponenty, týmy, dodavatelé, prostředí, a hlavně seznam otázek, které neleží v žádném z nich. (kapitola 1)

– ***Z**ákaznické závislosti*. Data, prostředí, součinnost, přístupy: ověřit s konkrétním člověkem na druhé straně, ne s tabulkou. (kapitola 8)

– **Adresná rizika a vlastníci.** Projít seznam rizik a u každého se ptát: má vlastníka, srozumitelně popsaný dopad a datum dalšího rozhodnutí? Co má jen záznam v tabulce, není řízeno. (kapitoly 6 a 8)

– ***S**cénáře produktu*. Projít dva až pět scénářů očima zákazníka a poznamenat si všechno, co nedává smysl. Nezdůvodňovat, jen zapsat, vysvětlení přijdou později. (kapitola 5)

Za dva dny to nahodíme

O úsudku, odpovědnosti a práci Technical Delivery Leada

Verze 1.12 · září 2026

© Valdemar Závadský · valda@cloudfield.cz

CLOUDFIELD a.s.