Jaká vidíte omezení použití principů TOC a CC v řízení projektů IS/ICT?

Tomáš Maršálek

2. ledna 2015

TOC (Theory of Constraints) je idea původně zamýšlená pro výrobní procesy, o kterých se dá tvrdit, že jsou jasně definované, kdežto vývoj softwaru je spíše výroba nehmatatelných ideí a připomíná komplexní dynamický systém. Definovat TOC pro vývoj softwaru je složité, protože je obtížné definovat základní veličiny k tomu potřebné. TOC vyžaduje definovat míry, podle kterých sledujeme přinesenou hodnotu. Ve výrobě lehce nadefinujeme produkt jako hotový výrobek. Ve vývoji softwaru lze použít pouze vágní jednotky jako např. jeden use case, story point, nová vlastnost, položka z backlogu nebo podobné, které jsou ale velmi těžce měřitelné.

Pro tyto jednotky je velmi obtížné přesně odhadnout jejich náročnost a užitek. Odhad náročnosti je složitý, protože se jedná o implementaci nového nápadu a ne pouze dalšího produktu v rutinní výrobě. Užitek této nové vlastnosti je pouze předpoklad užitku vlastníka. Skutečný užitek se projeví až v produkčním nasazení vlastnosti.

V analogii k výrobnímu procesu, kde se podle TOC snažíme minimalizovat zásoby na skladě, ve vývoji SW ani nevíme, co považovat za jednotku na skladě. Velké množství úkolů ve frontě v Kanbanu nebo v backlogu ve Scrumu nevidíme jako nic , co by brzdilo propustnost systému - constraint. Pokud se ve Scrumu úkol nestihl do konce sprintu, je jednoduše zrušen nebo přesunut do následujícího sprintu.

Pokud nepoužijeme TOC přímo na vývoj SW, ale použijeme aplikaci TOC na řízení projektů - metodu kritického řetězce, narazíme na několik jejích předpokladů. Metoda kritického řetězce vyžaduje zdroje (vývojáře) s možností rychlého přeřazení mezi úkoly nebo řetězci. Přepnutí mezi úkoly je ale velmi časově nákladné ve vývoji SW, protože programátor musí najednou v hlavě držet velké množství informací. Přepnutí kontextu mezi úkoly navíc způsobí, že programátor vypustí ze své krátkodobé paměti úkol, který opustil, a při navrácení musí na spoustu věcí přijít znovu.

Flexibilita v přehazování programátorů mezi úkoly a minimalizace nečinnosti mohou způsobit nejen zbytečnou ztrátu času kvůli přepínání kontextu, ale vystavují projekt dlouhodobému riziku jevům „software bloat“ (kypění kódu) a „feature creep“. V softwaru totiž neplatí, že čím více vlastností, tím je software kvalitnější. Dlouhodobě mohou některé vlastnosti, které nejsou ani důležité, způsobit přílišnou provázanost softwaru, mohou přidat závislosti na softwarových knihovnách a další nežádoucí jevy. Tyto jevy mohou dále zapříčinit pomalejší vývoj a v dlouhodobém horizontu neudržitelnost a dlouhodobé zaučování nových vývojářů.

Řízení vývoje softwaru i samotný vývoj softwaru jsou velmi nepředvídatelné disciplíny, které maji před sebou ještě dlouhou trať a spoustu pokusů a omylů, než se dostanou na relativně stabilní úroveň srovnotalnou například s výrobními procesy.

Pro vývoj softwaru je vhodné vypůjčit si z Theory of Constrains její prvky, ale nepoužívat je striktně. Lze ji efektivně zkombinovat s populárními agilními a lean metodologiemi.