PADÁ PROJEKT, RÝCHLO SI NIEČO ŽELAJTE

Expert je ten, kto pozná niektoré najhoršie chyby, ktoré sa môžu prihodiť v jeho odbore a vie, ako sa im vyhnúť. (W.K. Heisenberg)

Róbert Móro

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
robo.moro@gmail.com

Abstrakt. *Keď sa pozrieme na štatistiky úspešnosti softvérových projektov, odkryje sa pred nami nie celkom povzbudivý obraz. Približne každý piaty projekt končí úplným neúspechom, pričom pri takmer každom druhom projekte dochádza k problémom v podobe prekročenia rozpočtu, termínov alebo výsledný produkt nenapĺňa v plnej miere požiadavky zákazníka. Medzi najväčšie riziká pritom patrí použitie nesprávnej metodiky, nedostatočná zaangažovanosť zákazníka či neustále sa meniace požiadavky. Zlyhanie projektu je však vždy dôsledkom súhry viacerých príčin. Ako teda rizikám úspešne predchádzať? Jedným zo spôsobov je riadiť sa agilnou vývojovou metodikou Scrum, ktorá podľa môjho názoru už zo svojej podstaty niektoré riziká eliminuje, alebo ich aspoň do značnej miery minimalizuje. Avšak každá minca má dve strany a Scrum nie je všeliek, naopak prináša so sebou niektoré špecifické riziká, s ktorými musíme počítať.*

Kľúčové slová: *riziko, zlyhanie, plán, Scrum*

Úvod

Na softvérové projekty sa vynakladá čoraz viac finančných prostriedkov, len v rokoch 1997-2001 sa do oblasti IT investovalo zhruba 2,5 bilióna dolárov [6]. Nepríjemne však prekvapí zistenie, že z toho až jeden bilión dolárov sa minul na problémové projekty, z ktorých mnohé napokon úplne zlyhali, čo malo za následok obrovské finančné straty.

Podľa iných štatistík [3] takmer polovica softvérových projektov trpí prekročením rozpočtu alebo termínov, prípadne nedokáže v dostatočnej miere naplniť požiadavky zákazníka a takmer každý piaty projekt sa končí úplným neúspechom.

Sú to veľmi nelichotivé a hrozivo vyzerajúce čísla, ale mali by nás viesť k zamysleniu sa nad príčinami problémov a neúspechu softvérových projektov a k hľadaniu východísk, ako sa im do budúcnosti vyhnúť. Pretože hlúpy je ten, čo sa nedokáže poučiť z vlastných chýb, ale múdry sa učí z chýb druhých.

Na počiatku bolo riziko

Ak by sa ľudia, ktorí boli súčasťou problémových či neúspešných projektov, zamysleli nad príčinami ich problémov a neúspechu, zistili by, že na počiatku všetkého bolo *riziko*. Riziko, ktoré prehliadli, riziko, ktoré nedocenili, alebo keď s ním boli konfrontovaní, nedokázali mu správne čeliť.

Ako tvrdí Brooks vo svojej zbierke esejí [2], všetci programátori sú optimisti. Vyjadrenia ako: *„Teraz to určite pobeží,"* alebo *„práve som našiel poslednú chybu,"* sú nám všetkým, myslím, až príliš dobre známe. A pretože sú optimisti, programátori, či presnejšie softvéroví inžinieri, manažéri softvérových projektov pri plánovaní často podceňujú riziko prekročenia termínu a predpokladajú, že všetko pôjde ako má. Problém je, že plány sú založené len na odhadoch, koľko ktorá úloha bude trvať, a tie bývajú skoro vždy veľmi nepresné. Koniec koncov, keby sme to vedeli povedať presne, nemuseli by sme odhadovať.

Ďalšou príčinou, prečo podľa Brooksa [2] dochádza k prekročeniu termínov, je, že sa to deje postupne, deň po dni. Keby v nasledujúcom okamihu hrozila „katastrofa", tak by na ňu každý okamžite reagoval, snažil by sa vyvinúť čo najväčšie úsilie na to, aby jej zabránil. Ak však každý deň dôjde k malému oneskoreniu, nesplneniu nejakej malej čiastkovej úlohy, tak z toho takpovediac nikto nerobí veľkú vedu, pretože času je dosť, veď čo sa nestihlo dnes, dorobí sa zajtra. A takto sa postupne kopia veľké problémy. Toto sa dá podľa mňa zovšeobecniť a netýka sa to výhradne rizika prekročenia termínu. Riziká sa totiž väčšinou prejavujú postupne, pomaly a je ľahké si ich nevšimnúť alebo podceniť.

Je zaujímavé, že hoci Brooks vydal svoju zbierku esejí pred tridsiatimi piatimi rokmi, stále je veľmi aktuálna a riziká a ich príčiny, ktoré popisuje, trápia softvérové projekty dodnes. Zdá sa, že sme sa nedokázali dostatočne poučiť z chýb minulosti, ale neustále ich opakujeme.

Podľa mňa jediné východisko z tejto situácie je dôsledne spoznať riziká, ktoré sprevádzajú softvérové projekty a aké sú ich spúšťače, pretože len tak im budeme vedieť predchádzať.

Príčiny zlyhania

Ako som už spomínal, príčinou zlyhania projektov býva prehnaný optimizmus ľudí, ktorí sa na projekte podieľajú a podcenenie alebo prehliadnutie rizík. Toto je však, aj keď pravdivé, ale príliš všeobecné tvrdenie. Preto sa teraz zameriam na dve štúdie, ktoré analyzovali príčiny zlyhania projektov a konfrontujem ich výsledky.

Cerpa v [3] uvádza ako najčastejšie príčiny zlyhania projektov tieto:

1. dátum dodania zasiahol do procesu vývoja
2. projekt bol podhodnotený
3. riziká neboli počas trvania projektu prehodnocované, kontrolované alebo manažované
4. zamestnanci neboli odmenení za nadčasy
5. rozhodnutie o dodaní bolo vykonané bez dostatočnej znalosti požiadaviek
6. práca na projekte bola pre zamestnancov nepríjemnou skúsenosťou

Dôležité je zistenie, že zlyhanie žiadneho zo skúmaných projektov nebolo dôsledkom jedinej príčiny, vždy išlo o súhru viacerých príčin. Pri 46% projektov mali na zlyhaní podiel prvé štyri uvádzané príčiny, všetkých šesť príčin pri 26% projektov. Toto zistenie podľa mňa nie je prekvapivé, pretože jedna príčina je často spúšťačom ďalších, ktoré sú navzájom previazané.

Z uvedených faktorov vyplýva, že väčšina z nich bola v kompetencii manažéra projektu, aby im dokázal včas zabrániť. Napríklad prvú a piatu príčinu možno odstrániť lepšou komunikáciou so zákazníkom. Na základe nej by sa totiž ozrejmili jeho požiadavky a zreálnil by sa termín dodania finálneho softvérového produktu. Naopak, pri nedostatočnej komunikácii nedokážeme zachytiť požiadavky zákazníka a odhadnúť, koľko úsilia bude treba vynaložiť. Výsledkom je nedostatok času na vývoj a testovanie a riziko zamietnutia finálneho produktu, alebo minimálne nenaplnenie požiadaviek zákazníka v dostatočnej miere.

Tieto výsledky tiež potvrdzujú, aké ťažké je plánovať softvérový projekt, a aké dôležité je analyzovať, prehodnocovať a manažovať riziká.

Okrem toho do popredia vystupuje aj ľudský faktor – ak chceme, aby bol projekt úspešný, musí ho vyvíjať skutočný tím ľudí, ktorí sú spokojní s podmienkami, v ktorých pracujú a zároveň sú za svoju prácu aj dostatočne odmeňovaní.

Tiwana sa v [6] zaoberá spúšťačmi rizík. Na základe analýzy reálnych projektov identifikoval tieto najčastejšie spúšťače:

1. použitie nevhodnej metodológie
2. nedostatočná zaangažovanosť zákazníka
3. nedostatok formálnych praktík projektového manažmentu
4. rozdielnosť voči predchádzajúcim projektom
5. zložitosť projektu
6. nestálosť požiadaviek

Ako vidíme, na prvom mieste sa umiestnilo použitie nevhodnej metodológie, čo je podľa mňa trochu prekvapivé. Vyplýva z toho, že každý projekt má špecifické požiadavky a im treba prispôsobiť metodiku vývoja a podriadiť tomu rozhodnutie, či použijeme klasickú iteratívnu a inkrementálnu metodiku, alebo niektorú z agilných metodík. Osobne by som s týmto trochu polemizoval. Podľa mňa môžu existovať dôvody, kedy je potrebné použiť klasické metodiky, napríklad v prípade, že na začiatku projektu musí existovať presná a dôkladná špecifikácia softvérového produktu. Ale v ostatných prípadoch môže podľa mňa agilný vývoj viac pomôcť, ako uškodiť, a preto nevidím dôvody, prečo by jeho použitie malo byť nevhodné.

Ostatné miesta v tomto zozname už také prekvapivé nie sú. O nedostatočnej zaangažovanosti zákazníka som už hovoril vyššie. Pod nedostatkom formálnych praktík projektového manažmentu sa skrýva nedostatok formálnych plánov, odhadov rozpočtu, termínov a podobne, čo sa v podstate zhoduje s 2. a 3. príčinou, ktorú uvádza Cerpa v [3].

Obidve štúdie sa teda do veľkej miery prekrývajú, v prípade [3] je dôraz kladený aj na ľudský faktor, ktorý v [6] trochu chýba, aj keď je čiastočne skrytý pod rozdielnosťou s predchádzajúcimi projektmi – tím, ktorý má skúsenosti z nejakej domény sa ťažšie preorientuje na nový, výrazne odlišný projekt z inej domény. Zistenia oboch autorov sa navzájom dopĺňajú a podporujú a sú podľa mňa dobrým základom pre pochopenie hlavných príčin zlyhania softvérových projektov a rizík s nimi spojenými.

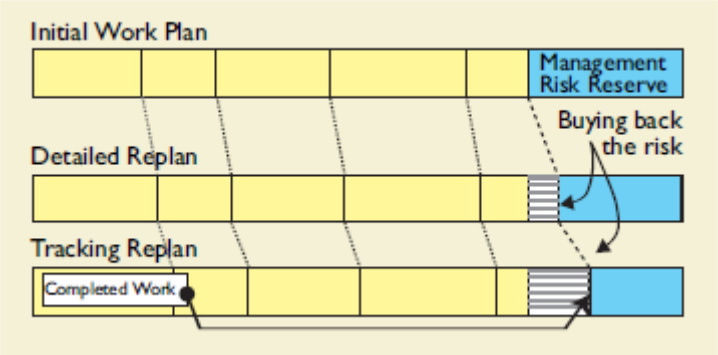
Ako sa vyrovnať s rizikami

Zo zistení, ktoré som uviedol vyššie, jasne vyplynula dôležitosť manažmentu rizík. Riziká pritom nestačí zanalyzovať na začiatku projektu, ale treba túto analýzu priebežne prehodnocovať, aby sme vedeli, ktoré riziká nám aktuálne hrozia najviac, čomu sa treba vyhnúť. Analýza je však tiež len jedna zložka manažmentu rizík. Na identifikované riziká je potrebné sa náležite pripraviť, vypracovať plán, ako im zabrániť a postupy, ktoré sa aktivujú v momente, keď zistíme, že sa nám niektorému riziku zabrániť nepodarilo. Pretože vtedy je väčšinou príliš neskoro, aby sme sa pýtali, čo teraz. V takom momente treba konať rýchlo, ale premyslene.

Dôležité je, aby mal manažér vždy prehľad o tom, v akom stave je jeho projekt. Toto väčšinou nie je problém zabezpečiť pri menších projektoch, ale v prípade, že ide o väčší projekt, v ktorom medzi ním a vývojármi a testermi stojí jeden alebo viacero medzičlánkov v podobe manažérov jednotlivých oddelení, sa situácia komplikuje. Ako píše Brooks v [2], záujmy „šéfa" a týchto manažérov sú v rozpore. Šéf potrebuje vedieť, v akom stave je projekt, aby mohol v prípade potreby zasiahnuť, ale tiež aby získal obraz a skúsenosti do budúcna. Na druhej strane, manažéri sa boja, že ak o problémoch povedia šéfovi, zasiahne a tým zníži ich autoritu, pretože by problém mohli zvládnuť aj sami. A preto problémy takpovediac zametajú pod koberec a čakajú s ich hlásením, až dokým zistia, že si s tým už neporadia. Vtedy však býva zvyčajne neskoro. Brooks vidí riešenie v redukovaní konfliktu rolí. Šéf musí rozlišovať medzi informáciami o stave a informáciami, ktoré by ho mali podnietiť k akcii. Musí sa ovládnuť a neriešiť problémy, ktoré dokážu zvládnuť jeho manažéri. Tí, ak budú vedieť, že ich nadriadený nezareaguje panicky alebo prehnanou akciou, budú otvorene informovať o stave a problémoch.

Riziko prekročenia termínu patrí medzi najčastejšie riziká v softvérových projektoch. O jeho príčinách som už písal, ako sa s ním však vyrovnať? Armour v [1] navrhuje vytvoriť dva plány – jeden pre zákazníka (záväzný plán) a jeden pre vývojový tím (pracovný plán). *Pracovný plán* má potom odrážať, koľko si myslíme, že nám projekt zaberie a riadime sa ním pri vývoji. Naproti tomu *záväzný plán* predkladáme zákazníkovi a zahŕňa pracovný plán spolu so započítanou mierou rizika a neistoty, ktorú vnímame na začiatku projektu v čase plánovania. V praxi to potom vyzerá tak, že ako projekt prebieha, získavame lepšiu predstavu o tom, koľko nám v skutočnosti zaberie, riziko sa preto zmenšuje a postupne ho *„odkupujeme naspäť"* z rezervy, ktorú sme si na začiatku alokovali v záväznom pláne. Dôležité preto je, aby sme na začiatku dobre odhadli potrebnú rezervu, a aby sme ju

nezneužívali, ale snažili sa dodržiavať pracovný plán. Situácia je názorne naznačená na obr. 1.



Obr. 1 Dva plány [1].

Je to podľa mňa zaujímavé riešenie, ale rovnako ako Armour, aj ja v tom vidím isté riziko. Ak pred vývojármi zatajíme že sa postupuje podľa dvoch plánov, a neskôr to zistia, budú sa cítiť podvedení. Naopak, ak im to povieme, je podľa mňa veľké riziko, že sa mentálne nastavia na plán, ktorý počíta s oneskorením, čo nakoniec bude viesť k prekročeniu aj tohto plánu. A to je už problém, pretože ide o termíny, ktoré sú záväzné a vznikli dohodou so zákazníkom. Armour však tvrdí, že treba byť voči vývojovému tímu otvorený a že si uvedomia potrebu dodržať pracovný plán a nezneužívať zdanlivú rezervu záväzného plánu. Podľa mňa je to optimistické tvrdenie, pretože človek je vždy len človek, a keď vie, že má časovú rezervu, tak ho nikto nedonúti nadčasovať a snažiť sa za každú cenu splniť úlohu v čase predpísanom pracovným plánom.

Riziká a Scrum

Ak hovorím o spôsoboch, ako sa vyrovnať s rizikami, nemôžem podľa mňa nespomenúť agilné metodiky vývoja, medzi ktoré patrí napríklad aj čoraz populárnejší *Scrum* [5]. Termín Scrum pochádza z anglického rugby a do slovenčiny sa v športovej terminológii prekladá ako mlyn, či priamočiarejší preklad skrumáž. Názov by mohol evokovať zmätok, dezorganizáciu, ale tak to v skutočnosti samozrejme nie je. Ako to teda je?

Softvérový produkt vyvíja Scrum tím, ktorý tvoria vývojári a testeri ako v bežnom softvérovom tíme. Projektový manažér sa nazýva *„Scrum master“* a je to niekto, kto je zodpovedný za celý tím, kontroluje priebeh práce, či usporadúva tímové stretnutia. Okrem toho do vývoja vstupuje tzv. vlastník produktu - *„Product owner“*, ktorý reprezentuje zákazníka a používateľa. Na produkt sa teda pozerá z ich pohľadu a rozhoduje, čo je priorita a ako si predstavuje výsledný produkt.

Zoznam všetkých funkcionalít a želaných vlastností produktu sa nazýva *„Product Backlog“*. Pridávať nové položky doň môže každý člen tímu, ale len vlastník produktu rozhodne, akú budú mať prioritu. Celý vývoj je rozdelený na tzv. *šprinty*, ktoré môžu trvať od niekoľko dní po mesiac, ale vždy rovnako dlho. Pred začiatkom šprintu sa vytvorí tzv. *„Sprint Backlog“*, čiže zoznam funkcionalít vybraných z produkčného zoznamu, ktoré sa

budú vyvíjať v danom šprinte. Dôležité je, že každá funkcionalita je v danom šprinte analyzovaná, navrhnutá, implementovaná a aj otestovaná, čiže prejde celým vývojovým cyklom. Výsledkom jedného šprintu je tak vždy fungujúci prototyp – časť budúceho systému.

Tento úvod do Scrumu tu uvádzam len preto, aby som na ňom mohol ukázať, ako nám pomáha manažovať riziká. Tradičný neagilný prístup k vývoju softvéru postupuje viac-menej vodopádovo – na začiatku sa urobí detailná analýza, v ktorej sa snažíme zachytiť všetky požiadavky zákazníka, následne urobíme architektonický návrh systému ako celku a až potom začneme s implementáciou. Nakoniec, keď je systém naimplementovaný, prejde do fázy testovania, v ktorej sa snažíme odhaliť a opraviť všetky chyby.

Riziká takéhoto postupu sú podľa mňa zrejmé. Ak nepochopíme požiadavky zákazníka vo fáze analýzy, môže sa nám stať, že nakoniec, keď mu predstavíme finálny produkt, nám povie, že si to tak nepredstavoval. Cena zmeny je vtedy príliš veľká, čo sa času aj financií týka. Ešte väčší problém spočíva v tom, že zákazník sám často nemá jasno v tom, čo by vlastne chcel. A úplná nočná mora je, že môže časom zistiť, že by si to vlastne predstavoval inak ako kedysi v začiatkoch a spýta sa, či by sme to nezmenili. Miera rizika nesplnenia zákazníkových požiadaviek je teda pri klasickom prístupe veľmi veľká. Naproti tomu pri agilných metódach, akou je napríklad Scrum, je zákazník priamo vtiahnutý do procesu vývoja. Vývojové cykly sú krátke, maximálne jeden mesiac a po každom cykle (šprinte) sa zákazníkovi predstaví prototyp. To je podľa mňa jedna z najväčších výhod Scrumu, pretože je už zo svojej podstaty odolný voči neustále sa meniacim požiadavkám zákazníka a nehrozí ani jeho nedostatočná zaangažovanosť, čo sú jedny z najväčších rizikových faktorov zlyhania projektu, ako som uviedol vyššie.

Ďalším častým rizikom je prekročenie termínu alebo rozpočtu. Aj na toto dáva podľa mňa Scrum odpoveď. Plánovanie prebieha vždy na začiatku šprintu a neskôr sa plán už nemení. Každý deň sa konajú krátke, maximálne pätnásťminútové stretnutia, na ktorých každý člen tímu povie, čo robil predchádzajúci deň, s akými sa stretol problémami, a čo bude robiť ďalej. Takto má manažér neustály prehľad o stave projektu. Postup prác sa dá jednoducho vizualizovať tzv. *grafom spaľovania („burndown chart“)*, z ktorého je zrejmé, koľko toho ešte zostáva do konca šprintu urobiť. Je teda jednoduché včas odhaliť riziko prekročenia termínu. Dôležité tiež je, že koniec šprintu sa za žiadnych okolností nedá posunúť a na jeho konci musí byť hmatateľný výsledok, ktorý je navyše otestovaný. Tak sa nemôže stať, ako to často býva, že sa odovzdá nedostatočne odladený produkt, pretože sme prekročili termíny na vývoj a na testovanie jednoducho nezostal dostatok času.

Samozrejme, Scrum ani iné agilné metodiky nie sú zázračným všeliekom, ktorý by vyriešil všetky problémy a riziká softvérových projektov. Denning v [4] porovnáva tradičné a evolučné (agilné) metodiky vývoja a hovorí, že zatiaľ čo vopred naplánovaný vývoj sa snaží rizikám vyhnúť, evolučný vývoj napodobňuje prírodu a snaží sa rizikám čeliť. Dokonca tvrdí, že vývojári by mali úmyselne vystavovať rodiace sa systémy rizikám, aby videli, kde zlyhávajú a mohli ich tak vylepšiť. Podľa neho je lepšie riziká vyhľadať a naučiť sa ich prežiť. V niečom s ním súhlasím, ale podľa mňa je agilný vývoj skôr spôsob, ako riziká minimalizovať, nie ako im ísť v ústrety. Myslím si, že Denning je v tomto prípade prehnaný optimista, tak ako mnohí iní pred ním, čo je však nám ako programátorom a softvérovým inžinierom asi vlastné, ako hovorí Brooks v [2].

Keď sa pozrieme do zoznamu najvýznamnejším spúšťačov rizík, ktorý som uviedol vyššie, tak nevhodne použitá metodika je na prvom mieste. Podľa mňa však na vine nie je metodika, ale to, ako sa jej ľudia zhostia. Treba si len uvedomiť, že aj pri Scrume treba analyzovať a manažovať riziká. Navyše musíme poznať a počítať s rizikami, ktoré sú špecifické pre Scrum a agilný vývoj vo všeobecnosti.

Podľa mňa najväčšie riziko, ktoré nám pri vývoji podľa Scrumu hrozí, je strata predstavy o systéme ako celku. Preto hoci vyvíjame jednotlivé funkcionality, vždy musíme myslieť na to, že sú súčasťou väčšieho celku, a teda ako budú spolupracovať, ako zapadnú do širšej koncepcie. Pre strom nikdy nesmieme zabudnúť na zvyšok lesa. Aby sme to dosiahli, stačí podľa mňa po každých pár šprintoch (v závislosti od veľkosti projektu a dĺžky jedného šprintu) vypracovať celkový pohľad na prototyp a jeho architektúru.

Záver

Každý softvérový projekt už od počiatočných fáz sprevádza množstvo rizík. Ak sa im nedokážeme vyhnúť, je pravdepodobné, že skončí zlyhaním a neúspechom. Projekt však nikdy nezlyhá kvôli jednému neustriehnutému riziku, vždy ide o súhrn rizík, z ktorých väčšinu, ak nie všetky, vieme ovplyvniť. Je preto dôležité poznať riziká, ktoré nám hrozia. Musíme sa poučiť z minulých chýb, pretože len tak ich v budúcnosti nebudeme opakovať. Možné riziká však nestačí poznať, musíme ich cielene minimalizovať. Jedným zo spôsobov, ktoré som sa snažil načrtnúť, je agilný vývoj reprezentovaný metodikou Scrum. Hoci nejde o všeliek a prináša so sebou špecifické riziká, dôrazom na komunikáciu so zákazníkom, včasné prototypovanie a odolnosťou voči zmenám požiadaviek, dokáže podľa mňa množstvo najvážnejších rizík softvérových projektov eliminovať.

Dôležité je riadiť sa tým, čo už pred tridsiatimi piatimi rokmi pomenoval Brooks v [2] – nebuďme prehnaní optimisti a nikdy nepodceňujme riziká. Pretože katastrofy väčšinou neprichádzajú zo dňa na deň, ale postupne.

Použitá literatúra

1. Armour, P.G.: To Plan, Two Plans. In: *Communications of the ACM*, Vol. 48, No. 9, 2005, 15-19.
2. Brooks, F.P.: *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, 1995.
3. Cerpa, N., Verner, J.M.: Why did your project fail? In: *Communications of the ACM*, Vol. 52, No. 12, 2009, 130-134.
4. Denning, P.J. et al.: The Profession of IT: Evolutionary System Development. In: *Communications of the ACM*, Vol. 51, No. 12, 2008, 29-31.
5. Hu, Z. et al: Research on Agile Project Management with Scrum Method. In: *Proc. of the 2009 IITA International Conference on Services Science, Management and Engineering*, 2009, pp. 26-29.
6. Tiwana, A., Keil, M.: The One-Minute Risk Assessment Tool. In: *Communications of the ACM*, Vol. 47, No. 11, 2004, 73-77.

Annotation

Project's falling, quickly make a wish

When we look at the statistics of software projects' success rate, we'll see not a very encouraging picture. Approximately every fifth project ends as a total failure, while almost every other project has problems with overrunning the planned budget, schedule or the final product doesn't fully meet customer's requirements. Amongst the most severe risks belongs use of inappropriate methodology, lack of customer involvement or requirements volatility. However, project failure is always a result of a combination of multiple reasons. How can we prevent the risks? One possible approach is to use the agile development methodology Scrum, which in my opinion eliminates some of the risks by its nature or at least significantly minimizes them. However, every coin has two sides and Scrum isn't a magical cure, on the contrary, it carries some specific risks we have to take into account.