PLÁNUJETE AGILNE ?

Keď človek nevie, ku ktorému prístavu pláva, žiaden vietor mu nie je priaznivý.

(Seneca)

Ján Sivuľka

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

Abstrakt. *Úspešnosť vytvorenia softvérového projektu v termíne a podobe, v akej nám ho zákazník špecifikoval, závisí najmä od pripravenosti a rozdelenia práce, zdrojov a času pri vývoji. Kľúčovým faktorom je plánovanie, ktoré je neoddeliteľnou súčasťou nie len veľkých, ale aj malých projektov. Pre tvorbu a riadenie plánov vývoja softvéru máme k dispozícii široké spektrum metód, no ani jedna z nich nám nezaručí, že pri jej použití sa neobjavia nepredvídateľné skutočnosti. Z tohto dôvodu je vhodné byť pri tvorbe plánu agilný a vedieť tak promptne reagovať na nepredvídané udalosti alebo zmeny požiadaviek zadávateľa projektu. Prečo plánovať agilne? Problematikou plánovania a zmenami v projektoch sa zaoberá každý vývojový tím, ktorý chce priviesť projekt do úspešného konca. Čoraz viac sa využívajú moderné agilné metódy ako napríklad SCRUM. Je možné zmienený spôsob vývoja softvéru plánovať a zefektívniť tak vývoj softvéru? Ako odhadnúť náročnosť a čas potrebný pre úspešné vyriešenie úlohy? Táto esej sa snaží odpovedať na tieto otázky spolu s názormi a odporúčaniami ako plánovať agilný vývoj softvéru.*

Kľúčové slová: *plánovanie, agilné metódy, SCRUM, odhadovanie, vývoj softvéru*

Krátky príbeh na úvod

Pred pár rokmi som sa vybral spolu s mojou priateľkou na výlet do Košíc. Celú cestu sme si dôkladne vopred naplánovali. Chceli sme prísť do Košíc, pozrieť pamätihodnosti, prejsť sa po hlavnom námestí, poobzerať si ceny oblečenia v obchodoch a večer ísť späť autobusom tak, aby sme boli doma okolo ôsmej. Všetko išlo podľa plánu, až kým sme sa dozvedeli o koncerte, ktorý bol zadarmo prístupný verejnosti. Rozhodli sme sa teda na tomto koncerte zúčastniť. Po skončení koncertu sme prišli na stanicu, kde sme s hrôzou zistili, že priame autobusové alebo vlakové spoje domov už nepremávajú. Pravdepodobne jediná možnosť ako sa vrátiť späť ešte v ten deň a neplatiť taxík bola ísť vlakom asi 60km okľukou s viacerými prestupmi. Potrebovali sme ísť domov, takže sme túto možnosť využili. Počas cesty sme plánovali ako vysvetlíme rodičom naše „malé" meškanie, keď zrazu náš vlak zastal a sprievodca nám oznámil, že došlo k poruche rušňa a pravdepodobne sa to už v ten deň neopravia. Opäť sme sa ocitli v situácii, kedy sme nevedeli, ako sa dostať domov. Našťastie sme stretli priateľa, po ktorého prišli známi a ponúkli nám odvoz. Domov sme aj napriek tomu dorazili až hlboko v noci.

Poučenie z tohto príbehu znie: bez ohľadu na to, ako usilovne, detailne a dôkladne plánujete, realita vás stále môže zaskočiť a narušiť vám vaše plány. Tak ako pri každom vážnejšom projekte, i v prípade tohto výletu sa vyskytlo mnoho skutočností a udalostí, ktoré sa jednoducho nedali predvídať. Navyše, čím ďalej do budúcnosti siahajú vaše plány, o to vyššia je pravdepodobnosť, že vaše plánovanie sa bude musieť aktualizovať.

Tu môžem nadviazať na agilné projektové plánovanie, v ktorom sa odporúča plánovať s náležitou detailnosťou: krátkodobé plánovanie nech je hĺbkové a dlhodobé zasa len v náčrtoch.

Agilný vývoj

V poslednej dobe začal agilný prístup nahrádzať tradičné plánovanie. Agilný prístup pritom zdôrazňuje ľudský a remeselný aspekt softvérového vývoja oproti inžinierskym aspektom. Uprednostňuje jednotlivcov a ich interakcie pred procesmi a nástrojmi, fungujúci softvér pred zrozumiteľnou dokumentáciou, spoluprácu so zákazníkom pred rokovaním o zmluve a prispôsobenie sa zmenám pred dodržiavaním plánu [2]. Agilné metódy vývoja sa vyvinuli z dobre známych inkrementálnych a iteratívnych modelov vývoja softvéru. Vznikli z presvedčenia, že prístup, ktorý je viac založený na učení, ľuďoch, inovácii a zmene môže viesť k lepším výsledkom.

Oficiálne uvedenie agilných metód vývoja sa viaže k roku 2001. Vo februári tohto roku 17 odborníkov v oblasti softvérových systémov vydalo takzvaný Manifest pre agilný vývoj softvéru. Tento manifest pojednáva okrem iného o 12 základných zásad vývoja softvéru [2].

Prvé skúsenosti s uplatňovaním agilného prístupu naznačujú, že takýto prístup môže napomôcť pri vyrovnávaní sa s problémami softvérovej krízy [1] a taktiež zdôrazňujú, že tradičné plánovanie a agilný prístup sa vzájomne nevylučujú [3]. Skutočne, takéto nepravdepodobné zlúčenie týchto dvoch prístupov môže vyústiť do flexibility a môže nastoliť rovnováhu medzi strnulosťou tradičného plánovania a potrebami prispôsobiť sa aktuálnemu vývoju situácií.

Agilné princípy kladú dôraz na tvorbu softvéru tak, aby sa čím skôr dospelo k nejakým fungujúcim fragmentom žiadaného softvéru, na rozdiel od trávenia množstva času nad písaním špecifikácie pre tvorbu cieľového softvérového projektu. Je to vhodné ako pre zákazníka, tak aj pre vývojárov. Pretože z vlastnej skúsenosti viem, že predstava zákazníka a vývojára o špecifikovaných požiadavkách môže byť do istej miery rozdielna. V prvom rade ide o spokojnosť zákazníka, ktorý má k dispozícii reálne výsledky v krátkych intervaloch a môže sa k nim vyjadriť, čím šetrí nie len čas a prácu celému vývojovému tímu ale takisto finančné prostriedky, ktoré by musel vynaložiť na prípadné zapracovanie zmien po odovzdaní celého projektu.

Mnoho tímov, ktoré uplatňujú agilný vývoj, zaznamenalo výrazné zlepšenie, ba dokonca úplnú premenu a to nie len na poli produktivity, ale i v oblasti pracovnej morálky. To má význam pre mnohých produktových vývojárov, z ktorých sa mnohí popálili na tradičnom spôsobe vývoja [5].

Začíname plánovať

Pri agilnom plánovaní vývoja softvéru sa nám núkajú tri úrovne plánovania a odhadovania.

Prvá úroveň sa týka začiatočného plánovania rozsahu celého projektu. Mali by sme získať celkový prehľad o tom, čo od nás zadávateľ očakáva, aké prostriedky a techniky sú potrebné pre dosiahnutie cieľov. To nie je až také náročné napríklad pri plánovaní cesty do Košíc, ale pri vývoji softvéru je to o trochu zložitejšie. Táto úroveň by mala obsahovať identifikovanie potenciálnych prípadov použitia, v ktorom nejde o ich špecifikovanie do úplných detailov, ale iba o zoznámenie sa s tematikou a získanie prehľadu o používateľských príbehoch[1] a určenie širokých odhadov pre ich realizáciu.

Druhou úrovňou plánovania je definovanie priorít a určovanie rozsahu. Ústrednou postavou v tejto fáze je zákazník, ktorý určuje priority a je zodpovedný za špecifikovanie toho, čo bude predmetom činnosti vývojového tímu, v akom poradí sa budú jednotlivé požiadavky vykonávať, prideľuje jednotlivé prípady použitia vývojovému tímu v iteráciách podľa obchodnej politiky a dostupného času, úsilia a zdrojov.

V tomto bode plánovania je potrebné uvedomiť si, že mnohé projekty sú vopred odsúdené na neúspech kvôli zavedeniu nereálneho časového plánu a konečných termínov. Je pochopiteľné, že veľmi často sa stretávame s prípadmi, kedy chce mať zákazník hotový produkt čo najskôr, pretože čím skôr je softvér dokončený, tým skôr môže byť reálnym prínosom pre objednávateľa a priniesť mu zisk.

Agilné metódy vývoja žiadajú od tímov dosiahnuť hmatateľné výsledky práce na konci každej iterácie [3], preto posledná úroveň plánovania obsahuje detailný popis najbližšej iterácie, ktorá je krátka a časovo ohraničená. Ide o podrobný plán, ktorý obsahuje zoznam úloh a k nim aj časový odhad, koľko by malo splnenie jednotlivej úlohy trvať. Každá úloha má prideleného aspoň jedného člena tímu, ktorý je za jej splnenie zodpovedný. Plán vychádza z reality a pri každom ďalšom detailnom plánovaní sa bude brať do úvahy spätná väzba odhadov (ako sa podarilo odhadnúť časovú náročnosť

[1] Požiadavka používateľa pri agilnom prístupe vývoja softvéru, ktorá je formulovaná jednou alebo viacerými vetami v hovorovom alebo obchodnom jazyku používateľa.

jednotlivých úloh), spätná väzba od zákazníka (či je čiastkový výsledok podľa jeho predstáv), potenciálne dizajnové zlepšenia, nové požiadavky a potreba opraviť chyby.

Nevyhnutnosť spätnej väzby

Pravdepodobne najdôležitejším princípom agilného plánovania je nevyhnutnosť spätnej väzby. Tá môže byť veľmi nápomocná pri redukovaní množstva potenciálnych rizík, ktoré počas vývoja softvéru môžu nastať. Jedná sa napríklad o riziko zlého pochopenia funkčných požiadaviek, neprijateľného používateľského rozhrania, integrovania systému nesprávne, riziko zlej analýzy a pod. Aby sme hrozby v podobe týchto rizík eliminovali, prípadne redukovali čo najviac, potrebujem získať spätnú väzbu. Spôsob, ktorý nám to umožní je vytvoriť pracovnú verziu softvéru a prinajmenšom to konzultovať so svojimi kolegami a získať tak názory na konkrétne riešenie aj z inej perspektívy. Možno sa pri vývoji pozabudlo na nejaké detaily, ktoré je potrebné pre úspešné riešenie daného problému a spokojnosť zákazníka do riešenia zapracovať. Komunikáciou a získavaním spätnej väzby získame prehľad o tom, aké reálne riziká hrozia nášmu projektu. Je potrebné sa z nich poučiť a tieto hrozby pri tvorbe najbližšieho plánu zohľadniť, a teda vyriešiť problémy skôr, než bude neskoro.

Scrum a plánovanie

Medzi najpopulárnejšie agilné metódy patrí momentálne Scrum. Bol silne ovplyvnený recenziou Harvardu z roku 1986, ktorá sa venovala praktikám úspešného vývoja softvérových produktov. V tomto dokumente bol zavedený pojem „Scrum“, ktorý odkazoval na samostatne organizované ragbiové družstvo, ktoré sa samostatne pohybovalo po ihrisku (obr. 1.). Oficiálnou metódou pre agilný vývoj sa stal v roku 1993, ktorej dali formálnu podobu Ken Schwaber a Dr. Jeff Sutherland. Scrum dnes využívajú malé ale aj veľké spoločnosti vrátane Yahoo, Microsoft, Google, Motorola, SAP, Cisco, GE, CapitalOne a americká spoločnosť Federal Reserve [5].

Obr. 1. Scrum

Slovami Kena Schwabera, Scrum nie je metodológia, je to *systém* pre vývoj projektov alebo aplikácií. Rozdeľuje vývoj do cyklov, ktoré sa nazývajú *šprinty*. Tieto iterácie trvajú väčšinou jeden až štyri týždne. Každý šprint má fixne stanovenú dĺžku, končí k presne špecifikovanému dátumu, bez ohľadu na to, či je práca určená pre daný šprint dokončená alebo nie.

V každom šprinte sa vykonáva analýza problémovej oblasti, návrh a samotná implementácia. Výsledkom každého šprintu má byť funkčná časť softvéru, ktorá sa môže prezentovať produktovému vlastníkovi. Jednotlivé výsledky šprintov sa následne integrujú do celkového riešenia. Každý člen tímu je zodpovedný za úlohy, ktoré mu boli pridelené. Na stretnutí tímu prezentuje výsledky svojej práce a diskutuje s ostatnými svoje riešenie, otázky alebo problémy, s ktorými sa počas vývoja stretol. Po skončení každého šprintu sa uzavrú úlohy, ktoré boli ukončené, určia sa úlohy, ktoré ešte treba prerobiť a evidujú sa úlohy, ktoré sa nestihli dokončiť. Zároveň sa určuje nová funkcionalita a nové úlohy, ktoré by sa mali do skončenia ďalšieho šprintu splniť. Aktualizuje sa produktový zoznam, ktorý predstavuje prioritizovaný zoznam požiadaviek, vlastností alebo vecí, o ktoré má zákazník záujem. Požiadavky sú popísané terminológiou zákazníka. Tieto položky nazývame *stories* alebo *položky backglogu*.

Stories by mali obsahovať [4]:

- *ID* - jedinečná identifikácia, jednoduché automaticky zvyšované číslo. Umožňuje nám nestratiť sa keď premenujeme story.
- *Názov* – krátky, popisný názov story. Napr. "Vidieť moju históriu transakcií". Jasné dosť nato, aby vývojári a produktový vlastník približne chápali o čom sa rozprávajú a jasné dosť nato, aby jednotlivé story boli navzájom rozlíšiteľné. Normálna dĺžka je 2 – 10 slov.
- *Dôležitosť* – dôležitosť story podľa produktového vlastníka. Napr. 10 alebo 150. Čím väčšie číslo, tým dôležitejšia story.
- *Počiatočný odhad* – počiatočný odhad prácnosti implementácie úlohy v porovnaní s inými stories urobenými tímom. Jednotkou sú story pointy, ktoré zvyčajne približne korešpondujú s ideálnymi človekohodinami.
- *Ako robiť demo* – hrubý popis ako bude táto story demonštrovaná počas dema šprintu. Je to v podstate jednoduchá špecifikácia testu "Urob toto, potom toto a toto musí nastať".
- *Poznámky* – akékoľvek ďalšie informácie, vysvetlenia, referencie na iné zdroje atď. Často veľmi stručné.

Ako môžeme zo zoznamu vidieť, pri plánovaní každého šprintu musíme dôkladne zhodnotiť počiatočný odhad, koľko nás bude stáť implementovanie danej úlohy. Pretože na základe časových odhadov jednotlivých úloh vieme na základe našich ohraničení (fixný počet ľudí v tíme a tým aj obmedzený počet odpracovaných hodín počas šprintu) povedať, koľko práce sme schopní v danom šprinte úspešne zvládnuť. Ako teda najlepšie odhadnúť náročnosť a trvanie vypracovania úlohy, aby sme mohli vytvoriť reálny plán?

Taktiež dôležitým aspektom pri plánovaní šprintu je pridelenie úlohy konkrétnemu členovi tímu. Scrum „má radšej“ všestranne zameraných ľudí ako tých, ktorí sa špecializujú len na jediný typ úlohy, ako napríklad testovanie. Inými slovami, členovia tímu by mali robiť to, čo je aktuálne potrebné a pomôcť tým ako je najviac možné. Ak šprint obsahuje veľa testovacích úloh, tak na testovaní by sa mal spolupodieľať celý tím a nie len jedna osoba, ktorá má rada testovanie. To neznamená, že každý by mal byť všeobecne zameraný, ale členovia tímu by mali pracovať spolu a učiť sa od svojich kolegov.

Pri plánovaní a prideľovaní jednotlivých úloh nie je nutné, ba dokonca ani vhodné žiadať si len úlohy, ktoré „viem spraviť najlepšie“. Naopak, je lepšie vybrať si jednu úlohu, pre ktorú mám predpoklady a uvažovať na vybratí úlohy, ktorá si vyžaduje doplňujúce štúdium (možno v kooperácii s členom tímu, ktorý je v danej oblasti zručnejší). Treba však dbať na to, aby nedošlo k tomu čo sa prihodilo v prvom šprinte nášmu tímu, kedy sa priradila úloha kolegovi, ktorý sa s rozoberanou problematikou stretol prvýkrát a precenil svoje schopnosti. Nanešťastie nám to dal neskoro vedieť, čím spôsobil značne spomalenie šprintu, keďže na vypracovanie jeho úlohy nadväzovali ďalšie. To sa dá jednoducho vyriešiť častejšími stretnutiami tímu a otvorenejšou komunikáciou.

Ako presne odhadovať?

Ako som už spomínal, pri plánovaní je dôležitý odhad náročnosti a času potrebného pre úspešne vyriešenie úlohy. Je veľmi náročné dobre odhadnúť presný čas, koľko bude trvať vypracovanie úlohy. V prvom rade to závisí od jej náročnosti. Konkrétna úloha môže byť pre môjho kolegu o polovicu menej náročnejšia než pre mňa. Takisto je tu možnosť, že pri vypracovaní úlohy natrafíme na ťažko predvídateľné problémy atď. Z tohto vyplýva, že plánovanie času vyhradeného pre realizáciu konkrétnej úlohy je dosť náročné, pretože jeho definovanie sa môže ľahko stať ilúziou. Odhadovanie je tímovou aktivitou – každý člen tímu je obyčajne zahrnutý do odhadovania každej úlohy. Prečo?

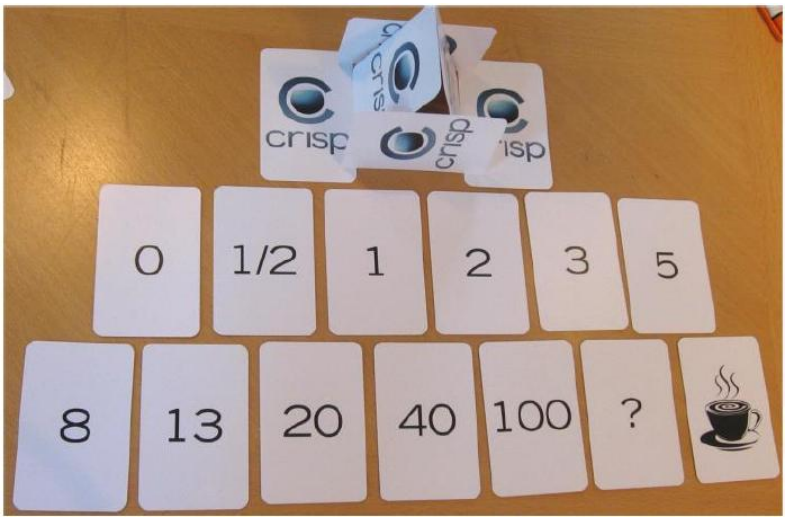
- Na začiatku plánovania väčšinou nemáme informácie o tom, aké úlohy budeme v rámci tímu implementovať.
- Zadania sa môžu týkať rôznych typov znalostí a na riešenie vyžadujú väčší počet členov tímu (dizajn používateľského rozhrania, programovanie, testovanie atď.).
- Aby sme stanovili odhad, každý člen tímu potrebuje porozumieť požiadavkám. To zvýši pravdepodobnosť, že členovia tímu si môžu navzájom pomôcť porozumieť pojednávaný kontext. Zároveň to zvýši pravdepodobnosť, že dôležité otázky sa objavia skôr.

Na podporu odhadov máme k dispozícii viacero metód. Uvediem dva príklady, s ktorými som mal osobnú skúsenosť:

1. *Odhadovanie času na základe priemeru jednotlivých odhadov každého z členov tímu* – v tejto metóde sa konečný odhad získa tak, že každý člen tímu vyjadrí svoj názor, aký čas je adekvátny pre úspešne vyriešenie úlohy. Následne sa spraví priemer zo

všetkých odhadov, ktorý predstavuje konečný odhad. Táto metóda nie je najideálnejšia, pretože členovia tímu sú ovplyvnení predošlými vyjadreniami ich kolegov. Ak si jeden z členov odhadne robotu na 3 story pointy a iný kolega si myslí, že na vypracovanie je potrebných 12 story pointov, môže byť ovplyvnený výpoveďou svojho kolegu a svoj odhad rázne zmenšiť. Avšak možno prvý člen tímu úlohu podcenil. Riešenie tohto problému ponúka druhá metóda.

2. *Odhadovanie času použitím plánovacieho pokru* – pri tejto metóde dostane každý člen tímu kôpku 13 kariet, podobných ako je na obrázku nižšie. Kedykoľvek je story odhadovaná, každý člen tímu si vyberie kartu, ktorá reprezentuje jeho odhad času (v story pointoch) a umiestni ju na stôl číslom dole. Keď ich už všetci položili, karty sa naraz odkryjú. Takto je každý člen tímu nútený premýšľať za seba a nie prikláňať sa k odhadu niekoho iného. Ak sú veľké rozdiely medzi dvoma odhadmi, tím diskutuje rozdiely a skúša si vytvoriť spoločný obraz o akú prácu pôjde. Potom tím odhaduje opäť. Tento cyklus je opakovaný pokiaľ sa časový odhad nezblíži [4].



Obr. 2. Príklad kariet plánovacieho pokru

Pri aplikovaní prvej metódy sa odhady v našom tíme od skutočného času potrebného pre vypracovanie konkrétnych úloh značne líšili. Dôvodov mohlo byť samozrejme niekoľko (zlé pochopenie úlohy, rozdielne vedomosti a zručnosti a pod.) ale po prvých vyjadreniach odhadov, bolo jasné k akým hodnotám odhad smeruje. Pretože ostatní si už vypočuli návrhy, čo ich do istej miery ovplyvnilo.

Použitím alternatívnej metódy plánovacieho pokru sa problém s ovplyvňovaním vyriešil, ba dokonca karty vniesli do odhadov viac aktivity zo strany členov tímu. Pretože už nikto nemohol čakať na vyjadrenie ostatných, podľa ktorých by sa riadil. Často sa využívala aj karta, na ktorej je zobrazená šálka kávy („nevládzem premýšľať, dajme si

krátku prestávku“), ktorá nás zakaždým milo pobavila a vytvorila tak príjemnejšiu atmosféru pri diskusiách.

Záver

Plánovanie je neodmysliteľnou súčasťou vývoja softvéru. Agilné metódy plánovania nám umožňujú vyvíjať softvér z inej perspektívy, ako sme možno boli doteraz zvyknutí. Oceňujú najmä spoluprácu v tíme, spätnú väzbu, pozitívne pristupujú k zmenám a vyžadujú si aktívnu účasť zákazníka, čím sa dosahuje jeho väčšia spokojnosť, ktorá je prioritou riešenia každého projektu. Medzi jednoduché a účinné metódy patrí Scrum, ktorý ponúka viacero možností ako dobre odhadnúť a naplánovať vývoj tak, aby doviedol projekt do úspešného konca.

Na záver uvediem citát na zamyslenie, ktorého autorom je vedec Charles Darwin: „Prežije nie ten druh, ktorý je najsilnejší, ale ten, ktorý sa dokáže najviac prispôsobiť“.

Použitá literatúra

1. Reifer, D.J., How good are agile methods? IEEE Software 19, 4, (2002), 16 - 18.
2. Beck, K. at al. Manifesto for Agile Software Development (2001). Dostupné na internete: http://www.agilemanifesto.org
3. Lycett, M., et al. Resolving the tensions of agility in standardized practice. *IEEE Computer 36* ,6, (June 2003), 79-85.
4. Kniberg, H., SCRUM and XP from the trenches, 2007. Dostupné na internete: http://www.infoq.com/minibooks/scrum-xp-from-the-trenches
5. Deemer, P., Benefield, G., Larman, C., The ScrumPrimer, 2008. Dostupné na internete: http://www.scrumalliance.org/resources/339

Annotation

Do you plan agile?

To what extent we are able to successfully realize a software project according to the customer's specifications depends predominantly on the fact how well the project has been prepared, i.e. how well work, resources and time have been divided during its development. The key factor is planning, which is an essential part of not only big but also small projects. There is a broad spectrum of methods for formulation and supervision of software development plans, however, none of them can prevent unforeseeable events. For this reason, it is advisable to show an agile approach by the plan formation and to be capable of a prompt reaction when the customers change their requirements. Why an agile approach to planning? Every development team that wants to bring the project to a successful end pays attention to the issue of planning and project changes. Modern agile methods such as SCRUM are becoming more and more popular. Is it possible to plan the mentioned way of software development and thus make the software development more effective? How can we estimate how demanding and time consuming it will be to successfully solve a task? This essay tries to

answer these questions and to offer some ideas and recommendations on how to plan an agile software development.