Ohjelmistuotuotanto

Luento 3

17.3.

Vaatimusmäärittely

- Ehkä keskeisin ongelma ohjelmistotuotantoprosessissa on määritellä asiakkaan **vaatimukset** (requirements) rakennettavalle ohjelmistolle
- Vaatimukset jakaantuvat kahteen luokkaan
  - **Toiminnalliset vaatimukset** (functional requirements)
    - Ohjelman toiminnot: mitä ohjelma tekee ja mitä sillä voidaan tehdä
  - **Ei-toiminnalliset vaatimukset** (nonfunctional requirements)
    - Koko ohjelmistoa koskevat ”laatuvaatimukset”
    - Ohjelmiston toimintaympäristön rajoitteet
- Vaatimusten selvittämistä, dokumentoimista ja hallinnointia kutsutaan **vaatimusmäärittelyksi**, engl. **requirements engineering**
- Käytettävästä prosessimallista riippumatta vaatimusmäärittelyn tulee ainakin alkaa ennen ohjelmiston suunnittelua ja toteuttamista
  - Lineaarisissa prosessimalleissa vaatimusmäärittely tehdään kokonaisuudessaan ennen ohjelmiston suunnittelua ja toteutusta
  - Iteratiivisessa ohjelmistokehityksessä vaatimusmäärittelyä tapahtuu paloittain ohjelmiston rakentamisen edetessä

Vaatimusmäärittelyn vaiheet

- Vaatimusmäärittelyn luonne vaihtelee paljon riippuen kehitettävästä ohjelmistosta, kehittäjäorganisaatiosta ja ohjelmistokehitykseen käytettävästä prosessimallista
- Joka tapauksessa asiakkaan tai asiakkaan edustajan on oltava prosessissa aktiivisesti mukana
- Vaatimusmäärittely jaotellaan yleensä muutamaan työvaiheeseen
  - Vaatimusten kartoitus (engl. elicitation)
  - Vaatimusanalyysi
  - Vaatimusten validointi
  - Vaatimusten dokumentointi
  - Vaatimusten hallinnointi
- Useimmiten työvaiheet limittyvät ja vaatimusmäärittely etenee spiraalimaisesti tarkentuen
  - Ensin kartoitetaan, analysoidaan ja dokumentoidaan osa vaatimuksista
  - Prosessia jatketaan kunnes haluttu määrä vaatimuksia on saatu dokumentoitua tarvittavalla tarkkuudella

Vaatimusten kartoituksen menetelmiä

- Selvitetään järjestelmän **sidosryhmät** (stakeholders) eli tahot, jotka ovat suoraan tai epäsuorasti tekemisissä järjestelmän kanssa
- Käytetään ”kaikki mahdolliset keinot” vaatimusten esiin kaivamiseen, esim.:
  - Haastatellaan sidosryhmien edustajia
  - Pidetään brainstormaussessioita asiakkaan ja kehittäjien kesken
- Alustavien keskustelujen jälkeen kehittäjätiimi voi strukturoida vaatimusten kartoitusta
  - Mietitään järjestelmän kuviteltuja käyttäjiä ja keksitään käyttäjille tyypillisiä käyttöskenaarioita
  - Tehdään paperiprototyyppejä ja käyttöliittymäluonnoksia
- Skenaarioita ja prototyyppejä läpikäymällä asiakas voi tarkentaa näkemystään vaatimuksista
- Jos kehitettävän järjestelmän on tarkoitus korvata olemassa oleva järjestelmä, voidaan vaatimuksia selvittää myös havainnoimalla loppukäyttäjän työskentelyä (etnografia)

Vaatimusten analysointi, validointi ja dokumentointi

- Vaatimusten keräämisen lisäksi vaatimuksia täytyy *analysoida*:
  - Onko vaatimuksissa keskinäisiä ristiriitoja
  - Ovatko vaatimukset riittävän kattavat
  - Ovatko vaatimukset sellaisia että niiden toteutuminen on mahdollista ja testattavissa
- Vaatimukset on myös pakko *dokumentoida* muodossa tai toisessa
  - Ohjelmistokehittäjiä varten: mitä tehdään
  - Testaajia varten: toimiiko järjestelmä niin kuten vaatimus määrittelee
  - Usein vaatimusdokumentti toimii oleellisena osana asiakkaan ja ohjelmistotuottajatiimin välisessä sopimuksessa
- Ja *validoida*:
  - Onko asiakas vielä sitä mieltä että kirjatut vaatimukset edustavat asiakkaan mielipidettä

Vaatimusten luokittelu – toiminnalliset vaatimukset

- Vaatimukset jakaantuvat toiminnallisiin ja ei-toiminnallisiin vaatimuksiin
- **Toiminnalliset vaatimukset** (functional requirements) kuvaavat mitä toimintoja järjestelmällä on
  - Esim.
    - *Asiakas voi lisätä tuotteen ostoskoriin*
    - *Onnistuneen luottokorttimaksun yhteydessä asiakkaalle vahvistetaan ostotapahtuman onnistuminen sähköpostitse*
- Toiminnallisten vaatimusten dokumentointi voi tapahtua esim.
  - ”feature-listoina”
  - UML-käyttötapauksina (ks. OTM)
  - Ketterissä menetelmissä yleensä **User storyinä**, joihin tutustumme kohta tarkemmin

Ei-toiminnalliset vaatimukst

- Jakautuvat kahteen luokkaan: **laatuvaatimuksiin ja toimintoympäristön rajoitteisiin**
- Laatuvaatimukset, ovat koko järjestelmän toiminnallisuutta rajoittavia/ohjaavia tekijöitä, esim.
  - Käytettävyys
  - Testattavuus
  - Laajennettavuus
  - Suorituskyky
  - Skaalautuvuus
  - Tietoturva
  - http://en.wikipedia.org/wiki/List_of_system_quality_attributes
- Toimintaympäristön rajoitteita ovat esim:
  - Toteutusteknologia
  - Integroituminen muihin järjestelmiin
  - Mukautuminen standardeihin
- Vaikuttavat yleensä ohjelman arkkitehtuurin suunnitteluun

Vaatimusmäärittely 1900-luvulla

- Vesiputousmallin hengen mukaista oli, että vaatimusmäärittelyä pidettiin erillisenä ohjelmistoprosessin vaiheena, joka on tehtävä kokonaisuudessaan ennen suunnittelun aloittamista
  - Ideana oli että suunnittelun ei pidä vaikuttaa vaatimuksiin ja vastaavasti vaatimukset eivät saa rajoittaa tarpeettomasti suunnittelua
- Asiantuntijat korostivat, että vaatimusten dokumentaation on oltava kattava ja ristiriidaton
  - Pidettiin siis ehdottoman tärkeänä että kerätään ja dokumentoitiin *kaikki* asiakkaan vaatimuksen
  - mielellään luonnollisen kielen sijasta formaalilla kielellä tehty jotta esim. ristiriidattomuuden osoittaminen olisi mahdollista
- Jos määrittelyvaiheessa tehdään virhe joka huomataan vasta testauksessa, on muutoksen tekeminen kallista
  - Tästä loogisena johtopäätöksenä oli tehdä vaatimusmäärittelystä erittäin järeä ja huolella tehty työvaihe
- Vaatimusdokumenttipohjia standardoitiin
  - *IEEE Recommended Practice for Software Requirements Specifications , ks.* http://ieeexplore.ieee.org

Vaatimusmäärittely 1900-luvulla – ei toimi

- Ideaali jonka mukaan vaatimusmäärittely voidaan irrottaa kokonaan erilliseksi, huolellisesti tehtäväksi vaiheeksi on osoittautunut utopiaksi
- Vaatimusten muuttumien on väistämätöntä
  - Huolimatta huolellisesta vaatimusten määrittelemistä, eivät ohjelmistokehittäjät osaa tulkita kirjattuja vaatimuksia samoin kuin vaatimukset kertonut asiakas
  - Ohjelmistoja käyttävien organisaatioiden toimintaympäristö muuttuu nopeasti, mikä on relevanttia tänään, ei ole välttämättä sitä enää 3 kuukauden päästä
  - Asiakkaiden on mahdotonta ilmaista tyhjentävästi tarpeitaan etukäteen
  - Ja vaikka asiakas osaisikin määritellä kaiken etukäteen, tulee mielipide muuttumaan kun asiakas näkee lopputuloksen
- Vaatimusmäärittelyä ei ole mahdollista/järkevää irrottaa suunnittelusta
  - Suunnittelu auttaa ymmärtämään ongelma-aluetta syvällisemmin ja generoi muutoksia vaatimuksiin
  - Ohjelmia tehdään maksimoiden valmiiden ja muualta esim. open sourcena saatavien komponenttien käyttö, tämä on syytä ottaa huomioon vaatimuksia laadittaessa
  - Jos suunnittelu ja toteutustason asiat otetaan huomioon, on vaatimusten priorisointi helpompaa: helpompi arvioida vaatimusten toteuttamisen hintaa

Vaatimusmäärittely 2000-luvulla

- Nykyään vallitsee laaja konsensus siitä, että useimmissa tilanteissa vaatimusmäärittelyä ei ole järkevä tehdä kokonaan suunnittelusta ja toteutuksesta irrallaan
- Syitä tälle
  - **Time to market:** ohjelmistotuotteet halutaan markkinoille nopeasti ja perinpohjaiselle, kuukausia kestävällä vaatimusmäärittelylle ei ole aikaa
  - Tämän takia kaikkia vaatimuksia ei edes teoriassa ehditä kartoittamaan ja siitä taas seuraa **muuttuvat vaatimukset**
  - **Uusiokäyttö, ohjelmistojen koostaminen palveluista:** ohjelmistoja tehdään enenevissä määrin räätälöimällä valmiista komponenteista ja verkossa/pilvessä olevista palveluista, vaatimukset riippuvat näin enenevissä määrin muustakin kuin asiakkaan tahdosta
- Ilman suunnittelun ja toteutuksen huomioimista vaikea tietää vaatimusten toteuttamisen hintaa
  - Riskinä että asiakas haluaa vaatimuksen muodossa, joka moninkertaistaa toteutuksen hinnan verrattuna periaatteessa asiakkaan kannalta yhtä hyvään, hieman eri tavalla muotoiltuun vaatimukseen

Ohjelmiston suunnitteluun ja toteutukseen integroitu vaatimusmäärittely

- 2000-luvun iteratiivisen ja ketterän ohjelmistotuotannon tapa on integroida kaikki ohjelmistotuotannon vaiheet yhteen
- Ohjelmistoprojektin alussa määritellään vaatimuksia tarkemmalla tasolla ainakin yhden iteraation tarpeiden verran
- Ohjelmistokehittäjät arvioivat vaatimusten toteuttamisen hintaa
- Asiakas priorisoi vaatimukset siten, että iteraatioon valitaan toteutettavaksi vaatimukset, jotka tuovat mahdollisimman paljon liiketoiminnallista arvoa
- Jokaisen iteraation aikana tehdään määrittelyä, suunnittelua, ohjelmointia ja testausta
- Jokainen iteraatio tuottaa valmiin osan järjestelmää
    - Edellisen iteraation tuotos toimii syötteenä seuraavan iteraation vaatimusten määrittelyyn
- Ohjelmisto mahdollista saada tuotantoon jo ennen kaikkien vaatimusten valmistumista
- *Kattavana teemana tuottaa asiakkaalle maksimaalisesti arvoa*

Vaatimusmäärittely ja projektisuunnittelu ketterässä prosessimallissa

Taustaa

- Seuraavassa esitellään yleinen tapa vaatimustenhallintaan ja projektisuunnitteluun ketterässä ohjelmistotuotantoprojektissa
- Tapa pohjautuu Scrumin ja eXtreme Programingin eli XP:n eräiden käytänteiden soveltamiseen
- Lähteenä on käytetty mm. seuraavia:
  - Kniberg Scrum and XP from the trenches, sivut 9-55
  - Shore: Art of agile development, osa luvusta 8
  - Rasmussen: The Agile Samurai, luvut 6-8
- Kaikissa edellisissä käydään läpi suunilleen samat asiat, terminologia ja painotukset hieman vaihtelevat (Kniberg käyttää Scrumin ja muut XP:n terminologiaa). Tärkeimmöt erot terminologiassa
  - Scrumin sprinttiä kutsutaan XP:ssä iteraatioksi
  - XP:n on-site customer on suunilleen sama kuin Scrumin Product owner
  - XP:ssä ei ole selvää vastinetta Scrum Masterille, koko tiimi jakaa vastuun prosessin noudattamisesta
- Erittäin kattavan kuvan asioihin antavat Mike Cohnin loistavat kirjat *Agile Estimation and Planning* ja *User stories applied*

User story

- Ketterän vaatimusmäärittelyn tärkein työväline on **User story**
  - Käsitteelle ei ole vakiintunutta käännöstä, joten käytämme jatkossa englanninkielistä termiä
- Alan suurimman auktoriteetin Mike Cohnin mukaan:

  A user story describes functionality that will be valuable to either user or purchaser of software. User stories are composed of three aspects:

  1) A written **description** of the story used for planning and reminder
  2) **Conversations** about the story to serve to flesh the details of the story
  3) **Tests** that convey and document details and that will be used to determine that the story is complete

- Mitä ylläoleva kuvaus tarkoittaa? Jatketaan user storyihin tutustumista käymällä samalla läpi esimerkkijärjestelmää Kumpula biershop:
  - https://github.com/mluukkai/ohtu2013/wiki/kumpula-biershop
  - http://kumpulabiershop.herokuapp.com/

User story

- User Storyt kuvaavat **loppukäyttäjän kannalta arvoa tuottavia toiminnalisuuksia**
- US:n ”määritelmän” alakohdat 1 ja 2 antavat ilmi sen, että User story on karkean tason tekstuaalinen kuvaus **ja** lupaus/muistutus siitä, että toiminnallisuuden vaatimukset on selvitettävä asiakkaan kanssa
- Seuraavat voisivat olla biersopin User storyjen tekstuaalisia kuvauksia:
  - Asiakas voi lisätä oluen ostoskoriin
  - Asiakas voi poistaa ostoskorissa olevan oluen
  - Asiakas voi maksaa luottokortilla ostoskorissa olevat oluet
- User story ei siis ole perinteinen vaatimusmääritelmä, joka ilmaisee tyhjentävästi miten joku toiminnallisuus tulee toteuttaa
  - User story on ”placeholder” vaatimukselle, muistilappu ja lupaus, että toiminnallisuuden vaatimukset tulee selventää tarvittavalla tasolla ennen kuin se toteutetaan
- Usein on tapana kirjoittaa User storyn kuvaus pienelle noin 10-15 cm pahvikortille tai postit-lapulle

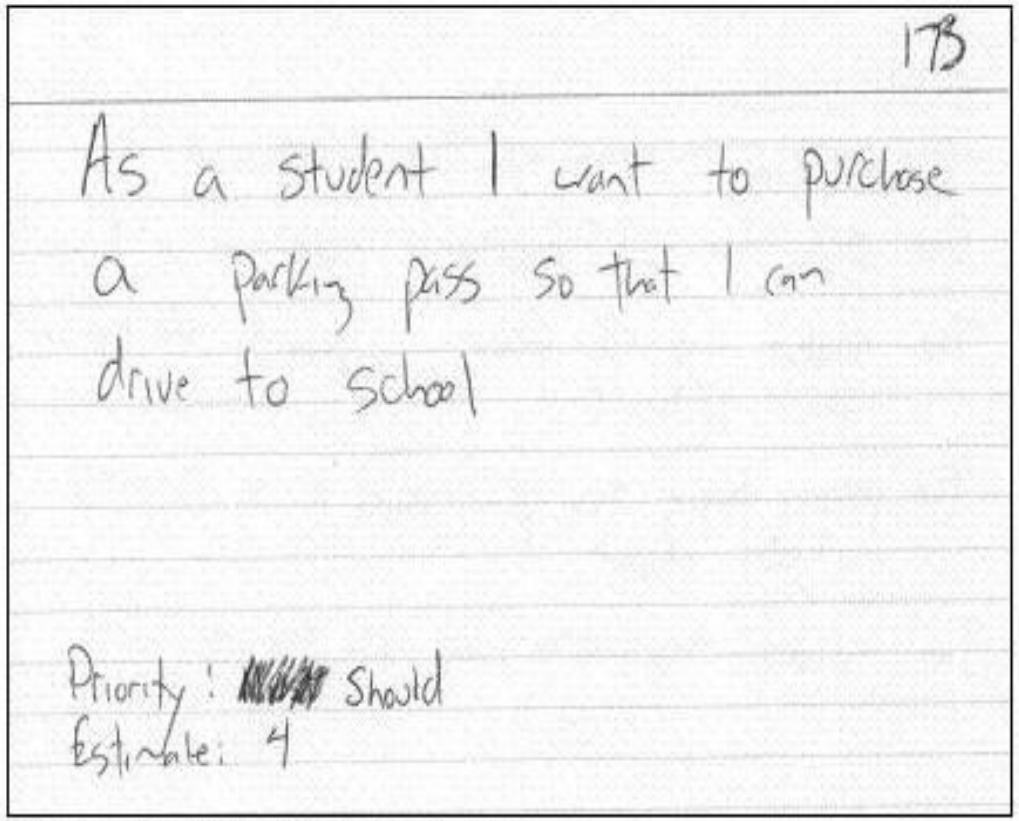
User story

- Kun User story päätetään toteuttaa, on pakko selvittää tyhjentävästi, mitkä ovat Storyn kirjaamaan toiminnon vaatimukset
- User storyn henkeen siis kuuluu, että Story on lupaus kommunikoinnista asiakkaan kanssa vaatimuksen selvittämiseksi
- ”määritelmän” kolmas alikohta sanoo että Storyyn kuuluu *”Tests that convey and document details and that will be used to determine that the story is complete”*
- User storyyn liittyviä testejä kutsutaan yleensä Storyn hyväksymätesteiksi (acceptance test)
- Hyväksymätesti tarkoittaa yleensä joukkoa konkreettisia testiskenaarioita joiden on toimittava, jotta User storyn kuvaaman toiminnallisuuden katsotaan olevan valmis
- Hyväksymätestien luonne vaihtelee projekteittain
  - Ne voitat olla Storyn kuvauksen sisältävän kortin takapuolelle kirjoitettavia tekstuaalisia skenaarioita (varsinkin jos projektissa on käytettävissä onsite customer, joka voi suorittaa hyväksymätestauksen)
  - Tai parhaassa tapauksessa automaattisesti suoritettavia testejä

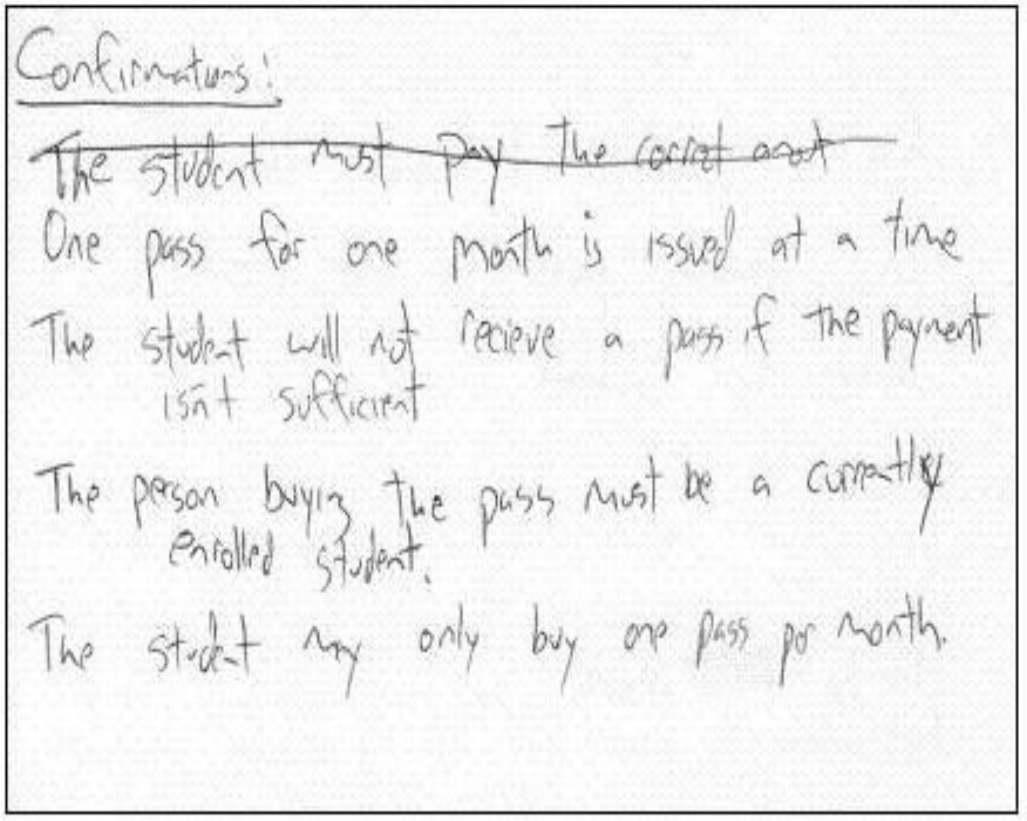
Esimerkki

- Alla esimerkki pahvikortille kirjoitetusta User storystä
- Kortin etupuolella kuvaus, prioriteetti ja estimaatti
  - Estimaatilla tarkoitetaan kortin toiminnallisuuden toteuttamisen työmääräarviota. Palaamme estimointiin pian tarkemmin
- Kortin takapuolella suhteellisen informaalilla kielellä kirjoitettu hyväksymistesti

Front of Card



Back of Card



Minkälainen on hyvä User Story

- Kuten jo mainittu, tulee User storyn kuvata asiakkaalle arvoa tuottavia toimintoja
  - Käytettävä asiakkaan kieltä, ei teknistä jargonia
- Hyvänä käytäntönä pidetään että User story kuvaa järjestelmän kaikkia osia koskevaa (esim. käyttöliittymä, bisneslogiikka, tietokanta) ”end to end”-toiminnallisuutta
  - Esim. ”lisää jokaisesta asiakkaasta rivi tietokantatauluun customrs” ei olisi suositeltava muotoilu User storylle
- Edellisen sivun esimerkki on formuloitu viimeaikaisen muodin mukaisessa muodossa
  - **As a** <type of user>, **I want** <functionality> **so that** <bussines value>
  - As a student I want to purchase a parking pass so that I can drive to school
- Näin muotoilemalla on ajateltu että User story kiinnittää huomion siihen kenelle kuvattava järjestelmän toiminto tuo arvoa
- Muoto ei oikein taivu suomenkielisiin kuvauksiin, joten sitä ei tällä kurssilla käytetä

Minkälainen on hyvä User Story

- Bill Wake luettelee artikkelissa *INVEST in good User Stories* kuusi User storyille toivottavaa ominaisuutta:
  - **I**ndependent
  - **N**egotiable
  - **V**aluable to user or customer
  - **E**stimable
  - **S**mall
  - **T**estable
  - http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/
- **Independent** User storyjen pitäisi olla toisistaan mahdollisimman riippumattomia
  - Riippumattomuus mahdollistaa eri käyttötapausten toteuttamisen mahdollisimman riippumattomasti toisistaan
- Esim biershopin Storyjen *Lisää olut ostoskoriin* ja *Poista olut ostoskorista* välillä on riippuvuus, jota on vaikea välttää
  - Voikin olla parempi yhdistää riippuvaiset User storyt yhdeksi: *tuotteiden lisäys ja poisto ostoskorista*

Minkälainen on hyvä User Story

- **Negotiable** hyvä User story ei ole tyhjentävästi kirjoitettu vaatimusmäärittely vaan lupaus siitä että asiakas ja toteutustiimi sopivat User storyn toiminnallisuuden sisällön ennen toteutusvaihetta
- **Estimatable** User storyn toteuuttamisen vaativa työmäärä pitää olla arvioitavissa kohtuullisella tasolla
- **Small** Työmäärän arviointi onnistuu paremmin jos User storyt ovat riittävän pieniä**.** User storyä pidetään yleensä liian isona jos se ei ole toteutettavissa noin viikon työpanoksella
- Liian suuret User storyt kannattaa jakaa osiin
  - Esim käyttötapaus *Olutkaupan ylläpitäjä voi kirjautua sivulle, lisätä ja päivittää oluiden tietoja sekä tarkastella asiakkaille tehtyjen toimitusten lista* kannattaa jakaa useaan osaan:
    - *Ylläpitäjä voi kirjautua sivulle*
    - *Ylläpitäjä voi lisätä ja päivittää oluiden tietoja*
    - *Ylläpitäjä voi tarkastella asiakkaille tehtyjen toimitusten listaa*
    - *Sivulle kirjautunut ylläpitäjä voi lisätä ja päivittää oluiden tietoja*
    - *Sivulle kirjautunut ylläpitäjä voi tarkastella asiakkaille tehtyjä toimituksia*

Minkälainen on hyvä User Story

- **Testability** Kuudes toivottu ominaisuus on testattavuus, eli User storyjen pitää olla sellaisia että niille on mahdollista tehdä testit, joiden avulla voi yksikäsitteisesti kertoa onko Story toteutettu hyväksyttävästi
- Ei-toiminnalliset vaatimukset (esim. suorituskyky, käytettävyys) aiheuttavat usein haasteita testattavuudelle
  - Esim. käyttötapaus O*lutkaupan tulee toimia tarpeeksi nopeasti kovassakin kuormituksessa* voidaan muotoilla testattavaksi esim. seuraavasti *käyttäjän vasteaika saa olla korkeinaan 0.5 sekuntia 99% tapauksissa jos yhtäaikaisia käyttäjiä sivulla on maksimissaan 1000*

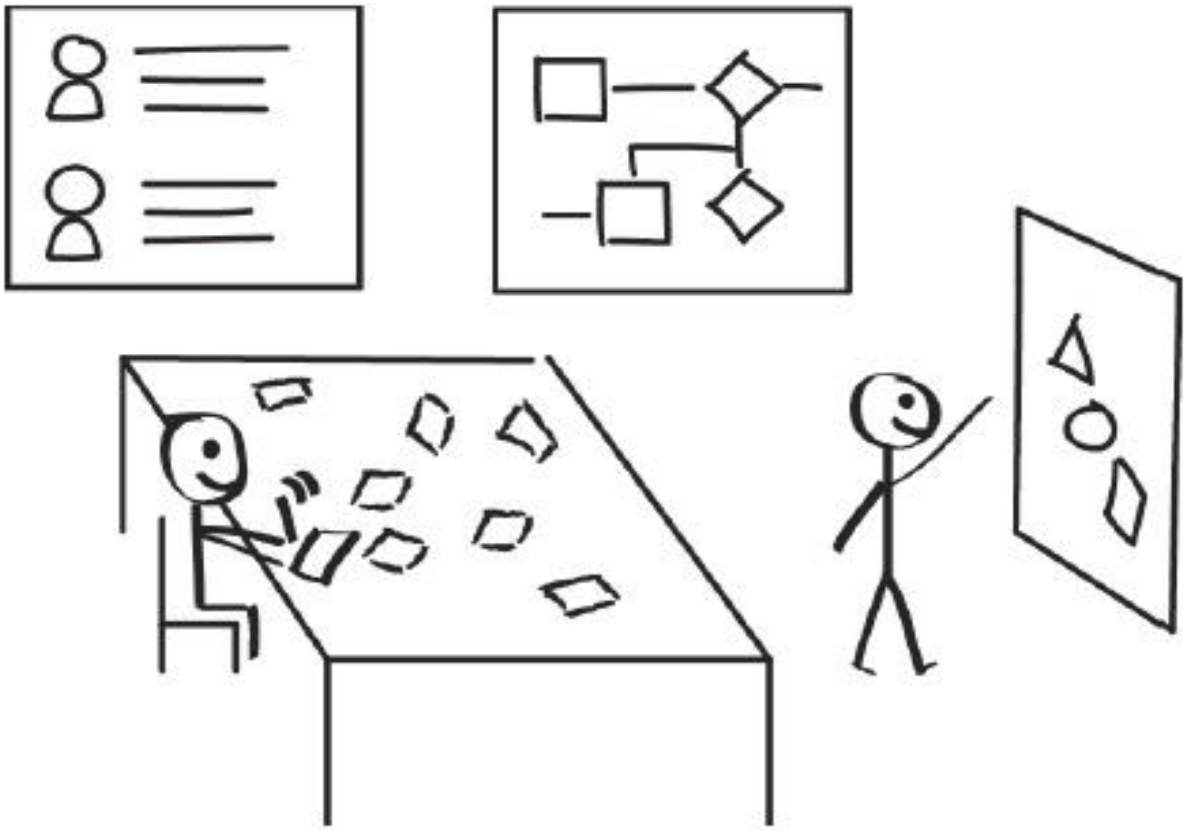
- Viime viikolla Scrumin yhteydessä puhuimme **product backlogista,** joka siis on priorisoitu lista asiakkaan tuotteelle asettamista vaatimuksista eli toivotuista ominaisuuksista ja toiminnoista
- Nykyään käytäntönä on, että product backlog koostuu nimenomaan User storyistä

Alustava product backlog

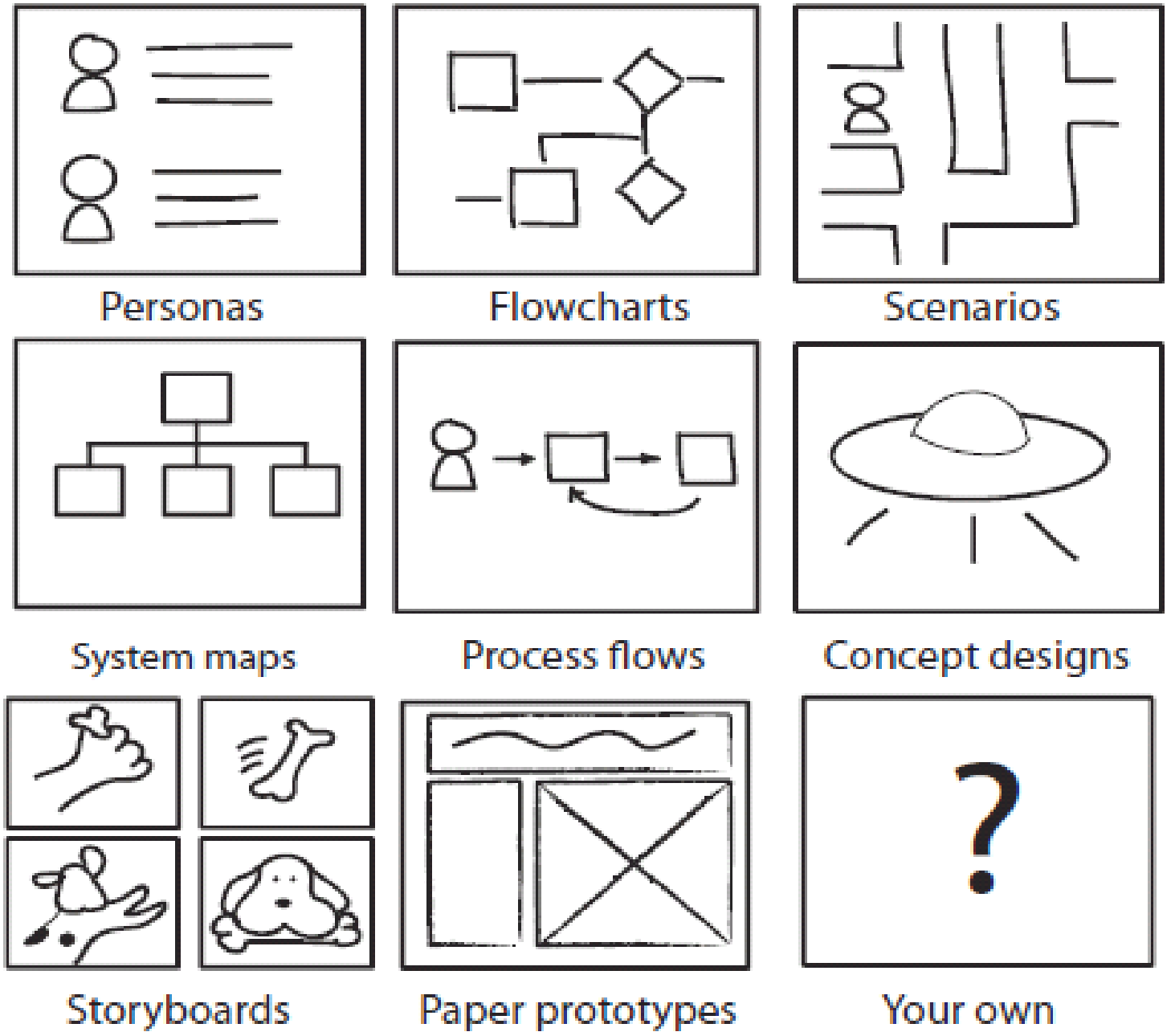
- Projektin aluksi kannattaa heti ruveta etsimään ja määrittelemään User storyja ja muodostaa näistä alustava Product Backlog
- Käytettävissä ovat kaikki yleiset vaatimusten kartoitustekniikat
  - Haastattelut
  - Brainstormaus, story gathering workshopit
- Alustavan User storyjen keräämisvaiheen ei ole tarkoituksenmukaista kestää kovin kauaa, maksimissaan muutaman päivän
- User storyjen luonne (muistilappu ja lupaus että vaatimus tarkennetaan ennen toteutusta) tekee niistä hyvän työkalun projektin aloitukseen
  - Turhiin detaljeihin ei puututa
  - Ei tavoitellakaan täydellistä ja kattavaa listaa vaatimuksista, asioita tarkennetaan myöhemmin
- Kun alustava lista User storyistä on kerätty, ne priorisoidaan ja niiden vaatima työmäärä arvioidaan karkealla tasolla
  - Näin muodostuu alustava Product Backlog, eli priorisoitu lista vaatimuksista

Story gathering workshop

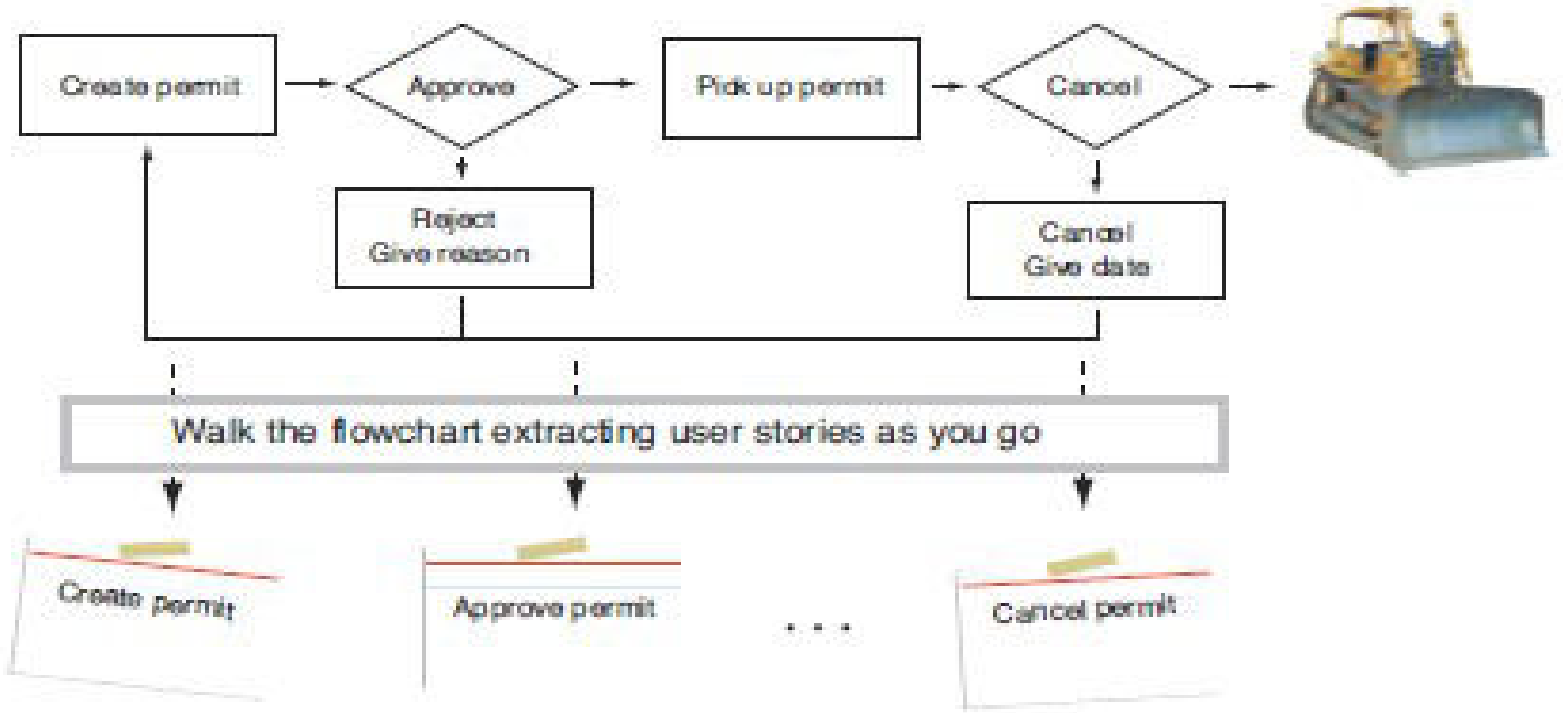
- Ennenkuin menemme tarkemmin User storyjen priorisointiin, esitellään nopeasti Johan Rasmussonin kirjassa Agile Samurai esittämä tapa Storyjen keräämiseen
- Step 1: **get a big room**
  - Huoneeseen kerääntyvät kaikki asianosaiset, asiakkaat ja ohjelmistotuotantotiimi

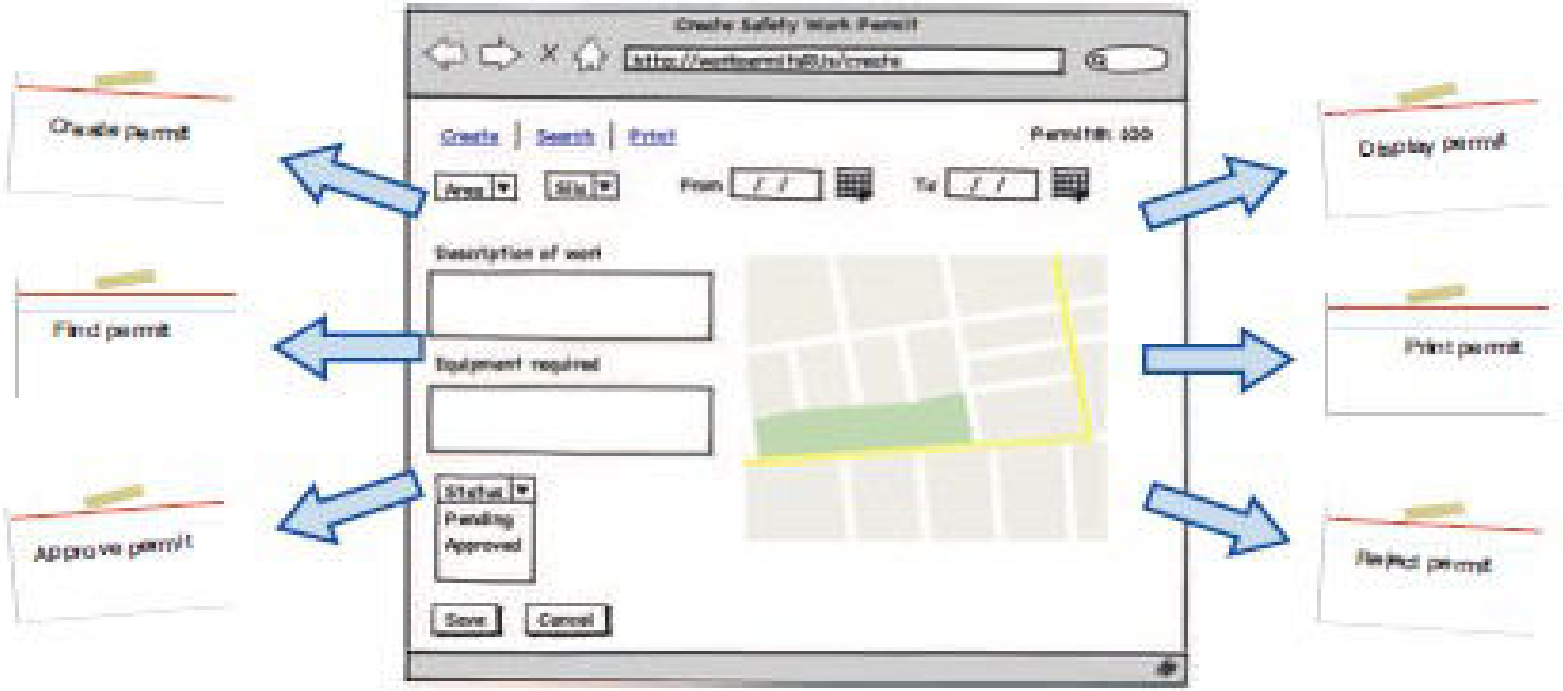


- **Step 2: draw a lot of pictures**



- **Step 3: Write lots of stories**





Story gathering workshop

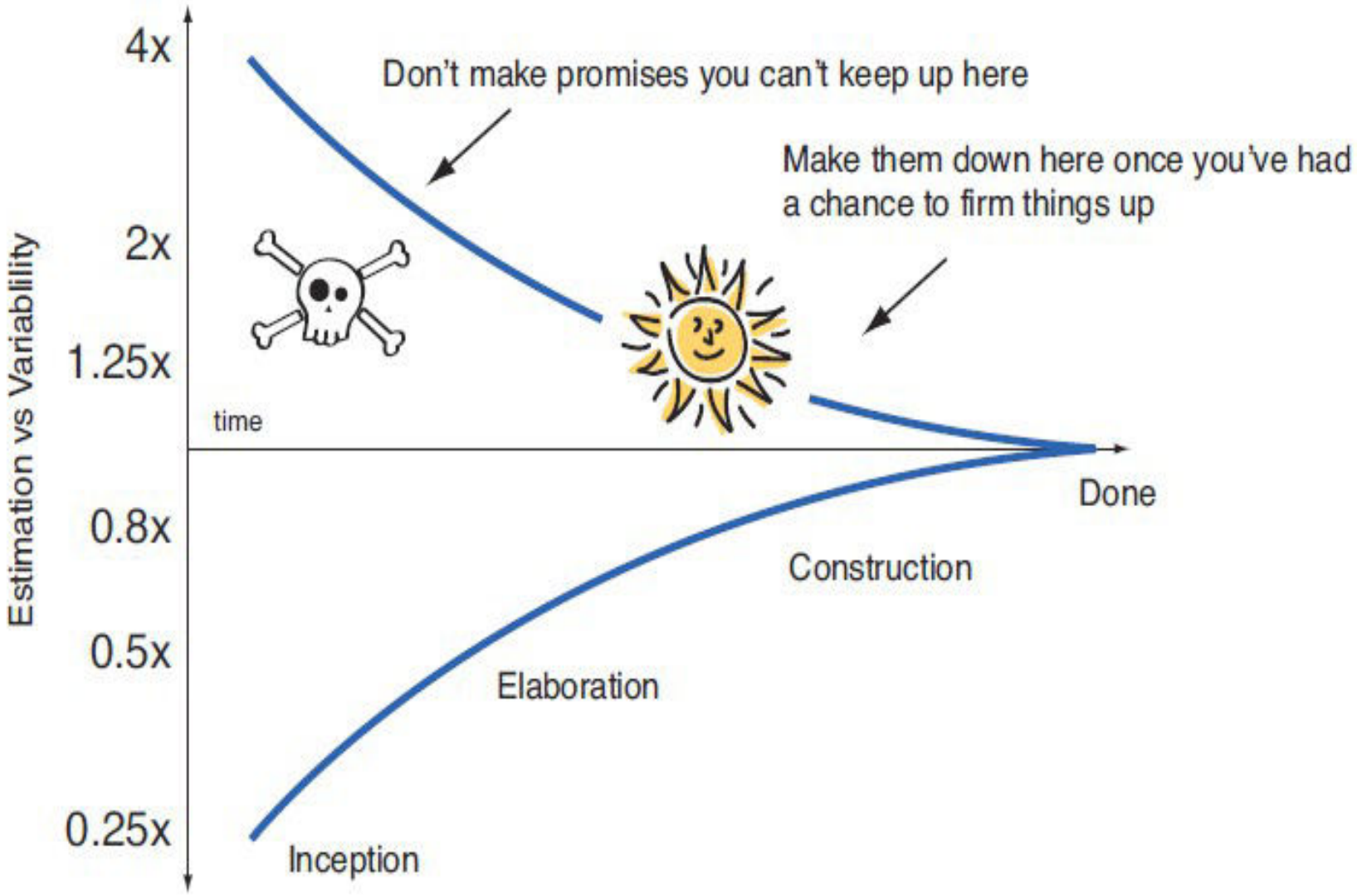
- **Step 4: Brainstorm everything else**
  - Kuvien piirtämisen ja siihen liittyvän brainstormauksen innoittamana saadaan yleensä kirjoitettua suuri joukko User storyjä
  - Kuvin ilmaistavien asioiden lisäksi mietitään muuta projektiin liittyvää ja kirjataan niitä vastaavat User storyt
- **Step 5: Scrub the list and make it shine**
  - Lopuksi siivotaan lista:
    - Poistetaan duplikaatit
    - Yhdistetään liian pienet toisiinsa liittyvät Storyt isommiksi
    - Kirjoitetaan User storyt koherentimpaan muotoon

Backlogin priorisointi

- Product Backlog siis on priorisoitu lista User storyjä
  - Kuten muistamme priorisoinnin hoitaa Product Owner, XP:ssä käytetään suunnilleen samassa roolissa toimivasta henkilöstä nimitystä on-site customer
- Prioriteetti määrää järjestyksen, missä ohjelmistokehittäjät toteuttavat ohjelmiston ominaisuuksia
- Priorisoinnin motivaationa on pyrkiä maksimoimaan asiakkaan kehitettävästä ohjelmistosta saama hyöty
  - Tärkeimmät asiat halutaan toteuttaa mahdollisimman nopeasti ja näin saada tuotteesta alustava versio markkinoille niin pian kuin mahdollista
- User storyjen priorisointiin vaikuttaa Storyn kuvaaman toiminnallisuuden asiakkaalle tuovan arvon lisäksi pari muutakin seikkaa
  - Storyn toteuttamiseen kuluva työmäärä
  - Storyn kuvaamaan ominaisuuteen sisältyvä tekninen riski
- Ei ole siis kokonaistaloudellisesti edullista tehdä priorisointia välttämättä pelkästään perustuen asiakkaan User storyistä saamaan arvoon

Estimointi eli User storyn toteuttamiseen kuluvan työmäärän arviointi

- User storyjen viemän työmäärän arvioimiseen on oikeastaan kaksi motivaatiota
  - Auttaa asiakasta priorisoinnissa
  - Mahdollistaa koko projektin viemän ajan summittaisen arviointi
- Työmäärän arvioimiseen on kehitetty vuosien varrella useita erilaisia menetelmiä.
- Kaikille yhteistä on se, että ne eivät toimi kunnolla, eli tarkkoja työmääräarvioita on mahdoton antaa
  - Joskus työmäärän arvioinnista käytetäänkin leikillisesti termiä *guestimation*
- Mitä kauempana tuotteen valmistuminen on, sitä epätarkempia työmääräarviot ovat
  - Cone of uncertainity, ks. seuraava sivu



- Ketterät ohjelmistotuotantomenetelmät ottavat itsestäänselvyytenä sen, että estimointi on epävarmaa ja tarkentuu vasta projektin kuluessa
  - Koska näin on, pyritään vahvoja estimointiin perustuvia lupauksia aikatauluista olemaan tekemättä

Suhteelliseen kokoon perustuva estimointi

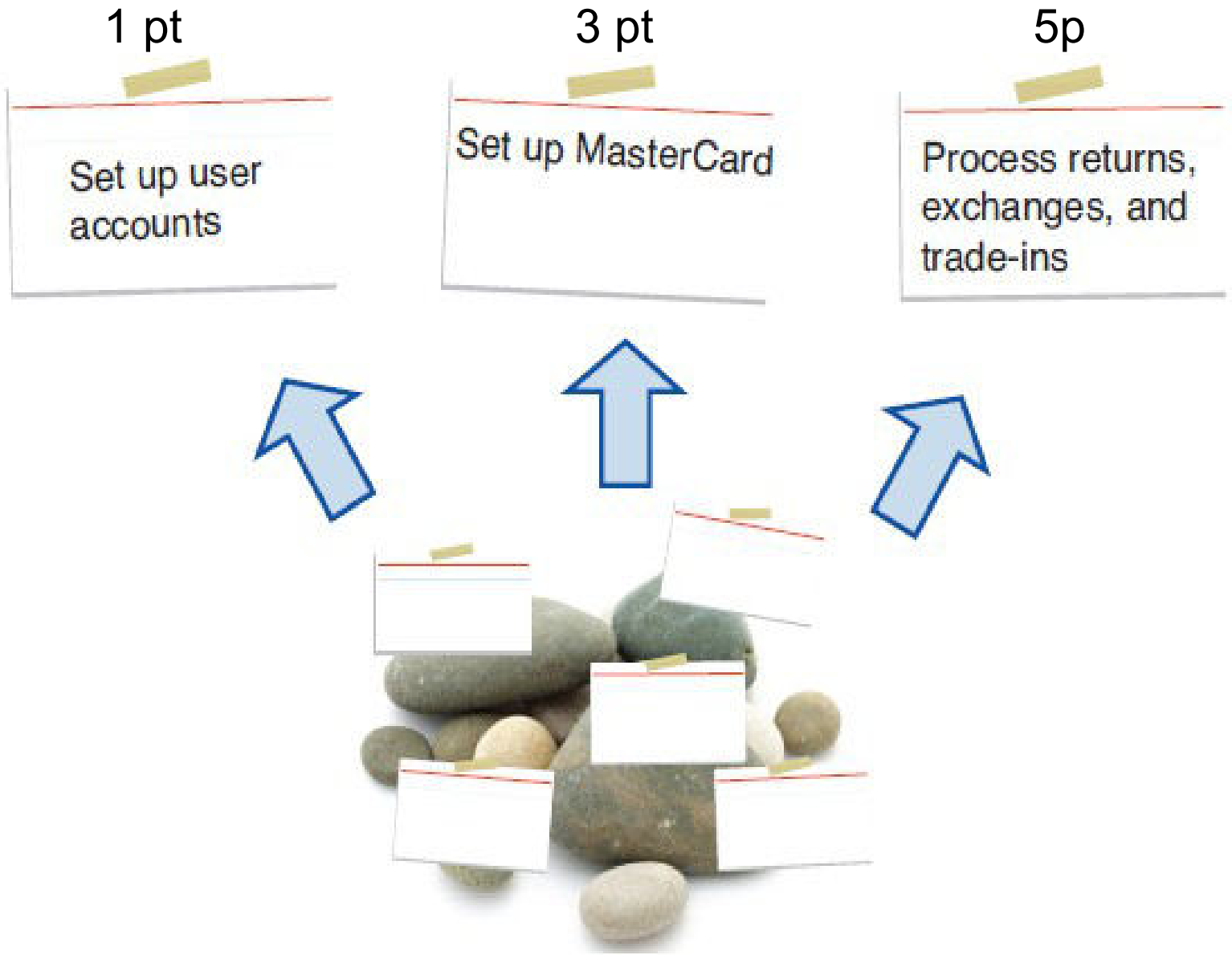
- On huomattu, että vaikka ominaisuuksien toteuttamiseen menevän tarkan ajan arvioiminen on vaikeaa, osaavat ohjelmistokehittäjät jossain määrin arvioida eri tehtävien vaatimaa työmäärää suhteessa toisiinsa
- Esim.
  - User storyn *Tuotteen lisääminen ostoskoriin* toteuttaminen vie yhtä kauan kuin User storyn *Tuotteen poistaminen ostoskorista* toteuttaminen
  - User Storyn O*stoskorissa olevien tuotteiden maksaminen luottokortilla* toteuttaminen taas vie kolme kertaa kauemmin kun edelliset
- Ketterissä menetelmissä käytetäänkin yleisesti suhteelliseen kokoon perustuvaa estimointia
- ”yksikkönä” arvioinnissa on yleensä **Story point**
  - Ei yleensä vastaa mitään todellista tuntimäärää
  - Biershop-projektissa voitaisiin esim. kiinnittää että User storyn *Tuotteen lisääminen olutkoriin* estimaatti on 1 Story point, muita voidaan sitten verrata tähän, eli *Ostoskorissa olevien tuotteiden maksaminen luottokortilla* estimaatiksi tulisi 3 Story pointia

Suhteelliseen kokoon perustuva estimointi

- Kun estimoitavana on suuri määrä User storyjä
  - Esimerkki Rasmussenin kirjasta Agile samurai

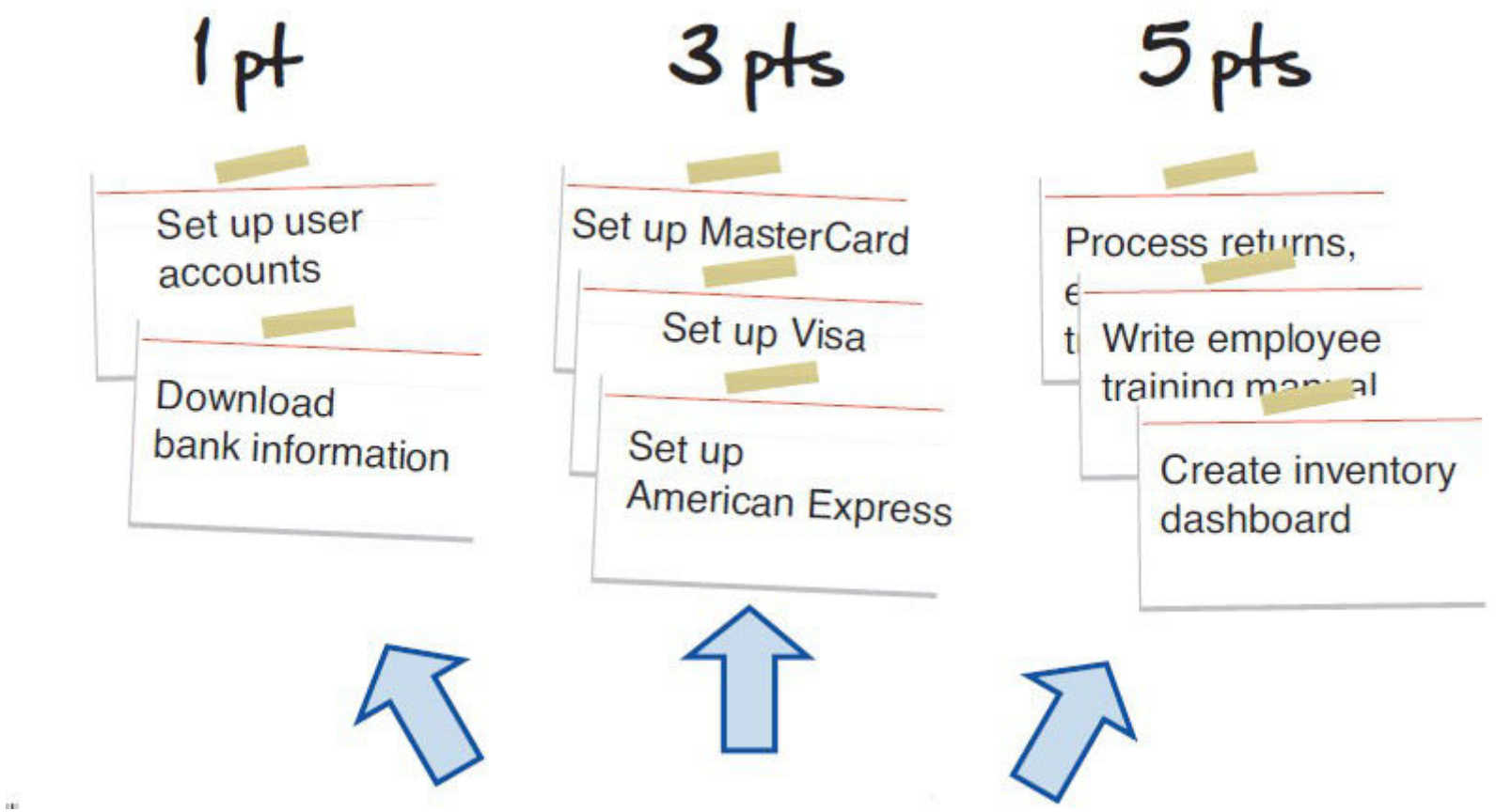
Suhteelliseen kokoon perustuva estimointi

- saattaa olla kannattavaa arvioida ensin muutama hieman erikokoinen Story ja valita nämä referensseiksi



Suhteelliseen kokoon perustuva estimointi

- Ja arvioida muut User storyt näiden suhteen



Kuka suorittaa estimoinnin?

- Estimonti tapahtuu **aina ohjelmistokehitystiimin toimesta**
- Asiakkaan on oltava läsnä tarkentamassa estimoitaviin User storeihin liittyviä vaatimuksia
- Usein estimointia auttaa User storyn pilkkominen teknisiin työvaiheisiin
  - Esim. User story *Tuotteen lisääminen ostoskoriin,* voisi sisältää toteutuksen kannalta seuraavat tekniset tehtävät (task):
    - tarvitaan sessio, joka muistaa asiakkaan
    - domain-olio ostoskorin ja ostoksen esittämiseen
    - html-näkymää päivitettävä tarvittavilla painikkeilla
    - Kontrolleri painikkeiden käsittelyyn
    - yksikkötestit kontrollerille ja domain-olioille
    - hyväksymätestien automatisointi
- Jos kyseessä on samantapainen toiminnallisuus kuin joku aiemmin toteutettu, voi estimointi tapahtua ilman User storyn vaatimien erillisten työvaiheiden miettimistä

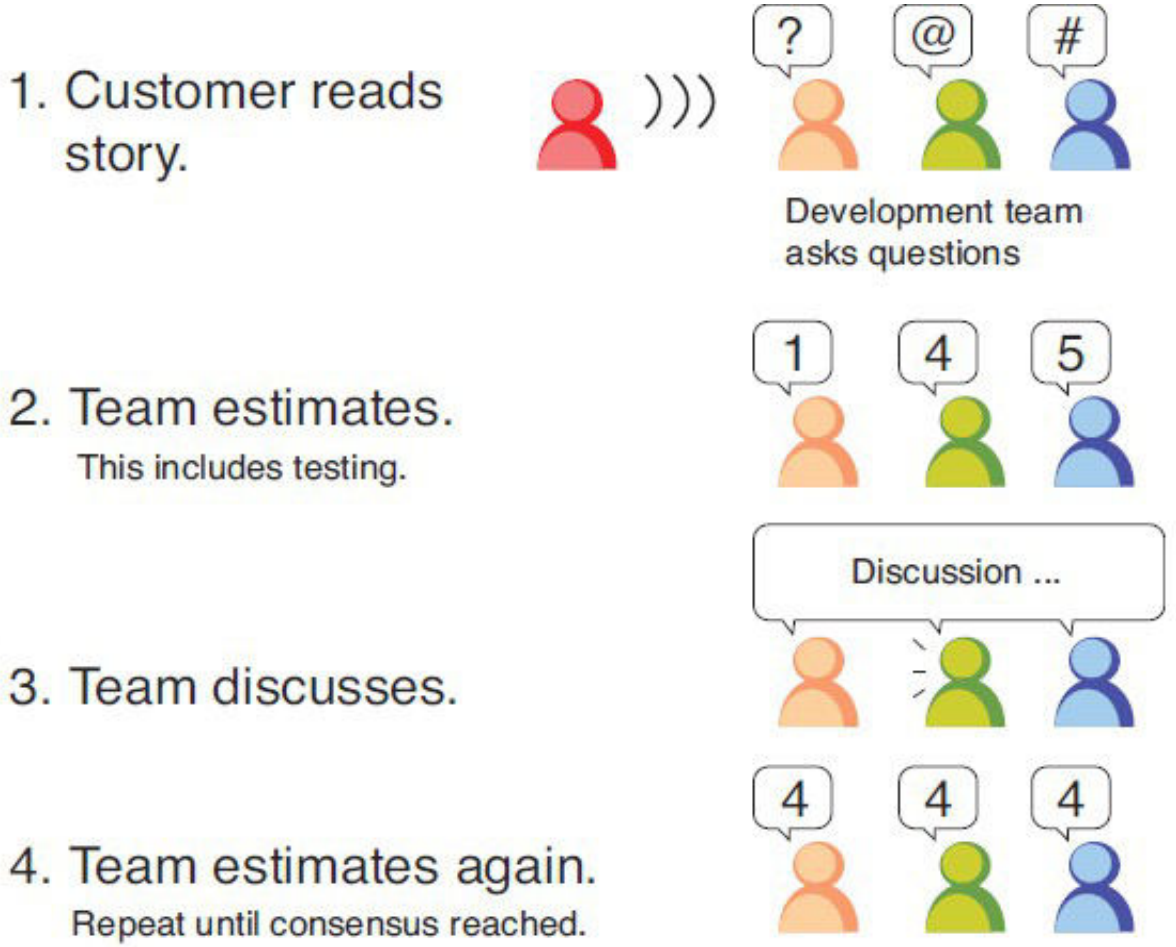
Estimoinnista

- Estimointi on joka tapauksessa suhteellisen epätarkkaa, joten estimoinnin on tarkoitus tapahtua nopeasti
  - yhden User storyn estimointiin kannattaa käyttää aikaa korkeintaan 15 minuuttia, jos se ei riitä, on todennäköistä että Storya ei tunneta vielä niin hyvin että se kannattaisi estimoida
- Kuten viime viikolla mainitsimme, määritellään ketterissä projekteissa yleensä ns. ”definition of done”
- Estimoinnissa tulee arvioida User storyn viemä aika ”definition of donen tarkkuudella”, tämä sisältää kaiken Storyn toteuttamiseen liittyvän:
  - määrittely, suunnittelu, toteutus, automatisoitujen tekstien tekeminen, testaus, integrointi ja dokumentointi

- Äsken mainitsimme että Story point ei vastaa yleensä mitään aikayksikköä
  - Jotkut kuitenkin mitoittavat Story Pointin ainakin projektin alussa ”ideal working day:n” suuruiseksi, eli työpäiväksi johon ei sisälly mitään häiriötekijöitä
  - Useimmat auktoriteetit suosittelevat olemaan sotkematta Story pointeja päiviin
  - ks. esim. http://blog.crisp.se/2008/12/05/tomasbjorkholm/1228470417545

Planning poker

- Hyvänä periaatteena pidetään että kaikki tiimin jäsenet osallistuvat estimointiin
  - Tiimille syntyy yhtenäinen ymmärrys User storyn sisällöstä
- *Planning poker* on eräs suosittu tapa estimoinnin tekemiseen



Planning poker

- Käydään läpi Backlogissa olevia User storyja yksi kerrallaan
- Asiakas lukee User storyn sisällön ja selittää tarkemmin Storyn luonnetta ja vaatimuksia
- Tiimi keskustelee Storystä, miettii kenties Storyn jakautumista teknisiin työvaiheisiin
- Kun kaikki kokevat olevansa valmiina arvioimaan, jokainen kertoo arvionsa (yksikkönä siis Story point)
- Usein tämä vaihe toteutetaan siten, että käytössä on pelikortteja, joilla on estimaattien arvoja, esim 1, 2, 5, 10, ... ja kukin estimointiin osallistunut näyttää estimaattinsa yhtä aikaa
- Jos estimaatit ovat suunnilleen samaa tasoa, merkataan estimaatti User storylle
- Jos seuraa eroavaisuutta, keskustelee tiimi eroavaisuuksien syistä
  - Voi esim. olla, että osa tiimin jäsenistä ymmärtää User storyn vaatimukset eri tavalla ja tämä aiheuttaa eroavaisuutta estimaatteihin
- Kun tiimi on keskustellut aikansa, tapahtuu uusi estimointikierros ja konsensus todennäköisesti saavutetaan pian

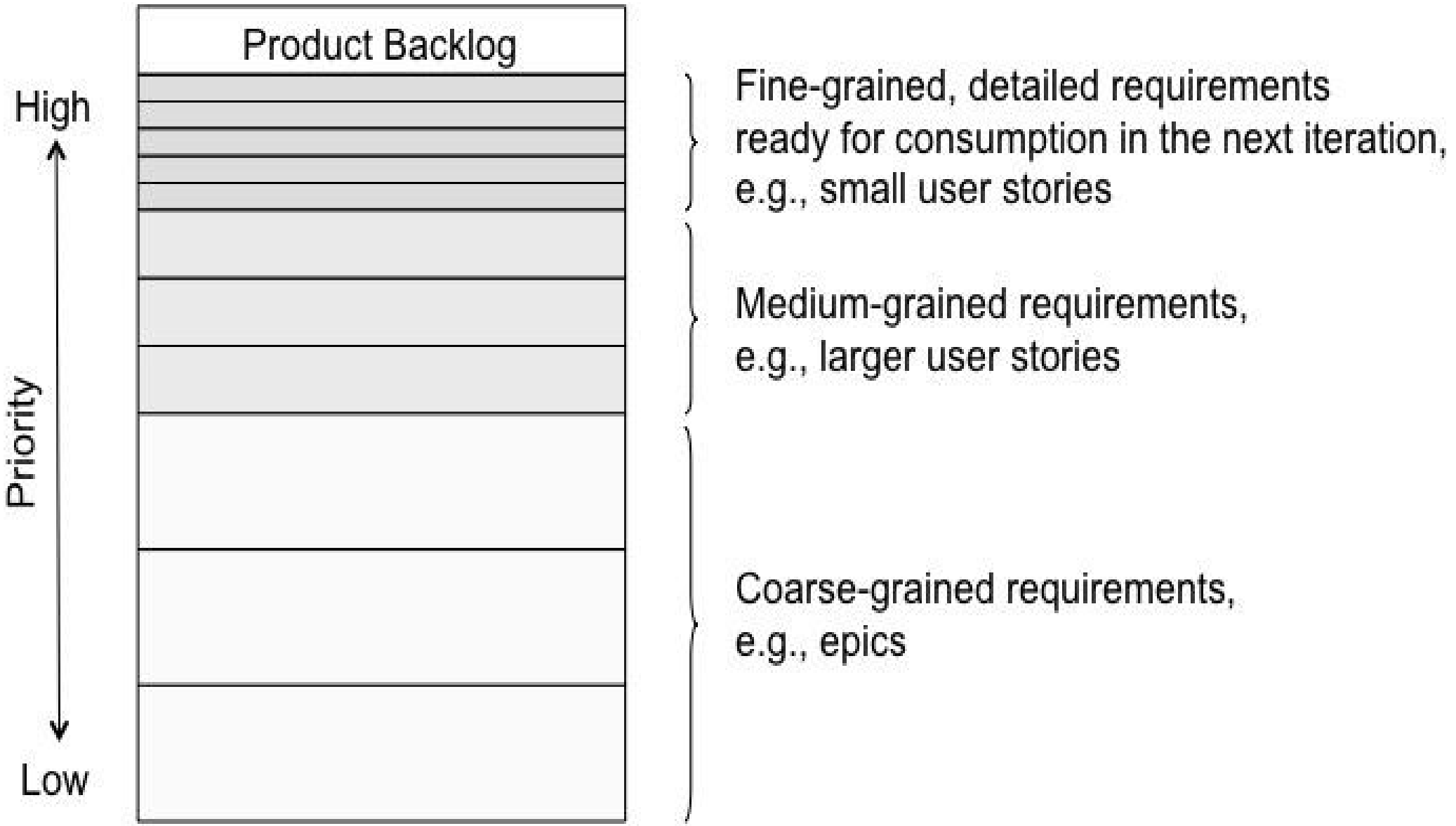
Estimaattien skaala

- Koska estimointi on jokatapauksessa melko epätarkkaa, ei estimoinnissa ole tapana käyttää kovin tarkkaa skaalaa
- Yleistä on esim. käyttää ainoastaan arvoja 1, 2, 3, 5, 10, 20, 40, 100 tai vastaavaa yläpäästä harvenevaa skaalaa
- Motivaationa se, että mitä suuremmasta kokonaisuudesta kyse, sitä vaikeampaa estimointi on, ja skaala yläpäässä on tarkoituksella harva jotteivat estimaatit antaisi valheellista kuvaa tarkkuudesta
- Joskus käytetään myös estimaattia *epic* jolla tarkoitetaan niin isoa tai huonosti ymmärrettyä User storyä että sitä ei voida vielä estimoida
- Alan suurin auktoriteetti Mike Cohn suosittelee käyttämään skaalaa 1, 2, 3, 5, 8 tai 1, 2, 4, 8 ja antamaan sitä suuremmille estimaatti epic

Hyvä product backlog on DEEP

- http://www.romanpichler.com/blog/product-backlog/making-the-product-backlog-deep/
- Mike Cohn lanseerasi lyhenteen DEEP kuvaamaan hyvän backlogin ominaisuuksia
  - **D**etailed appropriatly
  - **E**stimated
  - **E**mergent
  - **P**rioritized
- **D**etailed appropriatly eli sopivan detaljoitu:
  - Backlogin prioriteeteiltaan korkeimpien eli pian toteutettavaksi otettavien User Storyjen kannattaa olla suhteellisen pieniä ja näin tarkemmin estimoituja
  - Alemman prioriteetin User Storyt voivat vielä olla isompia ja karkeammin estimoituja

Hyvä product backlog on DEEP



Hyvä product backlog on DEEP

- DEEP ominaisuuksista **estimated** ja **prioritized** ovat meille tuttuja
- **Emergent** kuvaa backlogin muuttuvaa luonnetta:
  - The product backlog has an organic quality. It evolves, and its contents change frequently. New items emerge based on customer and user feedback, and they are added to the product backlog. Existing items are modified, reprioritized, refined, or removed on an ongoing basis.
- Muuttuvan luonteensa takia backlogia tulee hoitaa (**backlog grooming)** projektin edetessä
  - Backlogiin lisätään uusia User storyja ja vanhoja tarpeettomaksi käyneitä poistetaan
  - Isoja User storyja pilkotaan tarpeentullen pienemmiksi (erityisesti prioriteetin kasvaessa täytyy isot Storyt pilkkoa pienemmiksi)
  - Backlogiin lisättäviä uusia User storyjä estimoidaan ja vanhojen Storyjen estimaatteja tarkastetaan ymmärryksen kasvaessa
  - Backlogin hoitamiseen osallistuu koko ohjelmistotuotantotiimi, pääasiallinen vastuu on Product Ownerilla
- http://www.romanpichler.com/blog/product-backlog/grooming-the-product-backlog/

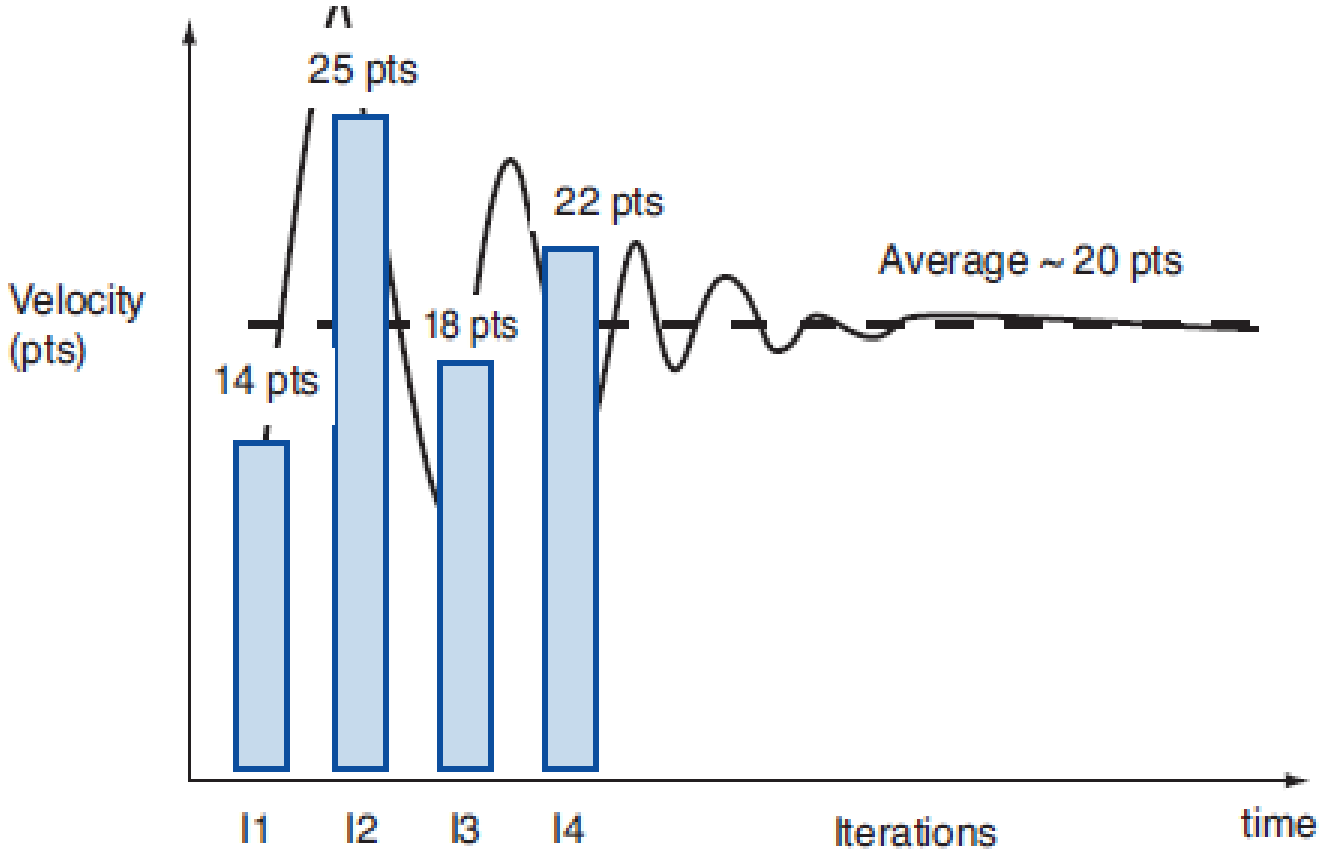
Julkaisun suunnittelu – release planning

- Estimoinnin toinen tarkoitus on, että se **mahdollistaa koko projektin viemän aikamäärän summittaisen arvioinnin** eli julkaisun suunnittelun (engl. release planning)
- Jos estimoinnin yksikkönä kuitenkin on abstrakti käsite Story point, miten estimaattien avulla on mahdollista arvioida projektin viemää aikamäärää?
- Kehitystiimin **velositeetti** (engl velocity) tarjoaa osittaisen ratkaisun tähän
- Velositeetillä tarkoitetaan Story pointien määrää, minkä verran tiimi pystyy keskimäärin toteuttamaan yhden sprintin aikana
- Jos tiimin velositeetti on selvillä ja projektissa toteutettavaksi tarkoitetut User storyt on estimoitu, on helppo tehdä alustava arvio projektin viemästä aikamäärästä

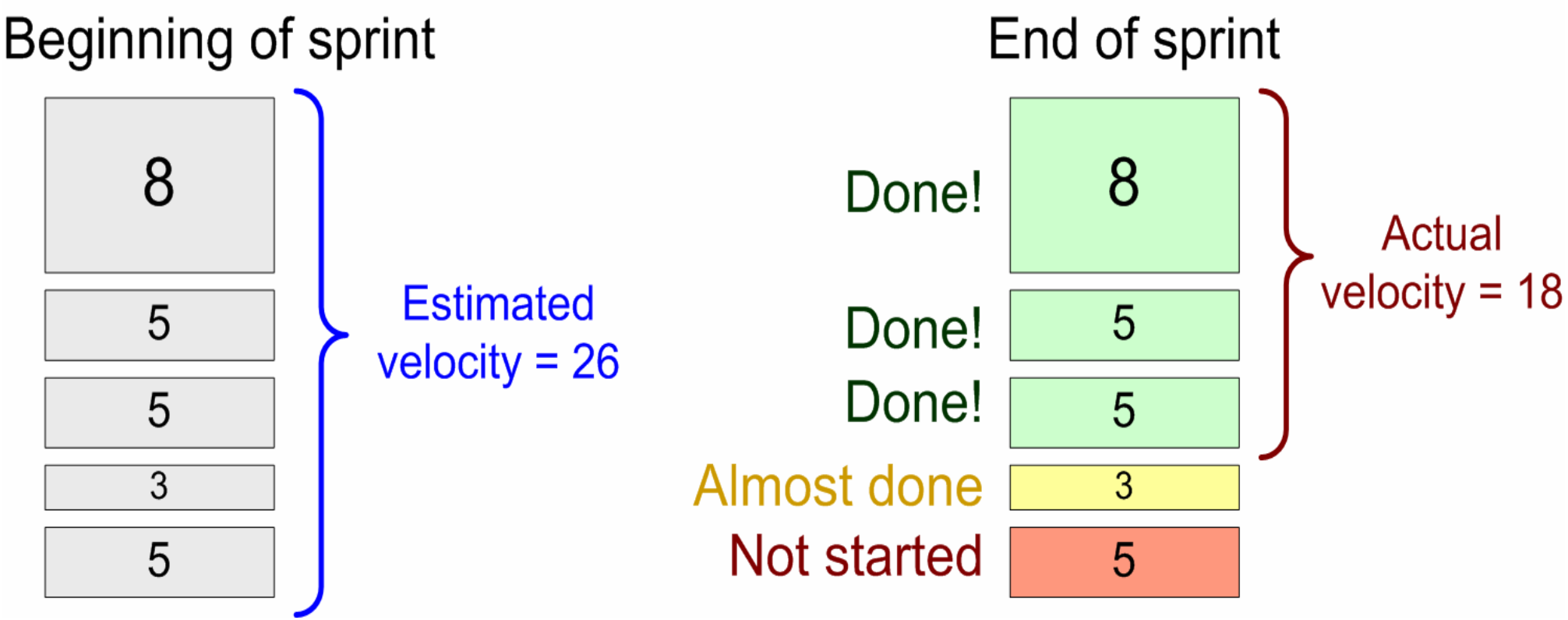
  (User storyjen estimaattien summa) / velositeetti * sprintin pituus
- Projektin alkaessa velositeetti ei yleensä ole selvillä, ellei kyseessä ole jo yhdessä työskennellyt tiimi
- On kehitetty tapoja joiden avulla velositeetti voidaan yrittää ennustaa
  - Hyvin epäluotettavia, emme käsittele niitä nyt

Velositeetti

- Velositeetti vaihtelee tyypillisesti alussa melko paljon ja alkaa stabiloitumaan vasta muutaman sprintin päästä
  - Estimointi on aluksi vaikeampaa varsinkin jos sovellusalue ja käytetyt teknologiat eivät ole täysin tuttuja
- Projektin kestoarvio alkaakin tarkentumaan pikkuhiljaa

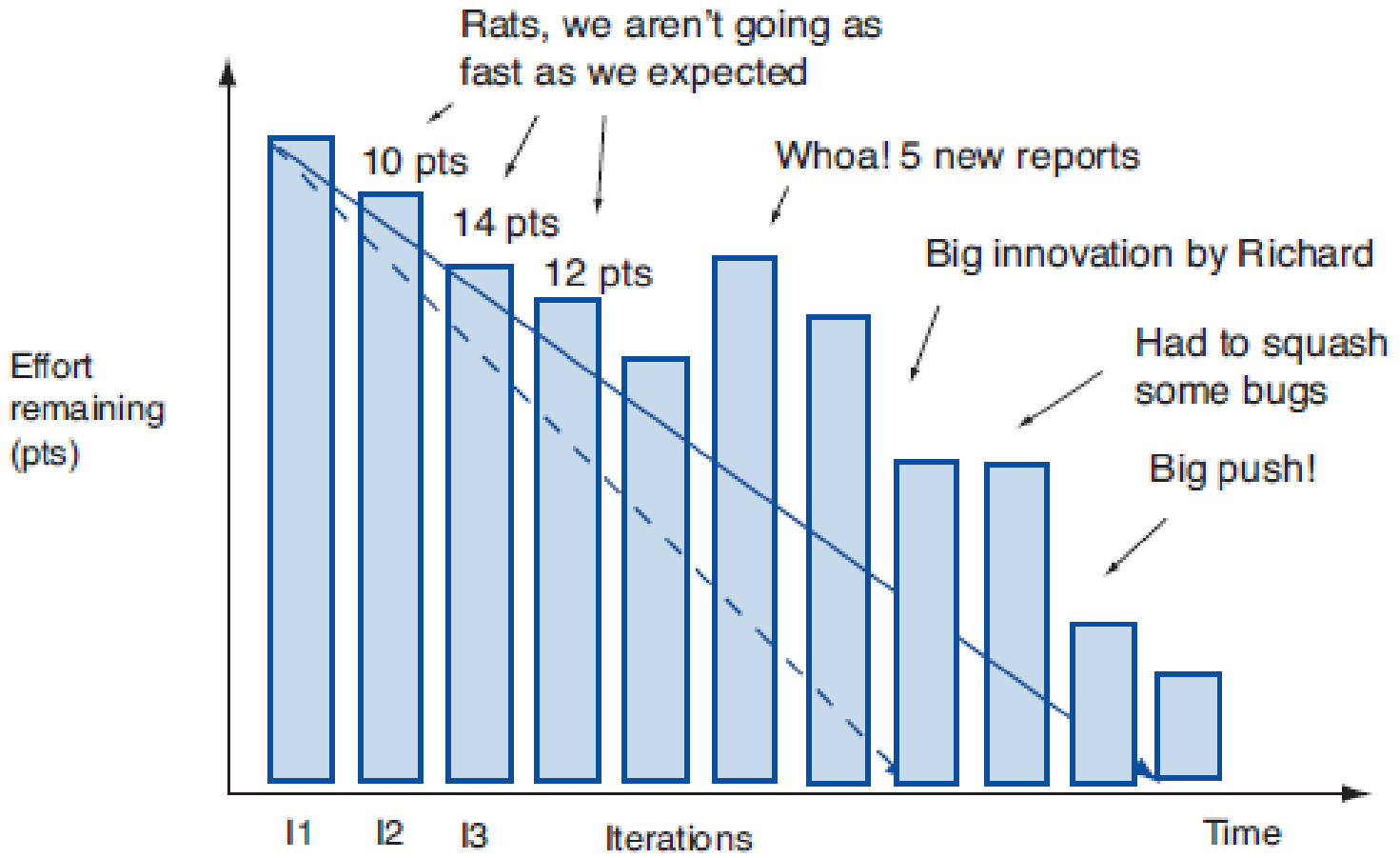


- Ketterissä menetelmissä on oleellista kuvata mahdollisimman realistisesti projektin etenemistä
- Tämän takia velositeettiin lasketaan mukaan ainoastaan täysin valmiiksi (eli Definition of Donen mukaisesti) toteutettujen User storyjen Story pointit
  - ”lähes valmiiksi” tehtyä työtä ei siis katsota ollenkaan tehdyksi työksi
  - http://jamesshore.com/Agile-Book/done_done.html



Julkaisun suunnittelu – release planning

- Ketterän projektin etenemistä kuvataan yleensä Release Burndown -kaavion avulla
  - Aika etenee x-akselissa sprintti kerrallaan
  - y-akselilla on jäljellä olevan työn määrä Story pointeina mitattuna



- Ketterässä projektissa vaatimukset saattavat muuttua kehitystyön aikana, siksi jäljellä olevan työn määrä ei aina vähene

Julkaisun suunnittelu – release planning

- Joskus käytetäänkin Burn Up -kaavioita joka tuo selkeämmin esiin kesken projektin etenemisen tapahtuvan työmäärän kasvun