Scrum en projectstart

Is software maken makkelijk of moeilijk?

- print('hello world')

of

https://tweakers.net/nieuws/98193/overheid-verliest-miljoenen-euros-op-mislukt-ict-project-uitkeringen.html

Waarom mislukken ICT-projecten?

- Niet goed aansluiten bij sociale context / wensen klanten/gebruikers
  - https://tbmnet.nl/overheid-verspilde-886-000-euro-aan-ontwikkeling-van-eigen-zoom-alternatief/
  - https://www.volkskrant.nl/nieuws-achtergrond/kuipers-was-gewaarschuwd-maar-kocht-in-2020-toch-softwaresysteem-om-ziekenhuisbedden-te-tellen~bf61e18c/?referrer=https://www.google.com/
- Overmatige complexiteit ('horse designed by committee')
  - https://www.agconnect.nl/artikel/it-complexiteit-bezorgt-omgevingswet-weer-uitstel
- Ongelukkige technische keuzes...

Ongelukkige technische keuzes?

- Kunnen ervoor zorgen dat project zo lang duurt of zo duur wordt dat stekker eruit wordt getrokken
- (Succesvolle software = baten – kosten)
- Voorbeelden:
  - Mijn "Molecule Evoluator" met Borland C++ builder
  - Tracey in .NET Core
  - The Piq in Xamarin
  - Excel als logica-engine pensioenpremie-app
  - 20 jaar niet refactoren bij DiskMagic
  - 'resume-driven-development' bij internetcontrolesite
- Is echter zelden de enige factor!

Slechte nieuws: je kunt als programmeur niet alles voorkomen

- "Visionaire" bazen
- Kameel-requirements
- Ongelukkige techkeuzes

Goede nieuws

- Sommige teams kan je bijsturen
  - Kans op mislukking valt te verkleinen met juiste maatregelen!
- Je kunt als goede programmeur altijd ergens anders een baan vinden!

Wat zijn de juiste/handige maatregelen?

1. Nauw en regelmatig gebruikersfeedback (ook 'inputgebruikers')
    1. Desnoods via schetsen en prototypen!
    2. Wel nuttig: 'persona' (hoofdgebruiker)
    3. Ook nuttig: scenarios en interviews als documentatie
2. Risico verminderen
    1. Goed vooronderzoek (ADRs)
    2. Tracer bullets en testen
    3. Zo snel mogelijk waarde proberen af te leveren
3. Software makkelijk aanpasbaar houden
    1. KISS, YAGNI, DDD, refactoring, 'evolutionair design'

Wat blijkt MINDER handig te zijn? - een greep

- Niet afspreken hoe je dingen gaat noemen (het was toch 'Job', ik zie geen 'Repair')
- Taken horizontaal verdelen ('ben je al klaar met die API, ik kan niets doen!')
- Haast die niet gebalanceerd is door discipline (geen unit tests of reviews maar snel pushen omdat je op vakantie gaat)
- Gecommitte code snel reviewen en niet uittesten
- Geen wireframes maken: 'ik dacht dat we het erover eens waren dat het er heel anders uit zou zien'
- Te weinig tijd besteden aan uitwerken taken ('Front-end')

Een tweede greep

- Op wankele basis technologieen kiezen
  - The client wants to be able to use the application across different platforms. To be precise: Mobile and Windows Desktops should be supported.
    Progressive Web Apps are not required. => Blazor
  - Mijn eigen keuze voor Borland C++ builder :(
- Focus op volledig design in plaats van eerste waardeleverende feature
- Te grote taken (bv 1 week), waardoor
  - Heel veel merge-conflicten!
  - Reviewer gaat scannen
  - Kans dat hele werk overmoet omdat er aan begin iets mis is gegaan...

vs

Belangrijk

- Olifant is niet 1000x zo grote muis
- Grote projecten hebben grotere teams, meer overhead, overleg, plannen, documentatie
- Niet altijd leuk
- Wel noodzakelijk!
- Deze projecten ook van programmeren in klein > groot!

Overgang van programmeur naar software developer

- Deel is mogelijk saaier
- En zal zeker trager aanvoelen
- Maar is deel van professionaliteit, dus als je betaald wilt worden...
- (of iets goeds, duurzaams en bruikbaars wilt afleveren)

"You rush a miracle man, you get rotten miracles."

- Miracle Max, 'The Princess Bride"

De 'Lameijer ideale volgorde' (2023/07/12)

1. Fysiek opzetten van project / repo
2. Fysiek opzetten van backlog met items als:
   2a. Bepaal persona (liefst met gebruiker)
   2b. Bepaal 1 of meer kernscenarios (liefst met gebruiker)
   2c. Maak schetsen / wireframes (liefst eerst op papier)
   2d. Maak ADRs
   2e. Maak mogelijk DDD-schetsen
   2f. begin aan 'ubiquitous language'
   2g. Eventuele overige code-conventies vastleggen
3. kies eerste features/PBIs, bespreek die (DoD) en estimate ze
4. Gebruik een iteratief proces, mogelijk met tracer bullets en spikes, vertical slicing etc.

1. Fysiek opzetten van project/repo

- Maak een repository aan
- Geef de juiste personen er toestemming voor
- Maak eventueel een development-branch aan naast de master/main
- Zet branch permissions (Settings>Rules>Rulesets)
- Je kan er een backlog aan toevoegen via GitHub Projects (of een Trello-board kiezen)

1. Fysiek opzetten van project/repo

- Demo…?

2. Fysiek opzetten van een backlog

- Demo? GitHub Projects?
- (Trello kan ook)

2a. Bepaal persona

- Wie is de ideale gebruiker/hoofdgebruiker?
- Voorkomt ruzie, complex makende compromissen

Voorbeeld persona

- Annette de Vries, 40 jaar, ambulant coach
- Moet van haar werkgever haar reiskostenvergoeding opmaken
- Moet daarvoor de afstanden van de client-client trajecten optellen
- Dat vindt ze niet leuk!

Zelf proberen: persona maken

2. Bepaal scenarios

- Wat is het doel/zijn de doelen van de persona?
- Wat zijn (voor die persoon) logische stappen om het te bereiken?

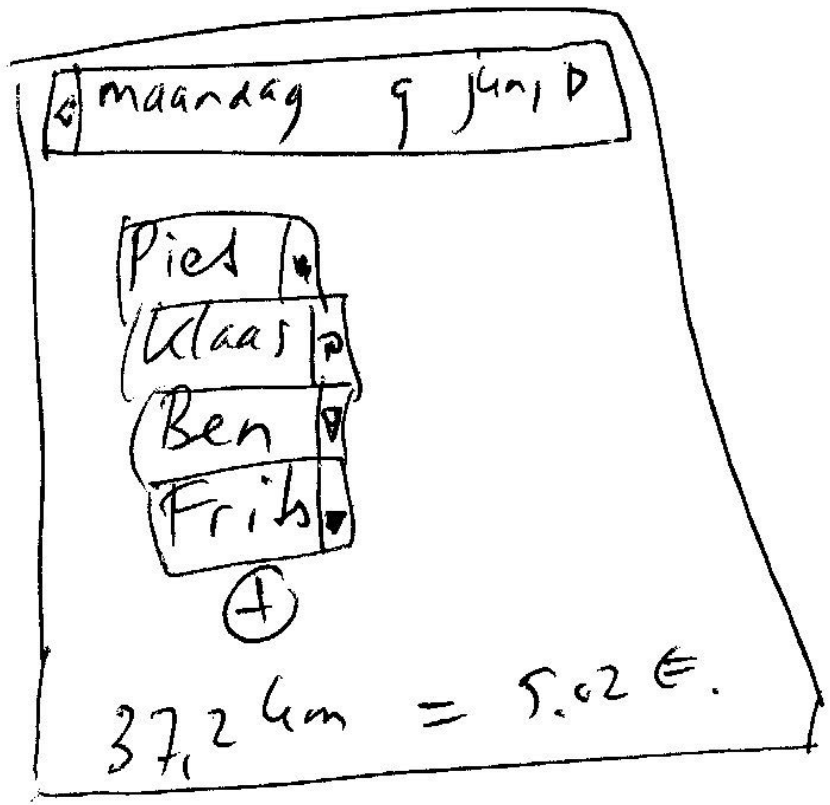
2b. Voorbeeld: Bepaal scenario(s)

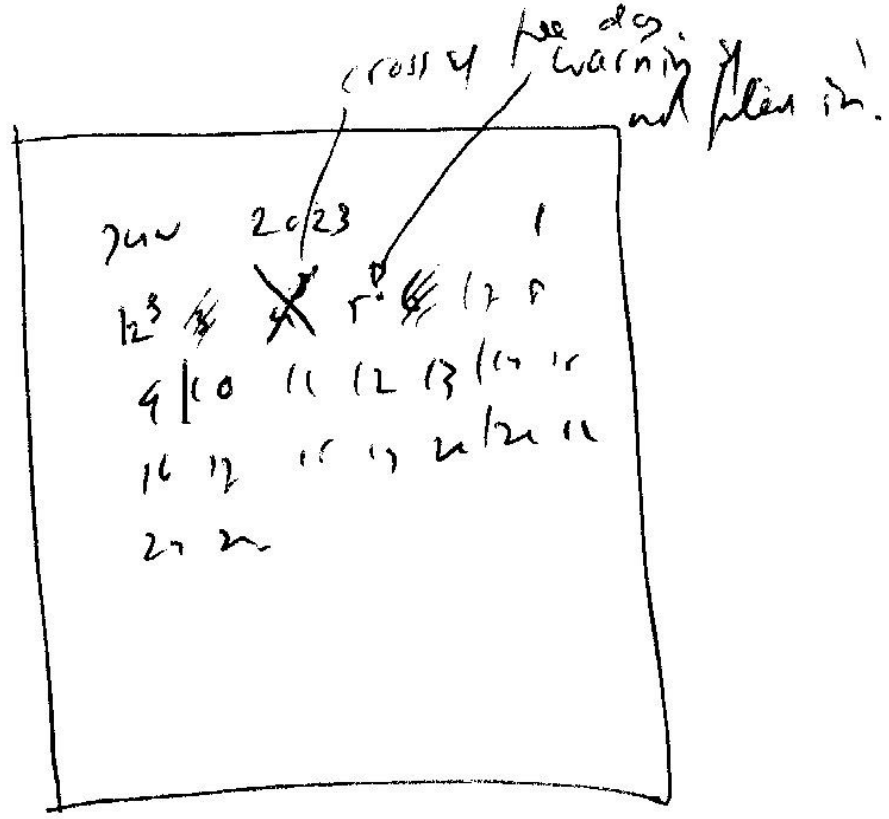
- Annette wil haar dagrooster in de app kunnen zetten
- Dus Flip, Bert, Wim, Piet
- (De adressen van die personen zijn er al eerder in gezet)
- De app berekent de reisafstanden vanaf Annette's huis naar de clienten
- Aan het eind van de werkweek verstuurt de app het overzicht naar Annette's manager

2. Probeer zelf: scenarios

2c.Maak schetsen/wireframes

- Begin met schetsen op papier
- Hoe lelijker, des te groter de kans dat klanten gaan letten op hoofdzaken
- Uiteindelijk wil je voor team wel ergens in wiki of version control wireframes hebben (liefst SVG ofzo)
- *Bij grotere teams worden wireframes vaak door anderen geleverd, maar het is goed ook te oefenen voor je portfolioproject


maandag 9 juni
Piet
Klaas
Ben
Frits
37,24m = 5.02 €.


cross = leerdag
warning
juni 2023

3a Probeer wireframes te schetsen

2d Maak ADRs (Architectural Decision Records)

- Bv https://github.com/joelparkerhenderson/architecture-decision-record
- https://betterprogramming.pub/the-ultimate-guide-to-architectural-decision-records-6d74fd3850ee
- Basaal: welke technologie-keuzes zijn mogelijk voor het project?
- Wat zijn de argumenten voor en tegen elke?
- Wat kies je? Wat zijn de consequenties? Zouden er redenen zijn om dit ooit te veranderen?
- // Is niet erg als je hier een paar uur aan besteedt bij een echt project
- Praten met andere developers kan ook helpen!
- In bedrijfsleven is er vaak CTO die die beslissing al maakt

2d: voorbeeld

- Title: framework for frontend
- Status: proposed
- Context: most users will want to use their mobile phone (to enter data when they’re waiting between clients). Which frontend framework would be best?
- Decision & arguments(should have more description than this): Swift, Kotlin, Flutter, React Native (can mention Xamarin, MAUI, Cordova, Kotlin Mulitplatform...). Summary would be: users may have IOs OR Android; so Flutter or React Native; Flutter more loved and popular, will go with that
- Consequences: Go for Flutter for now; consider going native route (Kotlin/Swift) if it is a success and competitors arise, possibly with KMP to reduce code duplication

2d: bonus: kijk ook 10 minuten naar de voorbeelden!

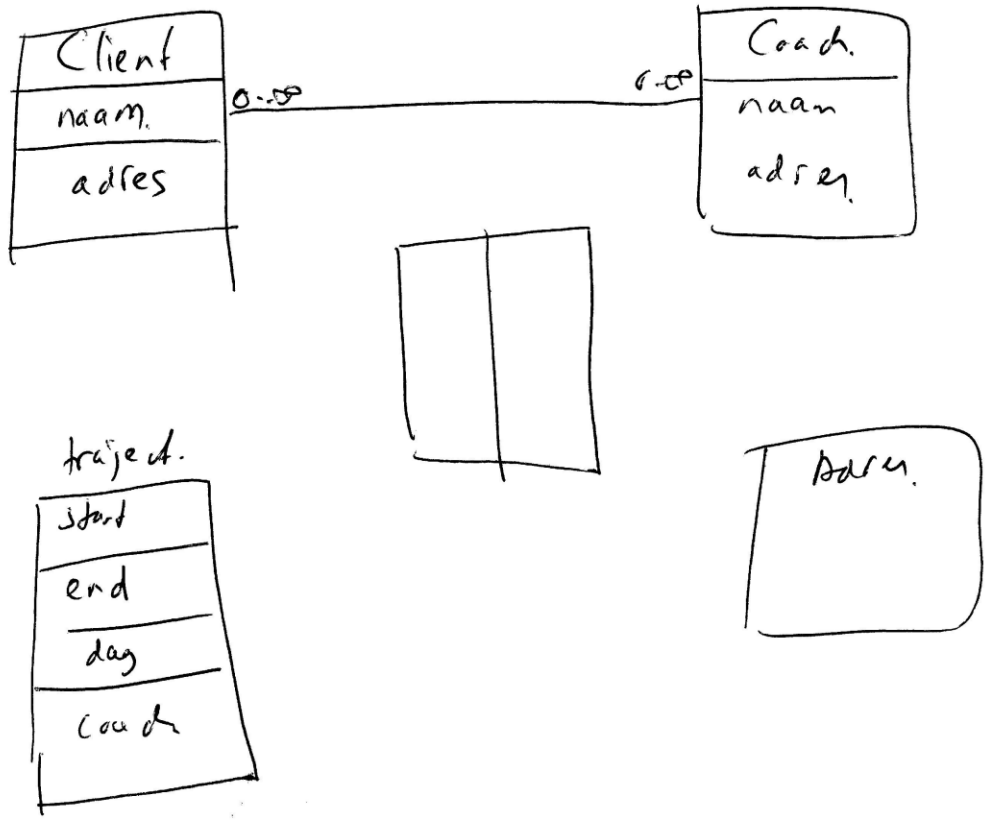
- https://github.com/joelparkerhenderson/architecture-decision-record
- https://www.lasssim.com/architecture-decision-records-example/
- https://github.com/omeerkorkmazz/adr-examples/blob/main/examples/adr-api-gateway.md

2d: zelf proberen

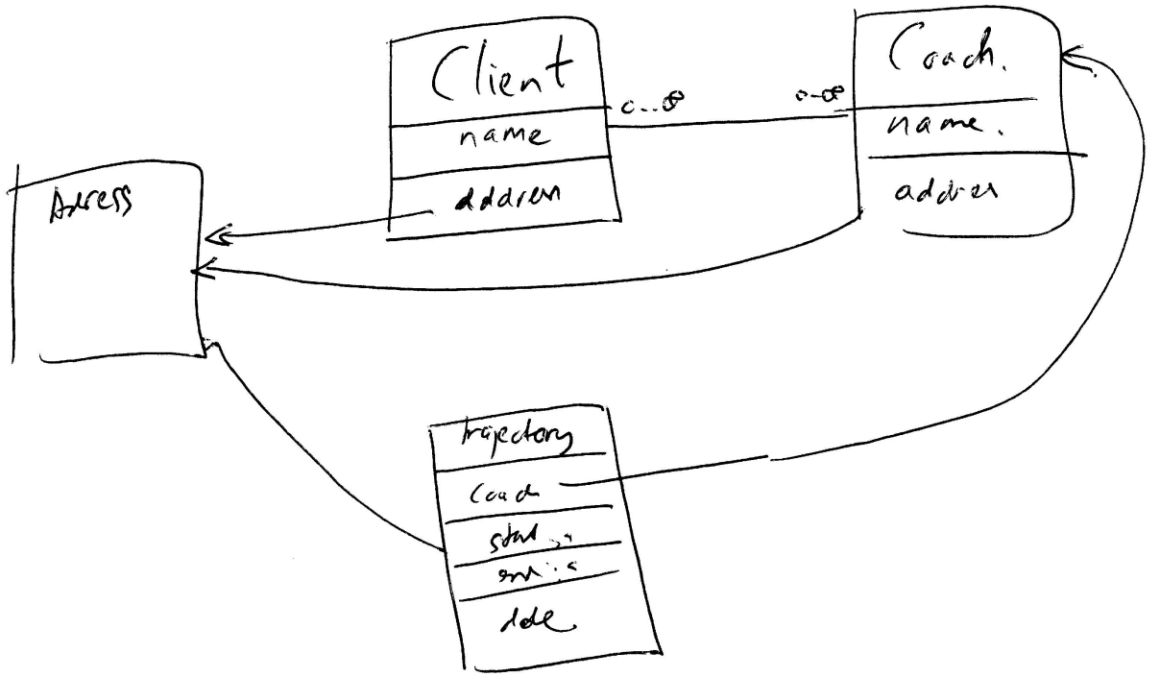
2e+f. DDD-schetsen en ubiquitous language

- Eric Evans: schetsen van 3-5 klassen per pagina, rest in tekst
- Maak lijst termen die je wilt gebruiken in project (dus niet repair == job ofzo)

2e Voorbeeld DDD-schets (versie 1)



2e Voorbeeld DDD-schets (versie 2)



Address is of course a location, and it could be that a client can be fetched from multiple different locations (future redesign?); for now, database will avoid data duplication, just recalculation of distances.

In future, may want to link to agenda of coach

2f Ubiquitous language

- Coach
- Client
- Address
- Trajectory (for day) - start and end at coache's house
- Leg (of journey)
- Distance (in km)
- Expenses (in euros)

2e+f Probeer dit zelf (diagram + ubiquitous language) voor

2g. Eventuele andere code-conventies

- Qua programmeertaal: meestal is editor met autoclean goed genoeg, evt Google code conventions
- Soms eerder opbouw: models map en repository map, of map voor model, controller en repository van bv een Trajectory?

- Zal voor deze app gewoon standaard IDEA-conventies gebruiken, en subject-based packages

3. Kies eerste taken om te doen

- Weet je niet of iets werkt?
  - Tracer bullet: hele lijn van frontend naar backend (react -> database -> react)
- Onderzoek nodig? Timebox het ("Spike") bv 2 uur
- Maak lijst taken (Backlog)
- Kies de taken voor de eerste sprint
  - Niet geblokkeerd door onaffe features
  - Liefst grootste/snelste waarde voor minste tijd (feedback klant!)
  - Taken ingeschat van max 1-2 dagen (liefst korter);
- Vertical slicing: laat iemand van frontend naar database werken
  - Itt horizontal slicing: jij doet de front-end!
- Voor iedere taak: DoD en dan planning poker/estimating
- Sleep die taak dan naar de komende sprint

4. Demo (met Trello of GitHub Projects)

4. Probeer het zelf (in teams!)

4. Ervaringen/wat viel je op?

4 Het dagelijkse werk

- Begint met daily scrum (wordt later behandeld)
- Je pakt een taak op het bord op, zet hem op jouw naam en 'In Progress'
- Je maakt een branch aan met daarin tenminste het nummer van de taak (en mogelijk ook de naam) bv task_52_add_password_validation
- Als je er tevreden over bent push je de branch, zet hem in 'Ready for Review' (en poke een teamlid om te reviewen)
- Dat teamlid zet de code in de "Under review" kolom (als die er is) reviewt de code (onderhoudbaar/duidelijk/getest/secure?) en test hem (happy/unhappy paths); maakt review bij pull request

- Jij bekijkt de commentaren, overlegt als je het (na nadenken en een pauze) niet mee eens bent, en past dingen aan. Je zet het weer in ready for review
- Het proces herhaalt zich, meestal is de code nu goed
- Je pullt de development branch (of master-branch), merget die in jouw code, kijkt of het nog werkt, fixt eventuele merge conflicts (mogelijk met reviewer).
- Dan merge je jouw gemergde branch in de master branch en pusht
- (soms mislukt dit door een tussenmerge, herhaal dan dit proces)
- Je zet de taak op 'done' en zoekt een

Dagelijkse werk - demo

Wat vaak moeilijk is...

1. Explosieve start: iedereen begint gelijk de eerste dag aan dezelfde files te programmeren => Waarom probleem? Hoe beter?
2. Mega-taken: soms nemen mensen taken op zich van 3 tot 5 dagen => Waarom probleem? Hoe beter?
3. Vertical slicing "Ik doe de backend, jij de frontend, en hij de database) => Waarom probleem? Hoe beter?
4. "Looks good to me"-reviews (niet kritisch, geen testen) => Waarom probleem? Hoe beter? *Taart
5. Vage taken "Maak frontend" => Waarom probleem? Hoe beter?
6. Wireframes, code conventies, ubiquitous language 'in het groepshoofd': "Ik dacht dat we het erover eens waren dat we het zo zouden gaan doen" => Waarom probleem? Hoe beter?
7. Denken in constructies, niet in features: "Maak een User-tabel" - waarom probleem? Hoe beter?
8. Code schrijven die niet aan een feature gelinkt is "We hebben de logica ervoor, alleen de UI niet) => Waarom probleem? Hoe beter?
9. Algemeen: focussen op produktie voor het proces goed onder de knie is => Waarom probleem? Hoe beter?

Daily scrum: theorie

- Timeboxed: 15 min
- Normaal “stand-up”
- Waaraan gisteren gewerkt
- Plannen voor vandaag
- Wat houdt je tegen?
  - NIET uitgebreid - dat voor subgroepje!

Daily Standup – oefenen

- Probeer het <= 5 min!
- Iedereen de drie vragen
- Blijft kwestie over?
  - Apart mini-overleg met 1-2 anderen

Daily Standup – demo

- Vrijwilligers?

Discussie/opmerkingen: de daily scrum

- Wat ging goed?
- Wat was nog moeilijk?
- Wat viel je op?
- Waar waren de twijfelpunten?