Szoftvertechnológia és -technikák

10. Előadás

A szoftverfejlesztés fázisai, dokumentáció, tesztelés alapjai

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Tartalom


It's alive!
IT'S ALIVE!
Szoftvertechnológia és -technikák
4


SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA
TERVEZÉS
IMPLEMENTÁLÁS
TESZTELÉS
ÜZEMBE HELYEZÉS, TELEPÍTÉS
ÜZEMELTETÉS, KARBANTARTÁS
Szoftvertechnológia és -technikák
15


MoonLanding – az ötlet
Szoftvertechnológia és -technikák
42

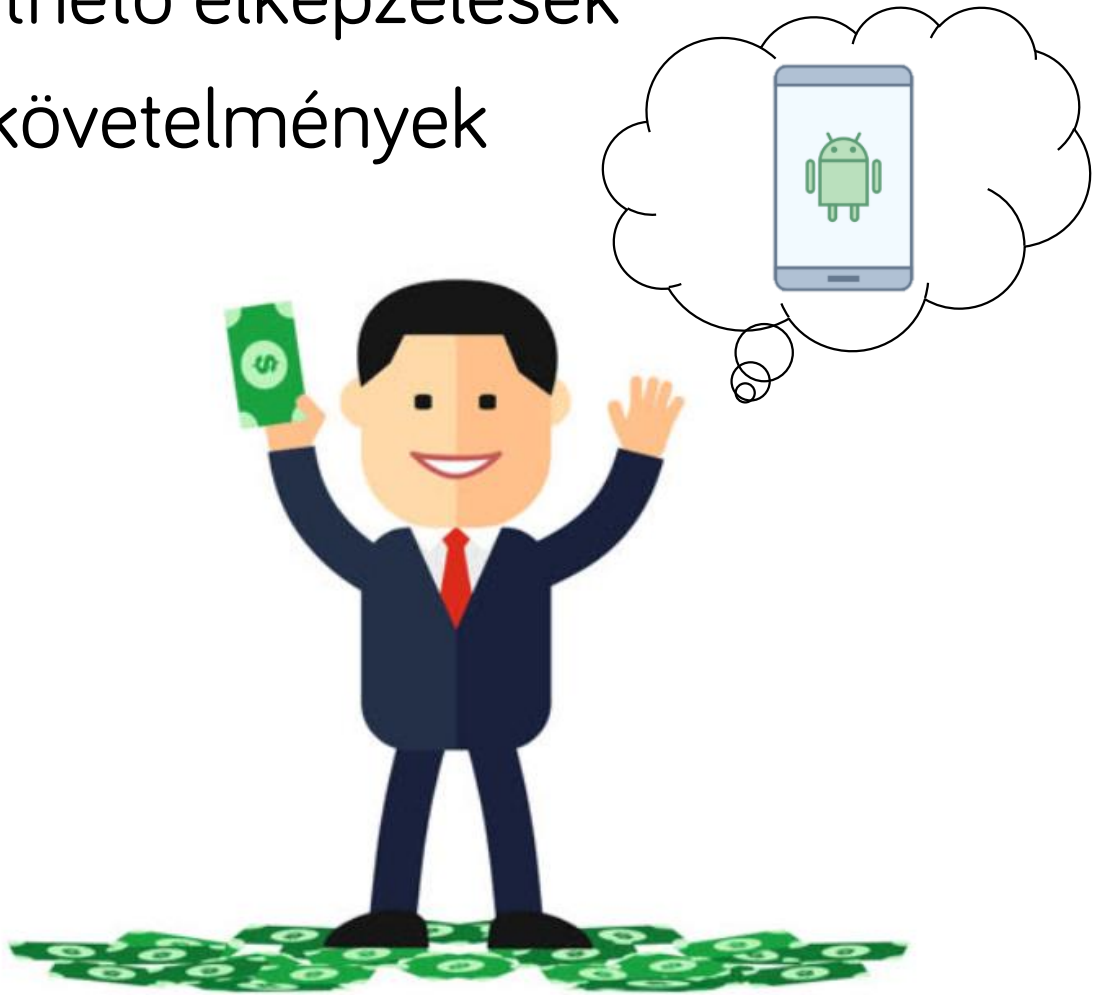

Miért fontos a tesztelés?
• A tesztelés minőségi mutatót állít elő
› Ha nincs tesztelés, nem tudjuk, hogy mennyire jó az alkalmazásunk
Ne adj át olyan kódot, amit még sosem indítottál el!!
Error!

It’s alive!

Ideális világ

Ideális világ - Megrendelő

- Jól követhető elképzelések
- Pontos követelmények



Ideális világ - Fejlesztés

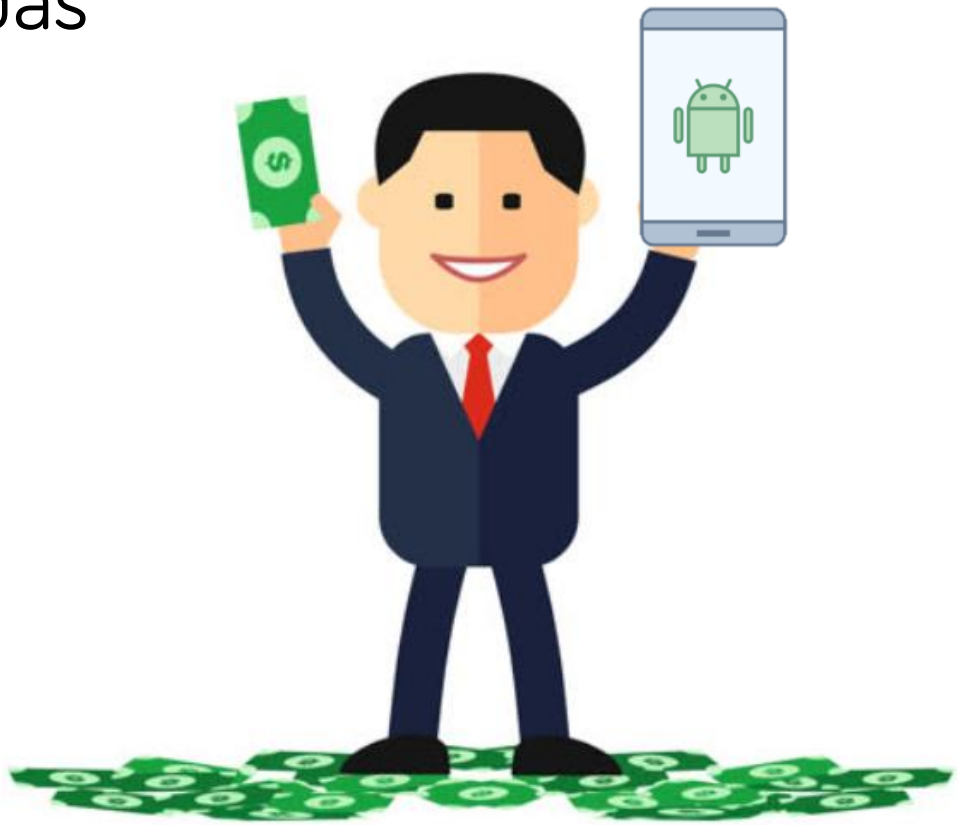
- Könnyen, tisztán megértjük a feladatot
- Mehet a fejlesztés!

Ideális világ – Kész termék

- Elkészül az alkalmazás határidőre

Ideális világ - Átadás

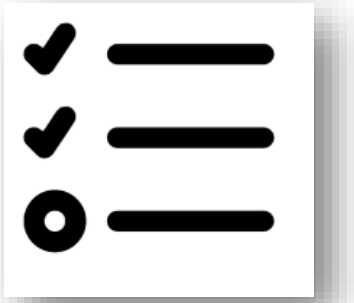
- Elégedett megrendelő
- Könnyű átadás



Ideális világ

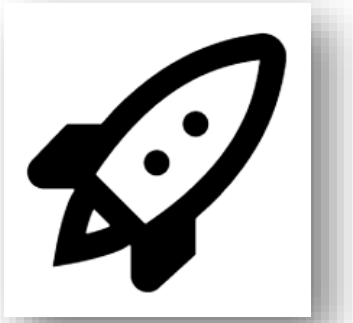
- Mehetünk nyaralni…

Ideális világban a szoftverfejlesztés fázisai



TERVEZÉS

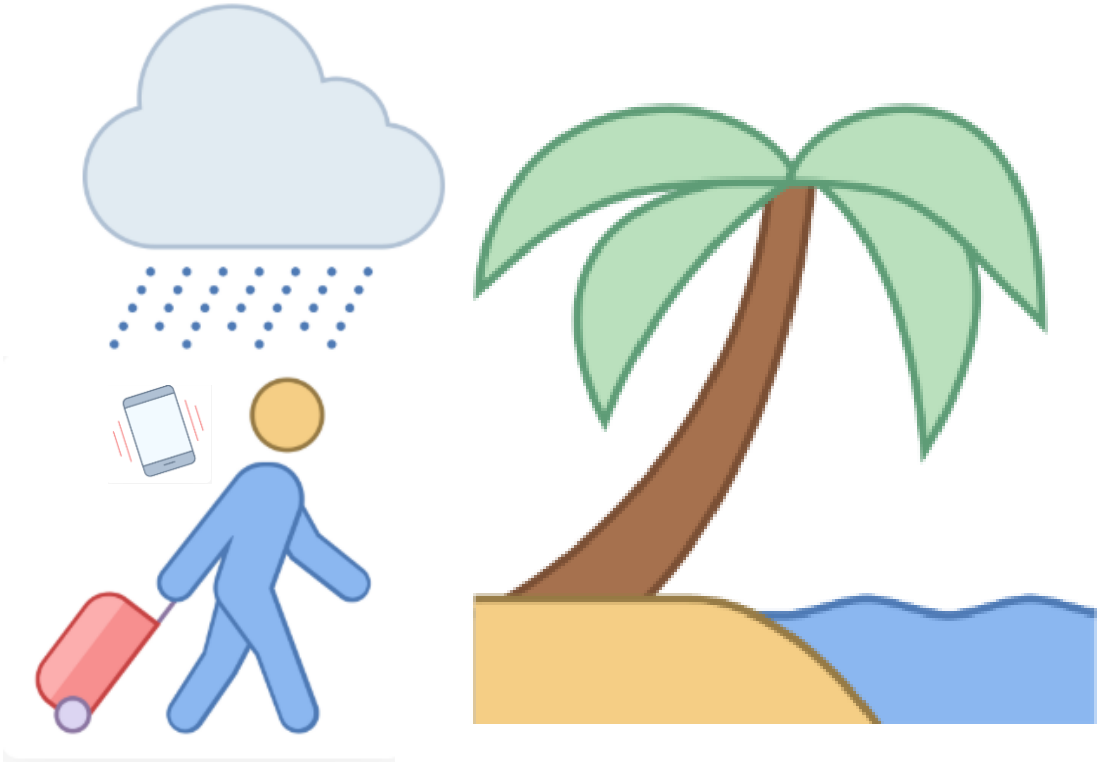
FEJLESZTÉS



TELEPÍTÉS

A valóság ennél árnyaltabb…

- Áramszünet után fura hiba



Elhangzó mondatok a valóságban

„Hm, otthon még működött!”

„Hogy is volt, emlékeztek milyen adatai vannak egy szerződésnek?”

„Valami olyasmit szeretnék, mint a facebook, csak egyszerűbbet.”

„Nem gondoltam volna, hogy ennyibe fog kerülni.”

„Nem megy a szerver egy darabig, mert átírjuk.”

„Ez így jó, de az jutott eszembe, hogy kéne egy olyan képernyő is…”

„Bocs, éjjel átírtam az adatbázis, mert nem tetszett.”

„Tegnap a fél csapat felmondott…”

„Kiderült, hogy a házszámot külön kell tárolni!”

„Nem mindenkinek Android telefonja van?”

„Ki gondolta volna, hogy ezt több, mint százan használják majd?!”

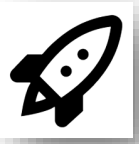
„Aki nem Chrome-ot használ, az egyébként is idióta.”

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS



ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS

ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

Követelmények definiálása

- A *klasszikus projekt* fogalma:
  - **egyedi jellemzőkkel rendelkező rendszer** kidolgozására vállalkozik
  - különféle **koordinált tevékenységek** alkotják
  - meghatározott a **célja**
  - jól definiált **kezdő és befejező dátummal** rendelkezik
  - **átmeneti** vállalkozás
    - a projekt végeztével a kialakított szervezeti és strukturális felépítés megszűnik vagy átalakul
  - egyéb keretek
    - például **költség**, információ biztonság stb.

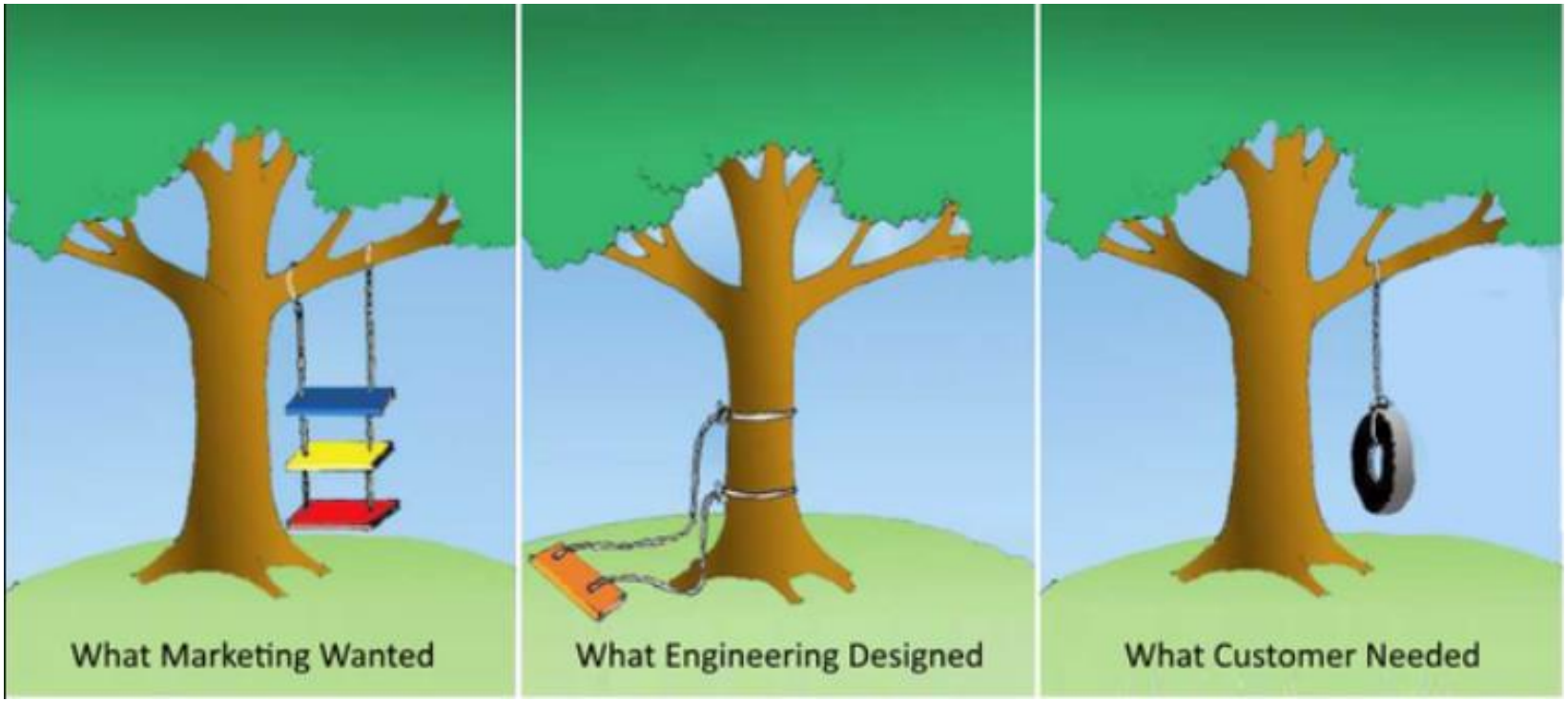
Követelmények definiálása

- Meghatározó tényezők: **idő, költség, teljesítmény és minőség**
- Példa: A megrendelő nem bízik a műholdakban, és szeretne egy saját navigációs, útvonal optimalizációs rendszert, ezzel bíz meg minket (gondoljunk a Waze-re)

Követelmények definiálása

- Mikorra szeretné ezt a megrendelő? Milyen határidővel?
  - 2 év múlva
- Milyen platformra szeretné a megrendelő?
  - Android, iOS? Mindkettő?
- Mit kell pontosan tudni az appnak?
  - Milyen pontossággal működjön?
  - Milyen gyorsan?
  - Milyen hibahatárral?
  - Milyen funkció vannak pontosan?

Követelmények definiálása



Követelmények rögzítése

- A követelmények részletei meghatározzák a költségeket és átfutást
- A pontos követelmények meghatározása kulcsfontosságú minden esetben!
  - Függetlenül attól, hogy milyen időtávban gondolkodunk: a növekedő időtáv leginkább a bizonytalanságot növeli
- A részletek adják a projekt funkcionális kereteit

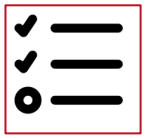
  → a szerződés kereteit

  → a change requestek alapját

Követelmények

Jó specifikáció	Hiányos specifikáció
Minden infó megvan a tervezéshez	A tervezést **hátráltatja** az információhiány. A terv rossz irányba mutat, mert alaptalan **feltételezésekre** építünk.
Pontos kérdéseket lehet megfogalmazni.	Még a megfelelő kérdésekig sem jutunk el.
Pontosabb költség és időkeretet lehet mondani.	A célok ismerete nélkül könnyen születnek szélsőségesen optimista vagy pesszimista becslések – nem tesz jót az üzletnek.
Jól látni a különböző üzleti és technikai függőségeket, szereplőket.	Homályos hivatkozások mellett nem látszanak az összefüggések.
Jól definiált keretek mentén egyértelmű pénzügyi megállapodásokat lehet kötni.	Ismeretlen elvárásokhoz kötött szerződések kódolják a feszültséget és konfliktust.

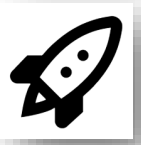
SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA



TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS



ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

Miért van szükség tervezésre?

- A specifikáció alapján nem tudunk még fejleszteni
  - Például nem tudjuk, hogy néz ki a képernyő
  - A következő absztrakciós szint
- Technikai döntéseket hozunk
  - Adatbázis típusa, UI technológia stb.
- A projekttervhez elengedhetetlen látni a függőségeket, technológiákat, erőforrásszükségletet stb.
- Tovább pontosítja a kereteket a megrendelővel
- Kockázatok felmérése, kezelése

Kompromisszum keresés

- A szoftvertervezési folyamatnak kell biztosítania, hogy a fejlesztés végén egy „jó szoftver” szülessen

- A „jó szoftver” egy megfelelő kompromisszum a különféle elvárások célkeresztjében
  - A „jó szoftver” ritkán az elképzelhető legjobb…

Milyan a jó szoftver 1.

- Funkcionálisan teljes
  - Ellátja a kitűzött funkcióit
- Robosztus
  - Hibatűrő (például felismeri a hibás bemenetet)
  - Jól viseli a változtatásokat
- Mérhető
  - Futásidő, például UI betöltése
- Debugolható
  - Például naplóbejegyzések
  - Monitorozható

Milyan a jó szoftver 2.

- Karbantartható
  - Architektúra, kommentek, dokumentáció
- Megbízható
  - Különféle környezetekben működik, sok „edge case” kezelve van benne
- Újra felhasználható
  - A komponensek, modulok, micro service-ek más projekten minimális módosítással használhatóak
- Tovább fejleszthető
  - Új funkció költsége az új funkció méretével jobban korrelál, mint a meglévő szoftver méretével

Példa a kompromisszumra

- A felhasználó néha nem fizeti meg a lehető legjobb minőséget → ???

- Megoldás:
  „egyszerűsített műszaki tartalmat ajánlunk”

- Példa:
  - Jobbgombos menüt sok idő lenne ide betenni
  - Jó lesz, ha lenyomja az ‚n’ betűt, létrejön az új elem
    - Beleírjuk egy tooltipbe és a kézikönyvbe

Szoftvertervezés

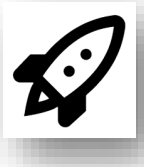
- A követelmények alapján elkészül a szoftverterv
- Kialakul a rendszer architektúrája:
  - Milyen egységek vannak?
  - Ezek hogyan kapcsolódnak
- A szoftvertervezés eredményei modellek és dokumentáció
  - Terméktípustól függően más és más jellegű és mennyiségű
  - A választott módszertan határozza meg
- **A terv akkor van kész, ha annak alapján kezdődhet a fejlesztés!**

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS



ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

Implementálás

- A szoftvertervezési fázisban készített szoftverterv ebben a szakaszban valósul meg
- A funkciók itt valósulnak meg
- Rengeteg apró döntés születik, ami a tervekben nem szerepelnek
  - Lásd a „milyen a jó szoftver” részt!
  - Teljesítmény, karbantartás, biztonság, funkciók, megbízhatóság, monitorozhatóság, …
- A mindennapi folyamatot a módszertan határozza meg

Fejlesztés != kódolás

- Kódolás: „gépelés”, írom a kódot
- Mini tervezés: hogyan írom meg ezt a funkciót
- Együttműködés másokkal, kód összefésülés
- Fejlesztői tesztek írása, tesztelés, hibajavítás
- Dokumentáció
- Projektműködés:
  - Státusz
  - Előrejelzés
  - Elakadás, segítség kérés és nyújtás

Módszertani elemek

- Klasszikus módszertani elemek
  - Például páros programozás stb.
- Kódolási konvenciók
- Legjobb gyakorlatok
  - Azokból melyeket alkalmazzuk
- Kódolási elvek
  - Például SOLID, stb.
- Minták
  - Példuál Command pattern stb.

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS, HIBAKERESÉS

ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

Hibakeresés

- A “tökéletes” és “hibátlan” szoftver legendája
- Tévedni emberi: hibakeresés != hibás keresés
  - Kapkodás
  - Monotonitás
  - Fáradtság
- Hibatípusok
  - Implementációs hiba (“`if (a=5)`…”)
  - Kommunikációs hiba
  - Rossz kódolás, verziókezelés, …

Mi a hiba?

- Elvárt viselkedéstől való eltérés?
- Specifikációtól való eltérés?
- Tervtől való eltérés?
- Felhasználói kézikönyvben leírtaktól való eltérés?
- Saját gyakorlatainktól való eltérés?
- Általános ipari gyakorlattól való eltérés?
- Reprodukálható vagy egyszeri anomália?

Tesztelés

- Tény: minden szoftverben vannak hibák
- A **minőség ellenőrök** megvizsgálják a szoftvert, hogy a **specifikáció szerint** működik-e?
- A hibák visszajutnak a fejlesztőkhöz és javítás után a tesztelés elölről (?) indul
- Gyakran használunk átmeneti állapotot: béta verzió, meghívott felhasználók stb.
- **A tesztelés célja, hogy megismerjük a szoftver minőségét!**

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS, HIBAKERESÉS

ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

Telepítés

- Telepítés, terjesztés
- Különböző környezeteket használunk
  - Fejlesztési, bugfix, belső teszt, megrendelői teszt, teljesítmény teszt, átadási, éles, több geolokáció stb.
- Más környezet – más működés
  - Milyen teszteket futtatunk a minőség ellenőrzéséhez?
- Pontosan ugyanaz a verzió menjen ki, mint amit a tesztelők jóváhagytak!
- Automatizálás mértéke módszertan függő
- Szoros együttműködés a megrendelővel

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

TERVEZÉS

IMPLEMENTÁLÁS

TESZTELÉS, HIBAKERESÉS

ÜZEMBE HELYEZÉS, TELEPÍTÉS

ÜZEMELTETÉS, KARBANTARTÁS

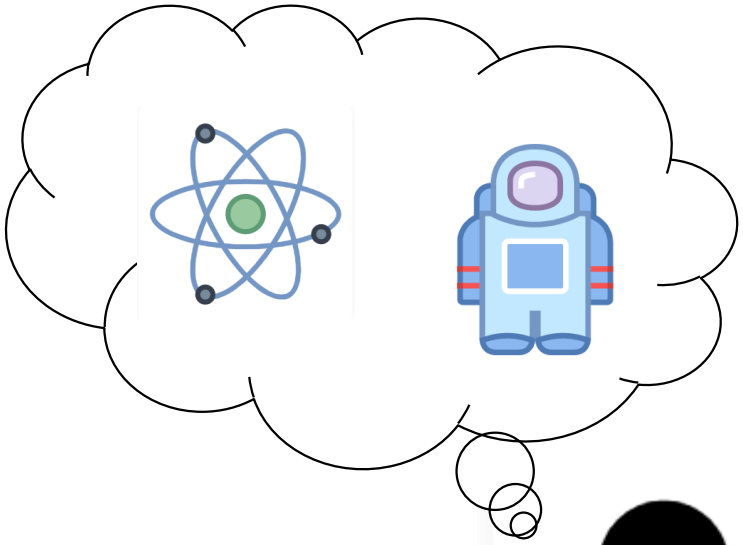
Üzemeltetés és karbantartás

- A szoftver életciklusának általában a leghosszabb szakasza
  - Gondoljunk az újonnan vett és a használt autókra! Mennyi költség megy el az autók szervizelésére, karbantartására!
- Továbbfejlesztési lehetőségek
  - Új követelmények fellépése
  - Implementáció/architektúra fejlesztése
- A hibák javítása
- Megváltozott környezethez illesztés
  - Keretrendszerek, jogi-, infrastrukturális környezet

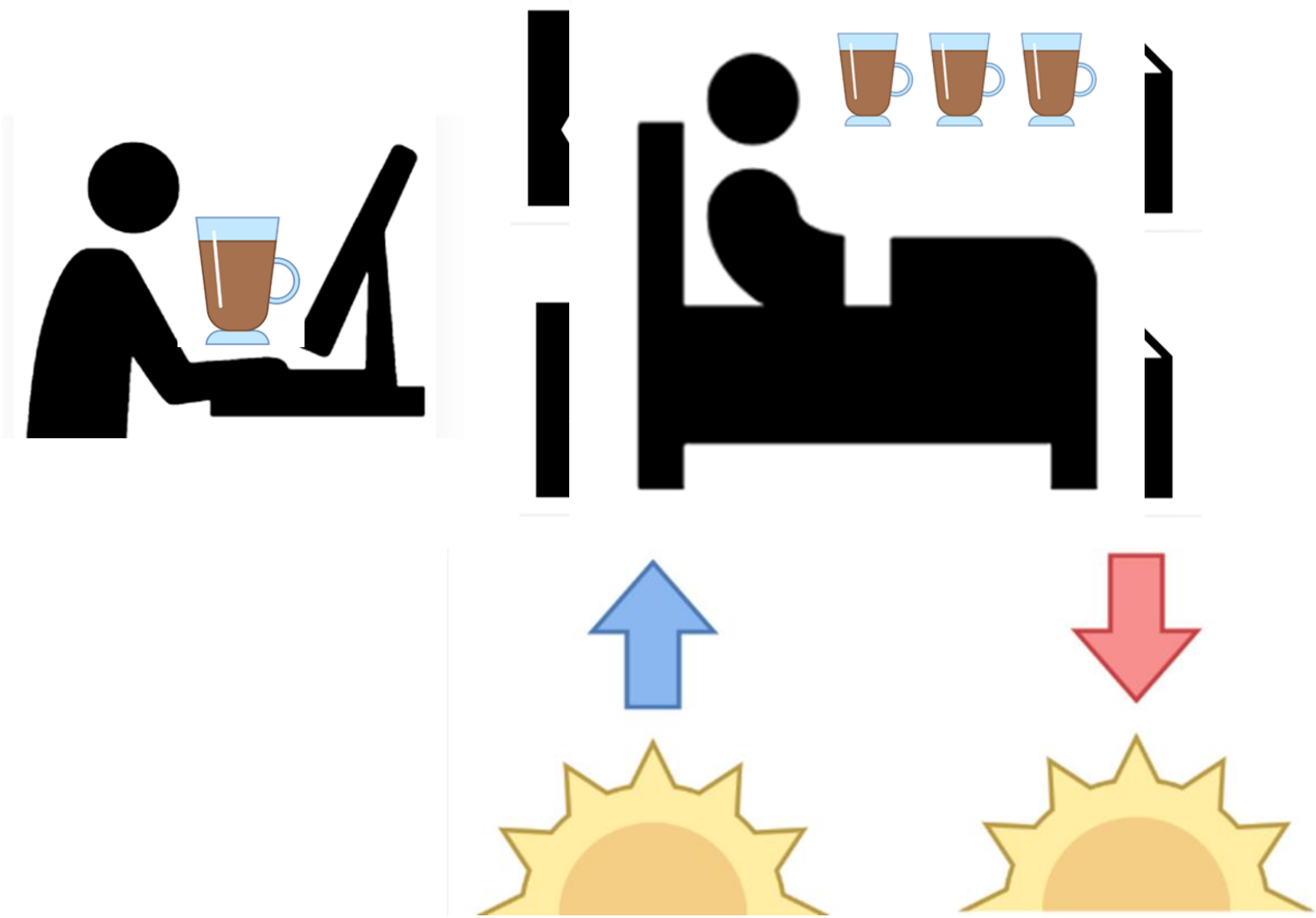
Eredmény a fázisok végén

- Specifikáció, követelmények definiálása
  - Modellek (Use Case), követelmény specifikáció
- Tervezés
  - Modellek (többi UML modell) és dokumentáció
- Implementálás
  - Forráskód
- Tesztelés
  - Tesztelt, javított forráskód, jegyzőkönyv
- Üzembehelyezés, telepítés
  - Telepített, működő alkalmazás
- Üzemeltetés, karbantartás
  - Alkalmazás újabb verziói

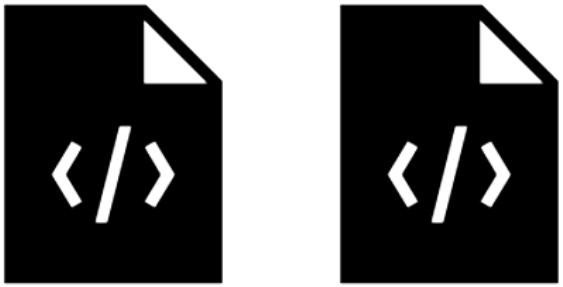
MoonLanding – az ötlet

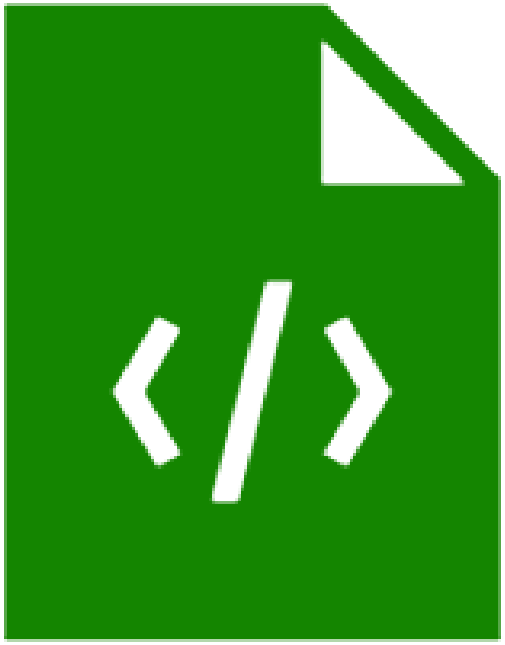


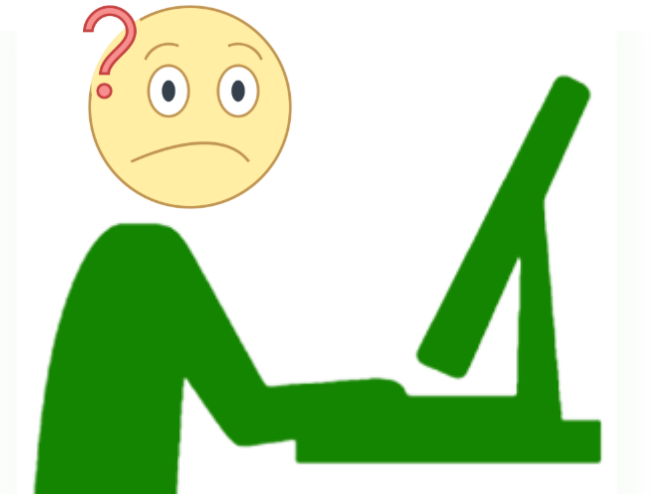
MoonLanding – az implemntáció

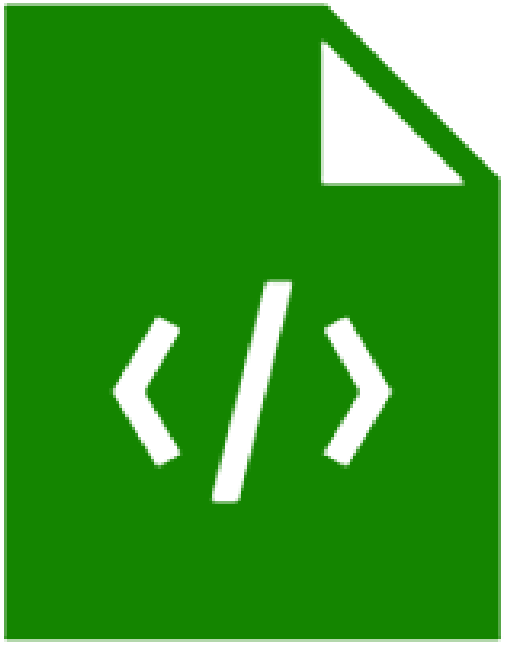


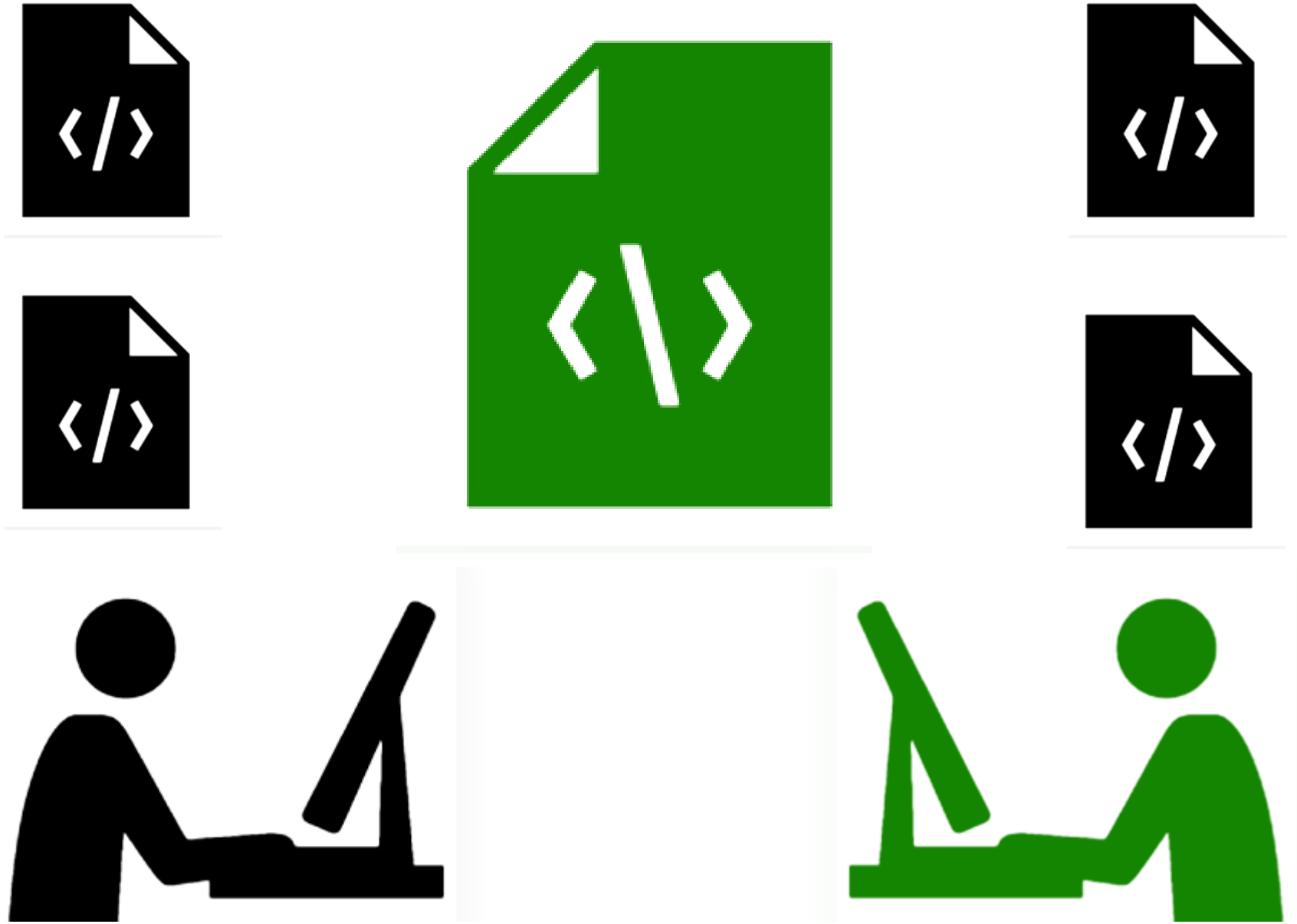
MoonLanding – a társ

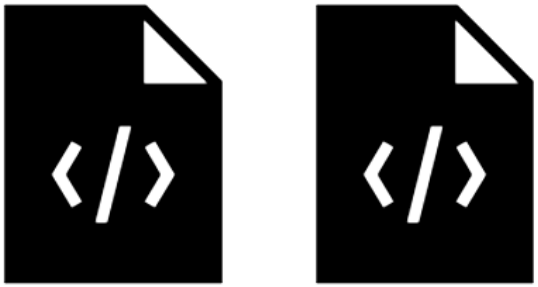


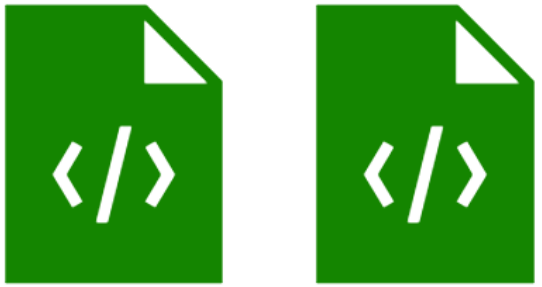


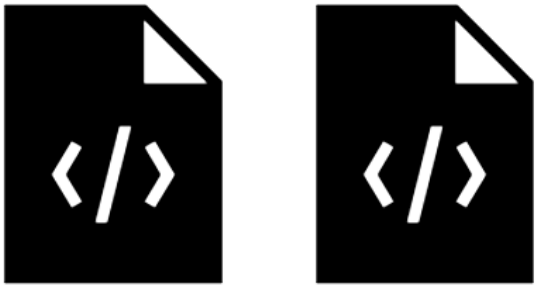


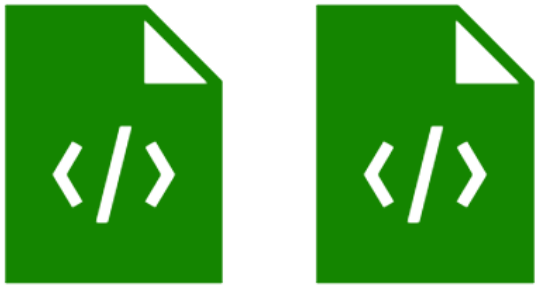


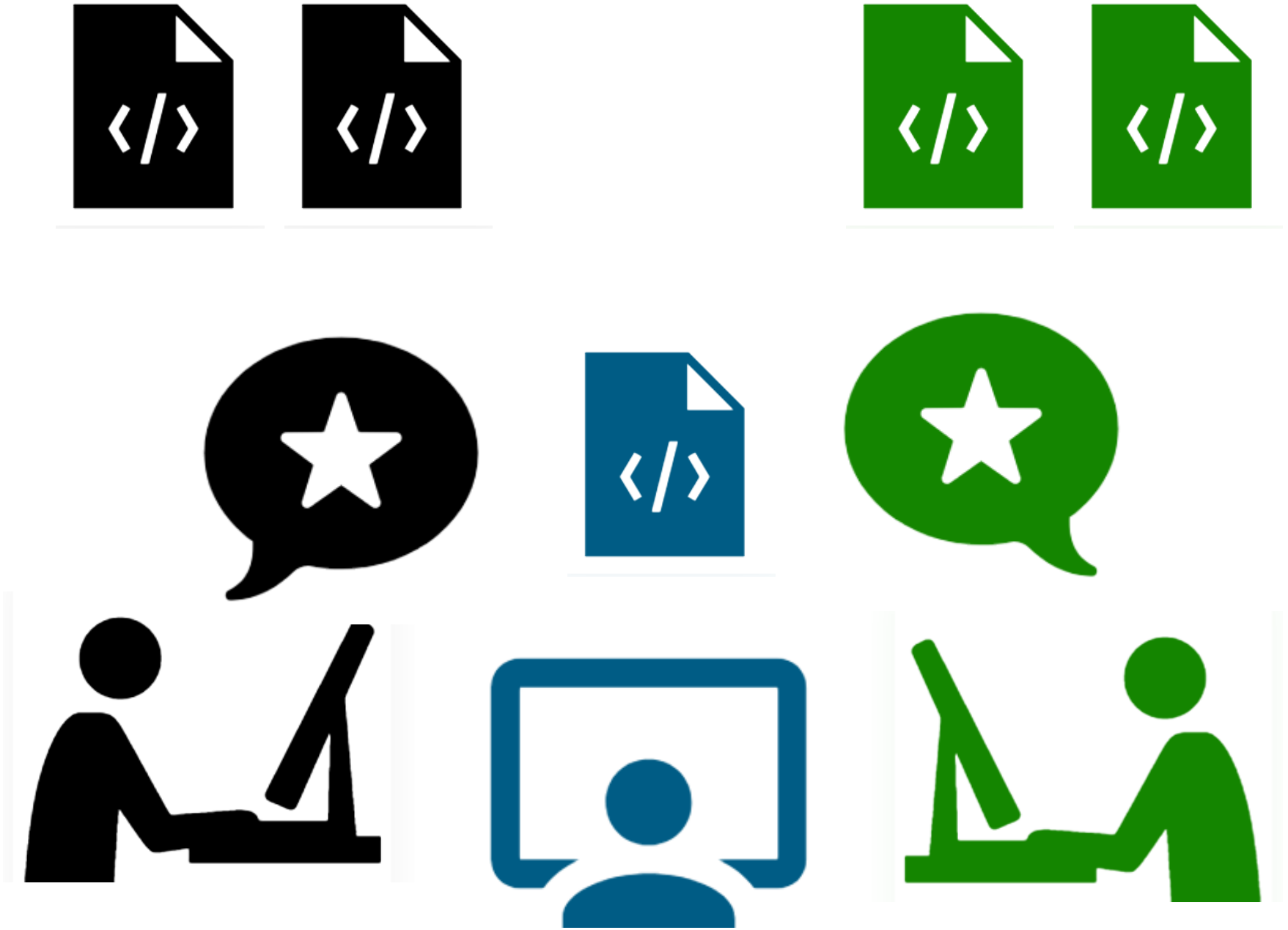


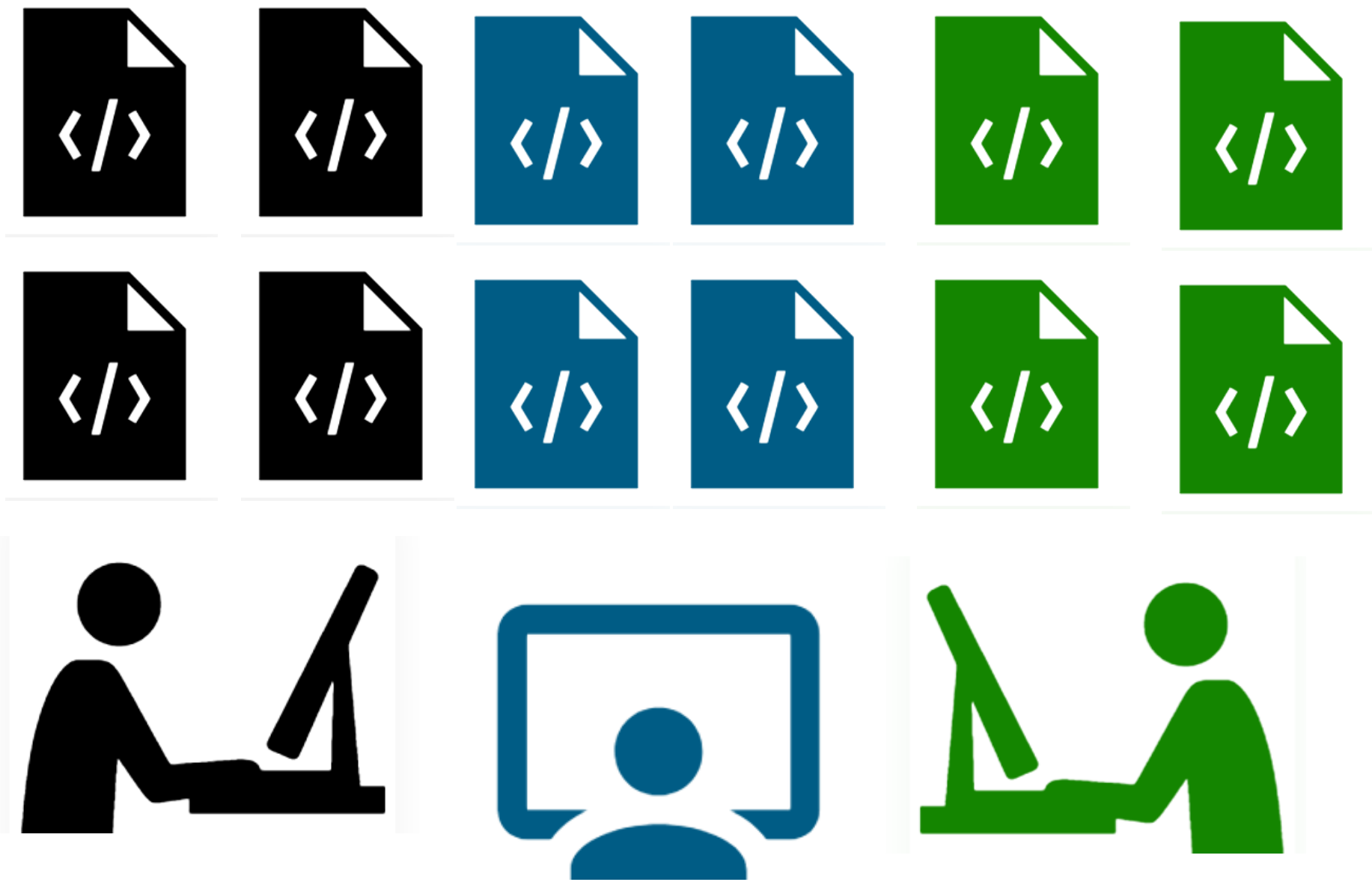


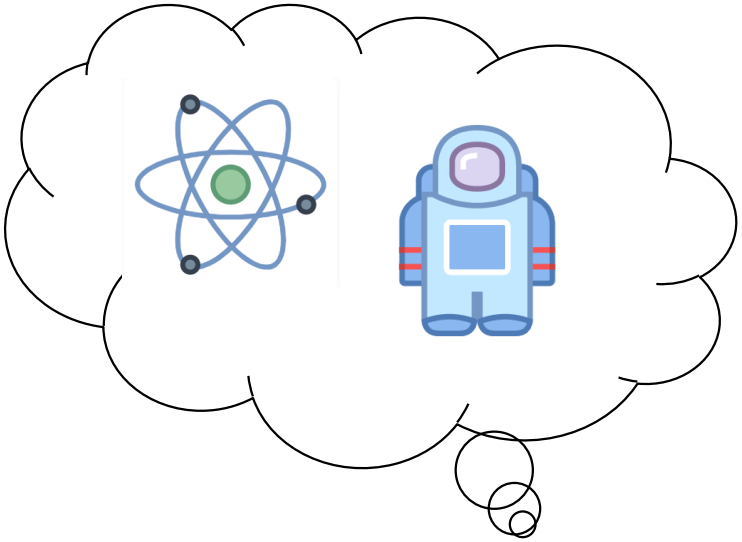


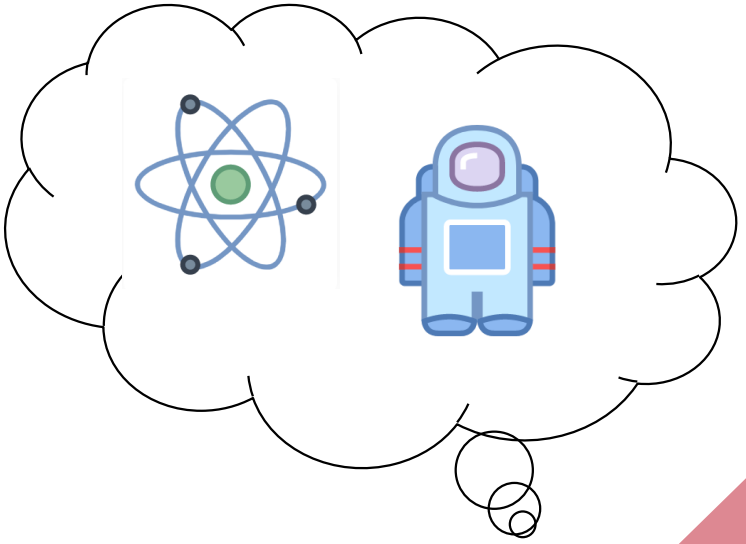


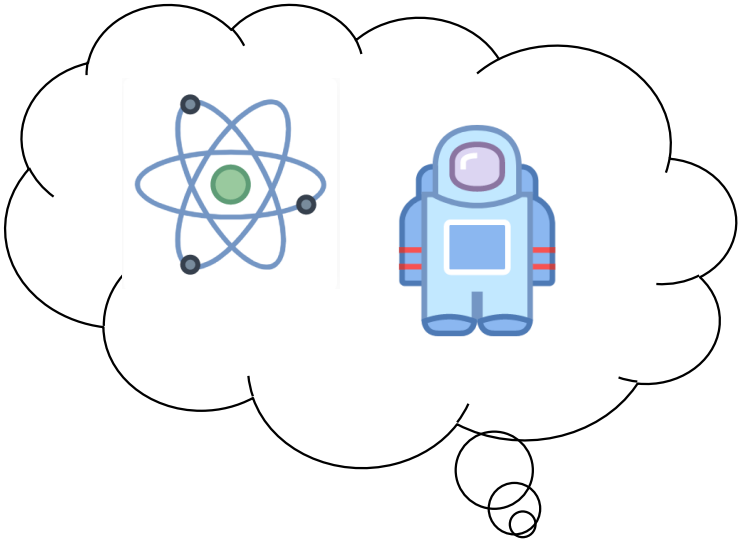


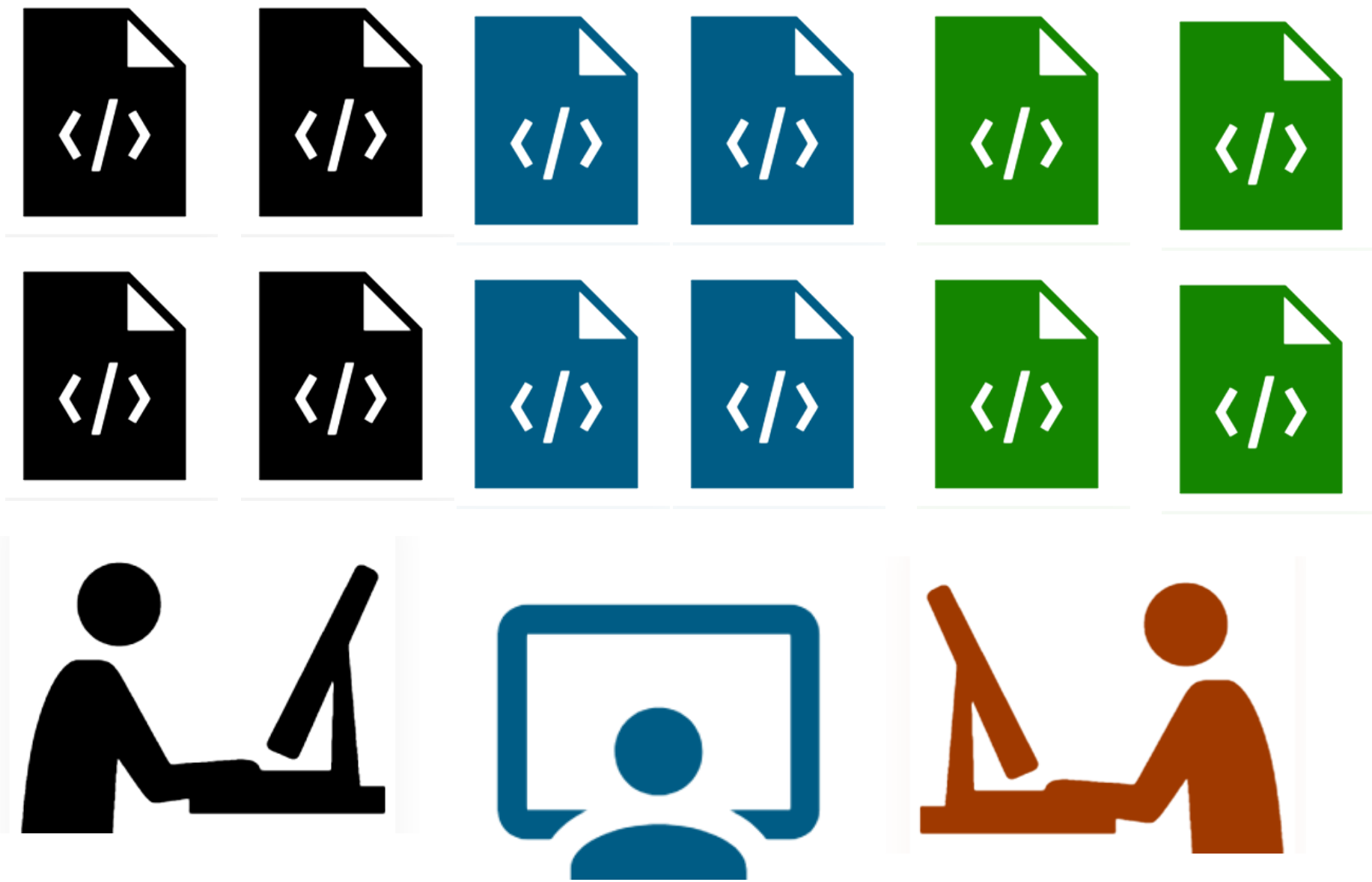


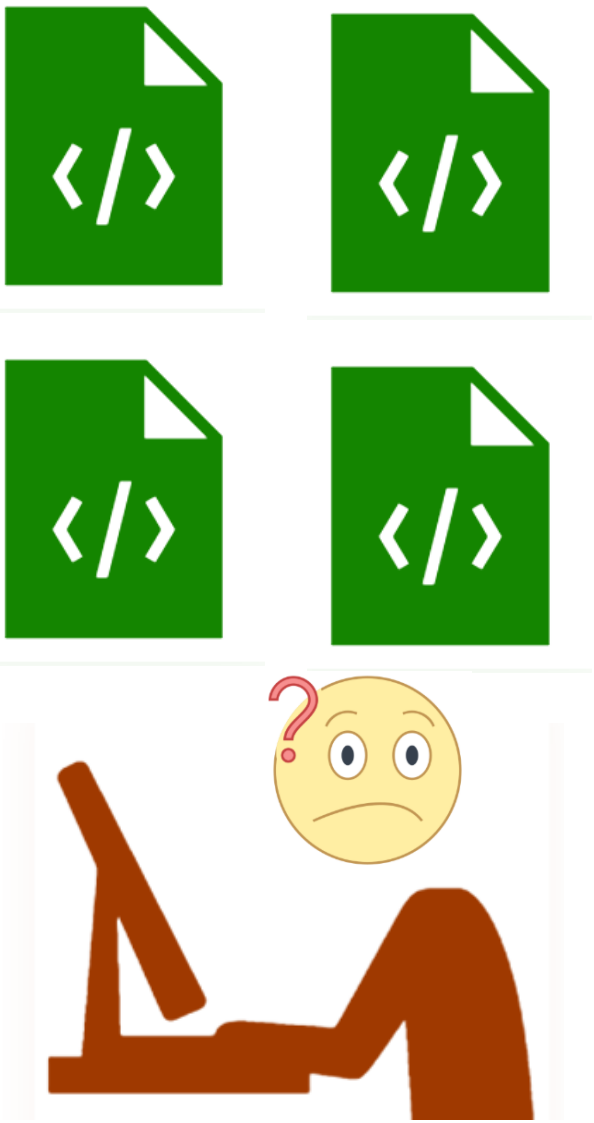


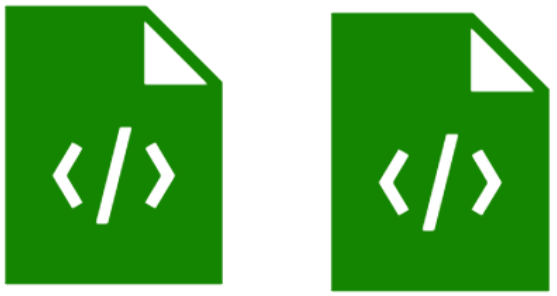


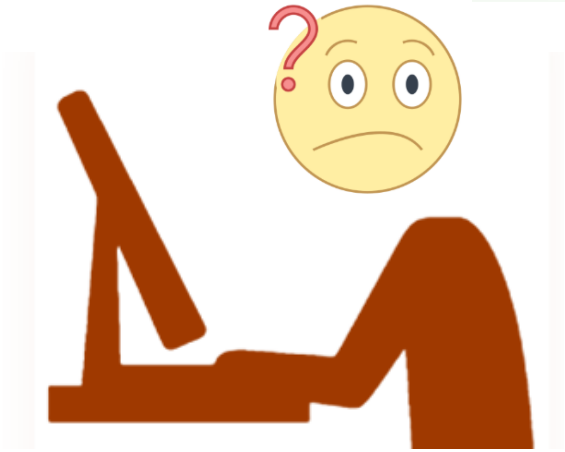


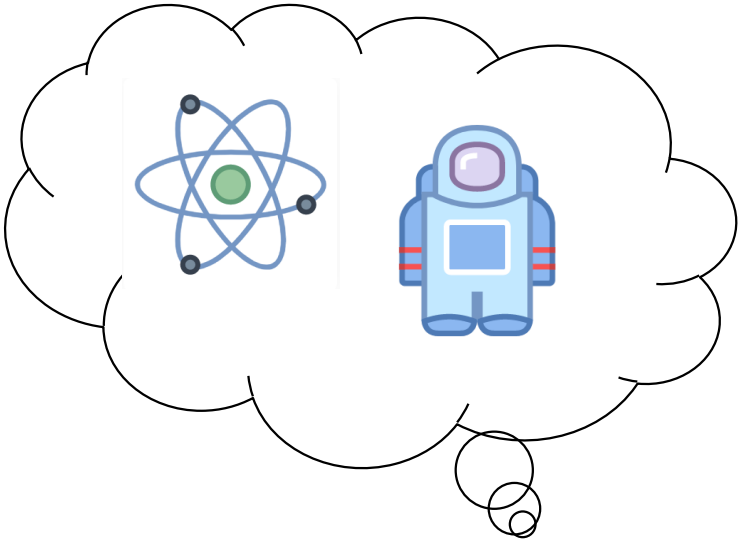












Egy másik példa

- .NET, JIT fordító
- 2001: 32 bit
  - A csapatot **szélnek eresztették**
- 2005: 64 bit, **teljesen más architektúra**
- 2009: **új architekturális tervek**: RyuJIT
- 2011: implementáció kezdete
- 2013: RyuJIT első verzió, 64 bit only
- 2015: RyuJIT élesbe kerül, 64 bit only
- 2018: RyuJIT teljes átállás

Hogyan lehetett volna ezt elkerülni?

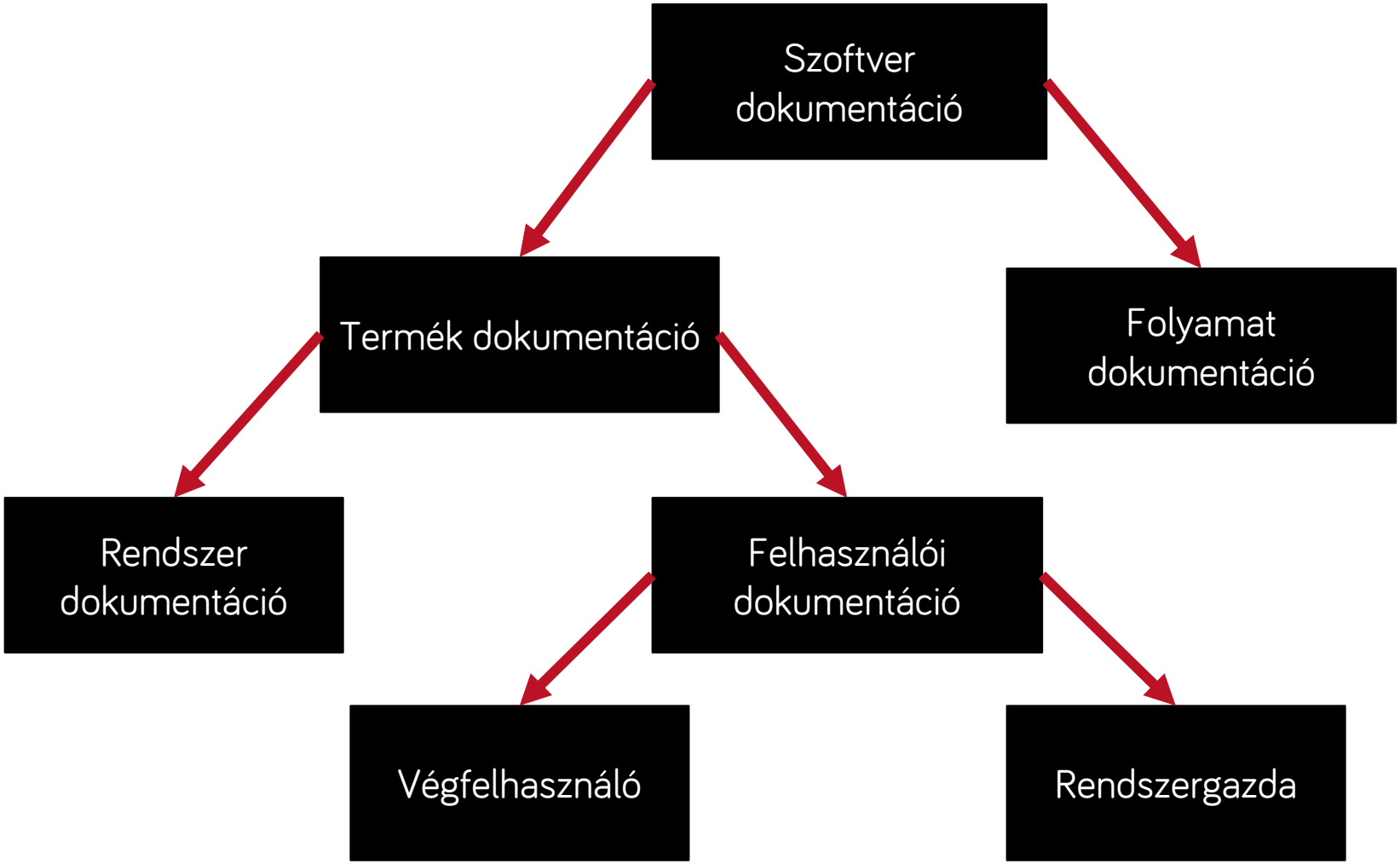
- „Dokumentálással”
- **A dokumentáció biztosítja a munka folyamatosságát**
- A következő fejlesztő generáció tudjon dolgozni
  - **hibákat javítani**
  - **új funkciókat fejleszteni**
- Lehetnek egyéb célok: például a megrendelő fizet érte, törvényi előírás stb.

Dokumentáció

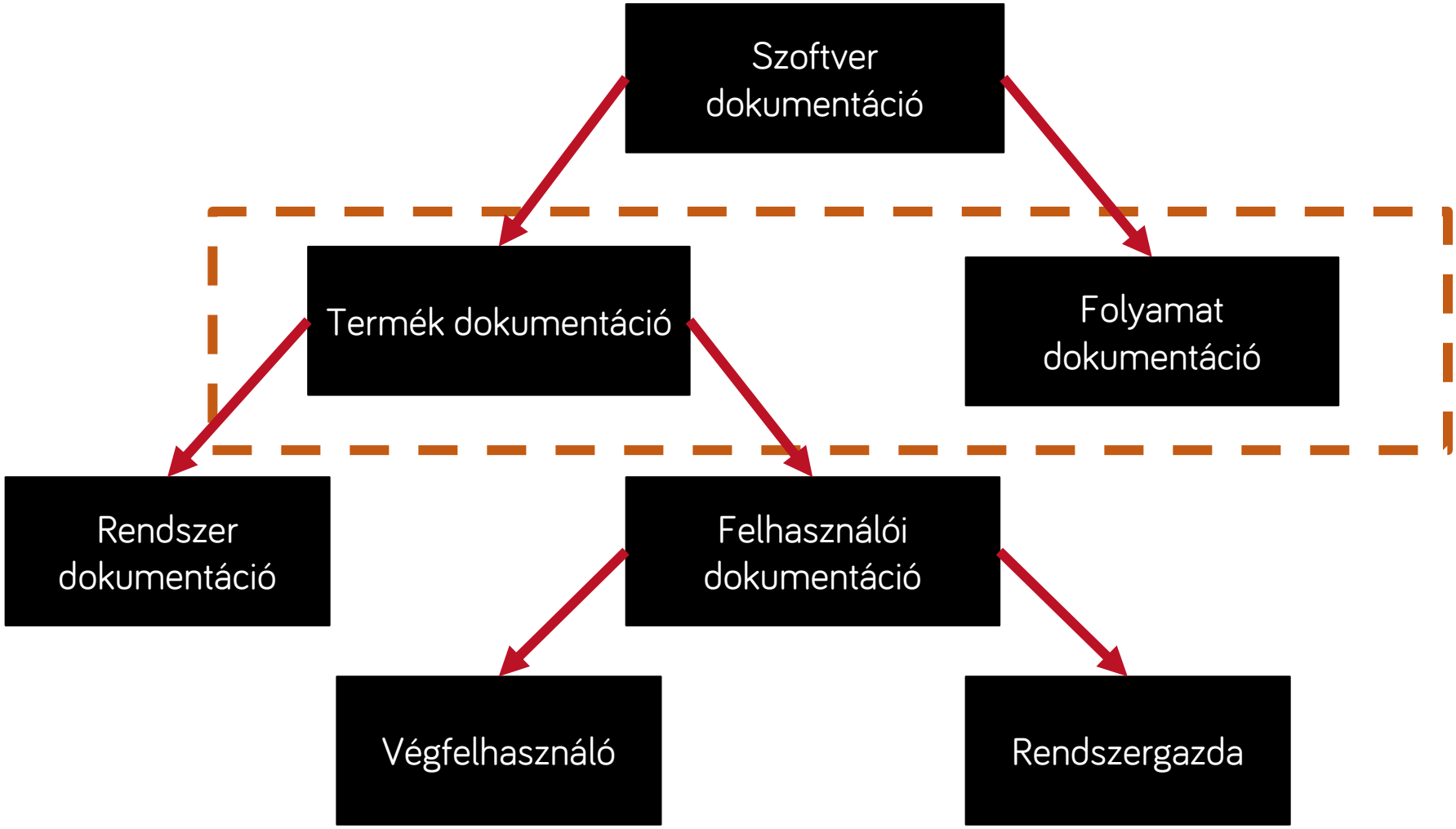
- A dokumentálás a szoftverkészítés folyamatát végigkísérő tevékenység
- Az összes munkafázishoz kapcsolódik dokumentáció, **eredménytermék**

- A szoftver nem csupán programkód!
- Szoftver = programok, dokumentációk, konfigurációk, adatok együttese

Dokumentáció csoportosítása



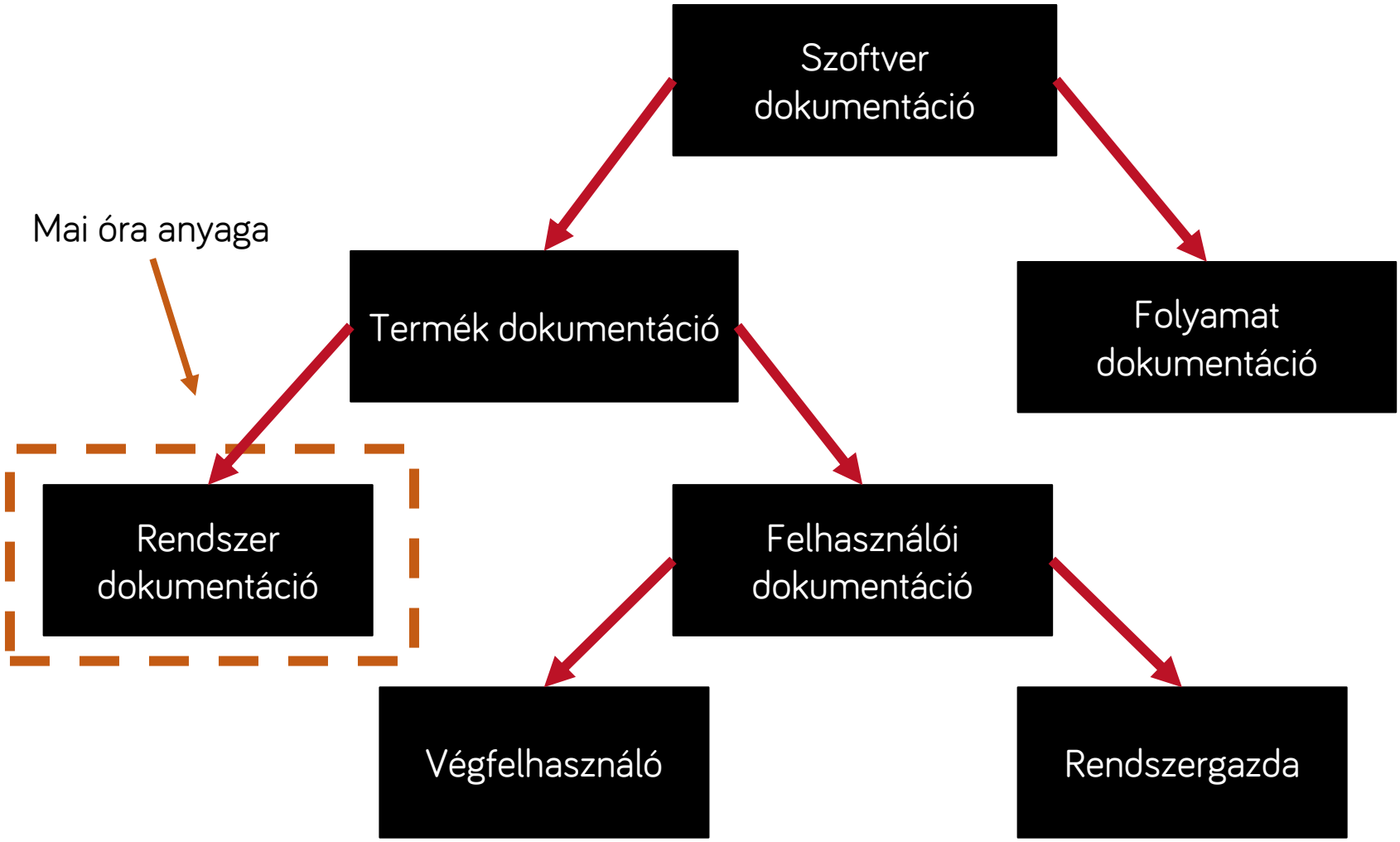
Dokumentáció csoportosítása



Termék vs folyamat

- Termék dokumentáció:
  - Leírja a szoftverterméket, ami éppen fejlesztésre kerül
- Folyamat dokumentáció:
  - A folyamat dokumentáció leírja a fejlesztési folyamatot
    - projekt terv
    - meeting jegyzetek
    - becslések, ajánlatok
  - Gyakran ISO leírás

Dokumentáció csoportosítása



Rendszer dokumentáció

- A termék dokumentáció része
- Egy áttekintést ad a szoftverről
  - Követelményspecifikáció
  - Rendszerterv
  - Forráskód dokumentáció
  - Tesztelési dokumentáció
  - Karbantartási dokumentáció
  - ...

Követelmény specifikáció 1.

- A követelmények definiálása fázishoz tartozik
- A követelmények alapján készítik el
- A gyakorlatban azt foglalja össze, hogy mit kell a rendszernek/modulnak/feature-nek tudnia
- A formátumának tisztának és jól követhetőnek kell lennie
- Elméletben a formája lehet: beszélt nyelven, formalizált leírással, formális matematikai leírással, formális leírónyelvekkel

SPECIFIKÁCIÓ KÉSZÍTÉS
KÖVETELMÉNYEK DEFINIÁLÁSA

Követelmény specifikáció 2.

- Funkcionális követelmények
  - Milyen funkciók vannak, azok hogy működnek
  - Kitérve minden al-esetre!
  - Interfészek, függőségek, **feltételezések**
- Nem-funkcionális követelmények
  - A rendszerre általában jellemző működés, például
  - Teljesítmény elvárások
    - 3 másodpercen belül fussanak le a lekérdezések
  - Biztonsági elvárások
    - Milyen ipari elvárásoknak feleljen meg
  - Rendelkezésre állási elvárások

Követelmény specifikáció példa

Requirements

#	User story title	User story description	Priority	Notes
1	Facebook Integration MDT-13 TO DO	A user wants to sign up via Facebook	Must Have	• We will need to talk to Cassie Owens. • There has also been some research done on this (see Facebook integration prototype)
2	Activity Stream MDT-14 TO DO	A user wants to view the latest updates via the mobile dashboard so that they can get a better understanding of what is in place	Must Have	
3	Post Updates MDT-15 TO DO	A user wants to be able to post status updates on the go	Must Have	The key things we will need to support: • Text status updates • Mentions • Support for images • Smart embedding for YouTube vids
4	API MDT-16 TO DO	A developer wants to integrate with the mobile app so that they can embed the activity stream on their website	Should Have	• We should chat to Team Dyno as they did something similar.

Rendszerterv

- A tervezés fázis terméke
- A döntések dokumentálása a szoftver felépítésével kapcsolatban
- Része az architektúra kifejtése
  - Pl. ha microservice-eket használunk a fejlesztés során, ez mindenképpen említésre kerül itt
- **Ez alapján történik a fejlesztés**

Rendszerterv felépítése 1.

TERVEZÉS

- Előszó: célközönség, dokumentum-történet
- Bevezetés: szoftver célja, helye, szükségessége, előnyei, fejlesztési módszertan
- Fogalomtár: technikai áttekintés
- Rendszer architektúra
  - Magas szintű áttekintés: UML csomag-, komponens-, állapotdiagramok
  - Architekturális minták
  - Service-ek, modulok, komponensek
  - Funkcionális megfeleltetés
- Felhasználói történetek (user storyk) megfeleltetése

Rendszerterv felépítése 2.

- Belső és külső interfészek, függőségek leírása
- Adattervezés
  - Adattárolás módja
  - Adatbázis tervek, sémák
  - Használt formátumok leírása
- Rendszer tervezés: alacsony szintű tervezés
  - Statikus terv: UML osztály-, objektumdiagramok
  - Dinamikus terv: UML állapot-, szekvencia- és aktivitásdiagramok
  - Felhasznált algoritmusok és minták
- Felhasználói felület
  - áttekintés, felületterv terv: drótvázak / wireframe-ek
  - Dizájn, akár képernyő szinten
- Implementációs ajánlások
- Hardver szükségletek

Forráskód dokumentáció

IMPLEMENTÁLÁS

- Az implementációs fázis egyik eredménytermék
- Elmagyarázza, hogyan működik a kód
- Szoftverfejlesztőknek szól
- Többek között tartalmazhatja:
  - A felhasznált keretrendszereket
  - Adatkötés módját
  - Implementációs döntések hátterét
  - Tervezési minták példákkal
  - Formázási követelmények
  - API-k leírása, megjegyzések

Dokumentáció formái

IMPLEMENTÁLÁS

- A „dokumentáció” nem feltétlenül kiegészítő szövegesen írott dokumentum
- Alternatívák, kiegészítő megoldások
  - **„Öndokumentáló” kód**: kódolási stílus, beszédes nevek, általános és projekt specifikus tervezési minták stb.
  - **Unit tesztek**: egyértelműsítik a komponens határokat, felelősségeket!
  - **UML diagramok** – a közös nyelv! Részben generálható is

Tesztelési dokumentáció

TESZTELÉS

- Teztterv
  - Stratégiák leírása (milyen tesztek lesznek)
  - Milyen funkciókat kell tesztelni
  - Milyen módon kell tesztelni őket
  - Milyen prioritással
  - Szerepek/felelősségek
- Teszteset dokumentáció
  - Részletes leírása a tesztesetnek
  - Mikor verifikálható a teszteset, milyen feltételeknek kell megfelelni
- Teszt ellenőrzőlista

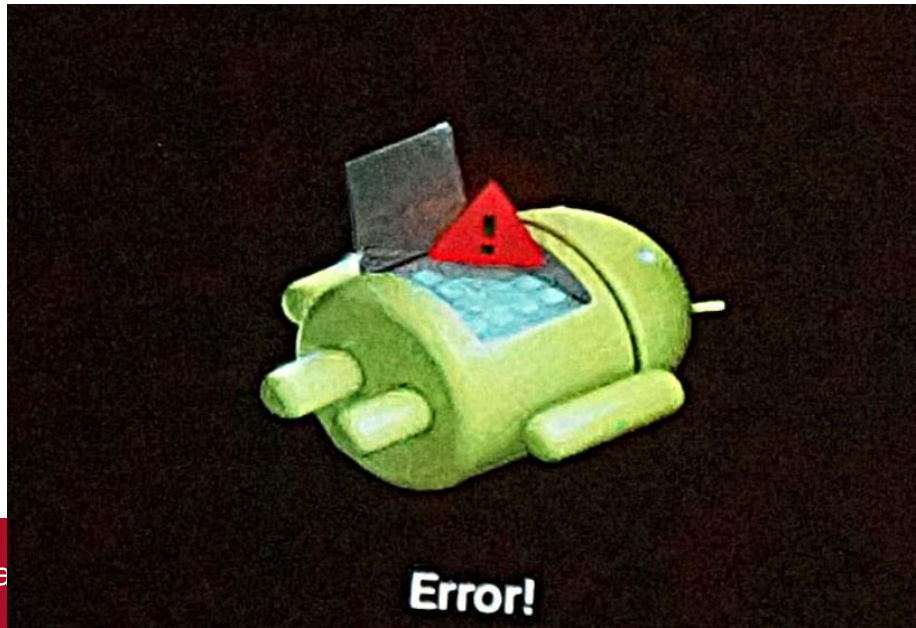
Telepítési, karbantartási dokumentáció


- Telepítési útmutató
  - Különböző környezetekhez, folyamatokhoz
- Üzemeltetési útmutató
  - Honnan tudom, hogy működik a rendszer?
  - Honnan tudom, hogy hibák vannak?
- Katasztrófa utáni helyreállítási útmutató
- Mentés – visszaállítás protokoll
- Archiválás folyamata
- Verzióváltozások leírása

Miért fontos a tesztelés?

- A tesztelés minőségi mutatót állít elő
  - Ha nincs tesztelés, nem tudjuk, hogy mennyire jó az alkalmazásunk

Ne adj át olyan kódot, amit még sosem indítottál el!!



Edsger W. Dijkstra, in 1970

„A tesztelés csak azt tudja megmutatni, ha a rendszerben hiba van, azt sose, ha nincs!”

- Lehetetlen ellenőrizni a program össze lehetséges lefutását
  - > Fontos, hogy okosan teszteljünk!

Szoftver tesztelés

- A **szoftver tesztelés** során ellenőrizzük, hogy a program úgy működik-e ahogy elvárjuk
- Tesztelés nélkül nem tudjuk, hogy azt csinálja-e, amit csinálnia kéne
  - Tesztelés = visszajelzés a minőségről, lehetőség a javításra
- A tesztelést érdemes a fejlesztés elején bevezetni
  - Ez meghatározza azt is, hogy hogyan fogod írni a kódot!
  - Minél korábban találod meg a hibát, annál olcsóbb/egyszerűbb kijavítani

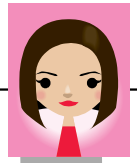
A javítás költsége attól függően, hogy hol és mikor találjuk meg a hibát

	Mikor ismerjük fel a hibát….				
Hol van a hiba	Spec. elemzés	Tervezés	Megvalósítás	QA	Kiadás után
Követelmény	1x	3x	5-10x	10x	10-100x
Terv	-	1x	10x	15x	25-100x
Megvalósítás	-	-	1x	10x	10-25x

Ki van bevonva a tesztelésbe?

Fejlesztő

- Megírja a kódot amit tesztelni lehet
- Átnézi a teszt tervet
- Unit teszteket készít

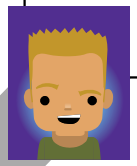


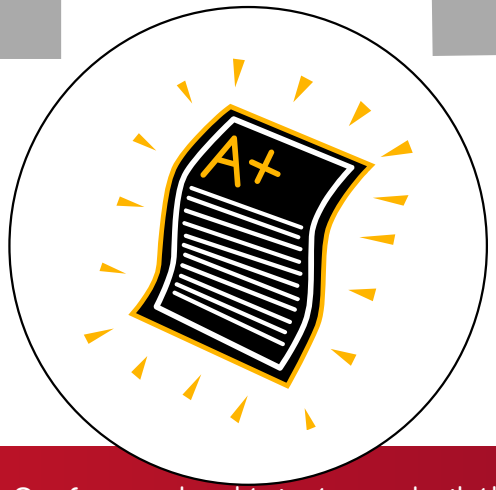
Üzleti elemző / tervező / …

- Funkcionális specifikációt ír, ami alapján a teszt terv készül
- Ellenőrzi a teszt tervet

Tesztelő mérnök

- Teszt tervet készít
- Teszt eseteket ír le és végrehajtja őket
- Jóváhagyja a kiadni kívánt terméket





Megrendelő

- A kiadás előtt az alfa/béta tesztelésben vesz részt
- Átadás / átvételi tesztelés végzi

Végfelhasználó

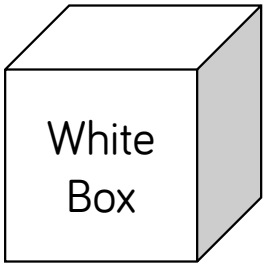
- A felhasználó, akit a ki nem javított hibák érintenek
- Részt vehet béta tesztelésben vagy átvételi tesztben

Automatikus vs manuális

- A tesztelés lehet automatikus vagy manuális
- Manuális tesztelés:
  - Egy *ember* végig kattint az alkalmazáson
  - Lehet ad-hoc, forgatókönyv mentén stb.
- Automatikus tesztelés:
  - A gép végzi
  - Nagy rendszereknél szükség van automatikus tesztelésre

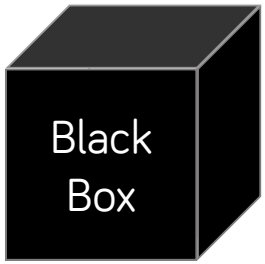
Tesztelési módszerek

- Szokásos „box” analógia a tesztelő mérnök szemszögéből



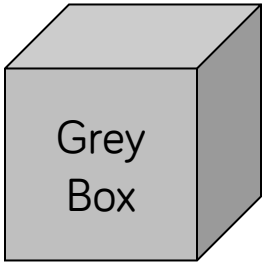
A tesztelő ismeri a szoftver belső felépítését és algoritmusait és felhasználja az információkat

- API tesztelés
- Kódlefedettség vizsgálat
- Hibakezelés vizsgálat



Nem használja a belső működésről ismerteket

- Bemenetekre adott válaszok vizsgálata
- Szélső érték elemzés
- Alpha/beta tesztelés, “dogfooding”



A tesztelő ismeri a szoftver belsejét de felhasználói/fekete doboz szinten tesztel

- Integrációs teszt: interfész szinten
- Reverse engineering

Ellenőrzés és validáció

- Validáció - validation

  “A megfelelő szoftvert készítjük?”

- Ellenőrzés - verification

  “Jó irányban haladunk a szoftverrel?”

- A végfelhasználói teszt a validációról szól a többi teszt a verifikációról

A teszt terv

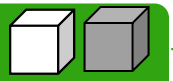
- A teszteléshez tartozó követelmény rendszer
- Teszt terv sablon / tartalomjegyzék
  - A teszt terv célja
  - Szkóp
    - Tesztelt funkciók
    - Nem tesztelt funkciók
  - *Tesztelési stratégia*
  - Teszt esetek
  - Nyitott kérdések

Tesztelési stratégia

- Megadja, melyik tesztet milyen módszerrel végezzük
  - Manuális / automata
  - White / black / grey box
  - Unit teszt, integrációs teszt stb.
- Melyik teszt keretrendszert, milyen eszközöket használunk majd
  - NUnit, Visual Studio, …
- Milyen előfeltételei vannak a tesztek végrehajtásának
  - Infrastruktúra, architektúra, input adatok stb.

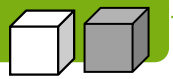
A tesztelés szintjei

Unit tesztelés



- A kód egy nagyon kis részét ellenőrzi
- Tipikusan white-box teszt

Integrációs teszt



- A komponensek közti interfészt ellenőrzi
- Tipikusan iteratív folyamat amíg a teljes rendszer nincs ellenőrizve

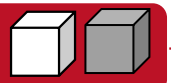
Rendszer tesztelés

- A rendszereket önmagukban ellenőrzi a követelmények tükrében

Rendszer integrációs teszt

- Külső rendszerekkel való együttműködést ellenőrzi
- Mindegyik rendszer átment a „Rendszer teszten”

Regressziós teszt



- Olyan teszt, ami a korábbi hibák újbóli felbukkanását jelzi
- Tipikusan a hibajavítás során készülnek

Elfogadási teszt

- “User Acceptance Testing”
- A végfelhasználó végzi saját környezetében

Alfa, béta teszt

- Nem kész verzió ellenőrzése belső vagy meghívott külső felhasználók által
- „dogfooding”

Biztonsági, terheléses tesztek

- Biztonsági: szakosodott cég végzi
- Terhelés: a várható terhelésre adott választ vizsgáljuk

Unit teszt

- Ez egy komponensteszt, vagyis a rendszer önálló részeit vizsgálja
- A legkisebb szoftveres egységre fókuszál
- A metódusokat kezeli: kimenet-bemenetre fókuszál
  - Jó-e a visszatérési érték

Unit teszt (példa)

```
public class PhoneValidator
{
    public bool IsValid(string phone)
    {
        return UseSomeRegExToTestPhone(phone);
    }
}
```

```
public void TestPhoneValidator()
{
    string goodPhone = "(123) 555-1212";
    string badPhone = "555 12"

    PhoneValidator validator = new PhoneValidator();

    Assert.IsTrue(validator.IsValid(goodPhone));
    Assert.IsFalse(validator.IsValid(badPhone));
}
```

A jó unit test

- Megbízható
- Hosszú távon futtatható, karbantartott
- Jól olvasható
- Egyszerű megírni
- Automatizált és megismételhető
- Bárki egyszerűen futtathatja
- Gyorsan lefut

Integrációs teszt

- A komponensek együttműködését teszteli
  - A komponensek közötti kapcsolatot
  - Az összeillesztéskor keletkező hibát
- Unit tesztek után szokott lenni
- Interfészek definiálásánál kezdődik el a tesztelés
- Hibakeresése nehézkesebb

Rendszerteszt

- Az integrációs teszt után szokott következni
- A rendszerteszt a már kész szoftverterméket teszteli
  - Hogy megfelel-e a követelményeknek
    - Elsősorban funkcionális teszt
  - Megfelel-e a rendszertervnek
- Feketedobozos teszt (blackbox)
- Van, hogy független cég végzi
- Gyakran manuális

Nem funkcionális tesztek

- Naplózás, monitorozás
- Üzemeltetési folyamatok
- Memóriaszivárgás
- Teljesítmény
- Szűk keresztmetszetek – bottleneck
- Működés szélsőséges körülmények között
  - Nagy terhelés
  - Hálózati hibák
  - Adattároló sérülése, kiesése
  - ...

Átvételi tesztek

- Az átvételi folyamathoz tartozik
  - Alfa teszt: meghívott végfelhasználók végzik, nem a fejlesztőcsapat
  - Béta teszt: végfelhasználók szűk csoportja végzi
  - Felhasználói átvételi teszt
    - UAT: User Acceptance Test
    - Felhasználói köre projektfüggő
    - **Tipikusan ennek eredménye alapján kerül kiállításra a teljesítés igazolás**
  - Üzemeltetői teszt: rendszergazdák tesztelik a rendszer viselkedését, üzemeltetési protokollokat

Regressziós teszt

- Korábban megjelent hibát vagy forgatókönyvet tesztel újra
- Tipikusan valamilyen változtatás után végzik: jó-e még a rendszer?
  - Amikor fejlesztésnél írunk egy új funkciót, akkor azt szeretnénk, hogy a már meglévő funkciók ne sérüljenek
  - Ezt segítik az automata unit tesztek

További részletek

- https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f
- https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/

Köszönöm a figyelmet!