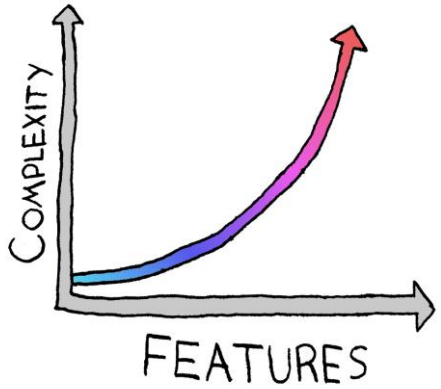
Dokumentácia, Markdown

Projekt 1

Prednáška 1

Motivácia



- Feri začne robiť projekt
- Zo začiatku vie ako všetko funguje
- Lebo všetko má v hlave
- Ale projekt rastie, Feri nestíha a prizve si pomoc.
- Čo sa stane, ak sa k projektu pridá Lucka?
  - Čuj Feri ako to spustím?
  - A furt mi tu eklips/intelidžej/veeskoud ukazuje červené.
  - Šak mám džavu nainštalovanú, nesom blbá.
  - Aháá, to treba dvadsaťjednotku a ja mám len jedenástku, si mal rovno povedať.
  - Ako to napojím k databáze?
  - A ako tam strčím ten apikej?
  - ...

Kruh sa uzatvára

- Feri je frustrovaný, lebo ho Lucka zdržiava s triviálnymi „somarinami“.
- Myslí si, že prizvať si pomoc bola chyba.
- Lucka je frustrovaná, lebo chcela Ferimu pomôcť a má pocit, že všetko robí zle.
- Lucka sa na to vykašle a Feri pokračuje sám
- Po chvíli si uvedomí, že fakt nestíha a potrebuje rozdeliť prácu na viacero ľudí.
- Prizve si na pomoc Jožka
  - Čuj Feri ako to spustím?
  - A furt mi tu eklips...

Ponaučenie

- Software veľmi rýchlo rastie na zložitosti
  - Každá fíčurka, každý IF, atď. zdvojnásobí komplexnosť programu
- Veľký projekt má veľmi veľa nefunkčných stavov a veľmi málo funkčných stavov (verzia jazyka, verzie knižníc, „naplnenie“ DB, typ a verzia OS, typ CPU)
- Pôvodní autori majú všetko v hlave.
- Ale noví programátori sa prakticky okamžite stratia a dlho sa v tom hľadajú.

Riešenie

Riešenie

- Dobré riešenie neexistuje
  - ~~Bože, nech už príde AI, čo bude všetko robiť za nás a ja sa už nebudem musieť prehrabávať sračkami cudzím kódom~~
- Vieme to ale slušne zjednodušiť


EXCUSE ME SIR
HAVE YOU HEARD ABOUT OUR LORD AND SAVIOR THE DOCUMENTATION?
imgflip.com

Dokumentácia projektu

- Každý projekt potrebuje dokumentáciu
- Stručný popis
- Ako ho spojazdniť
  - Aké OS sú podporované/vyskúšané
  - Čo potrebujeme mať nainštalované
    - JDK 21, NodeJS 20, Maven, Docker, ...
  - Čo treba spustiť
    - `build.sh, build.bat`
    - `mvn install`
    - `docker compose up`
- Rôzne dodatočné nastavenia
- Ako ho používať
  - Je to REST API? Dajte špecifikáciu.
  - Je to UI? Dajte obrázky/gify.
- Typické príklady použitia

Ako začať dokumentáciu

- Zvoľte si dokumentačný formát
  - **Markdown**
  - AsciiDoc
  - Čistý text
- V koreni projektu vytvorte súbor `README.md` (`.txt`, `.adoc`)
- Do README dajte základne informácie
  - Logo
  - Stručný popis
  - Ako spojazdniť projekt
  - Typický príklad použitia
- Komplexnejšie vysvetlenia a detaily dajte do samostatných `*.md` súborov v priečinku `docs`
  - A do README vložte iba relatívne odkazy kde je to vhodné

Príklady peknej dokumentácie

- https://github.com/ollama/ollama
- https://github.com/compiiile/compiiile

Markdown

- Veľmi jednoduchý značkovací jazyk.
- Stačí vám notepad.exe a Windows Explorer.
  - Samozrejme IntelliJ/VS Code majú vstavanú podporu
- Viete veľmi ľahko transformovať surový text do peknej web-stránky, PDF, atď.
- Návod https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
- Použitie
  - Dokumentácia projektu
    - Čeknite https://github.com/compiiile/compiiile na generovanie dokumentačnej webstránky
  - Tvorba webstránok
    - Čeknite https://gohugo.io/
  - Písanie kníh, bakalárok (ale na to asi radšej LaTeX)

Git, Vetvenie

Projekt 1

Prednáška 1

Git

- Git je verzovací kontrolný systém (VCS)
- Je to protokol na správu zdrojového kódu
- Zdrojáky vášho projektu budú hostované na vzdialenom serveri
  - git init – vytvor nový repozitár
  - git clone – stiahni vzdialený repozitár
  - git fetch – aktualizuj metadáta o projekte
  - git pull – stiahni aktuálne zdrojáky
  - git commit – lokálne ulož aktuálne zmeny v kóde do gitu
  - git log – pozri históriu komitov (zmien)
  - git revert – zruš konkrétny komit
  - git restore – vráti celý repozitár, alebo zadaný súbor do stavu posledného komitu
    - zruší všetky nekomitnuté zmeny v kóde

Git – fakt sa potrebujem učiť tie príkazy?

- Moderné IDE ako IntelliJ IDEA, alebo Visual Studio Code majú špičkovú GUI podporu pre Git.
- Git má tiež vlastný GUI klient
- Základná znalosť Gitu a týchto príkazov je však nutnou výbavou každého developera, alebo devopsáka
  - Najlepšie platené technické pozície v IT

Git vs GitHub vs GitLab

- Git je protokol pre správu projektov
- Niekde ale potrebujete server, kde projekty budete hostovať
- Je mnoho implementácií Git protokolu
- GitHub a GitLab sú hotové služby, ktoré, okrem iného, poskytujú Git protokol
- GitHub a GitLab sú konkurenčné služby
  - GitHub je známejší a je defacto štandardný hosting pre open-source SW
    - Aj keď ho vlastní Microsoft
  - GitLab umožňuje inštaláciu na vlastný server
    - Omnoho používanejši vo firmách, ktoré robia uzavretý SW

Git vs GitHub vs GitLab

- GitHub a GitLab dnes poskytujú omnoho viac ako len Git
- Vedia
  - Kompilovať/buildovať váš projekt
  - Nasadzovať ho na váš server
  - Poskytnúť binárky vášho programu na stiahnutie
  - Priamo hostovať webstránky
  - Robiť projektový manažment
    - GitLab je v tomto o dosť lepší ako GitHub, ale asi nič nemá na Jira
  - Robiť „code review“
    - Členovia tímu vedia kontrolovať a komentovať zmeny v kóde pred zverejnením do prevádzky

Vetvy projektu

- Na projekte je typicky viacero vývojárov
- Každý vývojár potrebuje vlastný „piesoček“
  - Každá úloha na projekte má vlastnú vetvu
  - Zoberiete si posledný známy funkčný kód
  - Z neho si vytvoríte novú vetvu
  - Vo vetve si robíte s kódom čo chcete
  - Zmeny vo vašej vetve neovplyvňujú iné vetvy
  - Na konci sa vetva zlúči s pôvodnou
  - Ostatní si svoje vetvy priebežne aktualizujú z pôvodnej vetvy

Vetvy

- Každý projekt má hlavnú vetvu
  - **master** (staršie projekty)
  - **main** (nové projekty, lebo slovo master je „rasistické“)
  - V hlavnej vetve môže zmeny robiť typicky iba tech-líder (hlavný vývojár)
- Vedľajšie vetvy
  - Každá úloha na projekte má vlastnú vetvu
  - To je váš „piesoček“
  - Tech-líder zlúči dokončenú úlohu do hlavnej vetvy

Stratégia vetvenia

- V každom projekte sa vetvy pomenovávajú trocha ináč.
- Existujú však „odporúčania“ ako správne pracovať s vetvami.
  - GitFlow
  - GitHub Flow
  - GitLab Flow
  - Trunk

GitFlow

- Najpoužívanejšia stratégia v korporátoch
- Vhodný pre obrovské projekty
- Ak potrebujete dlhšie udržiavať viac verzií projektu
  - Windows 10/11, Java 8/11/17/21, ...
- Máte viacero hlavných vetví
  - master/main – tu je posledné vydanie projektu (napr. v1.14)
  - develop – z toho sa stane ďalšie vydanie projektu
- Máte podporné vetvy
  - feature-* (napr. feature-credit-card-payments)
  - bugfix-* (napr. bugfix-nullpointer-in-login-when-empty-password)
  - release-* (napr. release-v1.12, release-v1.13, release-v1.14)

GitFlow cyklus - programujem

- Príklad: Chcem pridať prihlasovanie cez Google OAuth
- Z `develop` vetvy si vytvorím svoju *feature-google-oauth* vetvu (**git checkout** *develop* && **git pull** && **git checkout -b** feature-google-oauth)
- Implementujem svoje veci a spravím **git commit --am** “implements Google OAuth“
- Urobím **git fetch** a **git merge** *origin/develop* z hlavnej vetvy (develop) do mojej
  - Aktualizujem si svoju vetvu o najnovší kód z hlavnej vetvy
- Nahrám svoje zmeny cez **git push** –*u origin* feature-google-oauth
- Vytvorím „merge request“ („pull request“) na GitLabe (GitHube)
  - Chcem zlúčiť vedľajšiu vetvu s develop vetvou
- Vaši kolegovia a tím-líder (váš šéf) skontrolujú váš kód („code review“)
- Ak je všetko OK, tak tím-líder, alebo tech-líder (šéf tím-lídera) urobí **git merge** do hlavnej vetvy
  - Vedľajšia vetva sa zlúči s develop (a prípadne prestane existovať)

GitFlow cyklus – vydávam novú verziu

- Príklad: Som tech-líder a chcem urobiť nové vydanie
- Z `develop` vetvy si vytvorím svoju `release-[verzia]` vetvu
- Urobím potrebné zmeny (aktualizovať minor verziu v dokumentácii atď)
- Nechám QA tím (Quality Assurance) poriadne otestovať
- Fixnem, alebo nechám niekoho fixnúť prípadné bugy
- Vytvorím „merge request“ („pull request“)
  - Chcem zlúčiť vedľajšiu vetvu s main vetvou
- Ak je všetko OK, tak urobím „mérdž“
  - Vedľajšia vetva sa zlúči s main
- Urobím ešte dodatočné zlúčenie z main do develop

GitFlow poznámky pod čiarou

- `feature-*` a `bugfix-*` vetvy nemusia mať vždy funkčný kód
  - Tieto si programátori vytvárajú podľa potreby a robia si tam (takmer) čo chcú
  - Pri zlučovaní však kód musí byť (mal by byť) funkčný
- Ostatné vetvy by mali vždy obsahovať preložiteľný/spustiteľný/funkčný kód
  - Samozrejme chyby sa stávajú
    - Inak by už dávno programátori nemali čo jesť
    - ~~Bohužial ChatGPT nás tak skoro nenahradí, tá vec generuje chyby ako šialená, a nevio ich fixovať~~
  - Z chýb sa stávajú `bugfix-*` vetvy, ktoré sa po vyriešení chyby zlúčia do vetvy, kde bola chyba objavená

Verzie projektu

- `release-*` často existujú dlhšiu dobu
- Napr. sa backportujú mergnuté bugfixy z mainu ak sa bug týka aj týchto starších vydaní (`git cherry-pick`)
- Interne sa vo vetve release-v1.13 program dodatočne verzuje ako v1.13.4, v1.13.5.
- Každý bugfix zvýši verziu x.y.**z+1** a každá fíčurka zvýši verziu x.**y+1**.z
  - „Breaking change“, teda veľká zmena, ktorá narúša spätnú kompatibilitu zvýši verziu **x+1**.y.z
- Každá verzia projektu by mala mať `git tag` (napr tag v1.13.4)
  - Z tagov si viete vytiahnuť historický zdrojový kód v čase vydania danej verzie
- Treba užívateľom deklarovať dĺžku podpory vydania.
  - Typicky 6-12 mesiacov pre veľké projekty
  - Niektoré vybrané vydania môžu byť LTS (long term support)
    - Niekoľko rokov podpory
- Verzovanie a tagovanie môžeme robiť ručne,
  - https://git-scm.com/book/en/v2/Git-Basics-Tagging

  alebo automatizovane
  - https://semantic-release.gitbook.io/semantic-release/

GitHub Flow

- Populárna stratégia v mladších firmách
- Vhodný ak vám stačí podporovať iba poslednú verziu projektu
  - Napr. Steam, videohry, ...
- Máte iba jednu hlavnú vetvu
  - master/main – tu je posledné vydanie projektu (napr. v1.14)
- Staršie verzie majú stále svoj vlastný git tag (v1.13.5, v1.14.0)
- Máte podporné vetvy
  - feature-*
  - bugfix-*

GitHub Flow vs GitFlow

- Vedľajšie vetvy sa zlučujú rovno do main/master
- QA sa robí na každej vedľajšej vetve pred zlúčením
- Rýchlejšie vydávanie nových verzií

Iné stratégie

GitLab Flow

- Hybrid medzi GitFlow a GitHub Flow

Trunk

- Kašleme na vetvy a všetko strkáme do main/master
- Vhodný ak:
  - Ak som sám
  - Ak je tím malý (2-4 ľudia)
    - Členovia sú veľmi skúsení a veľmi dobre sa poznajú

hmm

- Git a verzovanie sú komplexné témy a nejestvuje jednotný prístup
  - Budeme zlučovať cez rebase alebo merge?
    - Ak neviete rozdiel, ostaňte pri git merge.
      - Je jednoduchší na použitie, aj keď robí trošička škaredšiu históriu zmien (git log)
  - Aký flow použijeme?
  - Ako budeme vydávať verzie?
- Každá firma/projekt/tím si veci robí po svojom
  - Niekedy dobre, niekedy nie-dobre

Ďakujem za pozornosť

Nech vás sprevádza Sila