Test driven development

Test-Driven Development (TDD) je vývojový přístup, který se zaměřuje na psaní testů před implementací kódu. Tím je zaručována jeho správnost a funkčnost. Již napsaný kód je v průběhu vývoje znovu testován pro zaručení kompatibility s novým kódem.

Vývoj tímto způsobem se dělí na jednotlivé fáze:

1. **Založení seznamu testů pro novou funkci:**
   * Prvním krokem v TDD je vytvoření seznamu očekávaných variant nového chování. To zahrnuje základní případ a další scénáře, například situaci, kdy dojde k timeoutu služby nebo když klíč ještě není v databázi. Specifikace lze zjistit prostřednictvím uživatelských příběhů a uživatelských případů.
   * Příkladem
2. **Přidání jednoho testu ze seznamu:**
   * Napsání automatizovaného testu, který projde, pokud nové chování splňuje očekávání.
3. **Spuštění všech testů:**
   * Nový test by měl selhat z očekávaných důvodů, což ukazuje, že je třeba nový kód k implementaci požadované funkce. To ověřuje, že testovací sada funguje správně a vylučuje možnost, že nový test je chybný a vždy projde.
4. **Napsání nejjednoduššího kódu, který projde nový test:**
   * I neelegantní nebo obtížný kód je akceptovatelný, pokud projde testem. Kód bude později vylepšen v kroku 6. Nepřidávejte žádný kód mimo testovanou funkcionalitu.
5. **Všechny testy by nyní měly projít:**
   * Pokud některý selže, nový kód musí být upraven, dokud všechny testy neprojdou. To zajišťuje, že nový kód splňuje požadavky testů a neporušuje existující funkce.
6. **Refaktorování podle potřeby, s použitím testů po každém refaktoru k zajištění zachování funkcionality:**
   * Kód je upravován pro čitelnost a udržitelnost. Výslovně se snažte odstranit tvrdé kódovaná testovací data. Spuštění testovací sady po každém refaktoru pomáhá zajistit, že žádná existující funkcionalita není narušena.

Příklady refaktorování: - Přesunutí kódu na nejlogičtější místo - Odstranění duplicitního kódu - Vytvoření názvů, které se samy dokumentují - Rozdělení metod na menší části - Přeorganizování dědičnostních hierarchií

1. **Přidání dalšího testu ze seznamu:**
   * Opakování procesu s každým testem na seznamu, dokud nejsou implementovány a všechny testy projdou.
2. **Opakování:**
   * Cyklus výše se opakuje pro každý nový kousek funkcionality. Testy by měly být malé a postupné, a commitovány by měly být často. To umožní, že pokud nový kód selže v některých testech, programátor může jednoduše vrátit zpět nebo revertovat, místo aby se musel nadměrně zabývat laděním. Při používání externích knihoven je důležité nepisovat testy, které jsou tak malé, že efektivně testují pouze samotnou knihovnu, pokud není nějaký důvod věřit, že knihovna je chybná nebo nedostačující pro potřeby vyvíjeného softwaru.

Prvním krokem TDD je napsat test, který popisuje požadovanou funkcionalitu. Tyto testy jsou krátké a jednoduché, zaměřují se na jednu konkrétní vlastnost. Příkladem při vývoji.

Poté, co je test napsán, je spuštěn a očekává se, že selže, protože odpovídající kód ještě neexistuje. Druhý krok spočívá v implementaci minimálního množství kódu nutného k tomu, aby test prošel. Tímto způsobem je postupně vytvářen funkční kód.

Po napsání kódu je opět spuštěn test. Jestliže test projde, víme, že kód dělá to, co by měl. Pokud ne, je třeba kód upravit tak, aby test prošel. Tento cyklus opakujeme a postupně přidáváme další testy, abychom zajistili, že kód je robustní a správně funguje.

Výhody TDD jsou mnoho. První z nich spočívá v tom, že testy slouží jako dokumentace. Studenti mohou představit testy jako instrukce k tomu, jak kód používat. TDD také zvyšuje důvěru ve správnost kódu, protože každá změna je okamžitě ověřena testy. Tím se minimalizuje riziko chyb a usnadňuje se údržba a rozšiřování kódu.

Celkově lze říci, že TDD je systematický přístup ke vývoji softwaru, který zdůrazňuje psaní testů před kódem. Tímto způsobem je dosaženo lepšího a spolehlivějšího softwarového produktu.

Důležité je si uvědomit, že TDD není pouze vývojová metodologie, ale také designový nástroj. Při psaní testů se programátori zamýšlejí nad tím, jak by měl jejich kód fungovat, což vede k lepší architektuře a modularitě. Studenti mohou přirovnat tuto část procesu k plánování před stavbou domu – testy jsou plánem, který zajišťuje, že každá část kódu má svůj přesný účel.

Dalším benefitům TDD je také rychlejší odhalování chyb. Když je chyba nalezena, může být opravena ihned, jelikož testy identifikují problémové místo. Tím se minimalizuje doba, kterou programátoři stráví laděním a hledáním chyb po dokončení celého projektu.

Navíc, TDD podporuje princip inkrementálního vývoje. Kód je postaven kousek po kousku, a každý kousek je ověřen testy. To umožňuje týmu dodávat funkční části produktu postupně a získávat zpětnou vazbu od zákazníků nebo ostatních členů týmu.

Pro studenty středních škol je pochopení TDD užitečné nejen pro softwarový vývoj, ale i pro zdokonalení dovedností v problematice logického myšlení, plánování a systematického přístupu k řešení problémů. TDD tak může být užitečným nástrojem pro rozvoj kompetencí potřebných nejen v oblasti programování, ale i v různých dalších disciplínách.

Další výhodou TDD je zvýšení produktivity a efektivity vývojového týmu. Přestože na začátku může psaní testů přidat trochu času, v dlouhodobém horizontu to může ušetřit mnoho času a námahy, protože odhalené chyby jsou odstraněny dříve v procesu vývoje. To také znamená, že nejsou potřeba rozsáhlé odladitelské fáze, protože každý kus kódu byl již podroben testům.

Důležitým prvkem TDD je také podpora refaktorování. Refaktorování znamená zlepšení kódu bez změny jeho vnějšího chování. S tímto přístupem mohou studenti postupně vylepšovat svůj kód bez obav, že něco rozbijí. Testy fungují jako bezpečný záchranný síť, který zajišťuje, že i po úpravách bude kód stále fungovat správně.

V praxi TDD vedoucím studentům umožňuje postavit své myšlenky na papír v podobě testů a následně implementovat funkce krok za krokem. To napomáhá lépe porozumět problému a poskytuje jasný postup, který snižuje zmatení při vývoji.

Celkově vzato, Test-Driven Development je metoda, která nejenže zlepšuje kvalitu kódu a snižuje chybovost, ale také usnadňuje spolupráci vývojového týmu a umožňuje efektivní rozvoj softwarových projektů. Je to přístup, který vzdělává studenty v systematickém a disciplinovaném přístupu k vývoji softwaru, což je klíčový aspekt moderního informačního věku.