Faza implementacji

Rezultatem fazy implementacji jest działający kod aplikacji.

Faza implementacji

Rezultatem fazy implementacji jest działający kod aplikacji.

Dobry kod jest:

- niezawodny
- przejrzysty i zrozumiały
- łatwo modyfikowalny
- optymalny

Faza implementacji

Rezultatem fazy implementacji jest działający kod aplikacji.

Dobry kod jest:

- niezawodny
- przejrzysty i zrozumiały
- łatwo modyfikowalny
- optymalny

Należy osiągnąć równowagę pomiędzy dążeniem do stworzenia kodu bliskiego idealnemu a przestrzeganiem ograniczeń czasu i zasobów.

Implementacja

W ostatnich latach faza implementacji uległa znacznemu przyspieszeniu i automatyzacji. Osiągnięto to dzięki:

- Używaniu języków wysokiego poziomu
- Wykorzystywaniu gotowych bibliotek i komponentów
- Używaniu automatycznych generatorów kodu
- Różnego typu narzędzi wspomagających szybkie tworzenie aplikacji

Implementacja

W ostatnich latach faza implementacji uległa znacznemu przyspieszeniu i automatyzacji. Osiągnięto to dzięki:

- Używaniu języków wysokiego poziomu
- Wykorzystywaniu gotowych bibliotek i komponentów
- Używaniu automatycznych generatorów kodu
- Różnego typu narzędzi wspomagających szybkie tworzenie aplikacji

Pomimo to żadnym narzędziom nie udało się zastąpić całkowicie uważnego programisty

Implementacja

W ostatnich latach faza implementacji uległa znacznemu przyspieszeniu i automatyzacji. Osiągnięto to dzięki:

- Używaniu języków wysokiego poziomu
- Wykorzystywaniu gotowych bibliotek i komponentów
- Używaniu automatycznych generatorów kodu
- Różnego typu narzędzi wspomagających szybkie tworzenie aplikacji

Pomimo to żadnym narzędziom nie udało się zastąpić całkowicie uważnego programisty – jak dotąd przynajmniej :).

Potencjalnie niebezpieczne techniki

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb
- Stosowanie wskaźników – możliwość dostępu do sektorów pamięci zajętych przez inne aplikacje lub system

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb
- Stosowanie wskaźników – możliwość dostępu do sektorów pamięci zajętych przez inne aplikacje lub system
- Obliczenia równoległe – trudne w analizie, możliwy niedeterminizm wyników, zakleszczenie, zagłodzenie

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb
- Stosowanie wskaźników – możliwość dostępu do sektorów pamięci zajętych przez inne aplikacje lub system
- Obliczenia równoległe – trudne w analizie, możliwy niedeterminizm wyników, zakleszczenie, zagłodzenie
- Przerwania, wyjątki – trudności w analizie

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb
- Stosowanie wskaźników – możliwość dostępu do sektorów pamięci zajętych przez inne aplikacje lub system
- Obliczenia równoległe – trudne w analizie, możliwy niedeterminizm wyników, zakleszczenie, zagłodzenie
- Przerwania, wyjątki – trudności w analizie
- Rekurencja – możliwość przepełnienia stosu, programy mogą być trudne do zrozumienia

Potencjalnie niebezpieczne techniki

- Używanie instrukcji `goto` – prowadzi do powstawania programów trudnych w analizie
- Stosowanie arytmetyki zmiennopozycyjnej – przybliżenia w obliczeniach mogą prowadzić do nieoczekiwanych wyników
- Używanie operacji bitowych – łatwo popełnić błędy, potencjalne problemy przy różnym kodowaniu liczb
- Stosowanie wskaźników – możliwość dostępu do sektorów pamięci zajętych przez inne aplikacje lub system
- Obliczenia równoległe – trudne w analizie, możliwy niedeterminizm wyników, zakleszczenie, zagłodzenie
- Przerwania, wyjątki – trudności w analizie
- Rekurencja – możliwość przepełnienia stosu, programy mogą być trudne do zrozumienia
- Dynamiczna alokacja pamięci – bez automatycznego zwalniania pamięci może prowadzić do wycieków pamięci

Potencjalnie niebezpieczne techniki

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji
- Funkcje, których działanie jest zupełnie odmienne dla różnych parametrów lub stanu zmiennych zewnętrznych

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji
- Funkcje, których działanie jest zupełnie odmienne dla różnych parametrów lub stanu zmiennych zewnętrznych
- Niestosowanie nawiasów w skomplikowanych wyrażeniach arytmetycznych – priorytet operatorów jest trudny do kontrolowania przez programistę

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji
- Funkcje, których działanie jest zupełnie odmienne dla różnych parametrów lub stanu zmiennych zewnętrznych
- Niestosowanie nawiasów w skomplikowanych wyrażeniach arytmetycznych – priorytet operatorów jest trudny do kontrolowania przez programistę
- Współdzielony dostęp do danych – bez odpowiednich mechanizmów synchronizacji może powodować (i z dużym prawdopodobieństwem spowoduje) błędy i niespójność danych

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji
- Funkcje, których działanie jest zupełnie odmienne dla różnych parametrów lub stanu zmiennych zewnętrznych
- Niestosowanie nawiasów w skomplikowanych wyrażeniach arytmetycznych – priorytet operatorów jest trudny do kontrolowania przez programistę
- Współdzielony dostęp do danych – bez odpowiednich mechanizmów synchronizacji może powodować (i z dużym prawdopodobieństwem spowoduje) błędy i niespójność danych
- Używanie funkcji z bibliotek bez dokładnej znajomości ich działania

Potencjalnie niebezpieczne techniki

- Efekty uboczne funkcji
- Funkcje, których działanie jest zupełnie odmienne dla różnych parametrów lub stanu zmiennych zewnętrznych
- Niestosowanie nawiasów w skomplikowanych wyrażeniach arytmetycznych – priorytet operatorów jest trudny do kontrolowania przez programistę
- Współdzielony dostęp do danych – bez odpowiednich mechanizmów synchronizacji może powodować (i z dużym prawdopodobieństwem spowoduje) błędy i niespójność danych
- Używanie funkcji z bibliotek bez dokładnej znajomości ich działania
- Stosowanie optymalizacji – utrudnia zrozumienie kodu, może powodować błędy, utrudnia automatyczne optymalizacje kompilatora

Potencjalnie niebezpieczne techniki

Uwaga

Wiele z tych praktyk jest niezwykle użytecznych i może znacznie usprawnić działanie programu. Wymagają jednak dużej uwagi i ostrożności.

Dobre praktyki

Dobre praktyki

- Hermetyzacja – ukrywanie wszystkiego, co tylko może być ukryte

Dobre praktyki

- Hermetyzacja – ukrywanie wszystkiego, co tylko może być ukryte
- Mocna kontrola typów – kompilator sprawdza poprawność przypisań (ok. 80% błędów poza syntaktycznymi)

Dobre praktyki

- Hermetyzacja – ukrywanie wszystkiego, co tylko może być ukryte
- Mocna kontrola typów – kompilator sprawdza poprawność przypisań (ok. 80% błędów poza syntaktycznymi)
- Tolerancja błędów – takie przygotowanie aplikacji, aby w razie wystąpienia błędu program mógł na to sensownie zareagować (wykryć błąd, zakończyć pracę odpowiedniego modułu lub spróbować naprawić błąd)

Poprawny styl kodowania

Dla wielu języków programowania (np. C++, Java, C#) istnieją zbiory zasad określających poprawny styl kodowania. Przestrzeganie ich ułatwia tworzenie programów zgodnych ze standardami języka.

Poprawny styl kodowania

Dla wielu języków programowania (np. C++, Java, C#) istnieją zbiory zasad określających poprawny styl kodowania. Przestrzeganie ich ułatwia tworzenie programów zgodnych ze standardami języka. Osobie czytającej kod będzie znacznie łatwiej go analizować, wiedząc że przestrzegane są zasady stylu kodowania.

Poprawny styl kodowania

Dla wielu języków programowania (np. C++, Java, C#) istnieją zbiory zasad określających poprawny styl kodowania. Przestrzeganie ich ułatwia tworzenie programów zgodnych ze standardami języka. Osobie czytającej kod będzie znacznie łatwiej go analizować, wiedząc że przestrzegane są zasady stylu kodowania.
Istnieje wiele aplikacji automatycznie analizujących kod i wskazujących ewentualnie usterki w stylu kodowania.
Przykład: **Gendarme**

Praca na współdzielonym kodzie

Przy wspólnej pracy nad projektem często występuje konieczność współdzielenia kodu.
Praca na oddzielnych kopiach skutkuje dużymi trudnościami przy łączeniu kodu.

Praca na współdzielonym kodzie

Przy wspólnej pracy nad projektem często występuje konieczność współdzielenia kodu.
Praca na oddzielnych kopiach skutkuje dużymi trudnościami przy łączeniu kodu.
Jednym z rozwiązań tego problemu są **repozytoria kodu**.

Repozytorium kodu

Jak działa repozytorium kodu:

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze
3. Zanim przystąpimy do pracy uaktualniamy lokalną wersję plików (ściągamy najnowszą z serwera)

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze
3. Zanim przystąpimy do pracy uaktualniamy lokalną wersję plików (ściągamy najnowszą z serwera)
4. Podczas pracy modyfikujemy lokalną kopię plików z kodem

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze
3. Zanim przystąpimy do pracy uaktualniamy lokalną wersję plików (ściągamy najnowszą z serwera)
4. Podczas pracy modyfikujemy lokalną kopię plików z kodem
5. Po skończonej pracy uaktualniamy wersję na serwerze

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze
3. Zanim przystąpimy do pracy uaktualniamy lokalną wersję plików (ściągamy najnowszą z serwera)
4. Podczas pracy modyfikujemy lokalną kopię plików z kodem
5. Po skończonej pracy uaktualniamy wersję na serwerze
6. Repozytorium kodu przechowuje kolejne wersje plików, łatwe jest więc śledzenie zmian

Repozytorium kodu

Jak działa repozytorium kodu:

1. Na serwerze tworzymy nowe repozytorium, gdzie będziemy wgrywać pliki zawierające kod
2. Wszyscy uczestnicy projektu synchronizują swoje foldery lokalne ze zdalnym folderem na serwerze
3. Zanim przystąpimy do pracy uaktualniamy lokalną wersję plików (ściągamy najnowszą z serwera)
4. Podczas pracy modyfikujemy lokalną kopię plików z kodem
5. Po skończonej pracy uaktualniamy wersję na serwerze
6. Repozytorium kodu przechowuje kolejne wersje plików, łatwe jest więc śledzenie zmian
7. W przypadku konfliktów możemy porównać oba pliki i usunąć niezgodności

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

1. Konflikty usuwamy na lokalnej wersji pliku, dopiero zsynchronizowane pliki umieszczamy na serwerze

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

1. Konflikty usuwamy na lokalnej wersji pliku, dopiero zsynchronizowane pliki umieszczamy na serwerze
2. Nie umieszczamy na serwerze kodu, który się nie kompiluje

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

1. Konflikty usuwamy na lokalnej wersji pliku, dopiero zsynchronizowane pliki umieszczamy na serwerze
2. Nie umieszczamy na serwerze kodu, który się nie kompiluje (a najlepiej takiego, który nie działa)

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

1. Konflikty usuwamy na lokalnej wersji pliku, dopiero zsynchronizowane pliki umieszczamy na serwerze
2. Nie umieszczamy na serwerze kodu, który się nie kompiluje (a najlepiej takiego, który nie działa)
3. Większe zmiany opatrujemy komentarzem opisującym, co się zmieniło

Repozytorium kodu

Przy korzystaniu z repozytorium kodu warto stosować kilka zasad:

1. Konflikty usuwamy na lokalnej wersji pliku, dopiero zsynchronizowane pliki umieszczamy na serwerze
2. Nie umieszczamy na serwerze kodu, który się nie kompiluje (a najlepiej takiego, który nie działa)
3. Większe zmiany opatrujemy komentarzem opisującym, co się zmieniło
4. Nie umieszczamy w repozytorium plików użytkownika, skompilowanych plików aplikacji itp.

Repozytorium kodu

Repozytorium kodu nie musi służyć do przechowywania kodu. Można za jego pomocą współdzielić wszystkie pliki tekstowe (np. pliki TEX).

Repozytorium kodu

Repozytorium kodu nie musi służyć do przechowywania kodu.
Można za jego pomocą współdzielić wszystkie pliki tekstowe (np. pliki TEX).
Można też współdzielić pliki inne niż tekstowe (np. `pdf`, `doc`, `exe`), jednak przy nich nie możemy korzystać z mechanizmu porównywania zmian.

Repozytorium kodu

Istnieje kilka popularnych protokołów współdzielenia kodu:

- Git
- CVS
- SVN (nowszy i lepszy CVS)
- Mercurial (Hg)

Repozytorium kodu

Istnieje kilka popularnych protokołów współdzielenia kodu:

- Git
- CVS
- SVN (nowszy i lepszy CVS)
- Mercurial (Hg)

Repozytoria możemy tworzyć na własnym serwerze lub na publicznie dostępnych serwerach świadczących takie usługi (np. *Github*, *Bitbucket*, *Assembla*).

Czas na nieco praktyki

Zacznijmy od połączenia się z istniejącym repozytorium.

1. Wejdź do folderu, gdzie chcesz utworzyć lokalną kopię repozytorium.
2. Wpisz
   `git clone https://pawel_rzazewski@bitbucket.org/pawel_rzazewski/pz-mni-1617.git`