Narzędzia programistyczne

Przydatne narzędzia

- **kompilator** + edytor tekstu
- **debugger** (debuger) służy do dynamicznej analizy programu w czasie jego działania, m.in. do śledzenia wartości zmiennych i stanu stosu oraz lokalizowania błędów.
- **profilowanie** polega na badaniu zachowania programu na podstawie danych zebranych podczas jego wykonywania, np. grafu wywołań, czasu wykonania lub zużycia pamięci.
- **analiza statyczna kodu (linter)** polega na analizie kodu bez uruchamiania programu, m.in. w celu wykrycia potencjalnych błędów i problemów ze stylem (tzw. *code smells*).
- **systemy kontroli wersji** śledzą zmiany w kodzie, ułatwiają współpracę zespołową i scalanie zmian.
- środowiska IDE - integrują wiele narzędzi

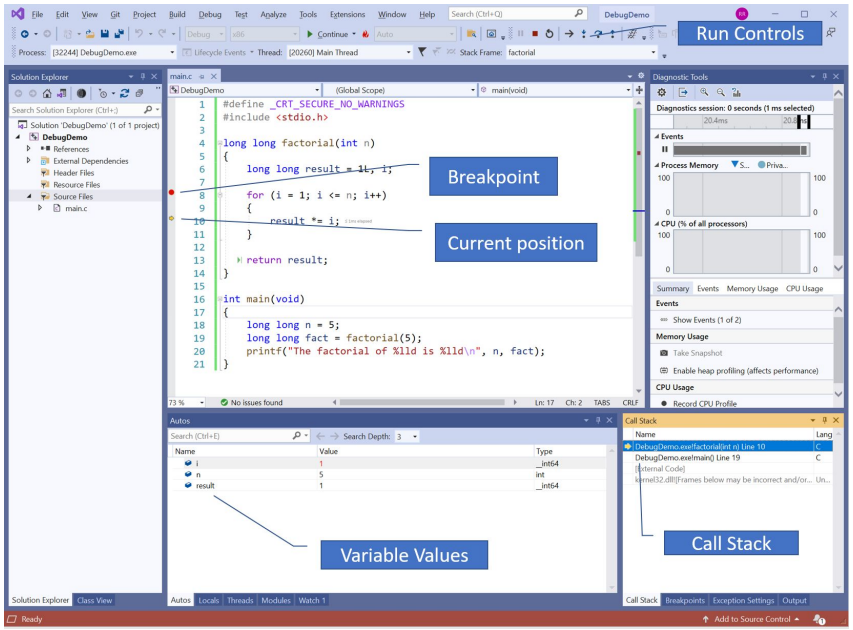
Przydatne opcje kompilatora GCC:

`-Wall`	włącza wiele ostrzeżeń o potencjalnych błędach
`-Wextra`	włącza dodatkowe ostrzeżenia
`-ansi`	ustawia tryb zgodny z ISO C90 (ANSI C89)
`-pedantic`	zgłasza ostrzeżenia dotyczące rozszerzeń wykraczających poza wybrany standard ISO C
`-fanalyzer`	uruchamia analizę statyczną kodu
`-g`	dodaje do pliku wynikowego informacje potrzebne debuggerowi
`-pg`	instrumentuje program na potrzeby profilowania za pomocą gprof
`-lm`	dołącza bibliotekę matematyczną libm podczas linkowania; deklaracje funkcji znajdują się w `math.h`

Przykład:

```
$ gcc -Wall -Wextra -ansi -pedantic -fanalyzer liczba-err.c
```

Debugger w Visual Studio



- punkty przerwania (*breakpoints*)
- krokowanie (*step in*, *step over*, *step out*)
- podgląd wartości zmiennych
- podgląd stosu wywołań

GDB - debuger konsolowy projektu GNU

```
$ gcc -g -o program program.c
$ gdb program
(gdb) run
```

Źródło: ☞ Visual Studio Debugging, Seneca, Software Testing.

Analiza statyczna kodu

- **Analiza statyczna kodu** polega na analizie kodu źródłowego lub skompilowanego bez uruchamiania programu
- **lint, linter** to narzędzie do analizy statycznej
- Narzędzia te wykorzystuje się do automatycznej weryfikacji jakości kodu
- Zawierają reguły i heurystyki wykrywające potencjalne podatności
- Mogą sprawdzać składnię, wykrywać potencjalne luki bezpieczeństwa, weryfikować styl kodu i sugerować usprawnienia wydajności
- Często koncentrują się na wybranym zestawie podatności, dlatego warto stosować kilka rozwiązań
- Mogą generować fałszywie pozytywne ostrzeżenia, czyli błędnie wskazywać na istnienie problemu

Przykładowe narzędzia analizy statycznej dla języka C/C++:

- **cppcheck** - analizator statyczny wykrywający błędy i inne potencjalne problemy.
  zob.: ☞lista testów
  ```
  $ cppcheck --enable=all liczba-err.c
  ```

- **clang-tidy** - wiele testów i bogate możliwości konfiguracji
  zob.: ☞lista testów
  ```
  $ clang-tidy --checks='*'  liczba-err.c
  ```

```
$ cppcheck --enable=all liczba-err.c
Checking liczba-err.c ...
liczba-err.c:18:0: information: Include file: <studio.h> not found.
#include<studio.h>;
^
liczba-err.c:28:13: style: Condition 'n=0' is always false
      if ( n=0 ) printf("To jest zero!\n");
             ^
liczba-err.c:33:14: error: Uninitialized variable: i [uninitvar]
      while( i<n );
              ^
liczba-err.c:36:12: style: Variable 'i' is assigned a value
                                          that is never used.
         i = i + 1;
           ^
```

Narzędzia pomagające dbać o styl kodu w języku C:

- ☞ Artistic Style

```
$ astyle  --style=kr nwd-balagan.c
$ astyle  --style=allman nwd-balagan.c
```

- ☞ ClangFormat

```
$ clang-format --style GNU nwd-balagan.c
```

- narzędzia online: ☞ formatter.org, ☞ Code Beautify

Przykłady wytycznych dotyczących stylu:

- wytyczne GNU ☞ Making The Best Use of C
- ☞ C Code Style Guidelines
- ☞ Google C++ Style Guide

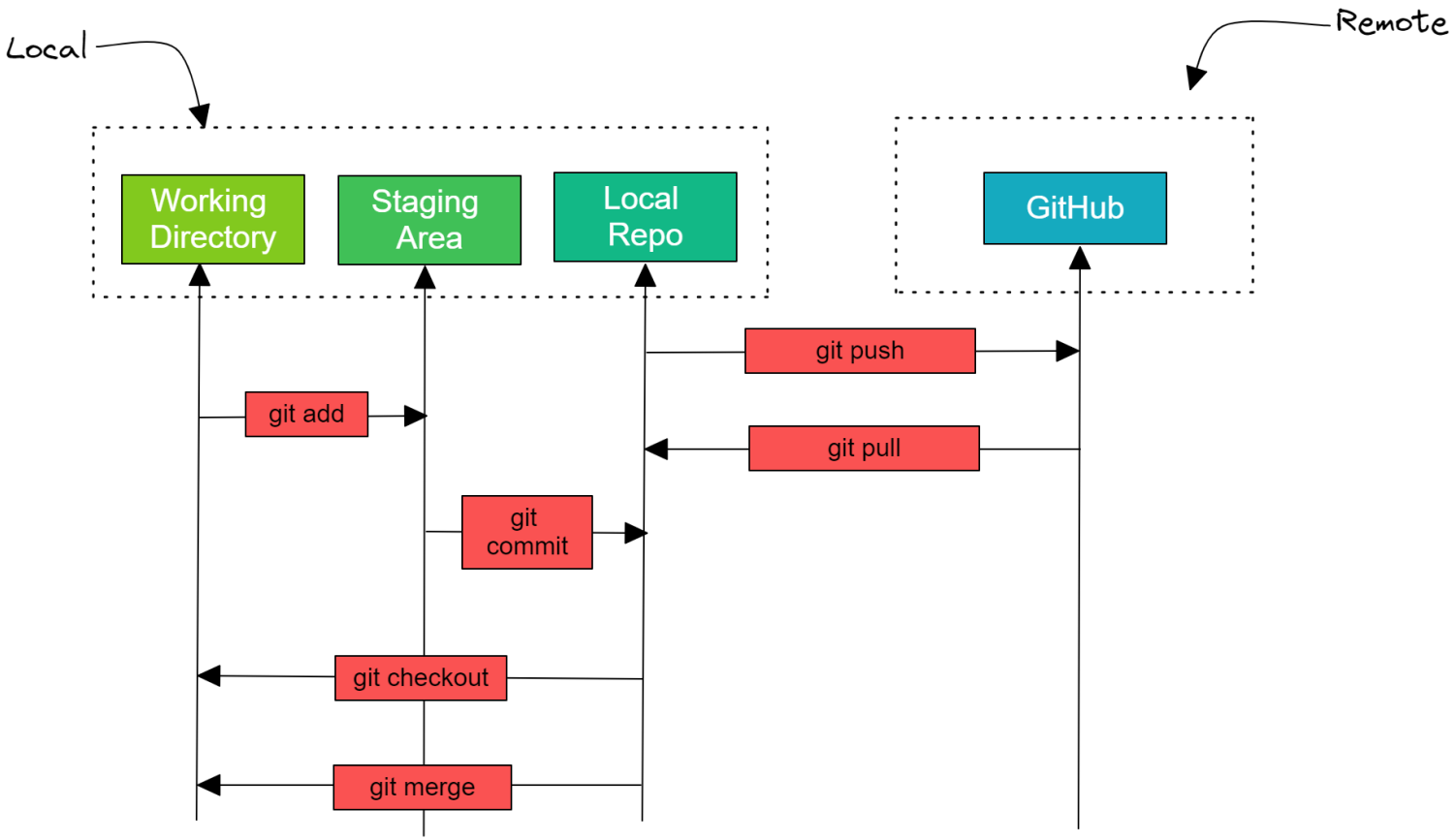
Git to rozproszony system kontroli wersji służący do zarządzania historią zmian w plikach i kodzie źródłowym projektów.

- przechowuje historię zmian w kodzie
- umożliwia współdzielenie kodu w zespołach
- umożliwia tworzenie i scalanie gałęzi
- umożliwia wymianę zmian między repozytoriami
- wiele operacji wykonuje lokalnie, dzięki czemu działa szybko
- jest elastyczny i integruje się z wieloma aplikacjami klienckimi

☞ GitHub - platforma internetowa wspierająca współpracę nad kodem i udostępniająca repozytoria Git.
Inne platformy: ☞ GitLab, ☞ Bitbucket, ☞ Azure DevOps.

`add`	dodanie zmian do indeksu, aby przygotować je do zatwierdzenia; dodanie nowego pliku rozpoczyna jego śledzenie
`commit`	zapisanie przygotowanych zmian w lokalnej historii repozytorium
`push`	wysłanie lokalnych commitów do zdalnego repozytorium
`pull`	pobranie zmian ze zdalnego repozytorium i zintegrowanie ich z bieżącą gałęzią
`merge`	scalenie historii gałęzi
`log`	historia zmian
`diff`	porównywanie zmian w kodzie
`status`	stan repozytorium, w tym indeksu i katalogu roboczego

Etapy pracy z Git



☞ *GitHub and Git Tutorial for Beginners*

Repozytorium z kodami z wykładu:

☞ `https://github.com/IS-UMK/pp_wyklad`

Klonowanie repozytorium:

```
$ git clone https://github.com/IS-UMK/pp_wyklad
```

Historia zmian:

```
$ git log
```

```
commit 01cc8a5dfb066e280ace0059ad05af9bbc951fbf (HEAD -> master)
Author: Marek <grochu@is.umk.pl>
Date:   Tue Oct 1 11:08:55 2024 +0200

      Dodanie euclid1.f euclid1.pas

commit 0552f871c029186cca1784270411ce77aad5e266
Author: Marek <grochu@is.umk.pl>
Date:   Tue Oct 1 10:51:51 2024 +0200

      Aktualizacja README
```

Dodanie pliku:

```
$ git add nowy.c
```

Status repozytorium:

```
$ git status
```

```
Na gałęzi master
Twoja gałąź jest na bieżąco z „origin/master".

Zmiany do złożenia:
  (użyj „git restore --staged <plik>...", aby wycofać)
 nowy plik:       nowy.c
```

Zapamiętanie zmian:

```
$ git commit -m "Opis wprowadzonych zmian"
```

Wypchnięcie zmian do zdalnego repozytorium:

```
$ git push
```

Pobranie i scalenie zmian pochodzących ze zdalnego repozytorium:

```
$ git pull
```

Pytania sprawdzające

1. Do czego służy git? Jaki jest związek z GitHub?
2. Które z wymienionych narzędzi diagnostycznych nie wymagają uruchomienia programu: debugger, profiler, linter?
3. Jak nazywają się narzędzia analizujące wydajność programu na podstawie pomiaru czasu wykonywania instrukcji?