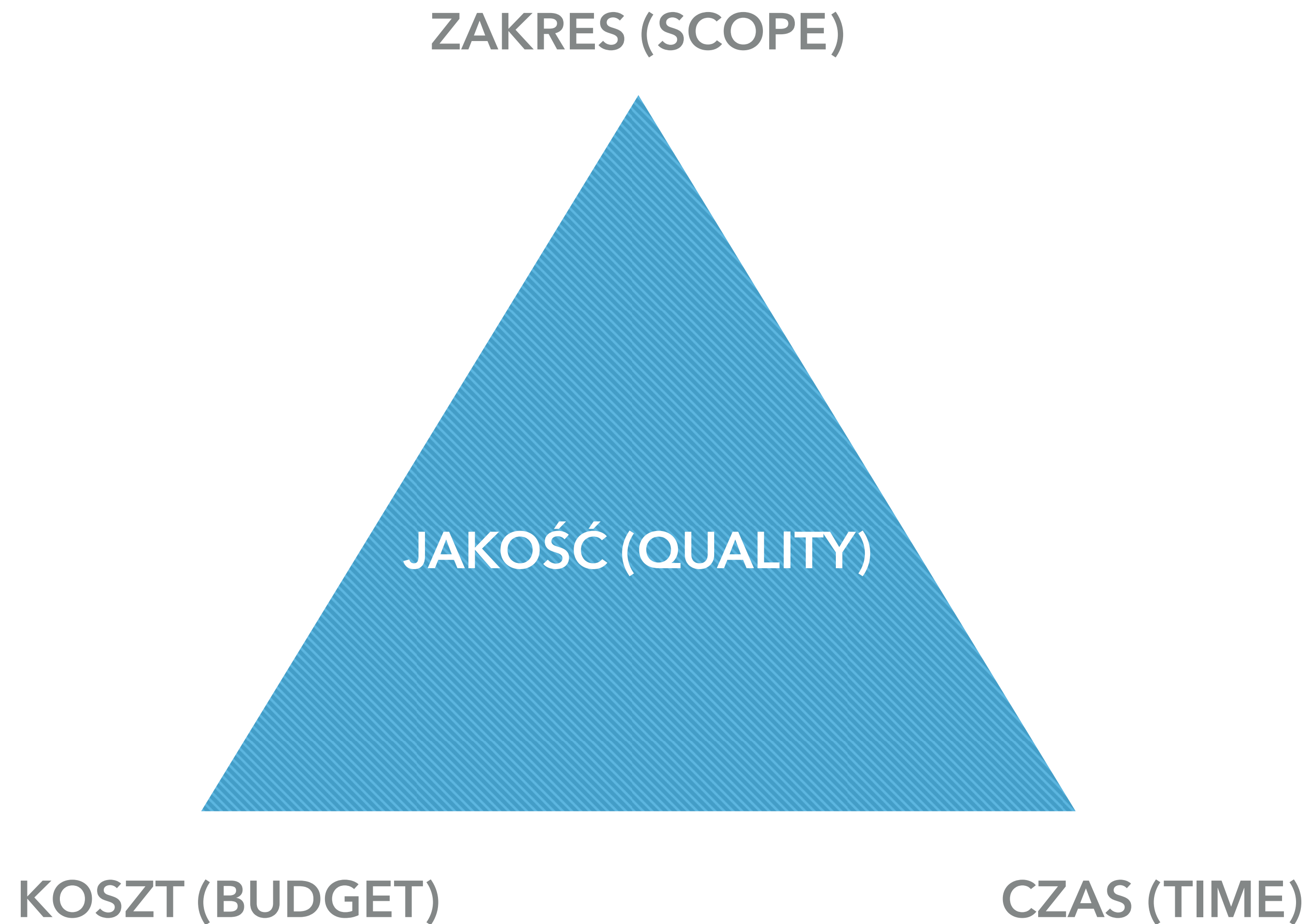


ZARZĄDZANIE CYKLEM ŻYCIA APLIKACJI

TESTOWANIE OPROGRAMOWANIA

TRÓJKĄT OGRANICZEŃ



**TESTOWANIE OPROGRAMOWANIA MOŻE
BYĆ UŻYTE JEDYNNIE W CELU POKAZANIA
OBECNOŚCI BŁĘDÓW, NIGDY ICH BRAKU.**

Edsger Dijkstra

WALIDACJA A WERYFIKACJA

▶ Weryfikacja

- ▶ „Czy budujemy produkt prawidłowo?”
- ▶ Poszukiwanie niezgodności pomiędzy cechami produktu a specyfikacją
- ▶ Weryfikacja statyczna lub dynamiczna

▶ Walidacja

- ▶ „Czy budujemy prawidłowy produkt?”
- ▶ Poszukiwanie błędów w specyfikacji systemu
- ▶ Kwestionujemy i weryfikujemy zasadność poszczególnych funkcji

STATYCZNA A DYNAMICZNA WERYFIKACJA

- ▶ Statyczna
 - ▶ Code review, inspekcja kodu
 - ▶ programy do analizy kodu źródłowego, Lint
- ▶ Dynamiczna
 - ▶ Uruchomienie programu z danymi testowymi i ocena uzyskanych wyników

TESTOWANIE

TYPY I POZIOMY

PODZIAŁ TESTÓW

TYPY

- ▶ Mówią o celu wykonania testu

POZIOMY

- ▶ Mówią o zakresie wykonania testu
- ▶ Występują w kolejnych fazach wytwarzania oprogramowania

TYPY TESTÓW

- ▶ Funkcjonalne
- ▶ Niefunkcjonalne
- ▶ Strukturalne
- ▶ Dotyczące wprowadzanych zmiany
 - ▶ Regresyjne
 - ▶ Retesty
 - ▶ Smoke testy

TESTY FUNKCJONALNE

- ▶ Weryfikacja wymagań funkcjonalnych
 - ▶ F - U - R - P - S
- ▶ Wykonywane w oparciu o wymagania funkcjonalne
 - ▶ User Stories
 - ▶ Przypadki użycia
- ▶ Testy czarnoskrzynkowe (black-box)

TESTY NIEFUNKCJONALNE

- ▶ Weryfikacja wymagań niefunkcjonalnych (URPS)
 - ▶ F - U - R - P - S
- ▶ Wykonywane w oparciu o wymagania pozafunkcjonalne
- ▶ Testy czarnoskrzynkowe (black-box)

TESTY NIEFUNKCJONALNE

- ▶ Wsparcie przeglądarek, systemów, rozdzielczości
 - ▶ <https://vmlpoland.atlassian.net/wiki/spaces/MDTMaster/pages/717717767/7.2+OS+Browser+screen+resolution+support+TBC>
 - ▶ <http://gs.statcounter.com/>

TESTY NIEFUNKCJONALNE

- ▶ Page speed

- ▶ <https://developers.google.com/speed/pagespeed/insights/>

TESTY REGRESYJNE

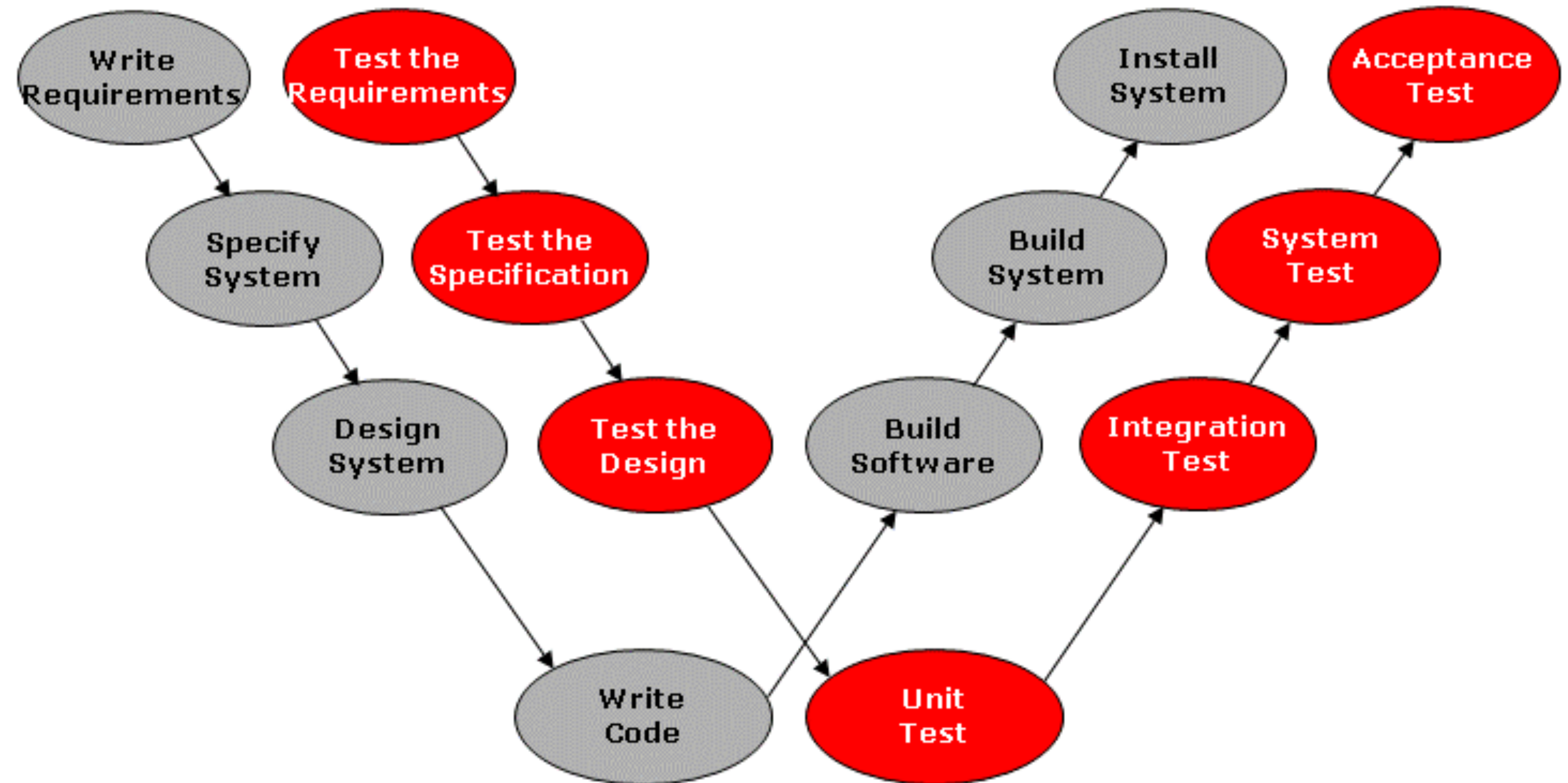
- ▶ Weryfikują działanie przetestowanych elementów systemu po dodaniu nowych elementów lub modyfikacji istniejących
- ▶ Powtarzalne = kandydat do automatyzacji
- ▶ JUnit, Selenium

RETESTY / SMOKE TESTY

- ▶ Retest
 - ▶ Weryfikacja działania komponentu po naprawieniu błędu
- ▶ Smoke test (testy dymne)
 - ▶ Sprawdzenie poprawności działania głównych funkcji systemu

POZIOMY TESTÓW

- ▶ Jednostkowe
- ▶ Integracyjne
- ▶ Systemowe
- ▶ Akceptacyjne



TESTY JEDNOSTKOWE / UNIT TESTY

- ▶ Testowanie najmniejszych jednostek oprogramowania (unitów)
- ▶ Realizowane w izolacji od innych unitów
- ▶ Weryfikacja Corner Case'ów (wartości brzegowych)
- ▶ Biblioteki narzędziowe
 - ▶ Java: JUnit, JMock
 - ▶ C#: NUnit, Moq

TESTY INTEGRACYJNE

- ▶ Weryfikacja integracji między komponentami
- ▶ Może dotyczyć mniejszych jednostek, ale też całych systemów

TESTY SYSTEMOWE

- ▶ Weryfikują scenariusze działania z perspektywy użytkownika
- ▶ Zazwyczaj testy manualne
- ▶ Realizowane na środowiskach DEV, STAGE

TESTY AKCEPTACYJNE

- ▶ Wykonywane w celu akceptacji produktu przez klienta
- ▶ Przeprowadzane na środowisku pre-produkcyjnym (STAGE, UAT) lub produkcyjnym

TESTOWANIE

**DOKUMENTACJA I BUG
TRACKING**

PRZYPADEK TESTOWY (TEST CASE)

Zbiór

- ▶ danych wejściowych
- ▶ wstępnych warunków wykonania
- ▶ oczekiwanych rezultatów
- ▶ końcowych warunków wykonania

opracowany dla warunku testowego.

PRZYPADEK TESTOWY (TEST CASE)

Obowiązkowo:

- ▶ Unikalne ID
- ▶ Nazwa
- ▶ Krótki opis
- ▶ Warunki wstępne
- ▶ Kroki do wykonania
- ▶ Oczekiwany rezultat
- ▶ Warunek końcowy

PRZYPADEK TESTOWY (TEST CASE)

Inne:

- ▶ Dane testowe
- ▶ Środowisko
- ▶ Identyfikator wymagań
- ▶ Priorytet
- ▶ Autor
- ▶ Numer wersji


NARZĘDZIA UTRZYMANIA DOKUMENTACJI


- ▶ Zephyr for Jira <https://marketplace.atlassian.com/vendors/17834/zephyr>
- ▶ TestLink <https://github.com/TestLinkOpenSourceTRMS/testlink-code>

FORMAT RAPORTOWANIA BUGÓW

- ▶ ID / Nazwa
- ▶ Krótkie podsumowanie, opis
- ▶ Środowisko
- ▶ URL
- ▶ Screenshot lub Video
- ▶ Kroki do zreprodukowania
- ▶ Oczekiwany i bieżący rezultat wykonania kroków

Inversed Scroll Indicator action after use of Focus CTA button

 Edit

 Comment

Assign

To Do

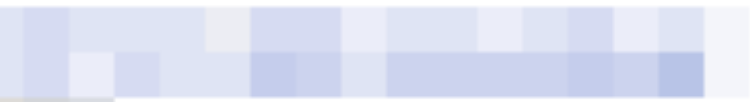
In Progress

Workflow ▾

Type:	 Bug	Status:	TO DO (View workflow)
Priority:	 Medium	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Labels:	None		
Sprint:	JF407 Sprint 4, JF407 Sprint 5, JF407 Sprint 6		

Description

Steps to reproduce:

1. open <http://stage-> 
2. click on scroll indicator
3. click on one of Focus CTA button "Partner with us"
4. scroll up to approx. position shown on attached image
5. click on scroll indicator again

Actual result: after step 5. page is scrolled to the top

Expected result: after step 5. page is scrolled to Focus CTA position



BUG TRACKING TOOLS

- ▶ Jira - <https://pl.atlassian.com/software/jira>
- ▶ Redmine - <https://www.redmine.org/>
- ▶ Trello - <https://trello.com/>
- ▶ Bugzilla - <https://www.bugzilla.org/>

NARZĘDZIOWNIK TESTERA

- ▶ <https://www.telerik.com/download/fiddler> - analiza API
- ▶ <https://www.wireshark.org/> - ruch sieciowy
- ▶ <http://home.snafu.de/tilman/xenulink.html> - analiza linkowań na stronie
- ▶ <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching> - materiały o http
- ▶ <http://testerzy.pl/artykuly> - baza wiedzy

ZWINNE METODYKI

**TDD: TEST DRIVEN
DEVELOPMENT**

TDD: TEST DRIVEN DEVELOPMENT

Technika tworzenia oprogramowania, zaliczana do metodyk zwinnych, stworzona w latach 90-tych przez Kenta Becka.

Polega na powtarzaniu kilku kroków (i od nich nazywana):

- ▶ RED - GREEN - REFACTOR
- ▶ RED - GREEN - CLEAN

KROK 1: RED

- ▶ Programista rozpoczyna kodowanie od napisania testu sprawdzającego dodawaną funkcję.
- ▶ Test na tym etapie powinien zakończyć się niepowodzeniem.

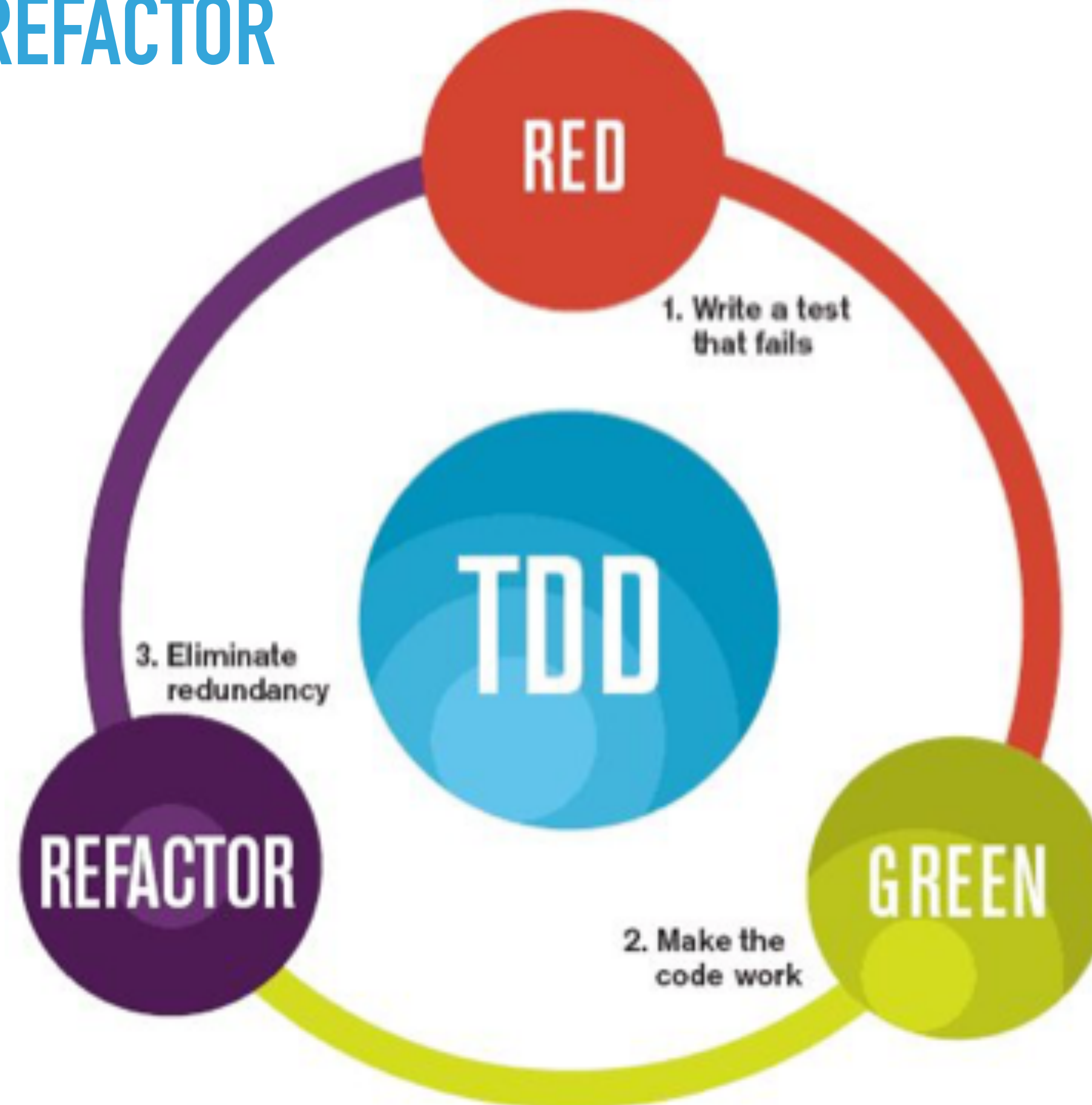
KROK 2: GREEN

- ▶ Po napisaniu testu następuje implementacja funkcji.
- ▶ Po zaimplementowaniu funkcji, napisany uprzednio test powinien przejść.

KROK 3: CLEAN (REFACTOR)

- ▶ W ostatnim kroku programista dokonuje refaktoryzacji napisanego kodu.

TDD: RED – GREEN – REFACTOR



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TDD: TRZY PRAWA

Pierwsze prawo

Nie pisz żadnego kodu produkcyjnego, dopóki nie napiszesz unit testu, który nie przechodzi.

Drugie prawo

Nie pisz więcej testów jednostkowych niż potrzeba, by testy przestały przechodzić.

Trzecie prawo

Pisz tylko tyle kodu produkcyjnego, żeby uprzednio nieprzechodzący test - przeszedł.

TDD: FUNDAMENTALNE ZASADY

- ▶ Myśl ciągle o tym, co zamierzasz zrobić.
- ▶ Stosuj zawsze cykl TDD i 3 prawa TDD.
- ▶ Nigdy nie implementuj nowej funkcji bez wcześniejszego napisania testu, który kończy się niepowodzeniem.
- ▶ Ciągłe dodawaj małe, przyrostowe zmiany.
- ▶ Upewnij się, że system jest ciągle w pełni funkcjonalny.