W poszukiwaniu praw rządzących inżynierią oprogramowania

Skąd się biorą i dlaczego wierzymy, że naprawdę obowiązują

Lech Madeyski
lech.madeyski@pwr.wroc.pl
http://madeyski.e-informatyka.pl/

Instytut Informatyki
Wydział Informatyki i Zarządzania

Politechnika Wrocławska

8 XI 2011, Wrocław

Agenda prezentacji

Agenda prezentacji

Agenda prezentacji

Agenda prezentacji

Tło i cele prezentacji

Tło: Inżynieria oprogramowania (IO) staje się coraz bardziej dojrzałą dziedziną nauki [Wohlin et al., 2000, Shull et al., 2008].

Obiekt rozważań: Prawa rządzące inżynierią oprogramowania (i związane z nimi obserwacje i teorie)

Cele prezentacji:

- przedstawienie wybranych praw na podstawie [Endres and Rombach, 2003],
- przybliżenie relacji jaka występuje pomiędzy obserwacjami, prawami i teoriami IO [Endres and Rombach, 2003].

Tło i cele prezentacji

Tło: Inżynieria oprogramowania (IO) staje się coraz bardziej dojrzałą dziedziną nauki [Wohlin et al., 2000, Shull et al., 2008].

Obiekt rozważań: Prawa rządzące inżynierią oprogramowania (i związane z nimi obserwacje i teorie)

Cele prezentacji:

- przedstawienie wybranych praw na podstawie [Endres and Rombach, 2003],
- przybliżenie relacji jaka występuje pomiędzy obserwacjami, prawami i teoriami IO [Endres and Rombach, 2003].

Tło i cele prezentacji

Tło: Inżynieria oprogramowania (IO) staje się coraz bardziej dojrzałą dziedziną nauki [Wohlin et al., 2000, Shull et al., 2008].

Obiekt rozważań: Prawa rządzące inżynierią oprogramowania (i związane z nimi obserwacje i teorie)

Cele prezentacji:

- przedstawienie wybranych praw na podstawie [Endres and Rombach, 2003],
- przybliżenie relacji jaka występuje pomiędzy obserwacjami, prawami i teoriami IO [Endres and Rombach, 2003].

Prawa a stojące za nimi dowody empiryczne – pierwszy kontrolowany eksperyment medyczny

Przykład zaczerpnięty z [Wilson, 2011]

- Czasy wojny 7-letniej (ang. *Seven Years War*).

- Wielka Brytania straciła:
  - 1512 marynarzy w wyniku starć z nieprzyjacielem,
  - i prawie 100000 marynarzy w wyniku szkorbutu – tej straty można było uniknąć!

Prawa a stojące za nimi dowody empiryczne – pierwszy kontrolowany eksperyment medyczny

Przykład zaczerpnięty z [Wilson, 2011]

- Czasy wojny 7-letniej (ang. *Seven Years War*).

- Wielka Brytania straciła:
  - 1512 marynarzy w wyniku starć z nieprzyjacielem,
  - i prawie 100000 marynarzy w wyniku szkorbutu – tej straty można było uniknąć!

James Lind i pierwszy kontrolowany eksperyment medyczny

- 1747 r.: szkocki lekarz James Lind na *HMS Salisbury* przeprowadza pierwszy w historii kontrolowany eksperyment medyczny.
- 12 marynarzy chorych na szkorbut dzieli na 6 par podając odpowiednio:
  1. vitriol,
  2. cydr,
  3. ocet,
  4. woda morska,
  5. woda Barleya,
  6. cytrusy!
- W 1753 (przed wojną!) Lind publikuje rozprawę doktorską.
- Admiralicja wykorzysta wyniki badań Lind'a dopiero podczas wojen napoleońskich!

James Lind i pierwszy kontrolowany eksperyment medyczny

- 1747 r.: szkocki lekarz James Lind na *HMS Salisbury* przeprowadza pierwszy w historii kontrolowany eksperyment medyczny.
- 12 marynarzy chorych na szkorbut dzieli na 6 par podając odpowiednio:
  1. vitriol,
  2. cydr,
  3. ocet,
  4. woda morska,
  5. woda Barleya,
  6. cytrusy!
- W 1753 (przed wojną!) Lind publikuje rozprawę doktorską.
- Admiralicja wykorzysta wyniki badań Lind'a dopiero podczas wojen napoleońskich!

James Lind i pierwszy kontrolowany eksperyment medyczny

- 1747 r.: szkocki lekarz James Lind na *HMS Salisbury* przeprowadza pierwszy w historii kontrolowany eksperyment medyczny.
- 12 marynarzy chorych na szkorbut dzieli na 6 par podając odpowiednio:
  1. vitriol,
  2. cydr,
  3. ocet,
  4. woda morska,
  5. woda Barleya,
  6. cytrusy!
- W 1753 (przed wojną!) Lind publikuje rozprawę doktorską.
- Admiralicja wykorzysta wyniki badań Lind'a dopiero podczas wojen napoleońskich!

James Lind i pierwszy kontrolowany eksperyment medyczny

- 1747 r.: szkocki lekarz James Lind na *HMS Salisbury* przeprowadza pierwszy w historii kontrolowany eksperyment medyczny.
- 12 marynarzy chorych na szkorbut dzieli na 6 par podając odpowiednio:
  1. vitriol,
  2. cydr,
  3. ocet,
  4. woda morska,
  5. woda Barleya,
  6. cytrusy!
- W 1753 (przed wojną!) Lind publikuje rozprawę doktorską.
- Admiralicja wykorzysta wyniki badań Lind'a dopiero podczas wojen napoleońskich!

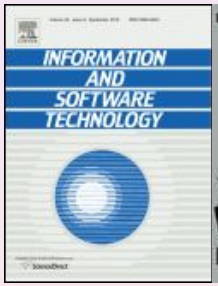
A jak to wygląda w inżynierii oprogramowania?

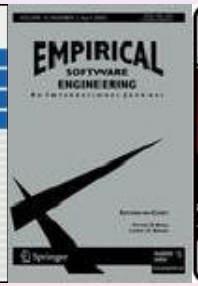
Czy badania empiryczne i eksperymenty są powszechne?

A jak to wygląda w inżynierii oprogramowania?

Czy badania empiryczne i eksperymenty są powszechne?

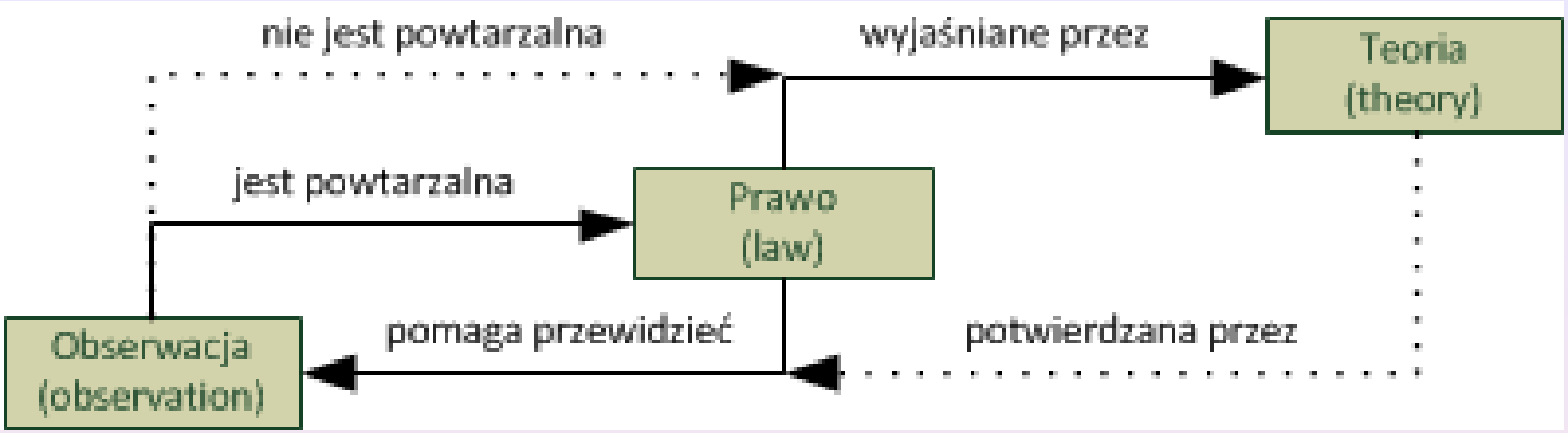
Badania empiryczne, w tym eksperymenty stanowiące systematyczny, zdyscyplinowany, wymierny i kontrolowany sposób oceny nowych rozwiązań [Wohlin et al., 2000] stają się fundamentalną składową badań w IO [Shull et al., 2008]:





Jakość metod ☞ rosnącą dojrzałość dyscypliny naukowej.

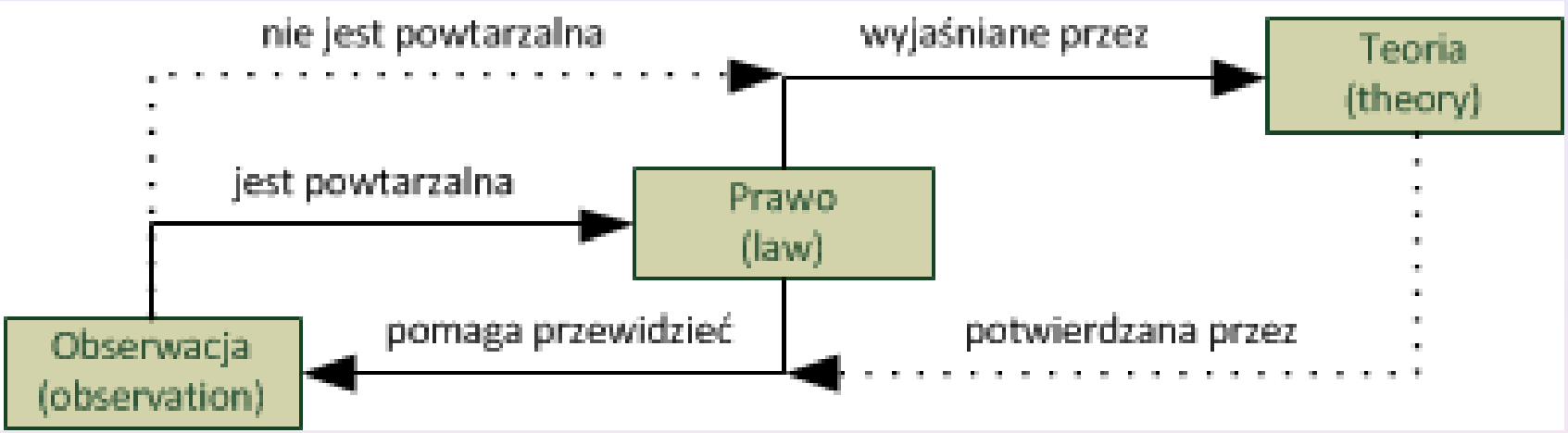
Obserwacje, prawa i teorie



[Endres and Rombach, 2003] przedstawili jak budowana jest wiedza nt. IO definiując 3 kluczowe pojęcia:

1. **Obserwacje** (ang. *observations*)
2. **Prawa** (ang. *laws*)
3. **Teorie** (ang. *theories*)

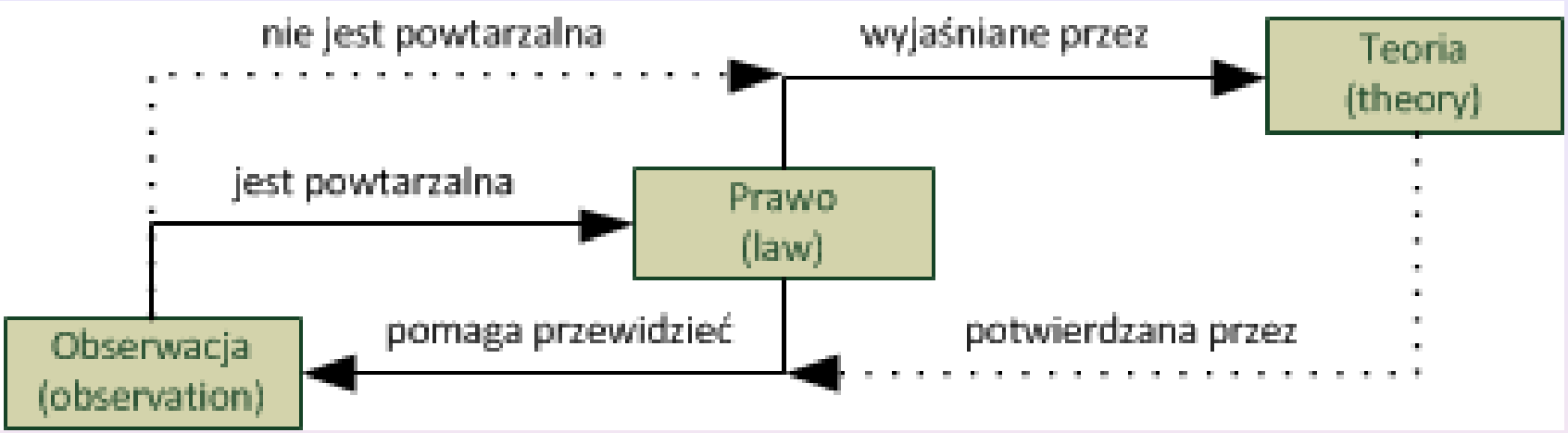
Obserwacje, prawa i teorie



[Endres and Rombach, 2003] przedstawili jak budowana jest wiedza nt. IO definiując 3 kluczowe pojęcia:

1. **Obserwacje** (ang. *observations*)
2. **Prawa** (ang. *laws*)
3. **Teorie** (ang. *theories*)

Obserwacje, prawa i teorie



[Endres and Rombach, 2003] przedstawili jak budowana jest wiedza nt. IO definiując 3 kluczowe pojęcia:

1. **Obserwacje** (ang. *observations*)
2. **Prawa** (ang. *laws*)
3. **Teorie** (ang. *theories*)

Hipotezy i przypuszczenia [Endres and Rombach, 2003]

- Hipoteza – twierdzenie wstępnie, tymczasowo zaakceptowane (*tentatively accepted*) na podstawie wstępnych badań.
- Przypuszczenie (*conjecture*) jest tylko i wyłącznie domysłem (*guess only*).
- Endres i Rombach [Endres and Rombach, 2003] zebrali wiele nieoczywistych praw, hipotez i przypuszczeń w IO.

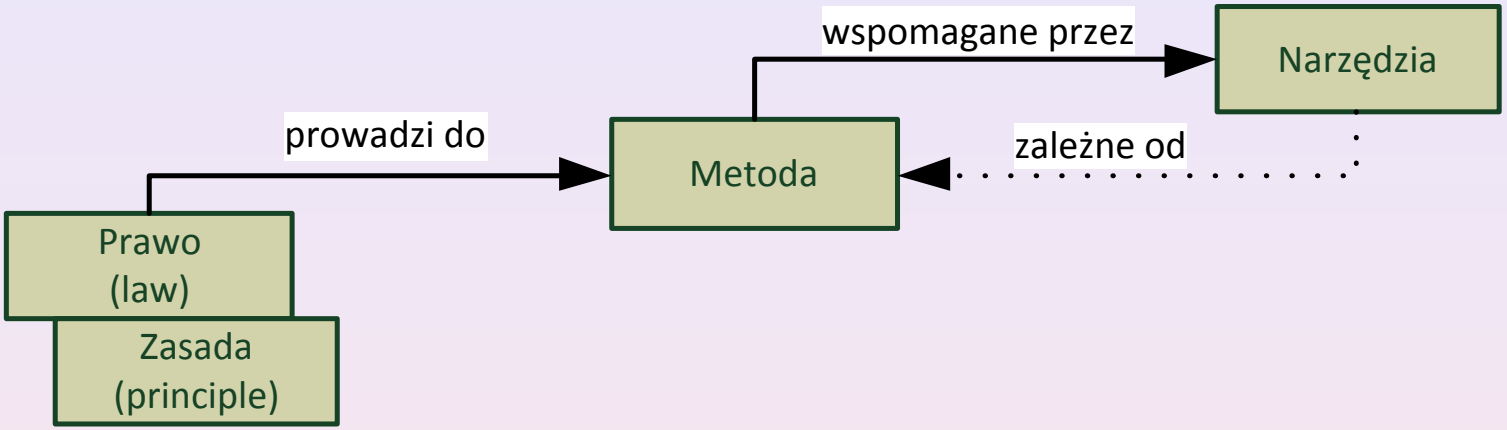
Hipotezy i przypuszczenia [Endres and Rombach, 2003]

- Hipoteza – twierdzenie wstępnie, tymczasowo zaakceptowane (*tentatively accepted*) na podstawie wstępnych badań.
- Przypuszczenie (*conjecture*) jest tylko i wyłącznie domysłem (*guess only*).
- Endres i Rombach [Endres and Rombach, 2003] zebrali wiele nieoczywistych praw, hipotez i przypuszczeń w IO.

Hipotezy i przypuszczenia [Endres and Rombach, 2003]

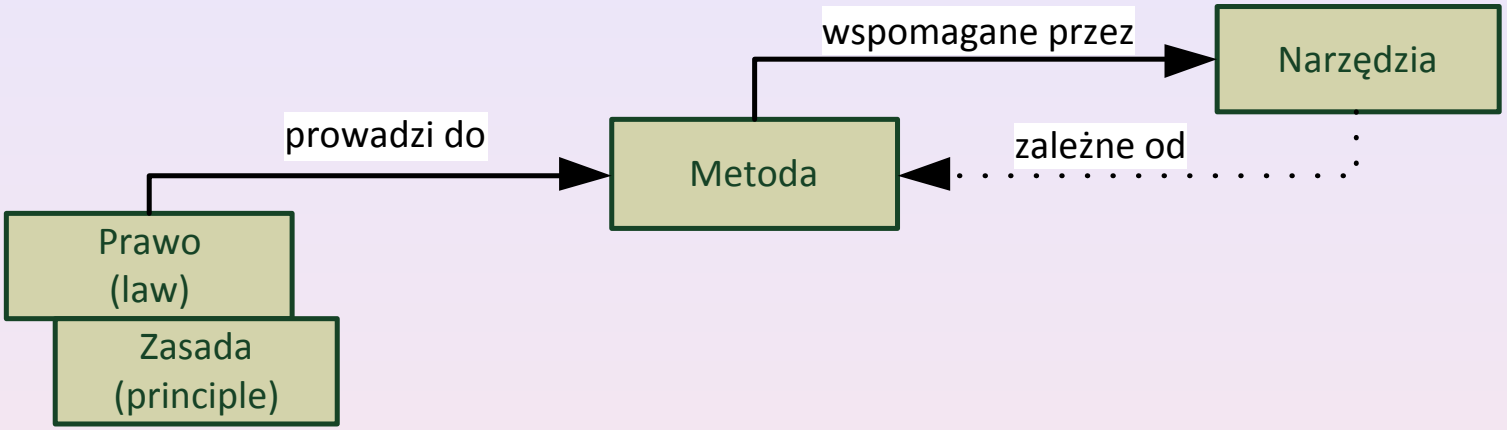
- Hipoteza – twierdzenie wstępnie, tymczasowo zaakceptowane (*tentatively accepted*) na podstawie wstępnych badań.
- Przypuszczenie (*conjecture*) jest tylko i wyłącznie domysłem (*guess only*).
- Endres i Rombach [Endres and Rombach, 2003] zebrali wiele nieoczywistych praw, hipotez i przypuszczeń w IO.

Zasady (*principles*), metody i narzędzia



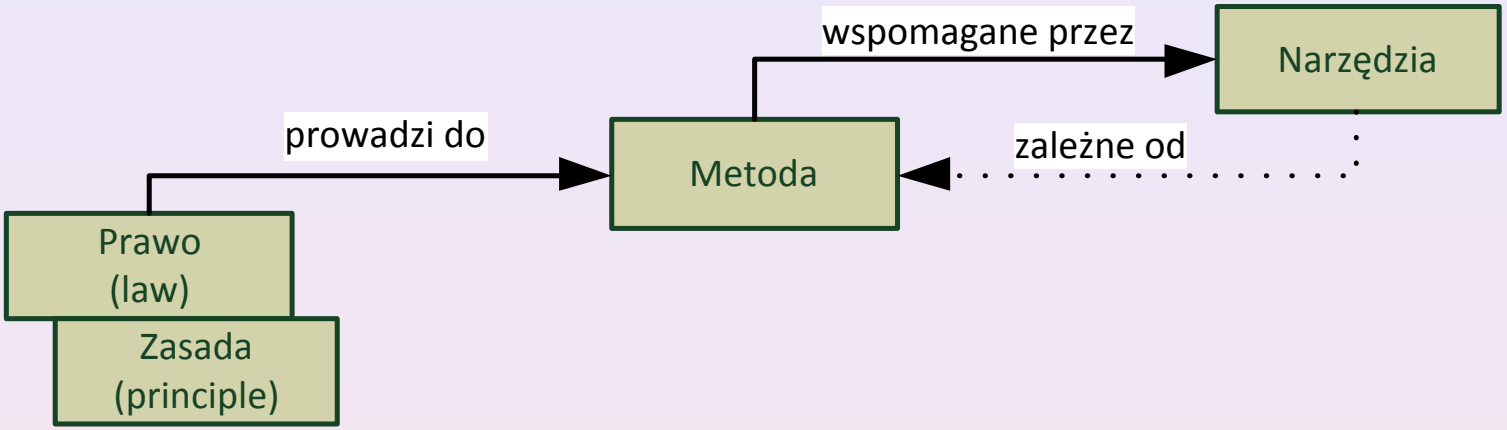
- Nie wszystkie wyniki doświadczeń mogą być nazywane prawami (np. ze względu na ich poziom szczegółowości) → pojęcie **zasady** (ang. *principle*) [Endres and Rombach, 2003].
- Na podstawie praw i zasad formułowane są metody w IO.
- Narzędzia wspomagają nasze postępowanie zgodnie z określoną metodą IO.

Zasady (*principles*), metody i narzędzia



- Nie wszystkie wyniki doświadczeń mogą być nazywane prawami (np. ze względu na ich poziom szczegółowości) → pojęcie **zasady** (ang. *principle*) [Endres and Rombach, 2003].
- Na podstawie praw i zasad formułowane są metody w IO.
- Narzędzia wspomagają nasze postępowanie zgodnie z określoną metodą IO.

Zasady (*principles*), metody i narzędzia



- Nie wszystkie wyniki doświadczeń mogą być nazywane prawami (np. ze względu na ich poziom szczegółowości) → pojęcie **zasady** (ang. *principle*) [Endres and Rombach, 2003].
- Na podstawie praw i zasad formułowane są metody w IO.
- Narzędzia wspomagają nasze postępowanie zgodnie z określoną metodą IO.

Prawo Glass'a (Glass' law): *Mankamenty w zakresie definiowania wymagań są głównym źródłem niepowodzenia projektów*

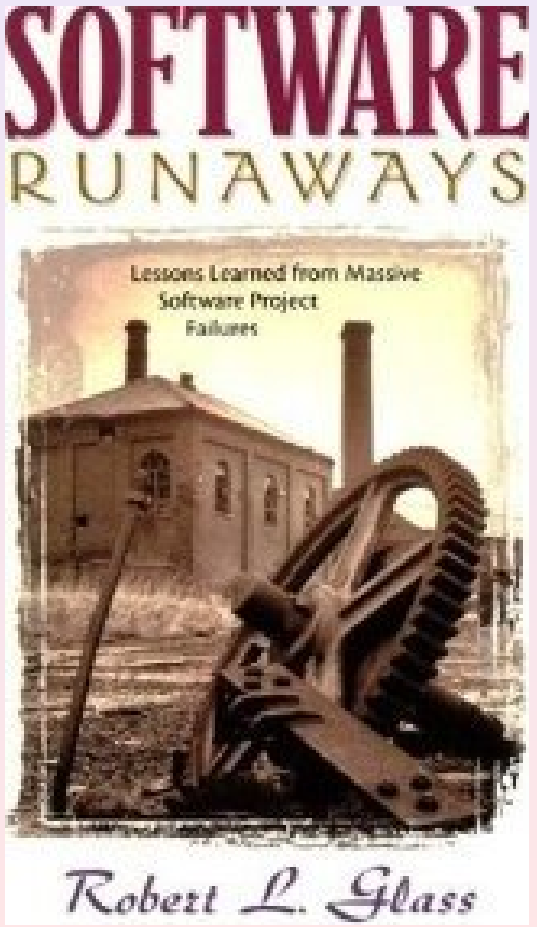
Prawo Glass'a (Glass' law): *Mankamenty w zakresie definiowania wymagań są głównym źródłem niepowodzenia projektów*

Dowody empiryczne: bazują tylko na studiach przypadków

[Glass, 1998] analizował przyczyny niepowodzenia projektów przez kilka dekad i przedstawił studia przypadków wskazując, że:

- **wymagań było o wiele za dużo,**
- **były one niestabilne, dwuznaczne i niekompletne**.

Analizy dotyczyły 3 dużych systemów.



Prawo Glass'a (Glass' law): *Mankamenty w zakresie definiowania wymagań są głównym źródłem niepowodzenia projektów*

Dowody empiryczne: bazują tylko na studiach przypadków

[Hofmann and Lehner, 2001] ankietowali 15 zespołów w 9 firmach. Najlepsze rezultaty osiągnęły zespoły reprezentujące właściwą kombinację:

- wiedzy,
- zasobów i
- procesu wytawarzania oprogramowania.

Prawo Glass'a (Glass' law): *Mankamenty w zakresie definiowania wymagań są głównym źródłem niepowodzenia projektów*

Dowody empiryczne: bazują tylko na studiach przypadków

Próba formułowania teorii:

- Definiowanie wymagań jest trudne ze względu na:
  - Różne potrzeby różnych grup użytkowników i naturalne konflikty interesów,
  - Trudności w nadawaniu priorytetów przy sprzecznych wymaganiach.

Pierwsze prawo Boehm'a: *Błędy pojawiają się najczęściej podczas definiowania wymagań i projektowania a im później zostaną usunięte tym bardziej są kosztowne*

Pierwsze prawo Boehm'a: *Błędy pojawiają się najczęściej podczas definiowania wymagań i projektowania a im później zostaną usunięte tym bardziej są kosztowne*

Dowody empiryczne: bazują tylko na studiach przypadków

[Boehm et al., 1975] analizowali błędy popełnione podczas projektu w firmie TRW:

- Błędy projektowania 64% vs błędy kodowania 36%
- Czas diagnozy i poprawy błędów projektowych 2 razy większy niż błędów kodowania.

Pierwsze prawo Boehm'a: *Błędy pojawiają się najczęściej podczas definiowania wymagań i projektowania a im później zostaną usunięte tym bardziej są kosztowne*

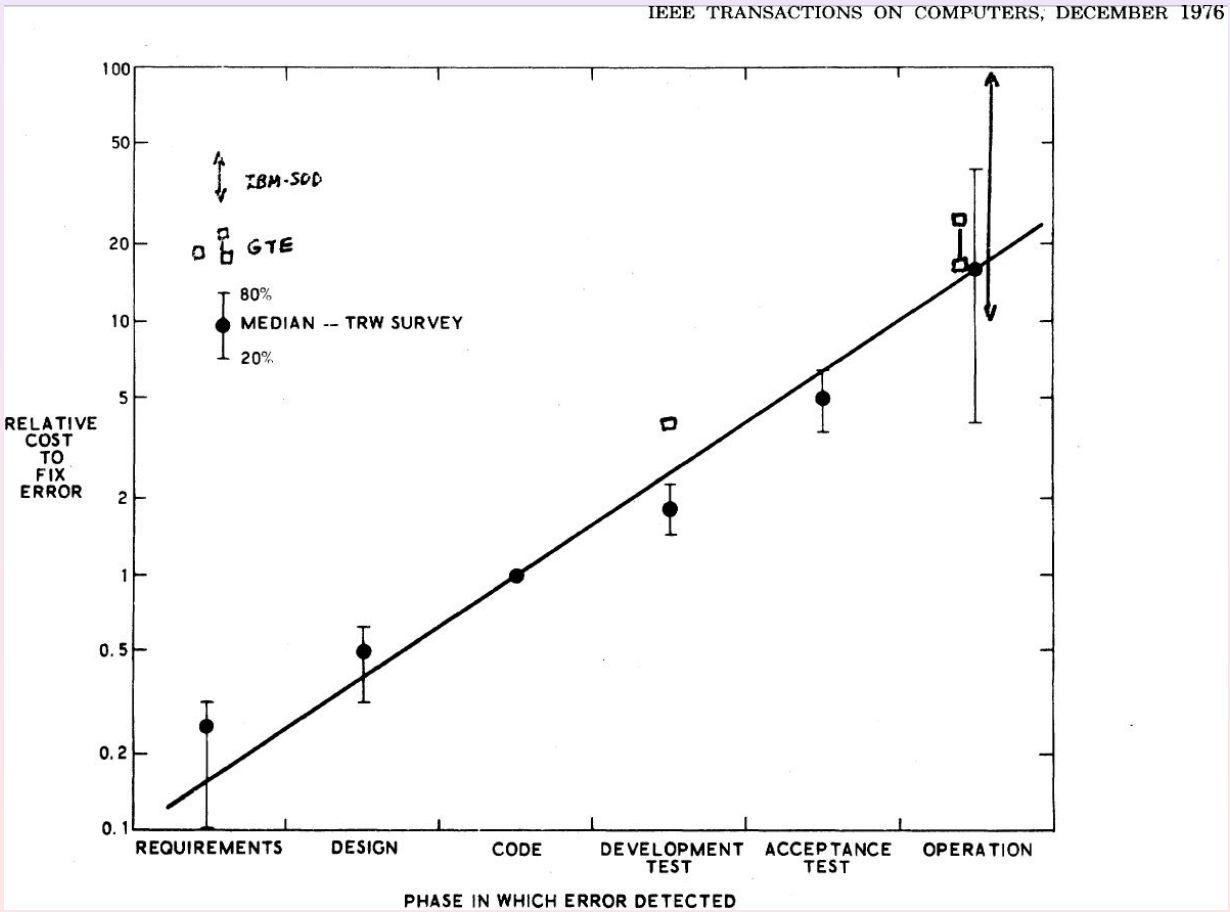
Dowody empiryczne: bazują tylko na studiach przypadków

- [Endres, 1975] – 60-70% błędów w projekcie DOS/VS to błędy podczas definiowania wymagań i projektowania systemu.
- [Basili and Perricone, 1984] potwierdzają te rezultaty dla programów w Fortranie, w przemyśle lotniczym, gdzie projektanci dobrze znali domenę aplikacji.

Pierwsze prawo Boehm'a: *Błędy pojawiają się najczęściej podczas definiowania wymagań i projektowania a im później zostaną usunięte tym bardziej są kosztowne*

Dowody empiryczne: bazują tylko na studiach przypadków

[Hiemann, 1975, Boehm, 1976, Daly, 1977, Boehm, 1981] przedstawiają studia przypadków z IBM, GE, TRW potwierdzając drugą część prawa Boehm'a.



Źródło: IEEE Transaction on Computers

Pierwsze prawo Boehm'a: *Błędy pojawiają się najczęściej podczas definiowania wymagań i projektowania a im później zostaną usunięte tym bardziej są kosztowne*

Dowody empiryczne: bazują tylko na studiach przypadków

Próba formułowania teorii [Endres and Rombach, 2003]:

- Problemy sprawia szczegółowe zgłębianie domeny aplikacji $\rightarrow$ przeoczenia w definiowaniu wymagań i projektowaniu są częstsze niż nieporozumienia podczas kodowania.
- Koszt zmian rośnie z kolejnymi krokami cyklu życia oprogramowania, bo kolejne inwestycje bazują na wcześniejszych decyzjach.

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Dowody empiryczne: kontrolowany eksperyment [Boehm et al., 1984]

Prototypowanie:

- pomost pomiędzy opisem systemu a jego implementacją,
- komunikuje wymagania,
- pomaga podjąć właściwą decyzję,
- pomaga poznać nową technologię, narzędzia.

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Dowody empiryczne: kontrolowany eksperyment [Boehm et al., 1984]

[Boehm et al., 1984] przeprowadzili w środowisku akademickim kontrolowany eksperyment dotyczący użyteczności prototypów:

- 7 zespołów
- mały system (2-4 KLOC).
- 4 zespoły stosowały podejście klasyczne (specyfikacja wymagań i projektu).
- 3 zespoły używały nowego podejścia (zamiast specyfikacji wymagany był wstępny prototyp).

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Dowody empiryczne: kontrolowany eksperyment [Boehm et al., 1984]

Rezultaty eksperymentu:

- Podejście wykorzystujące prototyp prowadziło do mniejszych systemów.
- Jakość systemów oceniono podobnie.

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Dowody empiryczne: kontrolowany eksperyment [Boehm et al., 1984]

Raport [Bernstein, 1993] z AT&T Bell Laboratories mówi o 40% redukcji czasu tworzenia oprogramowania dzięki prototypom, ale też o dwóch pułapkach:

1. Nawet wspaniały prototyp jest tylko prototypem.
2. Pokazanie prototypu podpowiada potencjalnym klientom jak zbudować taki system.

Drugie prawo Boehm'a: *Prototypowanie (istotnie) redukuje błędy specyfikacji wymagań i projektowe, szczególnie w przypadku interfejsów użytkownika*

Dowody empiryczne: kontrolowany eksperyment [Boehm et al., 1984]

Próba formułowania teorii:

- Prototyp oferuje widok systemu w taki sposób w jaki będzie on udostępniony użytkownikowi.
- W odróżnieniu od innych reprezentacji projektu systemu na podstawie prototypu łatwo jest wyobrazić sobie budowany system.

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to* .
Dijkstra: *Testowanie może wykazać obecność błędów, lecz* .
Pareto-Zipf: *Około 80% defektów pochodzi z* .
Corbato: *Produktywność i niezawodność zależy od długości* .
Conway: *System odzwierciedla strukturę organizacyjną,* .
Fagan: *Inspekcje istotnie zwiększają* .
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od* .
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością* .
Brooks: *Dodawanie ludzi do opóźnionych projektów* .

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz .*
Pareto-Zipf: *Około 80% defektów pochodzi z .*
Corbato: *Produktywność i niezawodność zależy od długości .*
Conway: *System odzwierciedla strukturę organizacyjną, .*
Fagan: *Inspekcje istotnie zwiększają .*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od .*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z .*
Corbato: *Produktywność i niezawodność zależy od długości .*
Conway: *System odzwierciedla strukturę organizacyjną, .*
Fagan: *Inspekcje istotnie zwiększają .*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od .*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości* .
Conway: *System odzwierciedla strukturę organizacyjną,* .
Fagan: *Inspekcje istotnie zwiększają* .
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od* .
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością* .
Brooks: *Dodawanie ludzi do opóźnionych projektów* .

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu.*
Conway: *System odzwierciedla strukturę organizacyjną, .*
Fagan: *Inspekcje istotnie zwiększają .*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od .*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu.*
Conway: *System odzwierciedla strukturę organizacyjną, która go tworzy.*
Fagan: *Inspekcje istotnie zwiększają .*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od .*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu.*
Conway: *System odzwierciedla strukturę organizacyjną, która go tworzy.*
Fagan: *Inspekcje istotnie zwiększają produktywność, jakość i stabilność projektu.*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od .*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu.*
Conway: *System odzwierciedla strukturę organizacyjną, która go tworzy.*
Fagan: *Inspekcje istotnie zwiększają produktywność, jakość i stabilność projektu.*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od ich organizacyjnej formy.*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością .*
Brooks: *Dodawanie ludzi do opóźnionych projektów .*

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte*.
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak*.
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów*.
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu*.
Conway: *System odzwierciedla strukturę organizacyjną, która go tworzy*.
Fagan: *Inspekcje istotnie zwiększają produktywność, jakość i stabilność projektu*.
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od ich organizacyjnej formy*.
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością defektów niż duże*.
Brooks: *Dodawanie ludzi do opóźnionych projektów* .

Inne ciekawe **prawa** w telegraficznym skrócie

Parnas: *Bez ryzyka może być zmieniane tylko to co jest ukryte.*
Dijkstra: *Testowanie może wykazać obecność błędów, lecz nie ich brak.*
Pareto-Zipf: *Około 80% defektów pochodzi z 20% modułów.*
Corbato: *Produktywność i niezawodność zależy od długości zapisu tekstu programu.*
Conway: *System odzwierciedla strukturę organizacyjną, która go tworzy.*
Fagan: *Inspekcje istotnie zwiększają produktywność, jakość i stabilność projektu.*
Porter-Votta: *Efektywność inspekcji jest niemal niezależna od ich organizacyjnej formy.*
Basili-Möller: *Małe zmiany charakteryzują się wyższą gęstością defektów niż duże.*
Brooks: *Dodawanie ludzi do opóźnionych projektów jeszcze bardziej je opóźnia.*

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia* .
Booch 2: *Projektowanie zorientowane obiektowo redukuje* .
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje* .
Beck-Fowler: *Metodyki zwinne redukują* .
Mays: *W przypadku błędów prewencja jest lepsza niż* .
Mills-Jones: *Jakość pociąga za sobą* .
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez* .

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje* .
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje* .
Beck-Fowler: *Metodyki zwinne redukują* .
Mays: *W przypadku błędów prewencja jest lepsza niż* .
Mills-Jones: *Jakość pociąga za sobą* .
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez* .

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów*.
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje* .
Beck-Fowler: *Metodyki zwinne redukują* .
Mays: *W przypadku błędów prewencja jest lepsza niż* .
Mills-Jones: *Jakość pociąga za sobą* .
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez* .

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów.*
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów.*
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu.*
Beck-Fowler: *Metodyki zwinne redukują* .
Mays: *W przypadku błędów prewencja jest lepsza niż* .
Mills-Jones: *Jakość pociąga za sobą* .
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez*
.

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów.*
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów.*
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu.*
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań.*
Mays: *W przypadku błędów prewencja jest lepsza niż .*
Mills-Jones: *Jakość pociąga za sobą .*
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż .*
Shaw-Garlan: *Właściwa architektura istotnie redukuje .*
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez .*

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów*.
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu*.
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań*.
Mays: *W przypadku błędów prewencja jest lepsza niż ich usuwanie*.
Mills-Jones: *Jakość pociąga za sobą* .
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez*
.

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów*.
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu*.
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań*.
Mays: *W przypadku błędów prewencja jest lepsza niż ich usuwanie*.
Mills-Jones: *Jakość pociąga za sobą produktywność*.
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż* .
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez*
.

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów*.
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu*.
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań*.
Mays: *W przypadku błędów prewencja jest lepsza niż ich usuwanie*.
Mills-Jones: *Jakość pociąga za sobą produktywność*.
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż większość innych podejść*.
Shaw-Garlan: *Właściwa architektura istotnie redukuje* .
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez*
.

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów.*
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów.*
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu.*
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań.*
Mays: *W przypadku błędów prewencja jest lepsza niż ich usuwanie.*
Mills-Jones: *Jakość pociąga za sobą produktywność.*
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż większość innych podejść.*
Shaw-Garlan: *Właściwa architektura istotnie redukuje koszty tworzenia i utrzymania systemu.*
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez*

.

Ciekawe **hipotezy** w telegraficznym skrócie

Gamma: *Użycie wzorców projektowych ułatwia utrzymanie (ang. maintenance) systemów*.
Booch 2: *Projektowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie projektów*.
Dahl-Goldberg: *Programowanie zorientowane obiektowo redukuje błędy i promuje ponowne użycie kodu*.
Beck-Fowler: *Metodyki zwinne redukują wpływ zmian wymagań*.
Mays: *W przypadku błędów prewencja jest lepsza niż ich usuwanie*.
Mills-Jones: *Jakość pociąga za sobą produktywność*.
Hamlet: *Testowanie bazujące na podejrzeniach może być bardziej efektywne niż większość innych podejść*.
Shaw-Garlan: *Właściwa architektura istotnie redukuje koszty tworzenia i utrzymania systemu*.
Boehm: *Ryzyka projektu mogą być odsunięte lub złagodzone przez wczesne zmierzenie się z nimi*.

Podsumowanie

Cele postawione przez Rombach'a [Rombach, 2011b] podczas sympozjum na cześć Barry'ego Boehm'a:

- Empiryczna ocena musi być integralną częścią dyscypliny IO a rezultaty badawcze muszą zawierać dowody empiryczne.
- Badania muszą być powtarzane (by były godne zaufania) i różnorodne (by poszerzać ich zakres).
- Indywidualne dowody empiryczne (obserwacje) powinny być agregowane, by odkrywać prawa rządzące w IO.
- Panele ekspertów mogą decydować o uznaniu dowodów empirycznych za prawo.

Dziękuję za uwagę.

Lech Madeyski
http://madeyski.e-informatyka.pl/

Prezentacja złożona w systemie LaTeX z wykorzystaniem pakietu Beamer

Obserwacje, prawa i teorie – rozróżnienie wg [Rombach, 2011a]

Obserwacja:

- Rezultat jednego badania w jednym kontekście.
- Nie daje gwarancji stabilności rezultatów jakościowych i ilościowych.

Prawo:

- Rezultat reprezentatywnego zbioru badań w pewnym kontekście.
- Jakościowa stabilność, ilościowa zmienność.

Teoria:

- Rezultat reprezentatywnego zbioru badań w pewnym kontekście (wszystkie ważne zmienne niezależne zidentyfikowane).
- Jakościowa stabilność, ilościowa stabilność (mała zmienność).

V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27:41–52, 1984.

L. Bernstein. Get the design right [software prototyping]. *IEEE Software*, 10(5):61–63, sep 1993. ISSN 0740-7459. doi: 10.1109/52.232402.

B. W. Boehm. Software engineering. *IEEE Transactions on Computers*, 25: 1226–1241, 1976.

B. W. Boehm. *Software Engineering Economics*. Prentice-Hal, 1981.

B. W. Boehm, R. K. McClean, and D. B. Urfrig. Some experience with automated aids to the design of large-scale reliable software. *IEEE Transactions on Software Engineering*, 1:125–133, 1975.

B. W. Boehm, T. E. Gray, and T. Seewaldt. Prototyping versus specifying: a multiproject experiment. *IEEE Transactions on Software Engineering*, 10:290–302, May 1984. ISSN 0098-5589. doi: 10.1109/TSE.1984.5010238. URL http://dx.doi.org/10.1109/TSE.1984.5010238.

E. B. Daly. Management of software development. *IEEE Transactions on Software Engineering*, 3:229–242, May 1977. ISSN 0098-5589. doi: 10.1109/TSE.1977.231132. URL http://dx.doi.org/10.1109/TSE.1977.231132.

A. Endres. An analysis of errors in their causes in system programs. *IEEE Transactions on Software Engineering*, 1:113–120, 1975.

A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering*. Addison-Wesley, 2003. ISBN 978-0321154200.

R. L. Glass. *Software Runaways. Lessons Learned from Massice Software Project Failures*. Prentice Hall, 1998.

P. Hiemann. A new look at the program development process. In C. Hackl, editor, *Programming Methodology*, volume 23 of *Lecture Notes in Computer Science*, pages 11–37. Springer Berlin / Heidelberg, 1975. doi: 10.1007/3-540-07131-8. URL http://dx.doi.org/10.1007/3-540-07131-8.

H. F. Hofmann and F. Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, 18:58–66, July 2001. ISSN 0740-7459. doi: 10.1109/MS.2001.936219. URL http://dx.doi.org/10.1109/MS.2001.936219.

D. Rombach. Towards a science of software engineering: Empirical models capturing cognitive laws, 2011a. URL http://research.microsoft.com/en-us/um/redmond/events/ss2011/slides/thursday/dieter_rombach.pdf. Microsoft Research Software Summit, Paris, 13-15. April 2011.

D. Rombach. Empirical software engineering models: Can they become the equivalent of physical laws in traditional engineering, 2011b. URL http://itechs.iscas.ac.cn/events/BoehmSymposium/Dieter%20Rombach.pdf. Symposium in Honor of Barry W Boehm, Beijing, China, 26-27 April 2011.

F. Shull, J. Singer, and D. I. K. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer, London, UK, 2008.

G. Wilson. Software engineering — empirical results, 2011. URL http://software-carpentry.org/4_0/softeng/ebse/.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 0-7923-8682-5.