Warszawa, 21 listopad 2011

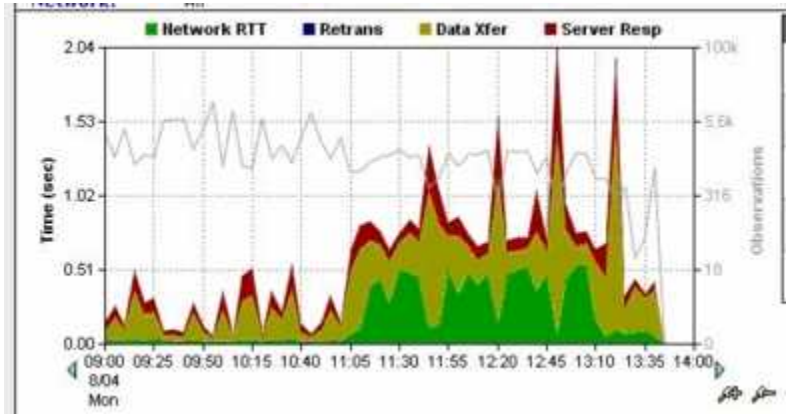
Cele prezentacji

- Zwiększenie świadomości dotyczącej jakości aplikacji
- Łatwiejsze określenie potencjalnych problemów z wydajnością/jakością
- Możliwość przewidzenia problemów przed ich wystąpieniem
- Możliwość wykorzystania sprawdzonych rozwiązań

Agenda

- Jak monitorować aplikację
- Po co nam testy?
- Domain Driven Development
- Interakcja z klientem
- Skalowanie komponentów aplikacyjnych
- Cloud Computing i okolice
- Rozwiązywanie problemów z aplikacja

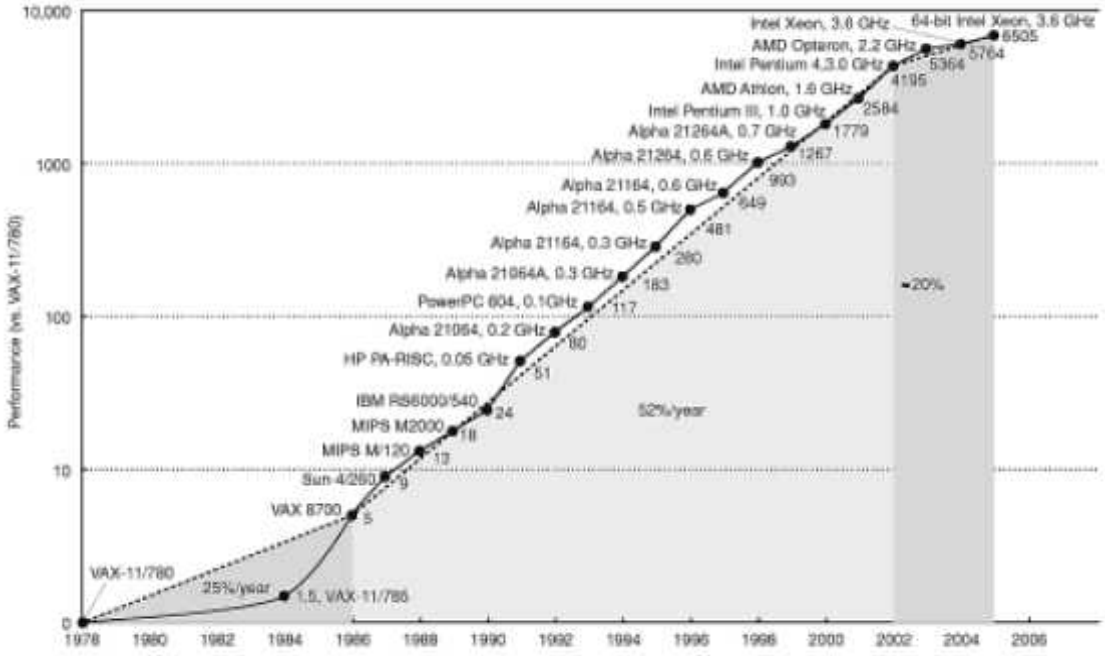
Kiedy aplikacja działa wolno?



Time	12:05 PM	12:10 PM	12:15 PM	12:20 PM	12:25 PM	12:30 PM	12:35 PM	Total 30 Mi Performance
Number of Seats Active	275,002	259,755	244,532	229,298	254,321	263,325	212,431	1,738,664
Seats without Reponse in 30 Seconds	1,003	369	1,589	4,800	320	4,350	850	13,281
Actual Non Performance	0.36%	0.14%	0.65%	2.09%	0.13%	1.65%	0.40%	0.76%
SLA Non-Performance Standard	0.50%	0.50%	0.50%	0.50%	0.50%	0.50%	0.50%	0.50%
Performance (Green=OK)	OK	OK	OK		OK		OK	OK

Jak sobie radzić z problemami z wydajnością aplikacji?

- 2 metody, które (już nie) działają
  - BHW – Buy HardWare
  - BMHW – Buy More HardWare
- Historia rozwoju CPU
  - CISC
  - Częstotliwość
  - Wielordzeniowość
- Teraz trzeba pisać lepsze aplikacje ☺

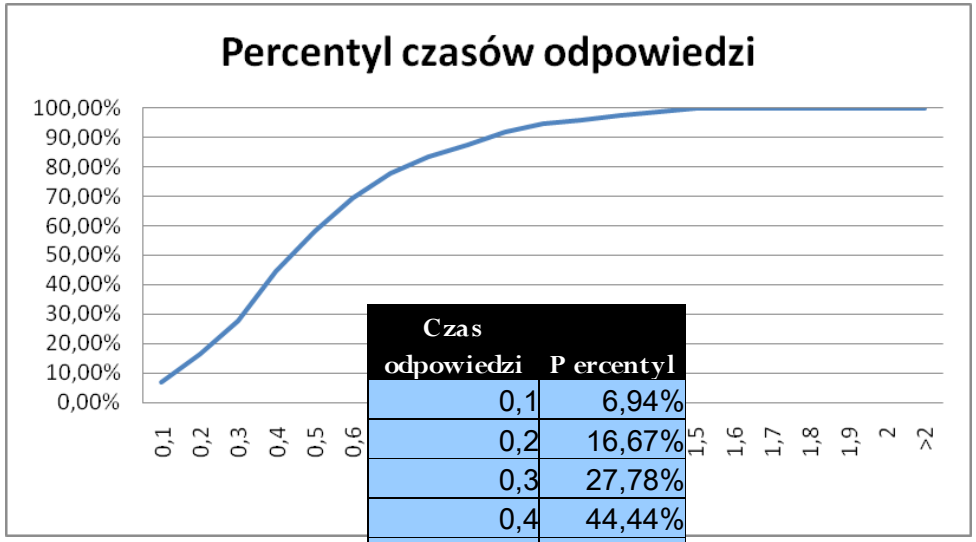


Memory Wall

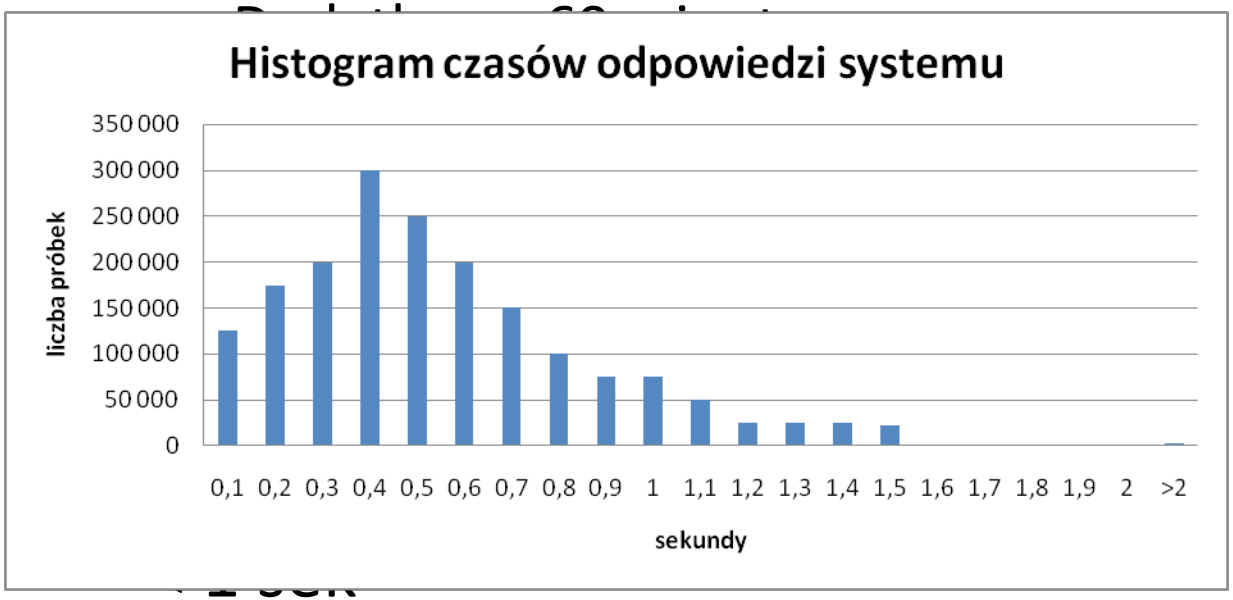
- Problem przewidziany w połowie lat 90
  - "Hitting the Memory Wall: Implications of the Obvious" - Wm. A. Wolf, Sally A. McKey, Computer Science Report No. CS-94-48, December 1994
- Przyczyna
  - Częstotliwość i moc procesorów rozwija się szybciej niż przepustowość i latency pamięci
- Efekt
  - „dobre lata 90" – aplikacja wykorzystuje ok. 50% RAM
  - XXI wiek – aplikacja wykorzystuje ok. 10% RAM

Testy, testy, testy…

- Wymagania niefunkcjonalne dotyczące testów
  - Czas trwania: 180 min

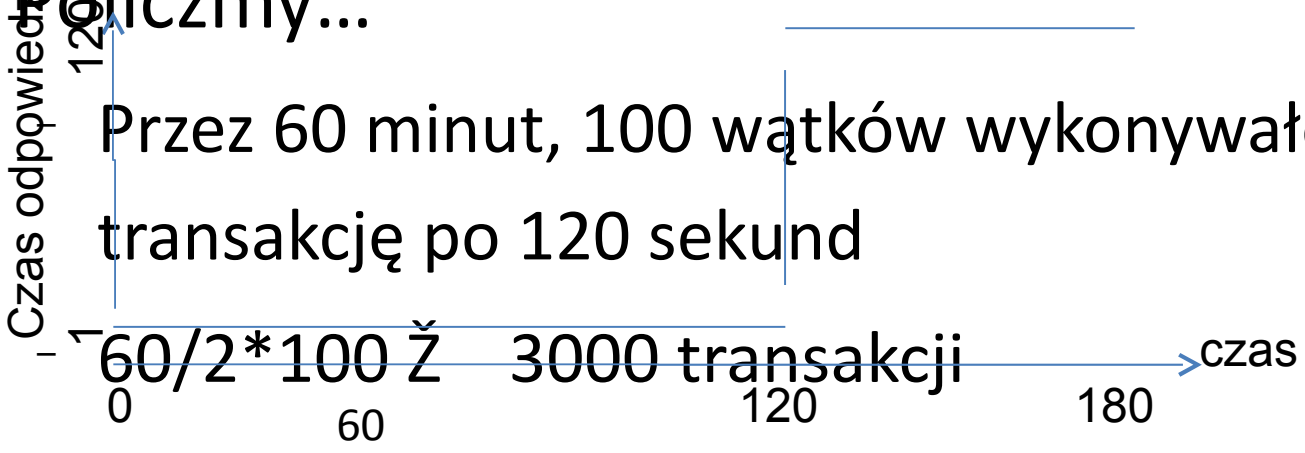


Czas odpowiedzi	P ercentyl
0,1	6,94%
0,2	16,67%
0,3	27,78%
0,4	44,44%
0,5	58,33%
>2	100,00%



Co poszło nie tak?

- Czasy odpowiedzi
  - Średnia – 0,75 sek
  - Mediana – 0,4 sek
- Zachowanie systemu
- Policzmy…
  - Przez 60 minut, 100 wątków wykonywało transakcję po 120 sekund
  - 60/2*100 Ž 3000 transakcji
  - Przez pierwsze 120 minut, 100 wątków



Czas odpowiedzi	Liczba próbek
0,1	125000
0,2	175000
0,3	200000
0,4	300000
0,5	250000
0,6	200000
0,7	150000
0,8	100000
0,9	75000
1	75000
1,1	50000
1,2	25000
1,3	25000
1,4	25000
1,5	22000
1,6	0
1,7	0
1,8	0
1,9	0
2	0
>2	3000

Test Driven Development

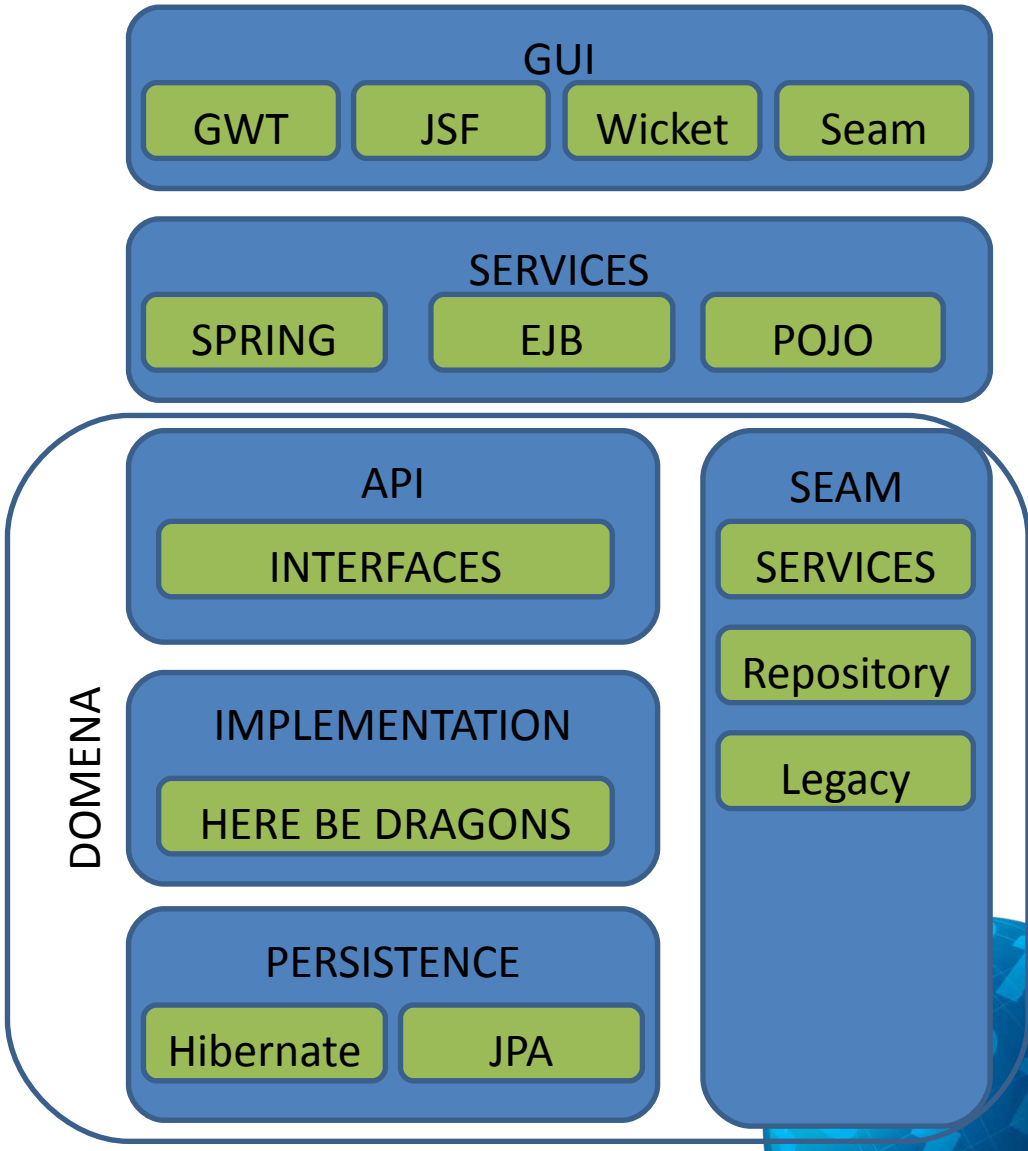
- Ankieta…
  - Ilu z Was robi testy jednostkowe?
  - Jaki procent kodu pokryty jest testami
    - 90%
    - 75%
    - 50%
  - Ilu z Was robi automatyczne testy GUI?
  - Ilu z Was używa ciągłej integracji?
- Po co nam testy?
  - Umożliwiają wykrycie błędów
  - Umożliwiają weryfikację kontraktów API

Wstęp od Domain Driven Architecture

- O czym należy zapomnieć
  - Projektowanie bottom-up zaczynając od modelu bazy danych
  - Umieszczanie logiki w oddzielnej warstwie (EJB)
  - Tworzenie prostych obiektów POJO
- O czym należy jednak pamiętać
  - W końcu i tak użyjemy bazy danych
  - Wykorzystujmy jak najwięcej wzorców projektowych
  - Testy, testy, testy

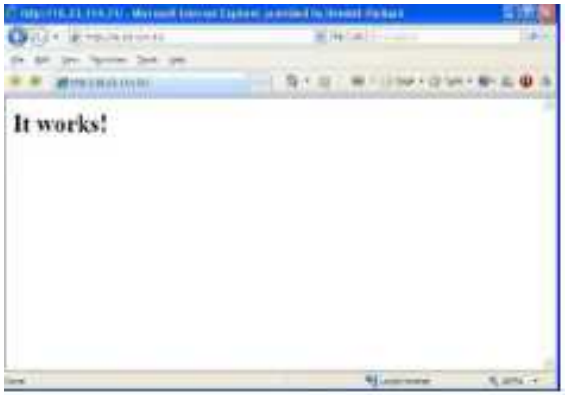
Domain Driven Design

- API
  - Interfejsy obiektów
  - Interfejsy usług
- Implementacja
  - Kod aplikacji
- Persystencja
  - Mapowania
  - Queries
- „Spinacz”
  - Wystawia domenę na



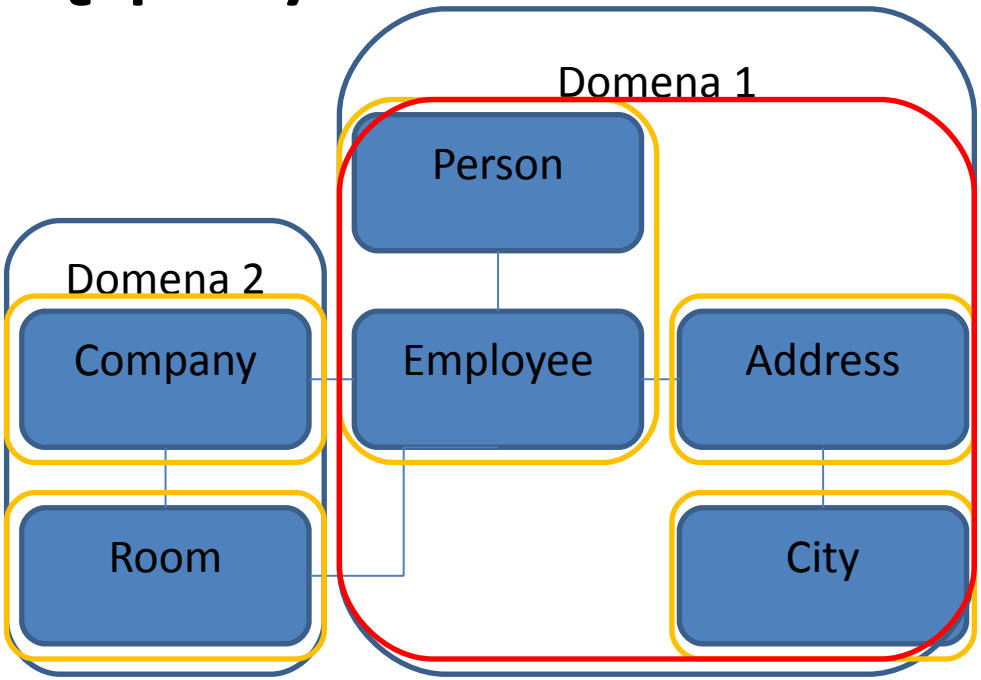
Wzorce projektowe i frameworki

- Rekomendowane
  - Domain Object (DO)
  - Data Transfer Object (DTO)
  - DTO Assembler
  - Repository
  - Generic DAO
  - Command
- Nierekomendowane
  - Anemic domain objects
  - Repetitive DAO
  - Fat Service Layer



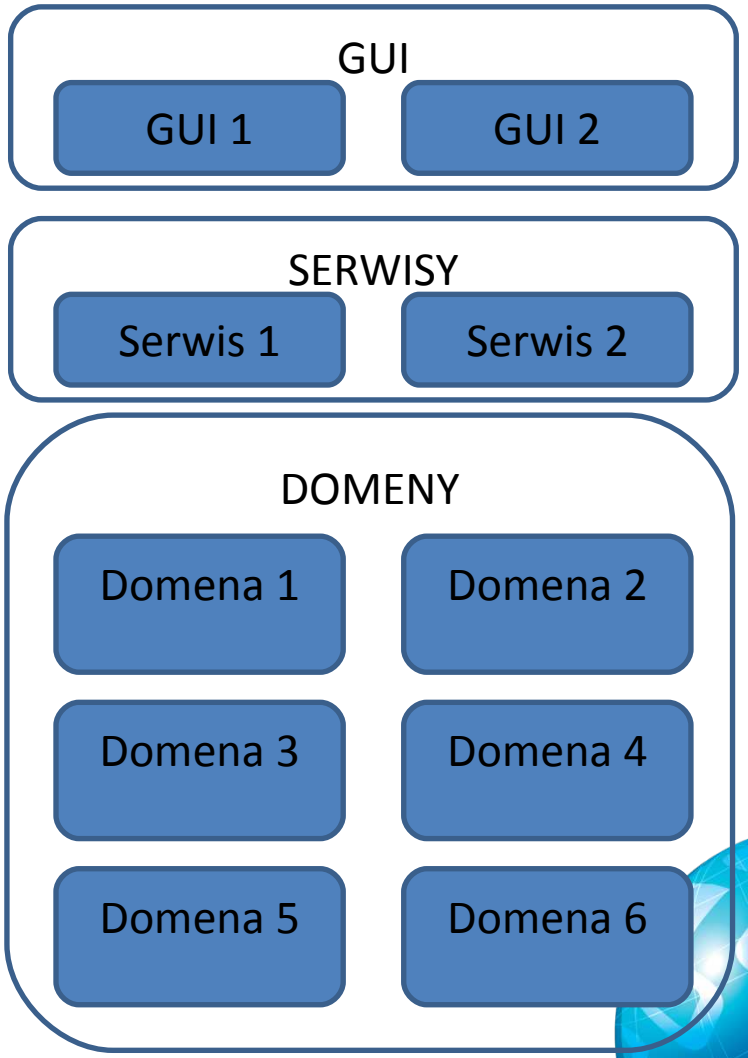
Trochę przykładów

- Jakie domeny można wydzielić?
- Mapowanie obiektów
  - Ile tabel zrobić?
  - Pliki XML czy anotacje?
- Repozytoria
  - RoomRepository
    - getAllRooms(Company)
    - findEmployeeRoom(Employee)
  - EmployeeRepository
    - getAllEmployees(Room)



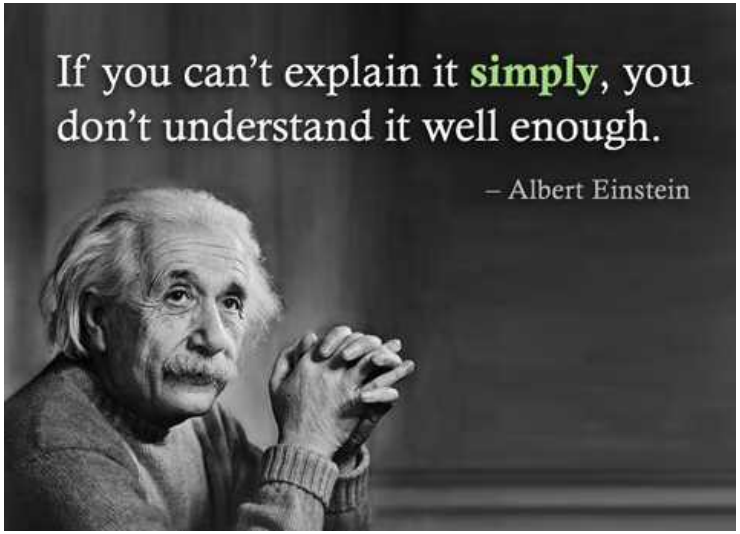
Duże projekty w DDD

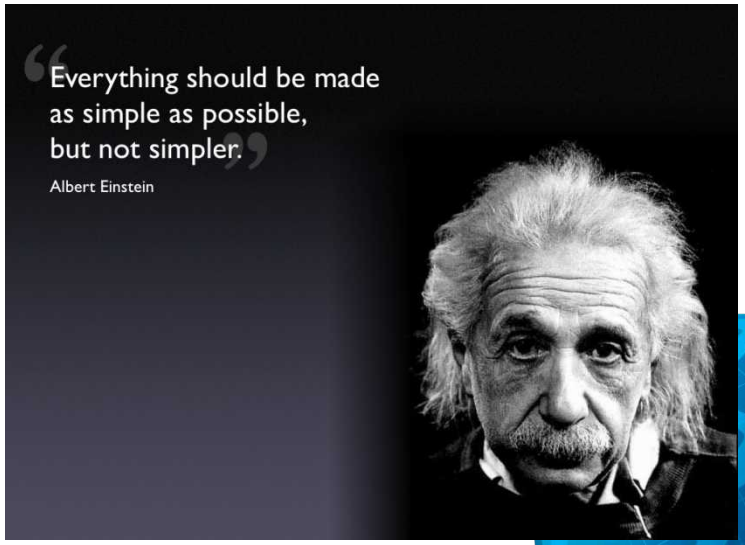
- Pamiętaj
  - Odpowiednio podziel API
  - Wypracuj stabilne API na początku projektu
  - Rób częste iteracje
  - Rób tyle testów ile się da (trust no one)
- Możliwość wymiany
  - Elementu domeny
  - Całej domeny
- Im więcej domen tym



Jak to wszystko podzielić?

- Typowe problemy i rozwiązania
  - Relacje dwukierunkowe
    - Ž Relacja jednokierunkowa i repozytorium
  - Zależności między obiektami
    - Ž Używanie Mock'ów
    - Ž Enkapsulacja logiki wewnątrz domeny
    - Ž Zaślepianie zależności
  - Transakcje pomiędzy domenami





Wpływ błędów w zależności od warstwy

- API
  - Bardzo bolesne i pracochłonne
  - Analogiczny problem jak zły projekt w modelu wodospadowym
  - Jak minimalizować skutki
    - Upewnij się, że jest dobra komunikacja z klientem
    - Tworzyć testy jednostkowe w innych warstwach
- Implementacja
  - Relatywnie łatwe do wykrycia i naprawienia
  - Jak minimalizować skutki
    - Testy, testy, testy

Najsłabsze ogniwo… klient

- Wymagania zawsze będą się zmieniać
  - Przeważnie można to przewidzieć
- Pracuj z klientem na prototypach
  - Wizualnych
  - Programistycznych
- Rób częste iteracje (2-4 tygodnie)
  - Szybko odkryjesz nieścisłości
  - Wymusisz dostarczanie „deliverables”
  - Klient będzie miał wrażenie, że coś się dzieje

Gdzie szukać problemów

Architektura rozwiązania	Infrastruktura	Tuning, monitoring i konfiguracja	Narzędzia do optymalizacji	Przykładowe Produkty
Aplikacja	Obiekty	Aplikacji	Application Performance Monitoring	CA Wily, dynaTrace, **JenniferSoft**
Serwer aplikacyjny	Framework	Serwera aplikacji	Infrastructure Monitoring	IBM Tivoli, HP OpenView
DB DB DB DB	SQL DML	Zapytań SQL, indeksów	Database Performance Monitoring	Quest
DBMS	SQL DDL, DCL	Serwera bazy danych	Database Monitoring	Oracle Resource Manager
System operacyjny	Procesy, zasoby	Systemu operacyjnego	Real time Optimization	**MoreVRP**
CPU I/O	Sprzęt	Rozbudowa sprzętu	Hardware Monitoring	HP SIM, IBM Director

Jak rozwiązywać problemy

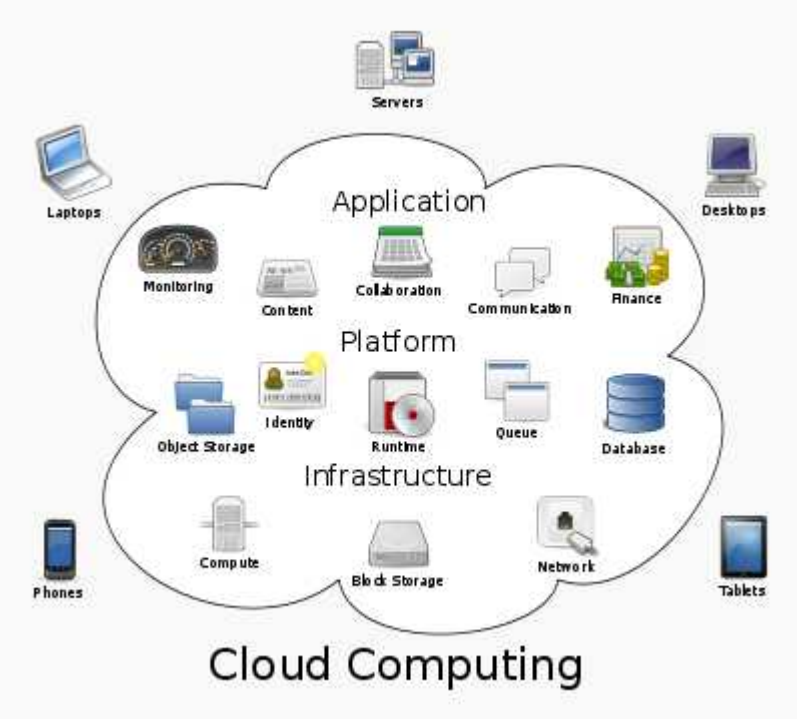
- Testy wydajnościowe powinny zidentyfikować ok. 75% problemów przed wdrożeniem
- Dodanie warstwy cache powinno być możliwe na każdym poziomie
- Implementacja nie powinna zależeć od technologii

Cloud i Grid - definicje

- Compute Grid
  - Równoległe i rozproszone przetwarzanie
- Data Grid
  - Równoległy i rozproszony storage
- Grid Computing
  - Compute Grid + Data Grid
- Cloud
  - Zestaw API dostarczający „pewną” funkcjonalność
- Cloud Computing
  - Datacenter + API (Cloud)

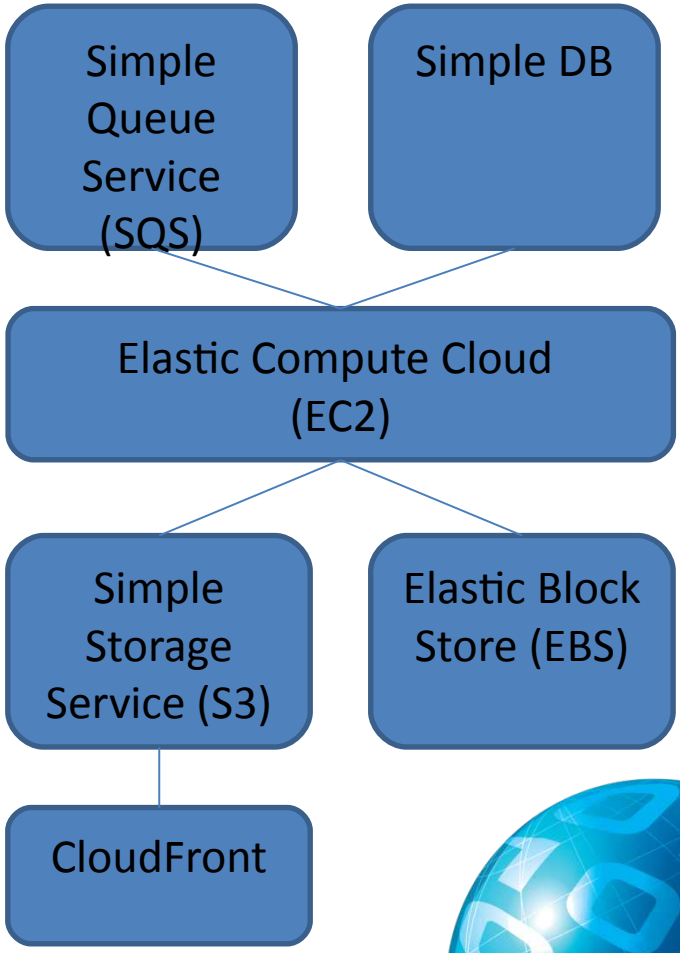
Grid Computing - rozwiązania

- Data Grid
  - Terracota (Software AG)
  - Oracle Coherence
- Grid Computing
  - GridGain
  - JBOSS Infinispan
- Cloud Computing
  - Amazon EC2
  - Google AppEngine
  - Microsoft Azure



Cloud na przykładzie Amazon EC2

- Amazon oferuje
  - Infrastrukturę dla aplikacji
  - Bazy danych
  - Kolejki
  - Storage
  - Dedykowane obrazy
  - …. I wiele więcej
- Możliwe wykorzystanie
  - Skalowane środowisko produkcyjne
  - Serwer FTP



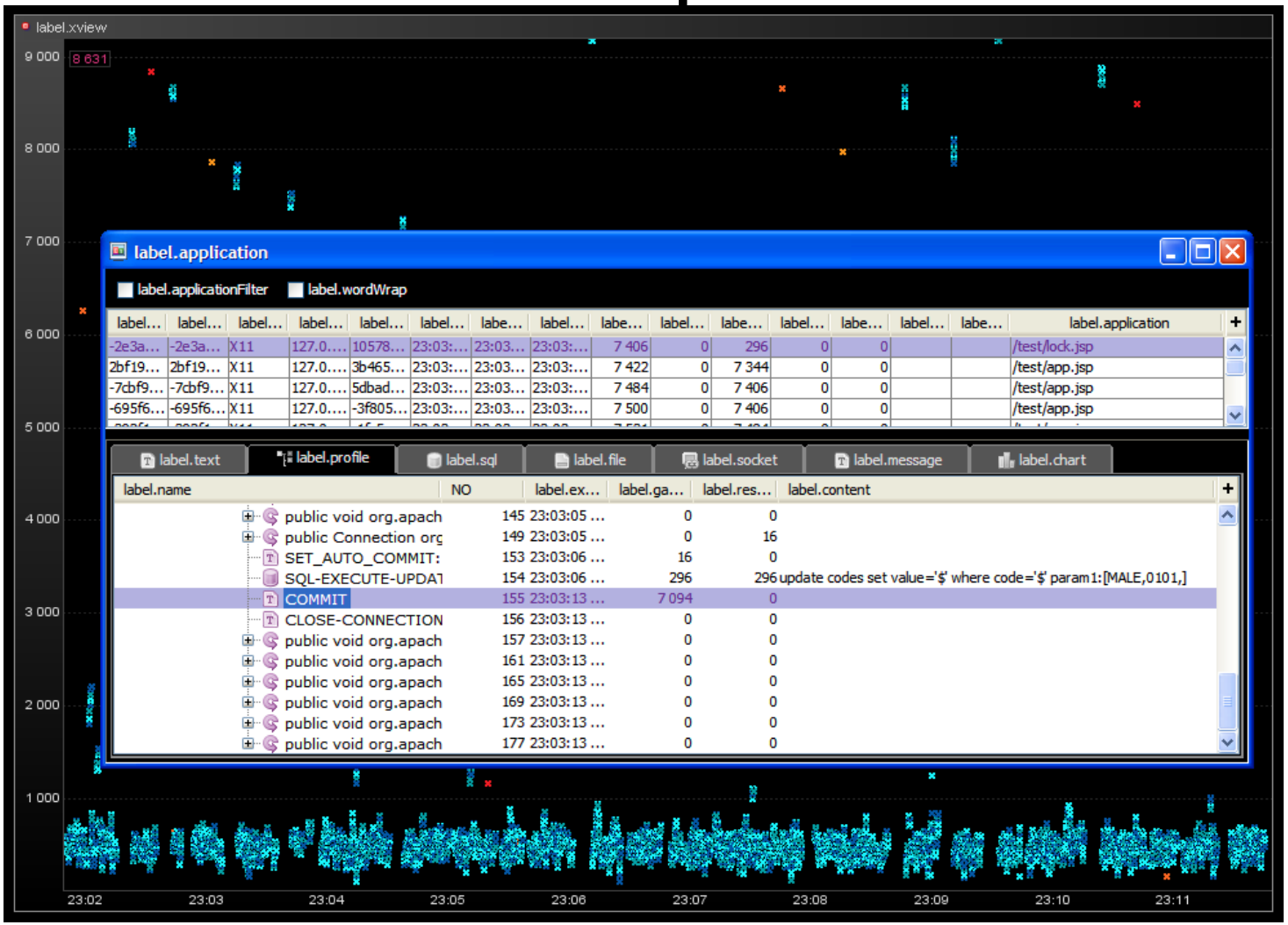
Wykorzystanie Grid Computing

- GUI
  - Przechowywanie sesji użytkownika
  - Przechowywanie obiektów współdzielonych
- Implementacja
  - Duże obszary danych dla obiektów (100+GB)
  - Niezawodne i koherentne kontenery danych
    - Przeważnie dostępne jako java.util.Map
  - Frameworki do przetwarzania
    - MapReduce
    - Job Scheduler
- Persystencja

A co jeżeli jednak musimy znaleźć problem?

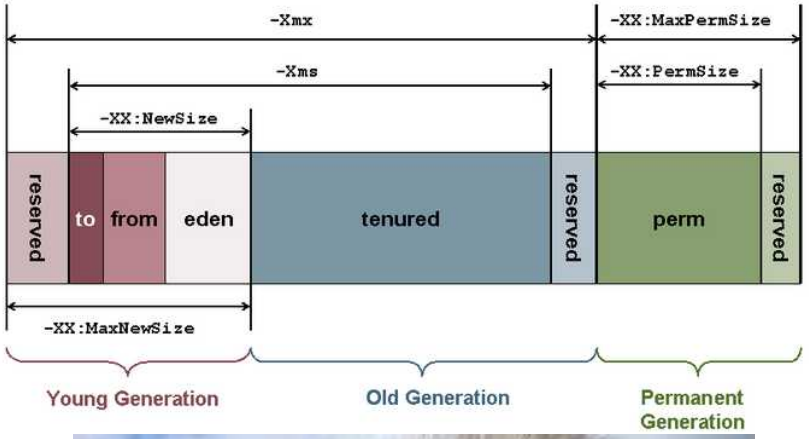
- Natywne narzędzia JDK
  - **JVisualVM** (wcześniej JConsole)
  - Jstat
  - Jmap
  - Jinfo…
- Logi Garbage Collectora
  - -Xloggc:<PLIK>
  - GCViewer (http://www.tagtraum.com/)
- Memory Analyzer (Eclipse)
- Sporo komercyjnych rozwiązań

Narzędzia do diagnozy problemów



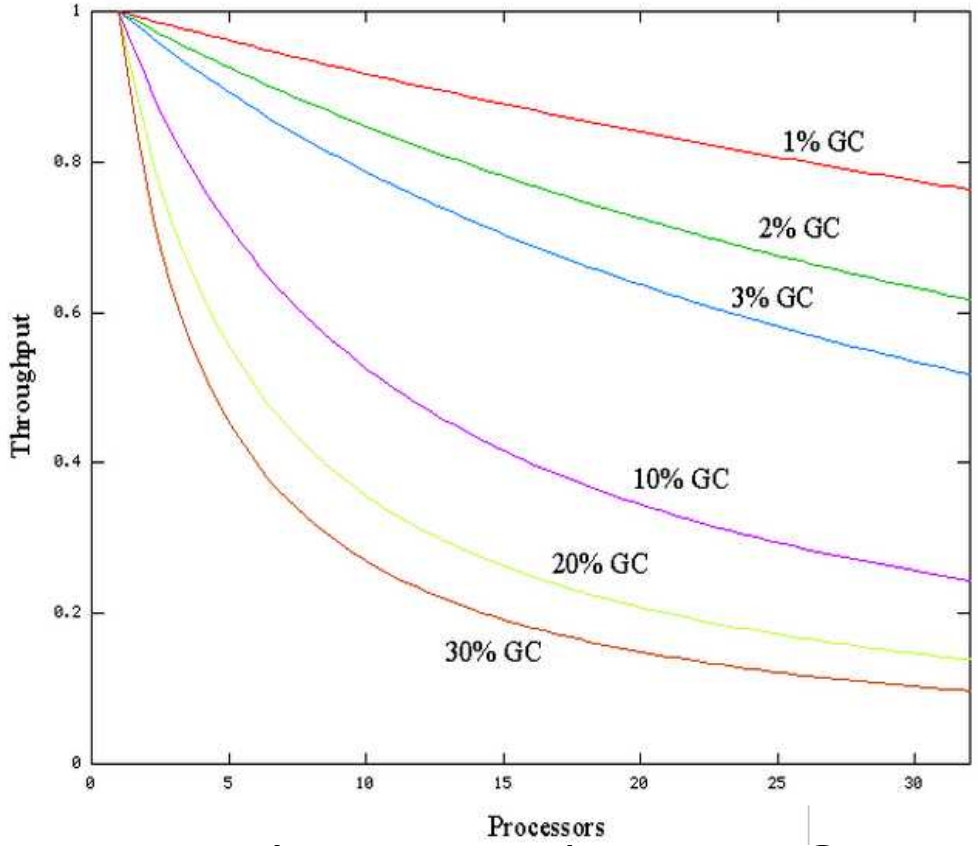
Garbage Collector

- Obiekty alokowane są w przestrzeni „young”
- Jeżeli przetrwają „X czasu” są przenoszone do obszaru „survivor”
- Często są 2 równoległe procesy GC
  - Young – przeważnie nie powodujący przestoju aplikacji
  - Old – przeważnie

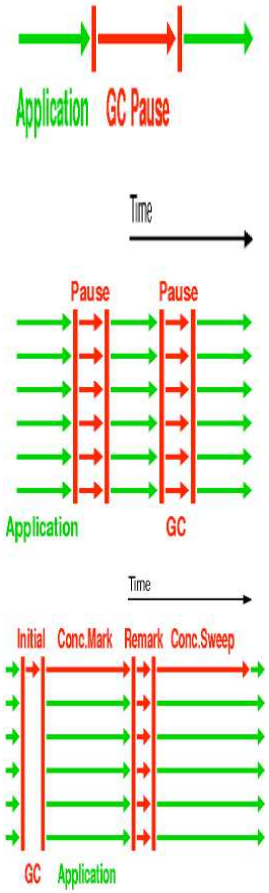


Garbage Collector – tryby pracy

- Sposoby działania GC



stoje



  - Mostly-Concurrent Collector
    - Minimalizuje latency (przestoje)

Tuning GC

Opcja nr 1

- Tryby pracy GC
  - -XX:+UseSerialGC
  - -XX:+UseConcMarkSweepGC
    - -XX:ParallelGCThreads=20
    - -XX:+UseParNewGC
    - -XX:SurvivorRatio=8
    - -XX:TargetSurvivorRatio=90 (default 50)

Nieskończona liczba roboczogodzin

Opcja nr 2

- Usprawnienia zwiększające przepustowość
  - Duże rozmiary sterty
  - Do 500 GB na JVM/instancja aplikacji
- 64 bitowa architektura JVM
  - Możliwosc wykorzystania

Skończona ilość dolarów

Find a bug…

- Klient mówi
  - Wszyscy użytkownicy muszą długo czekać na odpowiedź
  - Wykorzystanie CPU na maszynach jest na poziomie 10%
- Developer mówi
  - U mnie się skaluje rewelacyjnie
  - Przy testach obciążenie CPU jest >75%

Dziękuję