Architektura Projektów Bioinformatycznych

Katedry, bazary i inne budowle – o paradygmatach inżynierii oprogramowania

Bartek Wilczyński
bartek@mimuw.edu.pl

22.04.2020

Inżynieria oprogramowania

- Programowanie jest bardzo specyficzną działalnością
- Z jednej strony uprawnione są porównania do inżynierii (budowy dróg, mostów i układów elektronicznych)
- Z drugiej strony, istotne są zagadnienia twórcze, podobne np. do problemów autorów książek, czy innych dzieł artystycznych
- Analogie do architektury mogą być mylące, bo rzadko kiedy architekt budynku jest jednocześnie jego wykonawcą, zaś w programowaniu jest to niezwykle częste
- Dodatkowo, oprogramowanie jest tworem “ulotnym”, łatwym w kopiowaniu, co zwiększa podobieństwo do utworów artystycznych

Początki paradygmatów programowania

- 1970’s - różne modele strukturalizacji wielostopniowego procesu inżynieryjnego zapożyczane z innych dziedzin inżynierskich, jednocześnie akademickie źródła oprogramowania “naukowego”
- 1980’s - Pojawiają się pierwsze popularne systemy strukturalne, np. 7-stopniowy SSADM, stosowane w projektach dla przedsiębiorstw, jednocześnie rozkwit “garażowych” produkcji na mikrokomputery (np. Apple, Microsoft, gry komputerowe)

Product development from an IT failures perspective


How the customer explained it


How the project leader understood it


How the business consultant described it


How the analyst designed it


How the programmer wrote it


How the project was documented

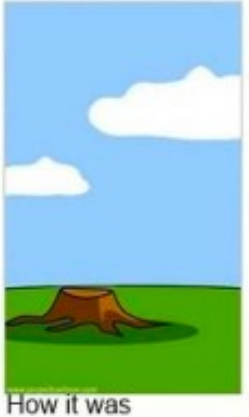
How they advertised the open source version


How they applied open source patches


What the beta testers received

What marketing advertised


What operations installed


How it was supported


What the customer really needed


How it performed under load


The disaster recovery plan


How the customer was billed

Rozkwit Inżynierii oprogramowania

- 1990’s – UML, rozkwit programowania obiektowego, Rational Unified Process, ale także zauważanie sukcesu małych zespołów w pomysłach typu Extreme programing, Scrum
- 2000’s – Rozczarowanie technologiami “scentralizowanymi” rozkwit różnych technologii typu “Agile”, również w dużej skali (np. Agile Unified Process)
- 2010’s – rozkwit technik rozproszonego rozwoju (e.g. github) i wykorzystania open-source również w komercyjnych projektach

Projekty komercyjne vs. open-source

- Ogromny rozwój komercyjnego przemysłu wytwórstwa oprogramowania w latach 80tych stworzył sytuację, gdy projekty akademickie i open-source były postrzegane jako stricte niekomercyjne i wręcz szkodliwe dla rynku oprogramowania komercyjnego
- Dopiero w ostatnich latach, dzięki sukcesom komercyjnym takich projektów jak MacOS X, GNU/Linux, Android, Apache, Open ssh/ssl, itp. Ten sposób myślenia odchodzi do lamusa

Inżynieria oprogramowania open-source?

- Projekty open-source przez dekady rozwijane były przez entuzjastów w małych zespołach, podczas gdy inżynieria oprogramowania była zwykle skupiona na ogromnych projektach bankowych i rządowych (Warto przeczytać “Mythical Man-Month”, F.P. Brooks’a)
- Dopiero E.S Raymond w swoim wpływowym eseju “The Cathedral and the Bazaar” (ca. 1998) dokonał porównania paradygmatów open-source i komercyjnego, wskazując wiele zalet open-source
- Później pojawiły się też inne dokumenty, m.in. Free Software Project Management (B.M. Hill, 2001), które dawały szczegółowe wskazówki dla projektów w modelu “bazarowym”

Katedra i bazar

- E.S. Raymond porównywał dwa modele rozwoju oprogramowania:
  - Katedra – gdzie tylko wąskie grono ludzi ma wgląd w kod i możliwość jego zmieniania (obejmujący wszystkie projekty closed-source i ówcześnie wiele projektów open-source, np. Emacs)
  - Bazar – model rozproszony, gdzie decyzje podejmowane są publicznie, często w ostrych sporach, tj. w przypadku systemu linux

Rozproszony model rozwoju oprogramowania

- Model “bazarowy” rozwoju oprogramowania był w latach 1990tych niezwykle egzotyczny dla “poważnych” firm, mimo, że wiele projektów software’owych powstawało w garażach i małych firmach
- Wymagał on też dobrej komunikacji internetowej, i publicznej dostępności kodu źródłowego, co wydawało się wtedy zupełnie pozbawione sensu z punktu widzenia komercyjnego

Zalety modelu “bazaar”

- Jednak okazało się, że w wielu wypadkach zalety przeważają nad wadami:
  - Wiele oczu łatwiej znajdzie błędy i problemy
  - Model płacenia przez klientów za kod źródłowy nie sprawdzał się (problem z wyceną kilo-linii kodu lub osobo-miesiąca)
  - Obawy, że ktoś, kto nie jest autorem kodu zabierze pracę jego autorowi się nie sprawdziły
  - W ostatecznym rozrachunku, nieopłacalne dla społeczeństwa jest płacenie programistom wielokrotnie za ten sam kod
  - Wraz z ogromnym rozwojem technologii internetowych, bariery w rozwoju rozproszonym zanikały

Open-Source i prawo autorskie

- Lata 1990te były okresem wielkich sporów dotyczących prawnych aspektów własności intelektualnej programów komputerowych
  - Ogromna batalia prawna pomiędzy SCO a różnymi firmami (IBM, Novell, Redhat) o prawa do kodu źródłowego linuksa rozstrzygnięta “na korzyść open-source”
  - Spór dotyczący patentowania algorytmów, ciągle różnie interpretowanych w prawie patentowym w różnych krajach, ale obecnie w praktyce wygasły

Różne licencje Open Source

- Na przestrzeni lat 80tych I 90tych wykształciło się wiele licencji na różnego rodzaju progamy open source.
- Obecnie w zasadzie głównie dominują licencje typu
  - BSD/MIT – Berkeley style distribution
  - GNU – General public license (GPL) i Lesser GPL (LGPL)
  - Apache
- Różnią się między sobą podejściem do redystrybucji kodu źródłowego przez jego użytkowników i modyfikatorów

License	Apache	BSD/MIT	GPL	LGPL	MPL/CDDL	CPL/EPL
Closed source	Yes	Yes	No	Maybe	Yes	Yes
Commercial	Yes	Yes	No	Maybe	Yes	Yes
Modification release	No	No	Yes	Yes	Yes	Yes
Patent	Yes	No	No	No	Yes	Yes
Jurisdiction	Silent	Silent	Silent	Silent	California	New York
Freedom	PR	Free	PR	PR	Free	PR

Table 2: The rights of the developer to redistribute a modified product. A comparison of open source software licenses listed as "with strong communities" on `http://opensource.org/licenses/category`. The main questions are: whether code can be used in closed source projects (Closed source); whether a program that incorporates the code can be sold commercially (Commercial) without releasing the incorporating program under the same license; whether the source code to modifications must be released (Modification release); whether it provides an explicit license of patents covering the code (Patent); the legal jurisdiction the license falls under (Jurisdiction); freedom to adapt licence terms (Freedom) (PR = Permission Required from license drafter). Apache: License used by the Apache web server; BSD: License under which the BSD Unix variant is released; MIT: developed by the MIT; GPL/LGPL: (lesser) GNU General Public License; MPL: License used by the Mozilla web browser; CDDL: Common Development and Distribution License developed by Sun Microsystems based on the MPL; CPL: Common Public License published by IBM; EPL: Eclipse Public License used by the Eclipse Foundation, derived from the CPL.

Oprogramowanie naukowe – komercyjne, czy nie?

- Początki programowania oczywiście wiążą się z projektami naukowymi i wojskowymi,
- w latach 80tych rozwinął się model “wyprowadzania” projektów software’owych ze świata akademickiego do firm komercyjnych
- Uniwersytety amerykańskie często same wspierały tworzenie spółek spin-off, które zarządzały i rozwijały oprogramowanie powstałe w ramach projektów badawczych
- Prowadziło to często do sytuacji, gdzie społeczność akademicka musiała płacić za rozwiązania powstałe w projektach naukowych

Powrót oprogramowania naukowego do open-source

- Obecnie, zauważa się powrót do oprogramowania open-source w środowiskach naukowych
- Ułatwia to dydaktykę
- Nie sprawia problemów licencyjnych
- Ułatwia stosowanie dobrych praktyk
- Wspiera reprodukowalność badań

Incentives and Integration In Scientific Software Production

James Howison
University of Texas at Austin
Austin, TX
jhowison@ischool.utexas.edu

James D Herbsleb
Carnegie Mellon University
Pittsburgh, PA
jdh@cs.cmu.edu

ABSTRACT
Science policy makers are looking for approaches to increase the extent of collaboration in the production of scientific software, looking to open collaborations in open source software for inspiration. We examine the software ecosystem surrounding BLAST, a key bioinformatics tool, identifying outside improvements and interviewing their authors. We find that academic credit is a powerful motivator for the production and revealing of improvements. Yet surprisingly, we also find that improvements motivated by academic credit are less likely to be integrated than those with other motivations, including financial gain. We argue that this is because integration makes it harder to see who has contributed what and thereby undermines the ability of reputation to function as a reward for collaboration. We consider how open source avoids these issues and conclude with policy approaches to promoting wider collaboration by addressing incentives for integration.

production of open knowledge in Wikipedia and open source software [4,27,30]. Given the communitarian principles of science it is not surprising that open collaboration is a promising candidate [19,40,41].

This paper examines openness as a road to more effective collaboration and co-creation of scientific software. We report on results of an empirical study of the socio-technical structure of innovation around a central piece of infrastructural software, the Basic Local Alignment Search Tool (BLAST). BLAST is arguably one of the most important pieces of scientific software ever written, to a large extent enabling the bioinformatics revolution. By 2003 the original paper describing the BLAST algorithm had become the third most cited paper of all time [38].

Understanding the shifting and sometimes conflicting incentives for sharing and collaboration in scientific software will deepen our understanding of coordinating collaborative work more broadly. This is because "the republic of science" [29] contrasts with environments

BRIEFINGS IN BIOINFORMATICS. VOL 7. NO 3. 287–296 doi:10.1093/bib/bbl026

Open source tools and toolkits for bioinformatics: significance, and where are we?

Jason E. Stajich and Hilmar Lapp

Submitted: 11th October 2005; Received (in revised f

Abstract

This review summarizes important v
couple of years. The survey is inter
developed or released under an Ope
Rather than creating a comprehensi
projects encompassing toolkit librari
show how freely available and modifi
applications, analysis workflows and r

Improving the Computer Science in Bioinformatics Through Open Source Pedagogy

John David N. Dionisio
Department of Electrical Engineering & Computer Science
Loyola Marymount University
Los Angeles, CA 90045, USA
dondi@lmu.edu

Kam D. Dahlquist
Department of Biology
Loyola Marymount University
Los Angeles, CA 90045, USA
kdahlquist@lmu.edu

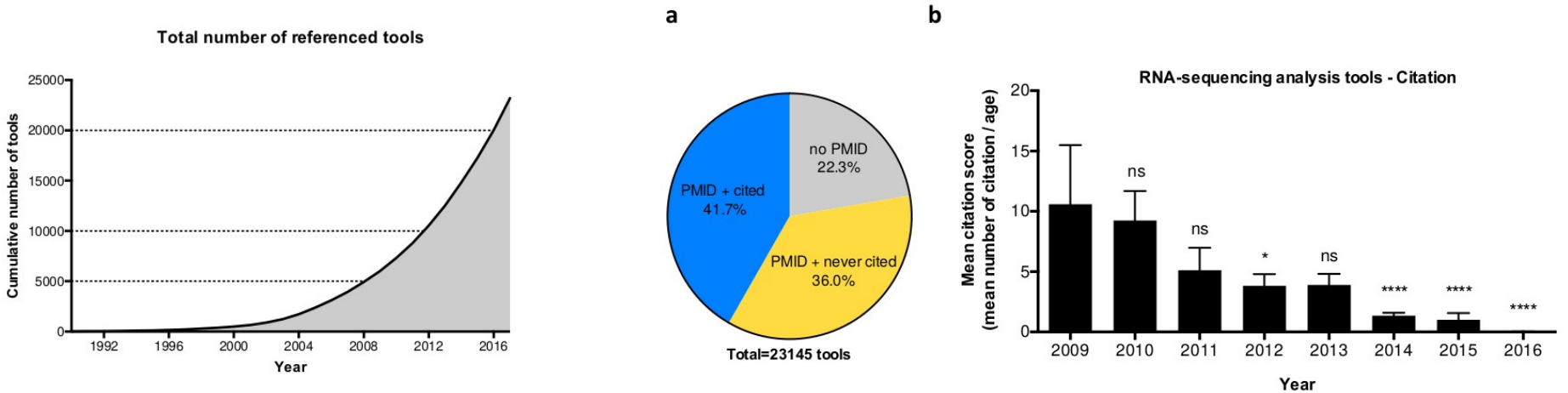
Abstract: Bioinformatics relies more than ever on information technologies. This pressures scientists to keep up with software development best practices. However, traditional computer science curricula do not necessarily expose students to collaborative and long-lived software development. Using open source principles, practices, and tools forms an effective pedagogy for software development best practices. This paper reports on a bioinformatics teaching framework implemented through courses introducing computer science students to the field. The courses led to an initial product release consisting of software and an *Escherichia coli* K12 GenMAPP Gene Database, within a total

Głosy krytyczne i dalszy rozwój

- Oczywiście sam model “bazarowy” czy licencje open-source nie rozwiązują wszystkich problemów związanych z oprogramowaniem
- W szczególności esej “Open source software development as a special type of academic research: Critique of vulgar Raymondism” (Bezroukov, 1999) opisywał wiele problemów, które pozostają do rozwiązania (finansowanie, problemy natury międzyludzkiej, itp.)

Obecny stan projektów bioinformatycznych

- Liczba dostępnych narzędzi przyrasta lawinowo, są one często publikowane, ale rzadko używane (i cytowane)



Levin et al.2016 https://omictools.com/bioinformatics-trends

Figure 5: Bioinformatic tools in the literature.

10 simple rules for open development of scientific software

1) Don’t reinvent the wheel
2) Code well
3) Be your own user
4) Be transparent
5) Be simple
6) Don’t be a perfectionist
7) Grow your community
8) Promote your project
9) Find sponsors
10) Science Counts

PLoS Comput Biol. 2012 Dec; 8(12): e1002802.