Intruduction to Electronic Participatory Culture

Fundamental Concepts and Definitions

Krzysztof Gajewski
Institute of Literary Research
Polish Academy of Science

Karlsruhe, 8-9 April 2015

Contents

Web 1.0 vs Web 2.0

Tim O'Reilly

1. open source
2. Web 2.0
3. architecture of participation
4. Government 2.0

open source

1. Software as Commodity
2. Network-enabled collaboration
3. Software customizability (software as a service)

> *Sites like Google, Amazon, or eBay – especially those reflecting the dynamic of user participation – are not just products, they are processes.*[1]

[1] Tim O'Reilly, Open Source Paradigm Shift, 2004, `http://archive.oreilly.com/pub/a/oreilly/tim/articles/paradigmshift_0504.html`

Web 2.0

Web 2.0 was not a new version of the World Wide Web; it was a renaissance after the dark ages of the dotcom bust, a rediscovery of the power hidden in the original design of the World Wide Web.[2]

[2] Tim O'Reilly, Government as a platform, in: Open Government, ed. D. Lathrop, L. Ruma, Mountain View, California 2010. `http://chimera.labs.oreilly.com/books/1234000000774/ch02.html`

Government 2.0

Government 2.0 vs. "vending machine government",

> *Government 2.0, then, is the use of technology especially the collaborative technologies at the heart of Web 2.0 to better solve collective problems at a city, state, national, and international level.*[3]

[3] Tim O'Reilly, Government as a platform

Architecture of participation

- Internet: "Standards were decided by participating individuals, irrespective of their company affiliations. The very name for proposed Internet standards, RFCs (Request for Comments), reflects the participatory design of the Net."[4],
- Web
  - "The HTML syntax for formatting a web page was not embedded in a proprietary document format. Instead, HTML documents are ordinary, human-readable text files."
  - "Anyone could link to any other page on the Web, without the permission or knowledge of the destination pages owner. This idea was the reversal of one taken for granted in previous hypertext systems, that links must always be two-way"[5]

---

[4] Tim O'Reilly, The Architecture of Participation, 2004, `http://archive.oreilly.com/pub/a/oreilly/tim/articles/architecture_of_participation.html`

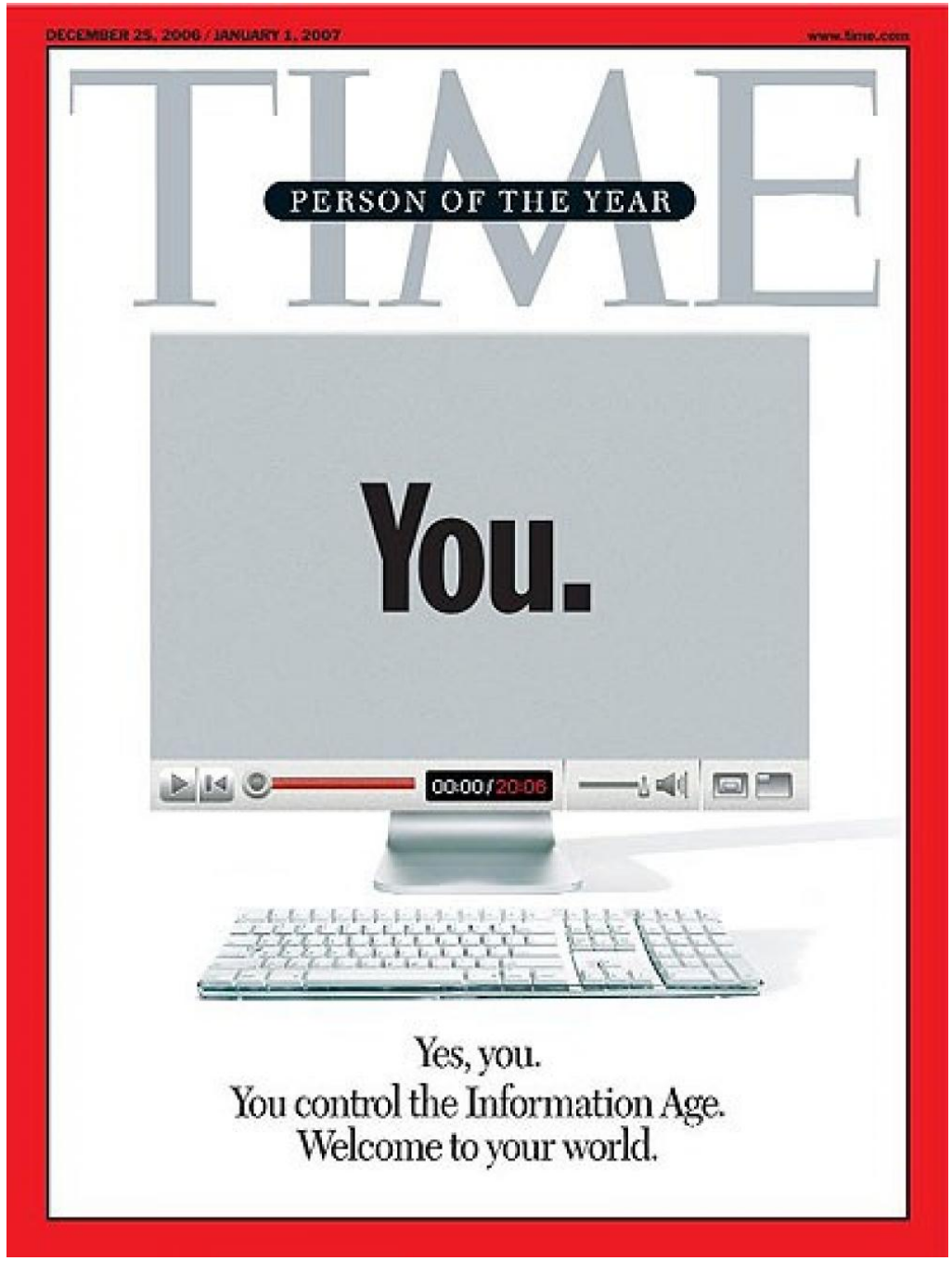
[5] Tim O'Reilly, Government as a platform

Web 1.0 → Web 2.0

DoubleClick	→ Google AdSense (Long Tail)
Ofoto(scanner software)	→ Flickr
Akamai	→ BitTorrent
mp3.com	→ Napster
Britannica Online	→ Wikipedia
personal websites	→ blogging
evite	→ upcoming.org and EVDB
domain name speculation	→ search engine optimization
publishing	→ participation
content management systems	→ wikis
directories (taxonomy)	→ tagging ("folksonomy")
stickiness (Sticky content)	→ syndication (web feeds)[6]

[6] Tim O'Reilly, What Is Web 2.0 Design Patterns and Business Models for the Next Generation of Software, 2005
`http://www.oreilly.com/pub/a//web2/archive/what-is-web-20.html`

User Generated Content


DECEMBER 25, 2006 / JANUARY 1, 2007
www.time.com
TIME
PERSON OF THE YEAR
You.
00:00 / 20:06
Yes, you.
You control the Information Age.
Welcome to your world.

User Generated Content

User Generated Content (UGC) aka. User Created Content (UCC)

1. "content made publicly available over the Internet"
2. "which reflects a certain amount of creative effort"
3. "which is created outside of professional routines and practices"[7]

[7] Sacha Wunsch-Vincent, Graham Vickery, Participative Web: User-Created Content, OECD Report 2007.

Cultural and social impact of UGC

- "increase in availability and a more diverse array of cultural content to find niche audiences"
- "platform enriching political and societal debates, diversity of opinion, free flow of information"
- transparency, "some watchdog functions" (Citizen journalism)[8]

[8] Wunsch-Vincent, Vickery, op. cit.

Challenges of UGC

- deeper digital divide
- cultural fragmentation
- content quality
- security and privacy[9]

[9] Wunsch-Vincent, Vickery, op. cit.

The Cathedral and the Bazaar (Eric Raymond)

"Linux is subversive"

Linux Kongress 1997, Würzburg

> *I believed that the most important software (...) needed to be built like* **cathedrals**, *carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time. Linus Torvalds's style of development (...) came as a surprise. (...) rather, the Linux community seemed to resemble a great babbling* **bazaar** *of differing agendas and approaches*[10]

10

Eric Raymond's Lessons

Lesson learned from the story of Fetchmail development

1. Every good work of software starts by scratching a developer's personal itch.
2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
3. "Plan to throw one away; you will, anyhow." (Fred Brooks, The Mythical Man-Month, Chapter 11)
4. If you have the right attitude, interesting problems will find you.
5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

Eric Raymond's Lessons

6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.
7. Release early. Release often. And listen to your customers.
8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.
(Or, less formally, "Given enough eyeballs, all bugs are shallow." I dub this: "Linus's Law")
9. Smart data structures and dumb code works a lot better than the other way around.
10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.

Eric Raymond's Lessons

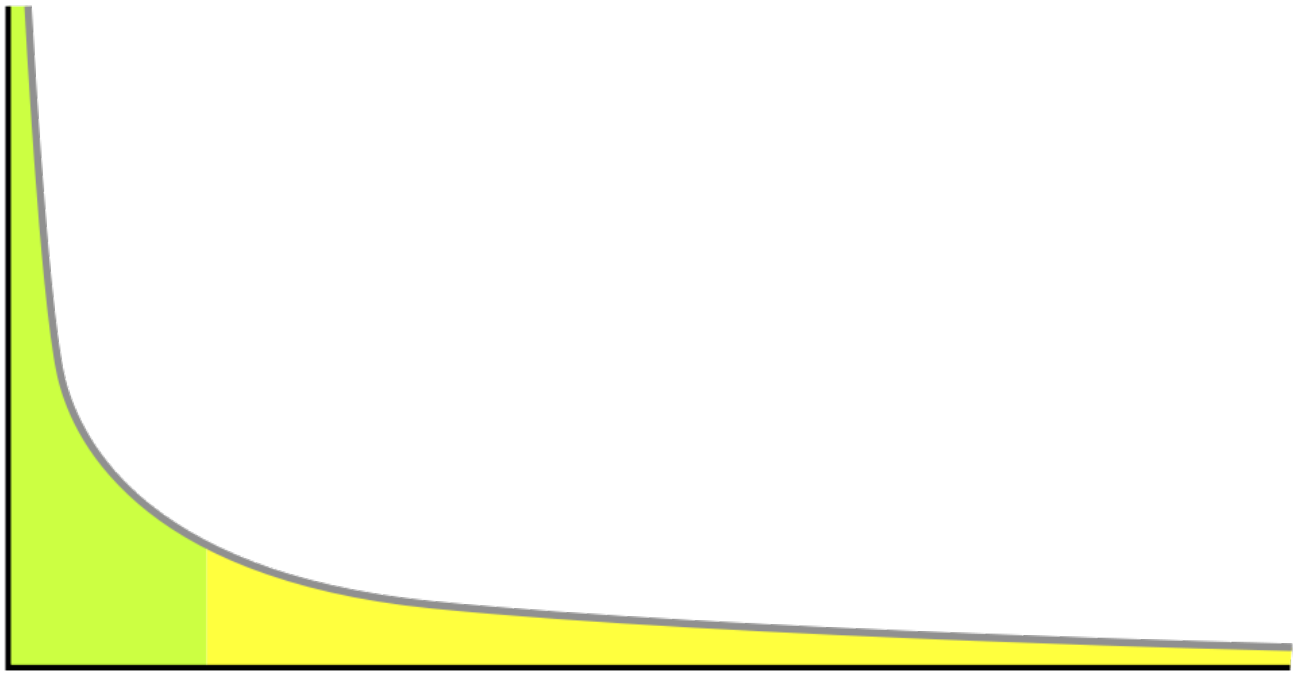
11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
13. "Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away." (Antoine de Saint-Exupry)
14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possibleand never throw away information unless the recipient forces you to!

Eric Raymond's Lessons

16. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.
17. A security system is only as secure as its secret. Beware of pseudo-secrets.
18. To solve an interesting problem, start by finding a problem that is interesting to you.
19. Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

Long Tail (Chris Anderson)

Long Tail



Forget squeezing millions from a few megahits at the top of the charts. The future of entertainment is in the millions of niche markets at the shallow end of the bitstream.[11]

[11] Chris Anderson, The Long Tail, 2004, `http://archive.wired.com/wired/archive/12.10/tail.html`

Cultural niche

- *Wal-Mart must sell at least 100,000 copies of a CD to cover its retail overhead and make a sufficient profit; less than 1 percent of CDs do that kind of volume. What about the 60,000 people who would like to buy the latest Fountains of Wayne or Crystal Method album (...)?*[12]
- Hit-driven economics vs. long tail

  If the 20th- century entertainment industry was about hits, the 21st will be equally about misses.

[12] Chris Anderson, The Long Tail