Intro. to Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

January 9, 2018

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Agenda:
  1. comments, questions, recap, then
  2. some (meta) *advice*, namely
     - the importance of design metrics and trade-offs, and
     - problem solving techniques.

Notes:

Point #1: an ETHZ-inspired [0] introduction to design (1)

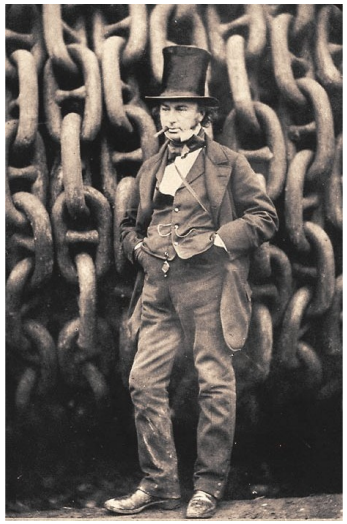
Notes:

Point #1: an ETHZ-inspired [0] introduction to design (1)

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg

Notes:

Point #1: an ETHZ-inspired [0] introduction to design (2)



Isambard Kingdom Brunel [4].

1. the Clifton suspension bridge,
2. Temple Meads station,
3. the SS Great Britain,
4. ...

http://en.wikipedia.org/wiki/File:IKBrunelChains.jpg

Notes:

- There's a nice overview of Brunel at

  http://hackaday.com/2017/12/15/hardware-heroes-isambard-kingdom-brunel

  noting the web-site: Hackaday is hardware-oriented, highlighting the link between general design and CS-specific design.

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we derive design requirements?

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we derive design requirements?
- Answer: in a rough sense,

$$\overset{\text{innovation}}{\rightsquigarrow} \text{ concept } \overset{\text{analysis}}{\rightsquigarrow} \text{ problem specification } \overset{\text{analysis}}{\rightsquigarrow} \text{ design requirements,}$$

although it's possible to identify (at least) two classes of requirement, namely

1. *constructive* : "the design must include functionality X"
2. *restrictive* : "the design must cost less than Y"

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

© Daniel Page ⟨dan@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we combat design complexity?

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we combat design complexity?
- Answer: abstraction (+ standardisation, + ...).
  - provide a clean enough separation st.

$$\textbf{design} = \left\{ \begin{array}{ccc} \textbf{function} & \simeq & \textbf{interface} \\ + & & + \\ \textbf{form} & \simeq & \textbf{implementation} \end{array} \right.$$

  - example:

$$\begin{array}{rcccl} \text{(instruction set) architecture} & \Rightarrow & \text{interface} & \Rightarrow & \text{x86} \\ \text{micro-architecture} & \Rightarrow & \text{implementation} & \Rightarrow & \text{Intel, AMD, VIA, Transmeta, ...} \end{array}$$

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

© Daniel Page ⟨dan@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we motivate design choices?

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we motivate design choices?
- Answer: design requirements + **design principles**.

> **Quote**
>
> *I am opposed to the laying down of rules or conditions to be observed in the construction of bridges lest the progress of improvement tomorrow might be embarrassed or shackled by recording or registering as law the prejudices or errors of today.*
>
> *– Brunel*

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

© Daniel Page (
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we evaluate design choices?

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

Point #1: an ETHZ-inspired [0] introduction to design (3)

Clifton Suspension Bridge

Maidenhead Railway Bridge

Royal Albert Bridge

- Question: how do we evaluate design choices?
- Answer:
  1. **design metric** : a quantifiable criteria used to evaluate a solution
  2. **design space** : the set of all viable (i.e., satisfying requirement) solutions
  3. **design trade-off** : a compromise wrt. metrics, favouring one over another

Notes:

http://en.wikipedia.org/wiki/File:Clifton.bridge.arp.750pix.jpg
http://en.wikipedia.org/wiki/File:Maidenhead_Bridge_-_fGWR_43132.JPG
http://en.wikipedia.org/wiki/File:Royal_Albert_Bridge_2009.jpg

© Daniel Page ⟨dan@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

An Aside: A counter-example (1)

- Beware: this can be a difficult topic to give precise answers about, with a famous essay [6] summarising the issue (roughly) as

Metric	do-the-right-thing (or, MIT philosophy)	worse-is-better (or, New Jersey philosophy)
Simplicity	interface > implementation	implementation > interface
Correctness	totally	mostly
Consistency	totally	mostly
Completeness	mostly	as given
Example		

Notes:

An Aside: A counter-example (1)

- Beware: this can be a difficult topic to give precise answers about, with a famous essay [6] summarising the issue (roughly) as

Metric	do-the-right-thing (or, MIT philosophy)	worse-is-better (or, New Jersey philosophy)
Simplicity	interface > implementation	implementation > interface
Correctness	totally	mostly
Consistency	totally	mostly
Completeness	mostly	as given
Example	MULTICS [10, 9] kernel OSI [5] network model	UNIX [12, 11, 15] kernel Internet [3] network model

where, perhaps surprisingly, the right-hand case "wins".

Notes:

© Daniel Page ⟨dan@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

An Aside: A counter-example (1)

- Beware: this can be a difficult topic to give precise answers about, with a famous essay [6] summarising the issue (roughly) as

Metric	do-the-right-thing (or, MIT philosophy)	worse-is-better (or, New Jersey philosophy)
Simplicity	interface > implementation	implementation > interface
Correctness	totally	mostly
Consistency	totally	mostly
Completeness	mostly	as given
Example	MULTICS [10, 9] kernel OSI [5] network model	UNIX [12, 11, 15] kernel Internet [3] network model

  where, perhaps surprisingly, the right-hand case "wins".
- Moral #1: *over*-design (cf. *over*-engineering) is often due to bad design trade-off(s).
- Moral #2: purely academic study of this topic *might* not be so worthwhile!

Notes:

An Aside: Some product-focused design principle examples (1)

http://en.wikipedia.org/wiki/File:IPhone_2G_PSD_Mock.png

Notes:

An Aside: Some product-focused design principle examples (1)

Quote

The broader one's understanding of human experience, the better design we will have.

– Jobs

Simple can be harder than complex: you have to work hard to get your thinking clean to make it simple.

– Jobs

$\therefore$ simplicity $\Rightarrow$ usability + user experience rule

http://en.wikipedia.org/wiki/File:IPhone_2G_PSD_Mock.png

Notes:

An Aside: Some product-focused design principle examples (2)

http://en.wikipedia.org/wiki/File:Game-Boy-Original.jpg

Notes:

An Aside: Some product-focused design principle examples (2)

Quote

Even if a video game doesn't have the power to display very complex graphics, I believe your imagination has the power to transform that perhaps-unrecognizable sprite called a "rocket" into an amazing, powerful, "real" rocket.

– Yokoi (http://shmuplations.com/yokoi)

I first take the character (or characters) which you're going to control and replace them with a dot as a placeholder, then I think about what kind of movement would be fun. Basically I'm trying to put myself in the player's shoes and figure out what they would enjoy.

– Yokoi (http://shmuplations.com/yokoi)

$\therefore$ user experience > cutting edge technology

http://en.wikipedia.org/wiki/File:Game-Boy-Original.jpg

Notes:

An Aside: Some product-focused design principle examples (3)

http://en.wikipedia.org/wiki/File:NES-Console-Set.jpg

Notes:

Quote

The main processor unit is operated according to a program contained in the external memory. To verify that the external memory is authentic, duplicate semiconductor devices, for example microprocessors, are separately mounted with the external memory and in the main unit, respectively. The semiconductor associated with the external memory device acts as a key device and the duplicate device mounted in the main unit acts as a lock device. The key device and the lock device are synchronized with each other, executing the same arithmetic operation according to the same program. The results of these operations are exchanged between devices, and compared. If the results agree, the external memory is determined to be authentic and the main processor unit is allowed to operate; but if the external memory is determined to be false (not authentic), the main unit is left in a reset (disabled) condition.

– Nakagawa (U.S. Patent 4,799,635 [8])

$\therefore$ IP control $\Rightarrow$ market control $\simeq$ profit > innovation

http://en.wikipedia.org/wiki/File:NES-Console-Set.jpg

Notes:

Point #2: problem solving (1)

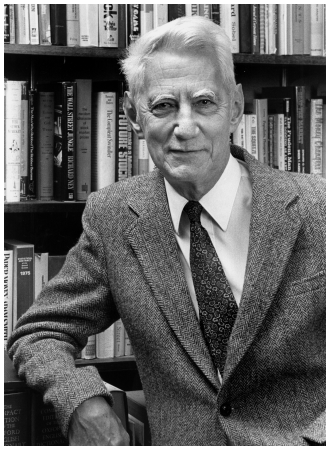
Quote

Genius is rarely able to give an account of its own processes.

– Lewes (http://en.wikiquote.org/wiki/George_Henry_Lewes)

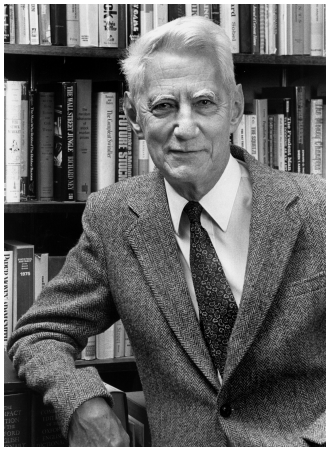
Notes:

Point #2: problem solving (2)



Notes:

Point #2: problem solving (2)

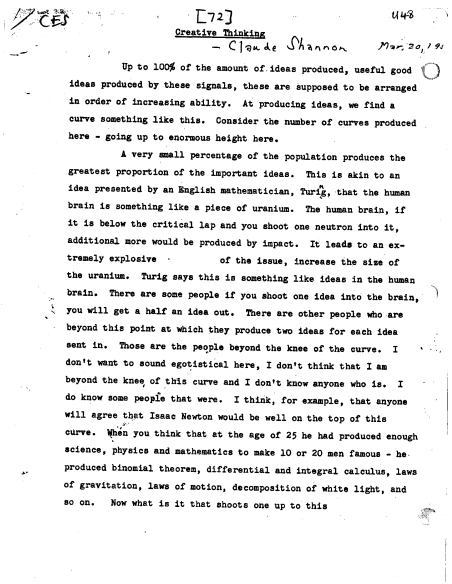


Claude Shannon [2].

1. switching circuits [14],
2. information theory [13],
3. ...

Notes:

http://www.bell-labs.com/news/2001/february/26/shannon2_lg.jpeg



CES [72] 448

Creative Thinking
– Claude Shannon Mar. 20, 19

Up to 100% of the amount of ideas produced, useful good ideas produced by these signals, these are supposed to be arranged in order of increasing ability. At producing ideas, we find a curve something like this. Consider the number of curves produced here - going up to enormous height here.

A very small percentage of the population produces the greatest proportion of the important ideas. This is akin to an idea presented by an English mathematician, Turig, that the human brain is something like a piece of uranium. The human brain, if it is below the critical lap and you shoot one neutron into it, additional more would be produced by impact. It leads to an extremely explosive of the issue, increase the size of the uranium. Turig says this is something like ideas in the human brain. There are some people if you shoot one idea into the brain, you will get a half an idea out. There are other people who are beyond this point at which they produce two ideas for each idea sent in. Those are the people beyond the knee of the curve. I don't want to sound egotistical here, I don't think that I am beyond the knee of this curve and I don't know anyone who is. I do know some people that were. I think, for example, that anyone will agree that Isaac Newton would be well on the top of this curve. When you think that at the age of 25 he had produced enough science, physics and mathematics to make 10 or 20 men famous - he produced binomial theorem, differential and integral calculus, laws of gravitation, laws of motion, decomposition of white light, and so on. Now what is it that shoots one up to this

http://www.scribd.com/document/355277426/Claude-Shannon-Creative-Thinking

Notes:

- Question: how did Shannon solve problems?
- Answer: he gave a talk [7, Chapter 25] in 1952, outlining a need for
  1. basic requirements
     - intelligence,
     - training and experience,
     - motivation $\Rightarrow$ drive to formulate problems and find solutions,
       - concrete purpose,
       - curiosity, desire for understanding,
       - constructive dissatisfaction with non-ideal solution.
       - ...
  2. effective problem solving strategies (or "tricks").

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y.

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y,

then we might

1. simplify problem (e.g., to avoid distraction, or improve intuition).
   - example:
     need $x - y$ for x and y in base-2 $\Rightarrow$ start with $x - y$ for x and y in base-10.
   - programming analogy: debugging via minimum working example.

© Daniel Page ⟨cdp@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y,

  then we might

  5. employ structural analysis (or decomposition).
     - example:

       need $x - y$ for 8-bit x and y $\Rightarrow$ start with $x - y$ for 1-bit x and y.
     - programming analogy: modularity, and hierarchical implementation.

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y,

  then we might

  2. leverage similarity to other, known problems
     - example:

       need $x - y$ $\Rightarrow$ leverage similarity to $x + y$.
     - programming analogy: sorting numbers $\simeq$ sorting strings.
     - advice: read widely, and broadly (within *other* fields); document your solutions.

© Daniel Page ⟨cdp@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y,

  then we might

  3. reformulate problem (e.g., to change perspective, reassess assumptions).
     - example:
       $$\text{need } x - y \quad \Rightarrow \quad \text{reformulate as } x + (-y).$$
     - programming analogy: you spend hours searching for bug, colleague finds it in seconds!
     - advice: write down and so *formalise* problem; draw and annotate diagrams of solution.

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Point #2: problem solving (4)

- Example: imagine we are tasked with producing a design that
  - computes $x - y$, and
  - can do so for 8-bit integers x and y,

  then we might

  4. generalise any solution.
     - example:
       $$\text{need } x - y \text{ for 8-bit } x \text{ and } y \quad \Rightarrow \quad \text{generalise solution to } n\text{-bit } x \text{ and } y.$$
     - programming analogy: reuse.

© Daniel Page ⟨cdp@bristol.ac.uk⟩
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09 University of BRISTOL

Notes:

- This is an important subtleties in the motivation requirement, namely the fact that formulating problems is a problem: doing so is not trivial at all, and, in a sense, captures the idea of innovation (i.e., that doing so rarely happens by accident, but rather by someone identifying and then solving an interesting or important problem).
- The numbering of these strategies follows the order they were presented by Shannon; the *re*ordering seemed to make more sense wrt. how you might apply them in practice.
- Shannon actually included a further technique, namely the idea that one could attempt to *invert* the problem at hand (i.e., work backward from an assumed solution). Of all of them, this seemed most aligned to the context of proof; it might be a useful technique elsewhere, but I couldn't really find a good analogy so excluded it.

Conclusions

- Take away points: developing solutions is hard, but good strategy helps: you'd *probably* agree (roughly) with

good :
- *first* understand
  - clear design requirements, principles, and metrics,
  - mastery of techniques and tools,
- *then* assess feasibility,
- *then* specify and design,
  - clear, precise, rationalised specification,
  - clean abstraction,
  - assess practicality,
- *then* implement (and verify and test, e.g., to verify practicality),
- *then* iterate (guided by principles and metrics).

bad :
- all manner of anti-patterns [1], e.g.,
- copy-and-paste development,
- development-by-permutation (i.e., hack-until-functional),
- ...

but coping with the unit demands you actually *do* it, not think or talk about it!

Notes:

Additional Reading

- *Wikipedia: Isambard Kingdom Brunel.* URL: http://en.wikipedia.org/wiki/Isambard_Kingdom_Brunel.
- *Wikipedia: Claude Shannon.* URL: http://en.wikipedia.org/wiki/Claude_Shannon.

Notes:

References

[1] *Wikipedia: Anti-pattern*. URL: http://en.wikipedia.org/wiki/Anti-pattern (see p. 69).

[2] *Wikipedia: Claude Shannon*. URL: http://en.wikipedia.org/wiki/Claude_Shannon (see pp. 49, 51, 71).

[3] *Wikipedia: Internet protocol suite*. URL: http://en.wikipedia.org/wiki/Internet_protocol_suite (see pp. 29, 31, 33).

[4] *Wikipedia: Isambard Kingdom Brunel*. URL: http://en.wikipedia.org/wiki/Isambard_Kingdom_Brunel (see pp. 11, 71).

[5] *Wikipedia: OSI model*. URL: http://en.wikipedia.org/wiki/OSI_model (see pp. 29, 31, 33).

[6] *Wikipedia: Worse is better*. URL: http://en.wikipedia.org/wiki/Worse_is_better (see pp. 29, 31, 33).

[7] J. Soni and R. Goodman. *A Mind at Play: How Claude Shannon Invented the Information Age*. Simon & Schuster, 2017 (see pp. 55, 57, 59, 61, 63, 65, 67).

[8] K. Nakagawa. *System for determining authenticity of an external memory used in an information processing apparatus*. U.S. Patent 4,799,635. URL: http://www.google.com/patents/US4799635 (see pp. 43, 45).

[9] F.J. Corbató, J.H. Saltzer, and C. T. Clingen. "Multics: The first seven years". In: 1972, pp. 571–582 (see pp. 29, 31, 33).

[10] F.J. Corbató and V. A. Vyssotsky. "Introduction and overview of the Multics system". In: 1965, pp. 185–196 (see pp. 29, 31, 33).

[11] D.M. Ritchie. "The UNIX System: Evolution of the UNIX Time-sharing System". In: *Bell Laboratories Technical Journal* 63.8 (1984), pp. 1577–1593 (see pp. 29, 31, 33).

[12] D.M. Ritchie and K. Thompson. "The UNIX Time-Sharing System". In: *Communications of the ACM (CACM)* 17.7 (1974), pp. 365–375 (see pp. 29, 31, 33).

[13] C.E. Shannon. "A mathematical theory of communication". In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423 (see pp. 49, 51).

[14] C.E. Shannon. "A Symbolic Analysis of Relay and Switching Circuits". In: *Transactions of the American Institute of Electrical Engineers (AIEE)* 57.12 (1938), pp. 713–723 (see pp. 49, 51).

[15] K. Thompson. "UNIX Implementation". In: *Bell System Technical Journal* 57.6 (1978), pp. 1931–1946 (see pp. 29, 31, 33).

Notes: