



First, an anti-announcement

- The first ICER abutted Rosh Hashanah (Jewish New Year)
- Likewise this ICER
 - (particularly hard for internationals)
- The announced ICER 2008 dates clash with it
 - ... I/we will get back to you on dates.




What are the barriers to learning computing?

ICER Discussion

Raymond Lister

University of Technology, Sydney,
Australia



What barriers, or conjectures about barriers, do we see in the ICER 2007 papers?



-
- Maybe tell my anthropologist story
 - Here or later?

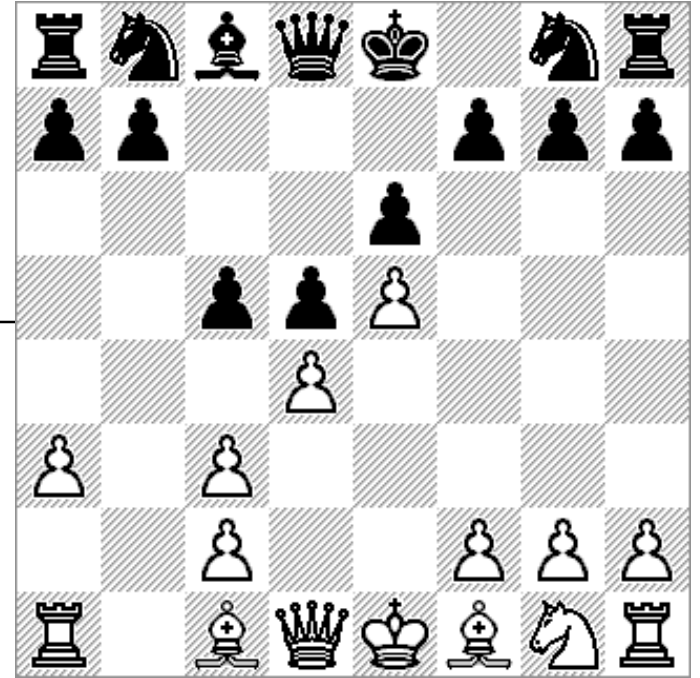
“Through the Eyes of Instructors: A Phenomenographic Investigation of Student Success”, Kinnunen, McCartney, Murphy, and Thomas

- 📁🕒 Nature of the subject
- 📄🕒 Intrinsic – the “geek gene”
- 📄🕒 Previous Experience
- 📄🕒 Attitude / Behaviour
- 📄🕒 Developmental

Do I believe this?
Ask me again in 10 years

Stasko Keynote

(Developmental)



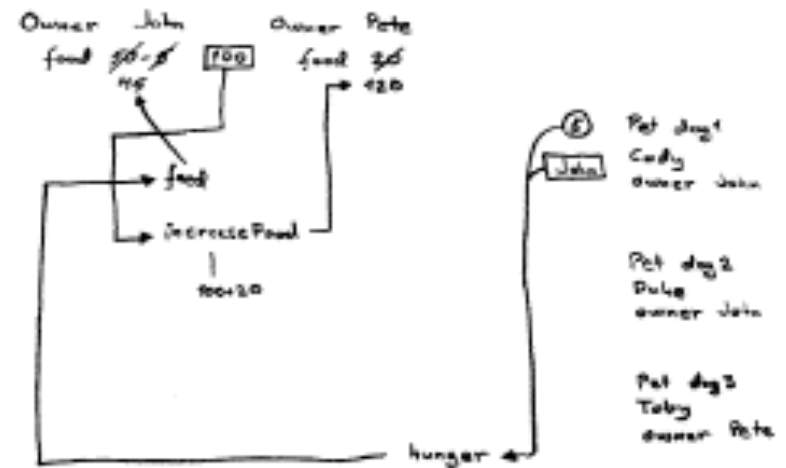
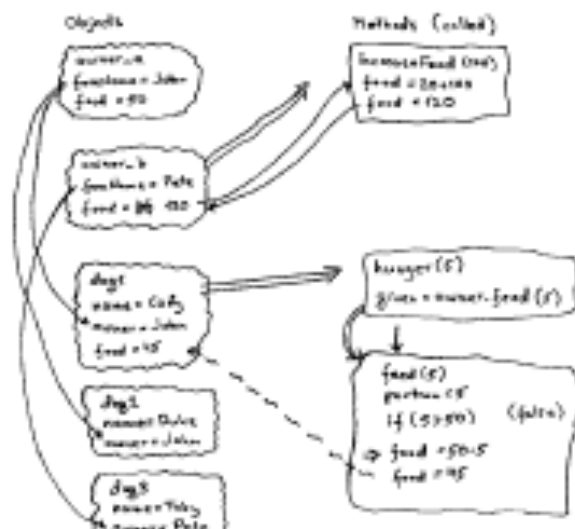
- Visualizations & other representations
 - An aid, or more to learn?
- **Barrier (or inclined plane):**
 - We under estimate how long it takes students to move from concrete to abstract
 - E.g. Classic chess studies of Chase and Simon
 - E.g. Computer Science: Adelson (1984)
 - 1, 3 and 7

A Study of the Development of Students' Visualizations of Program State during an Elementary Object-Oriented Programming Course

Jorma Sajaniemi
 University of Joensuu
 P.O.Box 111
 80101 Joensuu, Finland
 saja@cs.joensuu.fi

Marja Kuittinen
 University of Joensuu
 P.O.Box 111
 80101 Joensuu, Finland
 marja@cs.joensuu.fi

Taina Tikansalo
 University of Joensuu
 P.O.Box 111
 80101 Joensuu, Finland
 ttikans@cs.joensuu.fi



Developmental again? (or subject?)



Schulte and Knobelsdorf, “Attitudes Toward Computer Science ...”

“... learning problems are not always due to difficulties of understanding, but due to a kind of *unwillingness* to change the current conceptualization, caused by a lack of *meaningfulness* of the new concept for the learner...”

Previous Experience?
Attitude / Behaviour?

Compare with Yarosh and Guzdial, “Narrating Data Structures”.

Contrast with the next paper ...


Yardi & Bruckman, “What is Computing? Bridging the gap between Teenagers’ Perceptions and Graduate Students’ Experiences”

- Teenager’s have **perceptions**
 - ... superficial?
 - ... wrong?
- Grad students have **experiences**
 - ... and therefore the legitimate view?
 - ... or are they demented?
- Programming – ability or disability?

Previous Experience?
Attitude / Behaviour?

If we are constructivists, then we need to value student prior experiences, at least enough to help them build upon those prior experiences.

(Which I think Yardi and Bruckman advocate.)



Commonsense Computing (episode 3): Concurrency and Concert Tickets Lewandowski, Bouvier, McCartney, Sanders & Simon

- Respects the prior experience
 - At least enough to build upon it
- Replication! Not enough of it!
 - “We found that the categorizations developed by Ben-David Kolikant were also meaningful when applied to our data, and that our beginning CS1 students are more likely to give centralized solutions (as opposed to decentralized ones) than Ben-David Kolikant’s concurrency students”
 - “... 33% of the solutions in the Ben-David Kolikant study were centralized. Our study shows an even higher number of centralized solutions (55%)”
 - Statistical significance?
 - Is it even appropriate to compare these two groups of students?

Previous Experience?



Hanks and Simon, “First Year Students Impressions of Pair Programming”

Attitude / Behaviour? Developmental?

“I got stuck. I sat there for hours trying to figure out what was happening, and then somebody noticed some small error that I had, and I fixed it, and everything worked. **And I just sort of sat there and cried for a little bit.**”

- “*Low hanging (qualitative) fruit*”
- Qualitative research into pair programming now needs to connect to theory.

Eckerdal et al., “From Limen to Lumen”

- A welcome connection to a “theory”
 - If threshold concepts is a theory
 - How does threshold concepts relate to cognitive theory?
- “... *the student is being transformed ... acquiring a new identity, that of an insider ... This project fits squarely within the constructivist tradition*” (!?)
 - Back sliding objectivists?
 - Or a welcome attempt at transcending ye olde constructivist vs. objectivist dialectic?



General observation

Analysis is ...

“Through the Eyes of Instructors: A Phenomenographic Investigation of Student Success”, Kinnunen, McCartney, Murphy, and Thomas

- 📁👤 Nature of the subject
 - 📁👤 No papers from that perspective
- 📄👤 Intrinsic – the “geek gene”
- 📄👤 Previous Experience
- 📄👤 Attitude / Behaviour
- 📄👤 Developmental



In summary

- More qualitative than quantitative.
- Not a lot of theory
- Getting better (with experience) at method


Programming by permutation - Wikipedia, the free encyclopedia - Mozilla Firefox

File Edit View History Bookmarks Tools Help

W http://en.wikipedia.org/wiki/Programming_by_permutation Google

Customize Links Free Hotmail Windows Media Windows

Sign in / create account

 **WIKIPEDIA**
The Free Encyclopedia

navigation

- Main page
- Contents
- Featured content
- Current events
- Random article

interaction

- About Wikipedia
- Community portal
- Recent changes
- File upload wizard
- Contact us
- Make a donation
- Help

search

Go Search

toolbox

- What links here
- Related changes
- Upload file

article discussion edit this page history

Your continued donations keep Wikipedia running!

Programming by permutation

From Wikipedia, the free encyclopedia

Trying to approach a solution to a programming problem by iteratively making small changes ([permutations](#)) and testing each change to see if it behaves as expected is called "programming by permutation". This approach sometimes seems attractive when the programmer does not fully understand the code, and believes that one or more small modifications may result in code that is correct.

This tactic is rarely productive because:

- a series of small modifications can easily introduce bugs into the code, leading to a "solution" that is even less correct than the starting point
- many false starts and corrections usually occur before a satisfactory endpoint is reached
- it is rarely possible to measure, by empirical testing, whether the solution will work for all cases
- in the worst case, with poor code management, the original state of the code may be irretrievably lost

Programming by permutation gives little or no assurance about the quality of the code produced -- it is the polar opposite of [Formal verification](#).

Example

[\[edit\]](#)

For example, the following code sample (intended to find and copy a series of digits from a larger string) has several problems:

```
char* buffer = "123abc";
char destination[10];

int i=0;
int j=0;
int l = strlen(buffer);
while (i<l)
{
    if (isdigit(buffer[i])) destination[j++] = buffer[i++];
}
```

Find: education Next Previous Highlight all Match case

Done

Start SSH, Telnet and R... talks Programming by pe... Document1 - Microsoft ... 2007_07_NACCQ_List... EN << 4:04 PM

Cargo cult programming - Wikipedia, the free encyclopedia - Mozilla Firefox

File Edit View History Bookmarks Tools Help

W http://en.wikipedia.org/wiki/Cargo_cult_programming

Customize Links Free Hotmail Windows Media Windows

Sign in / create account

article discussion edit this page history

Your continued donations keep Wikipedia running!

Cargo cult programming

From Wikipedia, the free encyclopedia

Cargo cult programming is a style of [computer programming](#) that is characterized by the ritual inclusion of code or program structures that serve no real purpose. Cargo cult programming is typically symptomatic of a programmer not understanding either the bug they were attacking or the reasons behind adding comments in forms of code.

“the ritual inclusion of code or program structures that serve no real purpose”

The term '[cargo cult](#)', as an [idiom](#), typically refers to aboriginal religions which grew up in the South Pacific after [World War II](#). The practices of these groups centered on building elaborate mock-ups of airplanes and military landing strips in the hope of summoning the god-like airplanes that had brought marvelous cargo during the war. Use of the term in computer programming probably derives from [Richard Feynman's](#) characterization of certain practices as [Cargo cult science](#).

Contents [\[hide\]](#)

- [1 Cargo cult software engineering](#)
- [2 See also](#)
- [3 References](#)
- [4 Further reading](#)
- [5 External link](#)

Cargo cult software engineering

[\[edit\]](#)

A related term in [software engineering](#) is *cargo cult software engineering*, coined by [Steve McConnell](#). McConnell describes software development organizations that attempt to emulate more successful development houses, either by slavishly

Find: education Next Previous Highlight all Match case

Done


Start SSH, Telnet a... talks Cargo cult prog... Document1 - Micr... Presentation1 2007_07_NACCQ... EN << 4:02 PM

Voodoo programming - Wikipedia, the free encyclopedia - Mozilla Firefox

File Edit View History Bookmarks Tools Help

W http://en.wikipedia.org/wiki/Voodoo_programming Google

Customize Links Free Hotmail Windows Media Windows

 [Sign in / create account](#)

[article](#) [discussion](#) [edit this page](#) [history](#)

Your continued donations keep Wikipedia running!

Voodoo programming

From Wikipedia, the free encyclopedia

Voodoo programming (a term derived from [voodoo economics](#)) is a [tongue-in-cheek](#) term for using a [programming](#) device, system or language which one does not fully understand. The implication is that the end result should not actually work, or even if it does work one does not understand *why* it works properly. The term can also apply to doing something which you know should not work, but actually does work, such as successfully [recompiling](#) some code which refused to compile the first time. Some voodoo programming is probably due to spurious glitches, subtle bugs (such as uninitialized data), or incorrect/misleading documentation in the compiler, APIs, or OS.

It is similar to [black magic](#), except that black magic typically isn't documented and *nobody* understands it.

A person with experience of voodoo programming is sometimes called *medicine man* or *witch doctor*, such as "Java medicine man" or "C++ witch doctor" as equivalent for *guru* or *wizard*; the traditional terms imply high sophistication, study and knowledge over the matters and discipline, while voodoo programming implies getting things working but not fully understanding why.

See also

- [Anti-pattern](#)

This article was originally based on material from the [Free On-line Dictionary of Computing](#), which is licensed under the [GFDL](#).

Category: [Anti-patterns](#)

Find: Highlight all Match case

Done


Shotgun debugging - Wikipedia, the free encyclopedia - Mozilla Firefox

File Edit View History Bookmarks Tools Help

W http://en.wikipedia.org/wiki/Shotgun_debugging

Customize Links Free Hotmail Windows Media Windows

Sign in / create account


WIKIPEDIA
The Free Encyclopedia

navigation

- Main page
- Contents
- Featured content
- Current events
- Random article

interaction

- About Wikipedia
- Community portal
- Recent changes
- File upload wizard
- Contact us
- Make a donation
- Help

search

Go Search

toolbox

- What links here
- Related changes
- Upload file

Find: education Next Previous Highlight all Match case


Done

article discussion edit this page history

Your continued donations keep Wikipedia running!

Shotgun debugging



From Wikipedia, the free encyclopedia

 It has been suggested that this article or section be merged into *Programming by permutation*.
(Discuss)

Shotgun debugging is a process of making relatively undirected changes to software in the hope that a **bug** will be perturbed out of existence. This almost never works except in very simple programs, or when used as an attempt to work around **programming language** features that one may be using improperly; it usually introduces more bugs. These undirected, random changes can however cause more symptoms to occur, which assists in locating (and therefore fixing) problems.

Shotgun debugging can occur when working with **multi-threaded** applications. Attempting to debug a **race condition** by adding debugging code to the application is likely to change the speed of one **thread** in relation to another and could cause the problem to disappear. Although apparently a solution to the problem, it is a fix by pure chance and anything else that changes the behaviour of the threads could cause it to resurface — for example on a computer with a different **scheduler**. Code added to any part of the program could easily revert the effect of the "fix".

*This article is based in part on the *Jargon File*, which is in the public domain.*

 This *computer science*-related article is a *stub*. You can help Wikipedia by expanding it .

Categories: Articles to be merged since May 2007 | Computer science stubs | Anti-patterns

Start | 3 SSH, Telnet and R... | talks | Shotgun debugging ... | Document1 - Microsoft ... | 2007_07_NACCQ_List... | EN | << 4:03 PM



Preliminaries

- Sally & Josh, “Warren’s Question”
 - It’s not safe to say those things in your own institution
 - The Disciplinary Commons is a safe place
- ICER is **not** a commons
 - It’s a research conference, but ...
 - Can we find a way of critically engaging that doesn’t involve some of the traditional research bullshit?