Empirical Studies of Designers

Seth Holloway
David DeAngelis

EE382V: Software Architecture and Design Intent
Electrical and Computer Engineering
The University of Texas at Austin

Outline

- Questions
- High level overview
- Elevator design problem
- A model of cognitive processes in software design: An analysis of breakdowns in early design activities by individuals
- Designing the Design Process: Exploiting Opportunistic Thoughts
- Questions and comments

Question #1

- What are typical personality traits of gifted designers?

Answer #1

- The Hacker [Designer] Attitude
  - The world is full of fascinating problems waiting to be solved.
  - No problem should ever have to be solved twice.
  - Boredom and drudgery are evil.
  - Freedom is good.
  - Attitude is no substitute for competence.

Answer #1

- Bryan Dollery asserts that programmers are creative, artistic people who feel the “flow” of coding
- Possible link to autism (Asperger’s syndrome)
  - "I think *all* tech people are slightly autistic”-- *Microserfs* novelist Douglas Coupland
  - Richard Stellman (founder of GNU) and Bram Cohen (creator of BitTorrent) both have self-diagnosed Asperger’s syndrome

Question #2

- What benefits would studying designers have?

Answer #2

- This motivates an approach in which languages and tools are developed based on knowledge gained from empirical studies of programmers.
- This knowledge, applied within a tool development process, can lead to better support for programmers and software engineers.
- It can result in models of programmers and their tasks. It can result in data to compare different approaches to supporting programmers.

Question #3

- What are some issues in designing experiments for programmers?

Answer #3

- Humans are hard to study without using illegal or intrusive means. Many studies have to rely on external monitoring or interviews, however these threaten validity.
- Programmers being studied are students
- Not enough programmers are sampled
- Programmers are too similar in skill, culture, age, experience, background, etc
- No standard title for software designers, so it is harder to define and thus find “designers”
- We have determined what they know rather than how they know it

Empirical Studies of Designers

- The field seeks to understand designers in an effort to improve the programming experience (increase productivity, ease development, improve accessibility, enforce known-good programming practices)

Empirical Studies of Designers

- Research in the field includes
  - Comparisons of Expert vs. Novice programmers.
  - Models and strategies of program comprehension.
  - Models and strategies used when writing programs.
  - The importance of knowledge representation vs. strategies.

The Elevator Problem

- How would you program a system to efficiently control 2 elevators covering 10 floors?

Analysis of Breakdowns

- "A model of cognitive processes in software design: An analysis of breakdowns in early design activities by individuals" by Raymonde Guindon, Bill Curtis, and Herb Krasner, 1987.

Experimental Setup

- Elevator control problem
- Move ***n*** elevators between ***m*** floors
- 2 hours to develop logic "thinking aloud"

Experimental Setup

8 developers narrowed to 3 "best" subjects

- P6
  - PhD in Electrical Engineering with more than 10 years of professional experience
- P8
  - MS Software Engineering with 5 years of experience
- P3
  - PhD candidate in Computer Sciences with 3 years of experience

Designers' Approaches

- P6
  - Specialized design schemas
  - Issue-driven
  - Generation of simplifying assumptions
- P8
  - Less focused, less certain than P6. Uses a meta-schema to explore problem space at different level of abstraction. Attempts to apply partial solutions universally
- P3
  - Characterized by chaotic generate-test-debug design
  - Has problems doing mental simulations

Solutions

- P6
  - Communicating ring of distributed, independent elevators governed by FSMs (one for individual elevator and one for group).
- P8
  - Star architecture communicating through a central server. Design includes abstract data types, data flow diagrams and pseudocode.
- P3
  - Works on a central server system; represents behavior of the system by logical assertions

Breakdowns

- Knowledge-related breakdowns due to
  - lack of specialized knowledge of similar problems
  - lack of experience as a designer
  - lack of domain knowledge
- Cognitive limitations breakdowns result from
  - not enough working short-term memory (solution is too large)
  - unreliable retrieval of information from long-term memory
- Combination breakdowns caused by
  - Lack of specialized knowledge forces developer to use more cognitively-costly designs

Key Ideas

- Breakdowns result from a lack of knowledge
- The more schemas a designer knows, the quicker and more elegant the design
- Greater knowledge of possible solutions leads to designs with greater rationale

Exploiting Opportunistic Thoughts

- “Designing the Design Process: Exploiting Opportunistic Thoughts” Raymonde Guindon, 1990.

Design

- What is it?
  - Transform Spec to high level (semi) formal notation
    - Subsystems, info flow, data structures, interfaces
    - Most expensive errors
  - Everything you can do in advance to make coding easier.

Design

- Why is it hard?
- Ill-Structured Problem
  - Incomplete, ambiguous specs of goals
  - No predetermined solution path
    - a system may require novelty
  - Integrate several knowledge domains
    - problem domain, architecture, computer science

Top-Down Design

- Like breadth first search
- Overall system aspects designed first
- Progressively decomposed into subsystems with greater detail
- Fails in real systems
  - Designer faces novelty
  - Integration of multiple knowledge sources
  - Sub-problem is critical, difficult, or has an immediately known solution

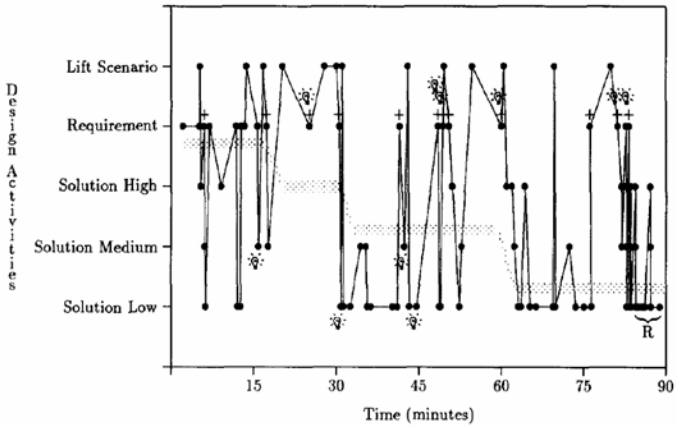
Opportunistic Design

- Data driven rules & associations
- Partial solutions
- If an opportunity is presented, follow it.
  - Worry about the bookkeeping later.

Experiment

- Same 8 designers were given the lift control problem (elevator problem)
- 2 hours, must produce a design solution with enough detail to be implemented by a programmer
- Thinking aloud reports were recorded, time-stamped notes, debriefing sessions
- Cog. Psychologist reviewed the sessions, prompted review session with participant

Results



Summary

- Top down decomposition is not as useful in practice as once thought
- Early stages of design are opportunistic
  - Bounce around various levels of abstraction
  - Some top-down decomposition when the thread is lost
- Supports eXtreme Programming

Review

- Questions
- High level overview
- Elevator design problem
- A model of cognitive processes in software design: An analysis of breakdowns in early design activities by individuals
- Designing the Design Process: Exploiting Opportunistic Thoughts
- Questions and comments

Questions? Comments.

- http://www.wired.com/wired/archive/9.12/aspergers.html
- **Autistic people have a hard time multitasking - particularly when one of the channels is face-to-face communication. Replacing the hubbub of the traditional office with a screen and an email address inserts a controllable interface between a programmer and the chaos of everyday life. Flattened workplace hierarchies are more comfortable for those who find it hard to read social cues. A WYSIWYG world, where respect and rewards are based strictly on merit, is an Asperger's dream.**

- **http://www.faifzilla.org/ Richard Stellman's Biography**
- **Richard Stellman, famed hacker and the creator of GNU Project, suggests that hackers have failed at other feats and fallen into programming**

Analyzing the Usability of a Design Rationale Notation

- **Semiformal, argumentation-based notations are one of the main classes of formalism currently being used to represent design rationale (DR). However, our understanding of the demands on designers of using such representations has to date been drawn largely from informal and anecdotal evidence. One way to tackle the fundamental challenge of reducing DR's representational overheads, is to understand the relationship between designing, and the idea structuring tasks introduced by a semiformal DR notation. Empirically based analyses of DR in use can therefore inform the design of the notations in order to turn the structuring effort to the designers' advantage. This is the approach taken in this chapter, which examines how designers use a DR notation during design problem solving. Two empirical studies of DR-use are reported, in which designers used the QOC notation (MacLean et al., this volume) to express rationale for their designs. In the first study, a substantial and consistent body of evidence was gathered, describing the demands of the core representational tasks in using QOC, and the variety of strategies which designers adopt in externalising ideas. The second study suggests that an argumentation-based design model based around laying out discrete, competing Options is inappropriate during a depth-first, 'evolutionary' mode of working, centered around developing a single, complex Option. In addition, the data provide motivation for several extensions to the basic QOC notation. The chapter concludes by comparing the account of the QOC–design relationship which emerges from these studies, with reports of other DR approaches in use.**

Supporting Systems Development by Capturing Deliberations During Requirements Engineering

- **Support for various stakeholders involved in software projects (designers, maintenance personnel, project managers and executives, end users) can be provided by capturing the history about design decisions in the early stages of the system's development life cycle in a structured manner. Much of this knowledge, which is called the process knowledge, involving the deliberation on alternative requirements and design decisions, is lost in the course of designing and changing such systems. Using an empirical study of problem-solving behavior of individual and groups of information systems professionals, a conceptual model called REMAP (representation and maintenance of process knowledge) that relates process knowledge to the objects that are created during the requirements engineering process has been developed. A prototype environment that provides assistance to the various stakeholders involved in the design and management of large systems has been implemented.**