COSC345 Week 7

Prototyping and User Evaluation

22 August 2017

Richard A. O'Keefe

Boehm's spiral model in a nutshell}

Planning determine objectives, alternatives, constraints

Risk analysis analyse alternatives, identify risks

Engineering develop the "next-level product"

Customer evaluation assess the results of engineering

Prototyping is concerned with *risk resolution* and customer evaluation.

Prototyping is about risk reduction

Ignorance breeds risk: what should detailed requirements be? Will an algorithm be efficient enough? What should the user interface be like? Will the new operating system work? Are the various products implementing (new buzzword) compatible?

If you have no question that's worrying you, *don't* build a prototype.

Developers don't think like users, so UI is usually a worry.

The Prototyping Cycle (Pressman p.27)

1. Gather requirements
2. Do quick design
3. Build cheap prototype
4. Evaluate prototype (maybe with customer)
5. If good enough, build real product.
6. Otherwise apply lessons and go back to 2.

“Cheap” is relative

INCIS and Novopay have shown the troubles that inadequate requirements can lead to in New Zealand.

The Queensland Health Payroll system blew out from \$6·2 million to \$1·2 milliard in large part because of inadequate requirements.

New Zealand approved a new Inland Revenue computer system budgeted to be \$1 milliard with contingency planning for \$1·5 milliard. What if *that* goes wrong on the same scale?

Spending \$10 million on prototypes for *that* could save milliards.

Kinds of Prototypes

— paper (stack of index cards, sketches of screens) as long as it can be evaluated.

— “gutless GUI” as produced by SpecTcl.

— working prototype with subset functionality

— working prototype with reduced performance

— working prototype with cost reduced some other way

— existing program that does some of the job

Developers and users are different

Developers tend to be deductive thinkers who like clarity and direction, to be introverts, and to score higher on autism scales. If we didn't like computers we wouldn't *be* developers.

Users tend to tolerate ambiguity and conflict better, to be extroverts, perhaps to be more intuitive.

Developers want crisp stable requirements. Users find it hard to provide them, but they aren't dumb. We just need to construct the requirements *together*.

Prototypes are for talking about

— Developer (mis)understands user

— Developer builds prototype

— User can see what developer means

— User can talk to developer about specifics

— Developer can change prototype cheaply

A Prototype is not a product (others)

— Customer sees “working” system, thinks product nearly ready.

So agree to prototype for requirements elicitation *only*.

— A manager does the same.

So get management approval for prototyping process and make prototype obviously unfinished.

A prototype is not a product (us)

— Developer gives in to demands to “fix” and ship the prototype. Result buggy in the extreme.

So don’t do that.

— Prototype shortcuts applied to real thing wrongly

So document design decisions; don’t economise there.

A product is not a prototype

Just because it's written in a language that is useful for prototyping doesn't mean a program is a prototype.

If a program was written to meet the functional and non-functional requirements of a project, and does meet them, it's a product, not a prototype.

Rewriting from a high level language to a low level one like C, C++, Java, or C# takes a lot of time and money, introduces additional bugs, and makes the program harder to maintain.

“Traditional” Languages

Fortran/Algol/C/Java

thin library, thick program

bulk of time in your code

major consequences from your errors

types give speed and safety

fight the compiler to a draw to run

Scripting Languages

sh/AWK/TCL/Lua/Python

thick library, thin program

bulk of time in your code

your errors can’t corrupt library

dynamic types for agile programming

code not run needn’t be type correct

Extension languages

Lua/elk/VBA/AppleScript/EmacsLisp

Large application in traditional language

Users customise and extend

GLIM/GenStat/SAS/R and Darwin here.

Prototyping languages 1

Essence of prototyping is to *build* and *change* prototype *quickly* and *rapidly*.

Most important factor: relevant libraries, *e.g.*, simple GUI kit, file handling kit, database interface, network interface, presentation graphics, numerical or statistical analysis kit, HTML or XML or SGML or JSON reader/writer . . .

N.B.: Java libraries can be called from Ada, Eiffel, Scheme, Groovy, Scala, Javascript . . . , but not C/C++.

Example: a visitor wrote a picture to cartoon program at Singapore airport, *given* a library with almost all he needed.

Prototyping languages 2

Debugging costs time: choose a language which makes bugs hard to write or easy to find. Look for automatic storage management, array bounds checks, unbounded strings. Look for unbounded arithmetic (Lisp bignums and ratios) or arithmetic overflow checks. Look for built-in relation or hash tables support. Variable trampling bugs: consider functional or logic based language. For text processing, look for regular expression and grammar support (via lex/yacc approach or special syntax). Consider an interactive language (not just IDE). Type system: either dynamic or less crippled than C/C++. Is network support required? Is XML support required? Is ASN.1?

Prototyping a spelling checker

```
#!/bin/sh
DICT=/usr/share/dict/words
case "$1" of
  *.tex)                 detex   "$1" ;;
  *.[1-9nt]*)            deroff  "$1" ;;
  *.sgm*|*.htm*|*.xml) unxml   "$1" ;;
  *.[hc]|*.java|*.erl) m2h -ix "$1" ;;
  *)                     cat     "$1" ;;
esac | ids -W -R | sort -u | comm -23 - "$DICT"
```

Prototyping language list 1

APL Funny alphabet, powerful tool, good Windows support

AWK Traditional UNIX prototyping language

Erlang Ideal for network prototyping

Haskell See ~`ok/COSC459/2000.d/index.html`

Java Huge libraries but clumsy (Java/Erlang about 5)

Lisp The King; world's most powerful OOPL

Prototyping language list 2}

Perl For puzzle addicts, lots of packages around

Python Clean, popular, 3D and numerics available

Prolog great for grammars, NLP, relations

Python Perl done better

Scala Java libraries but much nicer language

Smalltalk *the* OOPL, commercial+free & *fun*

Visual Basic For GUI prototypes in Windows&/Office

Evaluation is important

It's not enough to *build* a prototype, you must *evaluate it*.

Build prototype because you have a question; decide beforehand how prototype helps you find answer; how to evaluate prototype; what to do about answers.