Review of David L. Parnas’ “Designing Software for Ease of Extension and Contraction”

Alexander Sverdlov <alex@theparticle.com>

March 12th, 2004

Abstract
This talk reviews David L. Parnas’ paper titled “Designing Software for Ease of Extension and Contraction”, discusses some of the key ideas, and relates them to current times.
Introduction

- A way of viewing software requirements.
- Spotting subset requirements.
- Guessing extensions.
- Building software as a family of programs, with each program solving some distinct problem.
Requirement Sub-sets

Assuming reasonable requirement variation, what pieces are ‘constant’? (or are least impacted) by likely changes.

- Finding sub-sets.
- Starting implementation from these sub-sets.

- ‘Uses’ Relation
Requirement Extensions

After implementing the sub-sets, what other possible likely changes (or extensions) to requirements do we normally have to care about.

- We cannot predict these (no more than we can predict the future).
- Need to design system to be flexible.
Family of Programs

- Build software as many programs instead of one monolithic solution.
- Let programs use each other: the Uses Relation
Use Relation

We don’t want to duplicate functionality that may be common to many programs in the family-of-programs.

- If program $A$ needs functionality found in program $B$, then program $A$ should use program $B$ to accomplish the task.

- This needs clear inter-program interfaces (a real world example may be UNIX pipes)
Use Relation, Cont.

- The ‘uses’ relation must not create cycles. If a program $A$ is used by program $B$, then program $A$ shouldn’t use program $B$.

- The used programs should be very generally written to handle a specific task.
Chain of Programs

A software system may do processing by passing data from one program to the next, with each doing its own bit of processing before passing it on to another program (until it reaches the end of the line, which would be the output).

- Successfully implemented in UNIX, via ‘pipes’.

- Can cause problems if format is transformed (then pipe components aren’t removable—even if they’re not needed).
Comparison to Modern Day

Did the world improve?

- Most operating systems have many small programs to do distinct tasks.
- Sub-programs are generally not used (nor applicable) in simple situations (especially those presented in the article; the address book).
- Use of libraries instead of ‘programs’.
Not much Progress

As great as this paper is, and with many having tried these approaches (OOP, etc.) the software industry is still very much faced with exactly the same issues this paper is trying to address.

Software is still very difficult to develop in such a way that makes it easy to change in face of changing requirements, and easily expandable if needed.

This is the holy grail of software development.
Great educational & enlightening paper. Not very ‘real world’ practical due to its age, and the general lack of applicability to the way software is built today (very few people think of creating a few dozen separate executables to solve a single problem).