Out Of The Tar Pit(1-6)

I. Introduction

- Complexity
  - large-scale software systems 의 개발과 유지하는 것에서 가장 걸림돌이 되는 것
  - major: *state* handling
  - others: *code volume*, *flow of control*
- FP(functional programming) and OOP(object oriented programming) 양쪽에서 large-scale software systems을 handling하는 전략을 취해 소개하는 paper이다. (본 요약에서는 다루지 않음.)

II. Complexity

- Software 를 어렵게 만드는 4가지 속성이 ‘No Silver Bullet(Brooks)’ 에서 소개됨.
  : (1) **Complexity**, (2) Conformity, (3) Changability, (4) Invisiblity
- “The general problem with ambitious systems is complexity.”
  : 소프트웨어 개발에서 다양한 문제는 시스템에 대한 이해의 부족에서 야기되는데, complex 할수록 개발자가 시스템을 이해하기가 어렵다.
- **Simplicity is *Hard***

III. Approaches to Understanding

- Systems(or components of systems) 을 이해하기 위해 보통 사용되는 방법이 다음 두가지임.
- Testing
  : System을 블랙 박스로 두고, 그 바깥에서 이해해보려 하는 시도.
  : human - whole-system | machine - individual component
- Informal Reasoning
  : System inside에서 examine해보는 것.

- Testing은 bug 를 발견할 수 있도록 한다. 하지만 한계가 있고 informal reasoning 이 반드시 쓰여야 한다. Informal reasoning 으로 개선한다면 에러가 보다 적게 발생된다.

IV. Causes of Complexity

1. Caused By State
    a. state 의 존재 자체.
    b. Testing
    : 특정 state 에 대한 Testing 은 다른 stated 에 대해서는 어떠한 정보도 제공해주지 않는다. 또한 테스트들은 hidden internal state 에 관계 없이 같은 방식으로 동작할 것으로 기대되지만 실제로는 hidden internal state에 문제가 있을 수 있다.
    : Different set of inputs와 함께 testing의 불확실성의 주 원인이다.
    c. Informal Reasoning
    : State 에 관한 single bit의 추가는, state 수의 double로 이어진다. 즉 exponential하게 많아지고, 이는 testing과 informal reasoning 둘 다 복잡하게 만든다.
    : Contaminated
2. Caused By Control
: Order in which things happen

> When control is required to be specified, it is being forced to specify **how** the system should work rather than **what** is desired. This leads to **over-specifying** the problem, like choosing an arbitrary control flow in specifying only a relationship between values.

    a. Code 를 읽고 분석할 때, 순서를 신경써야 한다고 먼저 가정하고 읽은 후, 순서를 정말 고려해야하는지 판단하고 이후 작업을 수행한다. 이때, 잘못된 판단은 찾기 힘든 버그를 발생시킬 수 있다.
    b. Concurrency
        a. “shared-state concurrency” 이 가장 일반적.
        b. 프로그램의 분석에서 고려할 *the number of scenarios* 를 추가하는 데에 영향을 줌.

c. 시스템에서 test를 반복할 때 일관성을 보장할 수 없음.

3. Caused By Code Volume
   : 코드 볼륨의 증가는 곧 다른 문제로 이어진다.
4. Others
   - Complexity breeds complexity
   - Simplicity is Hard
   - Power corrupts

V. Classical approaches to managing complexity

1. OO
   a. State
      - Object : set of states + set of procedures for accessing and manipulating (Encapsulation)
      - Problems
        : 동일한 bit state 에 여러 precedure 로 접근가능하거나 관리할 때, constraint가 여러 위치에 걸려야 한다.
        : Encapsulation-based integrity constraint enforcement 는 single object constraint 에 적용되도록 최적화되어 있어 여러 object 의 복잡한 조건에 적용되기 어렵다.
   b. Control
      : “Shared-state concurrency” problem
2. FP(pure form)
   a. State
      : Avoiding state and side effects
   b. Control
      : Left-to-right sequencing은 있고, implicit sequencing에 대한 문제는 그대로이다.
      : explicit loop 보다 fold, map과 같은 funcional form 을 권장하므로 control 에 있어서 장점이다.

c. Kinds of State

d. State and Modularity

3. Logic Programming

   a. State
      : Pure 한 LP는 mutable state 를 사용하지 않는다. 하지만 많은 언어들이 stateful mechanism 을 지원하고, 그런 경우에 관련 문제가 발생한다.

   b. Control
      : Pure prolog 는 processing of sub-goals 을 위한 left-to-right implicit ordering 과 clause application 의 top-down implicit ordering 을 모두 가지고, 관련 문제점도 가진다.

VI. Accidents and Essence

1. Essential Complexity
   : User 에게 내장된 문제로, 이상적인 상황에서도 고려되어야 함.

2. Accidental Complexity
   : Essential complexity 외의 모든 것. 개발자가 처리하는 것을 지향.

: Software 의 고유한 특성에 complexity 가 있는 것은 아니다.