

# 소프트웨어 버그 관련 기본 개념

한동대학교 전산전자공학부

김인중

(ijkim@handong.edu)

# 목차

---



- 버그란 무엇인가?
- 자주 발생하는 버그 유형
- 버그에 관한 연구 결과들
- 버그와 프로젝트 규모
- 버그 최소화를 위한 방법론
- 질의 응답

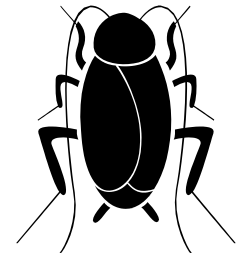
# 버그란 무엇인가?

## ■ 소프트웨어 버그란?

- 컴퓨터 프로그램이 의도한대로 수행되지 못하게 하는 error, flaw(결함), mistake, failure, or fault

## ■ 버그는 얼마나 많은가?

- 출시된 소프트웨어 평균: 1000 line 당 1~25 개
- Microsoft
  - 내부 테스트: 1000 line 당 10~20 개
  - 출시 제품: 1000 line 당 0.5 개



## ■ 버그는 개발 과정에 얼마나 영향을 미치는가?

- 개발 기간의 50% 이상은 디버깅에 소모됨
- 프로젝트의 규모가 클 수록 더욱 심각해짐.

# 컴퓨터에서 발견된 최초의 진짜 버그

9/9


0800 Antan started  
 1000 " stopped - antan ✓

1300 (032) MP-MC { 1.2700 9.037 847 025  
 (033) PRO 2 2.130476415 9.037 846 795 correct  
 correct 2.130676415 4.615925059(-2)

Relays 6-2 in 033 failed special speed test  
 in relay .. 10.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)  
 1525 Started Multy Adder Test.

1545  Relay #70 Panel F  
 (moth) in relay.

First actual case of bug being found.

1630 Antan started.  
 1700 closed down.

Relay 2745  
 Relay 3370

# 버그란 무엇인가?

## ■ 버그는 얼마나 줄일 수 있는가?

### ■ “Clean room development”

- 내부 테스트: 1000 line 당 3 개
- 출시 제품: 1000 line 당 0.1 개

cf. 공식적 개발 방법론 / 상세검토 / 통계적 테스트 적용 시 0 / 500,000 line 수준 실현

### ■ “Team software process”: 처음부터 결함을 만들지 않도록 개발자를 교육 및 개발과정 관리

- 1000 line 당 0.06개



# 버그는 누가 만드는가?



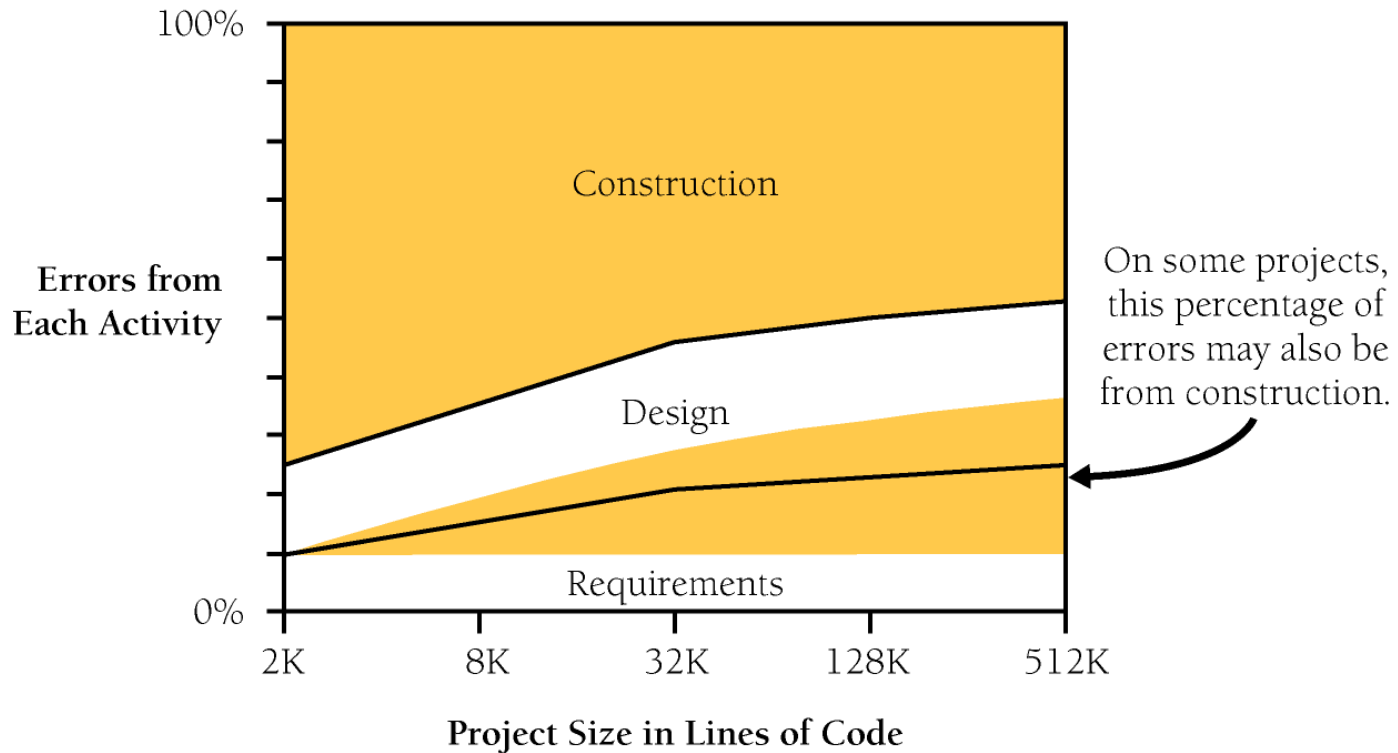
- 대부분의 버그는 개발자의 실수로 인한 오류

Cf. 오류 발생 원인에 대한 통계

- 95 %: 프로그램
- 2 %: 시스템 소프트웨어
- 2 %: 다른 소프트웨어
- 1 %: 하드웨어

➔ 버그를 찾을 때 일단은 개발자 자신의 실수로 가정하고 찾아야 한다.

# 소프트웨어 개발 단계별 오류 비율

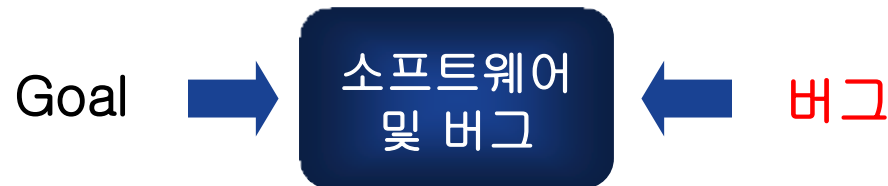


- 소규모 프로젝트 (1000 line 수준)의 경우
  - 75%: 코드작성 오류, 10%: 요구사항 오류, 15%: 설계 오류
- 대규모 프로젝트의 경우: 코드작성 오류가 35~75%

# 버그 없는 프로젝트를 위한 전략

## ■ 버그 없는 프로젝트를 위한 전략

1. 버그가 생기는 이유를 이해한다.



2. 그 이유에 대해 논리적이고 체계적인 대안 수립

- 개발자 차원의 대안
- 조직 및 시스템 차원의 대안

3. 실행에 옮긴다.

- 모든 개발자가 문제와 대안을 이해하게 한다.

예) 파라미터 체크에 의해 오류를 발견한 경우 타 개발자의 반응  
→ “ASSERT를 제거 요청!”





# 목차

---



- 버그란 무엇인가?
- 자주 발생하는 버그 유형
- 버그에 관한 연구 결과들
- 버그와 프로젝트 규모
- 버그 최소화를 위한 방법론
- 질의응답

# 일반적으로 자주 발생하는 버그들



## ■ Using uninitialized variables

```
int data[NoData];  
int sum;                      // uninitialized variable  
// some codes  
for(i = 0; i < NoData; i++)  
    sum += data[i];
```

## ■ Exceeding array bounds

```
for(i = 0; i < n; i++)  
    a[i] = a[i+1];             // what if i is n-1 ?
```

## ■ Off by one error

```
for(i = NoData - 1; i > 0; i--) { /* Body of the loop */ }
```

# 일반적으로 자주 발생하는 버그들

## ■ Accessing memory not owned (Access violation)

```
char *message;           // pointer to a garbage
strcpy(message, "Error");
```

## ■ Memory leak or Handle leak

- 할당되었으나 해제되지 않은 메모리, 또는 자원

## ■ Buffer overflow

## ■ Stack overflow or underflow

- 너무 많은 지역 변수 (대용량 array 또는 struct)
- 너무 깊은 함수 호출 (특히 recursion)

# 일반적으로 자주 발생하는 버그들

- **Divide by zero**

```
result = a / b;           // what if b == 0
```

- **Arithmetic overflow or underflow**

- **Loss of precision in type conversion**

```
int data[NoData];  
float ratio[NoData];  
for(i = 0; i < NoData; i++)  
    ratio[i] = data[i] / sum;           // sum is an integer var.
```

# 일반적으로 자주 발생하는 버그들

## ■ Infinite loops

```
for(i = 0; i < NoData; j++) { /* loop body */ }
```

```
for(i = 0; j < NoData; i++) { /* loop body */ }
```

```
while(i < NoData){  
    if(/* some condition */){  
        // some code  
        i++;  
    }  
}
```

## ■ Multi-thread, multi-process 환경에서의 오류

- Deadlock, race condition

# 버그 방지를 위한 전략

- 코딩 후 소스 코드 검토
  - Code checker 활용
- 모든 변수 초기화
- 함수 입출력에 대한 방어적 프로그래밍
  - 함수 입력 파라미터 체크 (오류처리 / assertion)
  - 함수 리턴 값 체크 (호출하는 함수 및 호출되는 함수 모두)
- 영역(**range**)의 **upper/lower** 표시 방법 통일  
for(i = 0; i < NoData; i++) { /\* loop body \*/ }
- 자원 할당과 해제는 모두 동일 함수에서 한다

# 목차

---



- 버그란 무엇인가?
- 자주 발생하는 버그 유형
- 버그에 관한 연구 결과들
- 버그와 프로젝트 규모
- 버그 최소화를 위한 방법론
- 질의응답

# 버그에 관한 연구 결과

## ■ Beizer의 오류 분류

오류의 종류	비율
구조적	25.18%
데이터	22.44%
구현된 기능	16.19%
구현	9.88%
통합	8.98%
기능적 요구 사항	8.12%
테스트 정의나 실행	2.76%
시스템, 소프트웨어 아키텍처	1.74%
기타	4.71%



# 버그에 관한 연구 결과

## ■ 실험 결과에 의한 유추 결과

- 대부분의 오류가 발생하는 범위는 상당히 제한되어 있다
  - 80% - 20%, 50% - 5%
  - 오류의 85%는 한 루틴을 변경함으로써 수정될 수 있다.
- 많은 오류들이 구현의 도메인 밖에 있다.
  - 도메인 지식 부족, 요구사항의 변동 및 모순, 의사소통/협동의 실패
- 대부분의 구현 오류들이 프로그램의 잘못이다
  - 시스템 S/W, 타 프로그램, H/W일 가능성은 5% 이내
- 구현 오류 중 오타 오류가 36 ~ 18%를 차지한다.

# 버그에 관한 연구 결과

## ■ 실험 결과에 의한 유추 결과 (계속)

- 프로그램 오류 중 설계를 잘못 이해하는 문제는 지속적으로 빈번히 발생한다.
  - 16%[Beizer90], 19%[Weiss75], ...
- 대부분의 오류들은 수정하기가 쉽다
  - 85%의 오류는 몇 시간 내, 15%의 오류는 며칠 내에 수정 가능
  - 그 이상 걸리는 경우는 1% 이내이다.
- 자신이 속한 조직의 오류에 대한 경험을 측정하는 것은 매우 유익하다.

# 목차

---



- 버그란 무엇인가?
- 자주 발생하는 버그 유형
- 버그에 관한 연구 결과들
- 버그와 프로젝트 규모
- 버그 최소화를 위한 방법론
- 질의응답

# 버그는 왜 발생하는가?



## ■ 다음 프로젝트에서 버그 없는 개발이 가능한가?

### 1. Data Structures HW #3: “Stack 구현”

- 개발자 수: 1명
- 소스 코드 크기:  $\leq 200$  line, 3개의 source file

### 2. Project ALPHA

- 개발자 수: 10명
- 소스 코드 크기: 수 십만 line, 수 백 개의 source file

# 버그와 프로젝트 규모의 관계

## ■ 프로젝트의 규모가 커질 때의 문제점

- 개발자들이 기억해야 할 정보량이 급격히 증가한다.
  - Data type 및 변수
  - 각 모듈, function의 기능
  - 오류 및 예외 발생시 현상
  - Pre/post condition, ...
- 코드의 각 부분이 만족해야 하는 조건들이 늘어난다.
- 개발자들간 교환해야 할 정보량이 급격히 증가한다.

# 버그와 프로젝트 규모의 관계



- 그러나, 개발자들의 기억 용량은 한계가 있다. 따라서...
  - 프로젝트의 규모가 커질수록 버그를 막는 것은 어려워 진다.
  - 대규모 프로젝트를 위한 방법론은 소규모 프로젝트를 위한 방법론과 달라야 한다.
  - 버그를 예방하기 위해서는 기억, 교환해야 하는 정보의 양과 만족해야 하는 조건을 최소화 해야 한다.

# 문제를 심화시키는 것들



## ■ 개발 종료일에 즈음하면...

- 버그를 방지하기 위한 충분한 검토 없이 코드의 추가/수정이 다량으로 이루어진다.

## ■ 개발 종료일에 즈음하여 개발자의 추가할 경우

- 개발을 위해 기억해야 할 정보를 기억하지 못한 신규 개발자들은 다량의 버그를 생산할 수 있는 잠재력을 갖는다.

# 목차

---



- 버그란 무엇인가?
- 자주 발생하는 버그 유형
- 버그에 관한 연구 결과들
- 버그와 프로젝트 규모
- 버그 최소화를 위한 방법론
- 질의응답



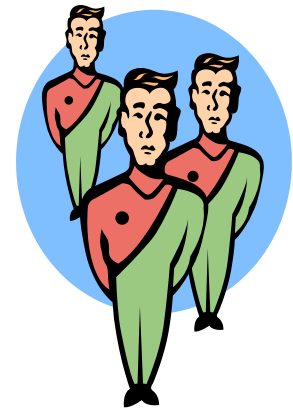
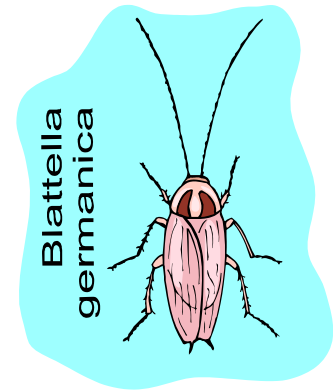
# 버그를 최소화를 위한 방법론



- 버그를 완전히 막을 수는 없으나 체계적인 방법론에 의해 크게 줄일 수 있다.
  - 소프트웨어 품질 관리를 실시할 경우 오류발생은 감소하지만 전체적인 개발 비용은 증가하지 않는다.
- 버그 최소화를 위한 방법론
  - 개발자 차원의 예방 방법
  - 조직 및 시스템 차원의 예방 방법

# 개발자 차원의 버그 예방

- 버그에 대한 올바른 접근 태도
- 버그의 발생 원인과 패턴에 대한 이해
  - 같은 버그를 반복하지 않기 위한 예방책
- 프로그래밍 기술 및 습관
  - 방어적 프로그래밍
  - 예외처리
  - 디버깅
- 코딩 후 검토
- 적절한 도구 이용
  - 디버거, static/dynamic code checker
- ETC.



# 조직 및 시스템 차원의 버그 예방

- 소프트웨어 품질에 관한 목표 설정 및 시스템적 반영
- 결함 감지 방법론
  - 요구 분석 / 설계 / 테스트 전략  
Ex) Quality Gate
- 코딩 스타일 표준화
- 적절한 도구 이용
  - Source code control, bug tracker, static/dynamic checker, ...
- ETC.



# 참고 문헌 및 추천 도서

---



- **Writing Bug-Free C Codes**
- **Code Complete 2<sup>nd</sup> edition**
  - Steve McConnell, Microsoft Press
- **Effective C++, 3<sup>rd</sup> edition**
  - Scott Meyers, Addison Wesley
- **More Effective C++**
  - Scott Meyers, Addison Wesley
- **Error Types**
- **GNU coding standard**
- **Bug Free Programming**
- **Exception Handling in C without C++**
- **Building bug-free O-O software**

# 질의 & 응답

---



# 감사합니다!

---

