조직 및 시스템 차원의 버그 최소화

한동대학교 전산전자공학부
김인중
(ijkim@handong.edu)

목차

- 소프트웨어 품질에 관한 목표
- 결함 방지 방법론
- 코딩 스타일 표준화
- 버그 추적 시스템
- 유용한 도구
- 질의 응답

소프트웨어 품질에 관한 목표

- 외적인 품질 특성
  - 정확성(correctness)
  - 유용성(usability)
  - 효율성(efficiency)
  - 신뢰성(reliability)
- 내적인 품질 특성
  - 유지 보수성(maintainability)
  - 유연성(flexibility)
  - 이식성(portability)
  - 재사용성(reusability)
  - 가독성(readability)
  - 테스트 용이성(testability)

품질 특성간의 상호 관계

- 사람에 따라 우선 순위가 다르다
- 일부 품질 특성은 서로 대립된다.

아래에 있는 요소가 오른쪽에 미치는 영향	정확성	유용성	효율성	신뢰성	무결성	적용성	정밀성	견고성
정확성	↑		↑	↑			↑	↓
유용성		↑				↑	↑	
효율성	↓		↑	↓	↓	↓	↓	
신뢰성	↑			↑	↑		↑	↓
무결성			↓	↑	↑			
적응성					↓	↑		↑
정밀성	↑		↓	↑		↓	↑	↓
견고성	↓		↓	↓	↓	↑	↓	↑

긍정적↑
부정적↓

목표 설정에 따른 성과

- 품질의 목표 설정과 수행 능력의 상관관계에 대한 실험

  Weinbergn and Schulman, “Goal and Performance in Computer Programming”

  - 동일한 다섯 개의 품질 목표
    - 최소 메모리 사용
    - 가장 읽기 쉬운 출력
    - 가장 읽기 쉬운 코드
    - 최소 코드
    - 최소 프로그래밍 시간
  - 다섯 개의 팀이 서로 다른 목표에 최적화 되도록 개발

목표 설정에 따른 성과

- 품질의 목표 설정과 수행 능력에 대한 실험 결과

최적화하도록 지시 받은 목표	최소 메모리 사용	가장 읽기 쉬운 출력	가장 읽기 쉬운 코드	최소 코드	최소 프로그래밍 시간
최소 메모리 사용	**1**	**4**	**4**	**2**	**5**
가장 읽기 쉬운 출력	**5**	**1**	**1**	**5**	**3**
가장 읽기 쉬운 코드	**3**	**2**	**2**	**3**	**4**
최소 코드	**2**	**5**	**3**	**1**	**3**
최소 프로그래밍 시간	**4**	**3**	**5**	**4**	**1**

결론: 조직 차원에서 품질의 목표를 명확히 설정해야 한다.

목차

- 소프트웨어 품질에 관한 목표
- **결함 방지 방법론**
- 코딩 스타일 표준화
- 버그 추적 시스템
- 유용한 도구
- 질의 응답

결함 감지 방법론

- 통일되고 명확한 소프트웨어 품질의 목표 설정
- 품질 보증 활동에 대한 높은 우선 순위
  - 구체적인 시스템(예: 평가 척도 등)에 반영
- 요구사항 / 아키텍처 / 설계와 동시에 테스트 전략 수립
- 소프트웨어 공학의 지침 적용
  - 문제 정의 / 요구사항 분석/ 아키텍처 / 구현/ 시스템 테스트 등

결함 감지 방법론

- **비공식적인 기술적 검토**
  - 설계/코드 검토, 동료와 함께 코드 Review
- **공식적, 주기적인 테스트, 또는 검토 (quality gates)**
  - 정밀검토, 동료 검토, 감사, …
- **수정 관리**
  - 통제되지 않은 수정은 품질 유지에 치명적
- **품질 관리 결과 측정**
- **Prototyping**
  - 시스템의 핵심 기능들에 대한 현실적 모델 개발

결함 감지 방법론의 상대적 효과

결함 감지 방법
비형식적 설계 검토
형식적 설계 정밀 검토
비형식적 코드 검토
형식적 코드 정밀 검토
모델링 또는 프로토타이핑
코드에 대한 개인 탁상 검사
단위 테스트
새로운 기능(컴포넌트) 테스트
통합 테스트
회귀 테스트
시스템 테스트
소량 베타 테스트
대량 베타 테스트

각 방법론의 결함 방지 효과는?

결함 감지 방법론

- 다양한 결함 감지 방법에 대한 연구 결과
  - 어떠한 기법도 타 방법에 의해 통계적으로 큰 이점을 갖지 못함
  - 사람과 방법론에 따라서 서로 다른 결함을 발견하는 경향성
  - 두 가지 기법을 조합할 경우 결함 발견율이 거의 2배로 증가

➔ 두 가지 이상의 방법론을 조합하여 사용함으로써 소프트웨어 오류를 크게 줄일 수 있다.

결함 감지 방법론

- **추천할 만한 결함 감지 방법의 조합**
  - 모든 요구사항/아키텍처, 시스템의 주요 부분에 대한 설계의 공식적 조사
  - 모델링 / Prototyping
  - 코드 읽기
  - 수행 테스트

목차

- 소프트웨어 품질에 관한 목표
- 결함 방지 방법론
- **코딩 스타일 표준화**
- 버그 추적 시스템
- 유용한 도구
- 질의 응답

코딩 스타일 표준화

- 코딩 스타일 표준: 개발자가 코드를 어떻게 작성해야 하는지에 대한 규약
  - 주석, 변수명, 함수명, 띄어쓰기, 등등
  - 각 개발자가 나름대로의 스타일로 작성하는 것이 아니라 공통의 표준 스타일로 작성하게 함

- 코딩 스타일 표준의 필요성
  - 대규모 프로젝트의 코드를 일관성 있는 스타일로 작성할 수 있다.
  - 새로운 개발자들이 쉽게 적응할 수 있다.
  - 개발자들이 자신만의 스타일을 개발하지 않아도 된다.
  - 코드를 이해하기 쉽게 하고, 실수를 줄일 수 있다.
  - 개발자들이 공통의 적을 갖게 한다. :-)

코딩 스타일 표준화

- **코딩 스타일 표준 적용의 문제점**
  - 적절하지 못한 표준은 개발 효율을 저하시킨다.
    - 작성자가 C/C++을 잘 이해하지 못한 경우
    - 실제 개발 담당자가 아닌 사람이 작성할 경우
  - 표준은 보통 지나치게 많은 구조를 포함한다.
    - 따라서 교육과 적용이 쉽지 않다.
  - 표준 스타일은 NIH (Not Invented Here)의 이유가 될 수 있다.

코딩 스타일 표준의 예: 주석

- 파일 헤더 주석
  - 다른 함수를 볼 필요없이 해당 프로그램이 어떤 동작을 하는지 이해할 수 있도록 설명하기

```
/*
** project : General Template **
** filename : TPL.H **
** version : 1 **
** date : April 15, 2007 **
**********************************************************************
** Copyright (c) All rights reserved. **
**********************************************************************
VERSION HISTORY:
Version : 1
Date : April 15, 2007
Revised by : manager
Description : Original version.
*/
```

코딩 스타일 표준의 예: 주석

- 함수 헤더 주석
  - 함수의 기능, 인자값, 리턴값에 대해 설명하기

```
/*
* NAME :             int KB_GetLine(pKbdBuf, MaxChars)
* DESCRIPTION :      Input line of text from keyboard
* INPUTS :
*            int     MaxChars  max chars to read before beeping
* RETURNS:
*            Type:   int                       Error code
*            Values: VALID_DATA                valid read
* PROCESS :
*                    [1]  Clear keyboard buffer
*                    [2]  Do
*                    [3]    Get character
*                    [4]    Translate characters
*                    [5]  Until CR or buffer full
* NOTES :            Unknown characters returned as '*'
*/
```

코딩 스타일 표준의 예: 주석

- 기타 주석
  - 블럭 주석

```
/*********************************************************************/
/**                                                                 **/
/**                    TYPEDEFS AND STRUCTURES                      **/
/**                                                                 **/
/*********************************************************************/
```

  - 한 줄 주석

```
/* loop to read and echo print score array */
for (student = 0; student < NUM_STDTS; student++)
{
    scanf ("%d", &scores [student]);
    printf ("%d", scores [student]);
}
```

  - 곁가지 주석

```
if (income < 0.0) /* handle bad input data */
```

코딩 스타일 표준의 예: 띄어쓰기

- 읽기 편한 띄어쓰기 사용하기
  - space보다 tab키 사용하기

```
int blah(int x, int y) {
    stmt1;
    stmt2;
    while (...) {
        stmt3;
        stmt4;
        if(...) {
            stmt5;
            stmt6;
            stmt7;
        } else {
            stmt8;
        }
        stmt9;
    }
    stmt10;
}
```

코딩 스타일 표준의 예: 명명법

- 변수, 함수, 상수의 용도를 설명할 수 있는 이름 붙이기

```
#define SCALE_FACTOR (5.0 / 9.0)
int height, length, width, volume;
```

- 함수를 설명할 수 있는 동사, 명사, 형용사로 이름 사용하기

```
float average(float a, float b) {…}
void print_count(int n) {…}
```

- C 스타일의 명명 규칙 사용하기
  - 반복 카운터 변수는 i 와 j 사용
    for (i = 0;i < n; i++), for (j = 0; j < n; j++)

코딩 스타일 표준의 예: 변수와 상수

- 전역변수 사용하지 않기
  - 전역변수 사용을 피하고, 함수에 변수 값을 인자로 넘기기

- 숫자 값을 직접 이용하기 보다는 상수 값을 정의하고 이를 이용

Do this:	Not this:

`#define MAX 50`
`int buf[MAX];`
`if(i < MAX) {…}`
	`int buf[50];`
	`if(i < 50) {…}`

코딩 스타일 표준의 예: 기타

- **C** 언어에서 일반적인 두 가지 블럭의 형태

while (expression) {
  statement1;
  statement2;
  statement3;
}
	while (expression)
	{
	  statement1;
	  statement2;
	  statement3;
	}

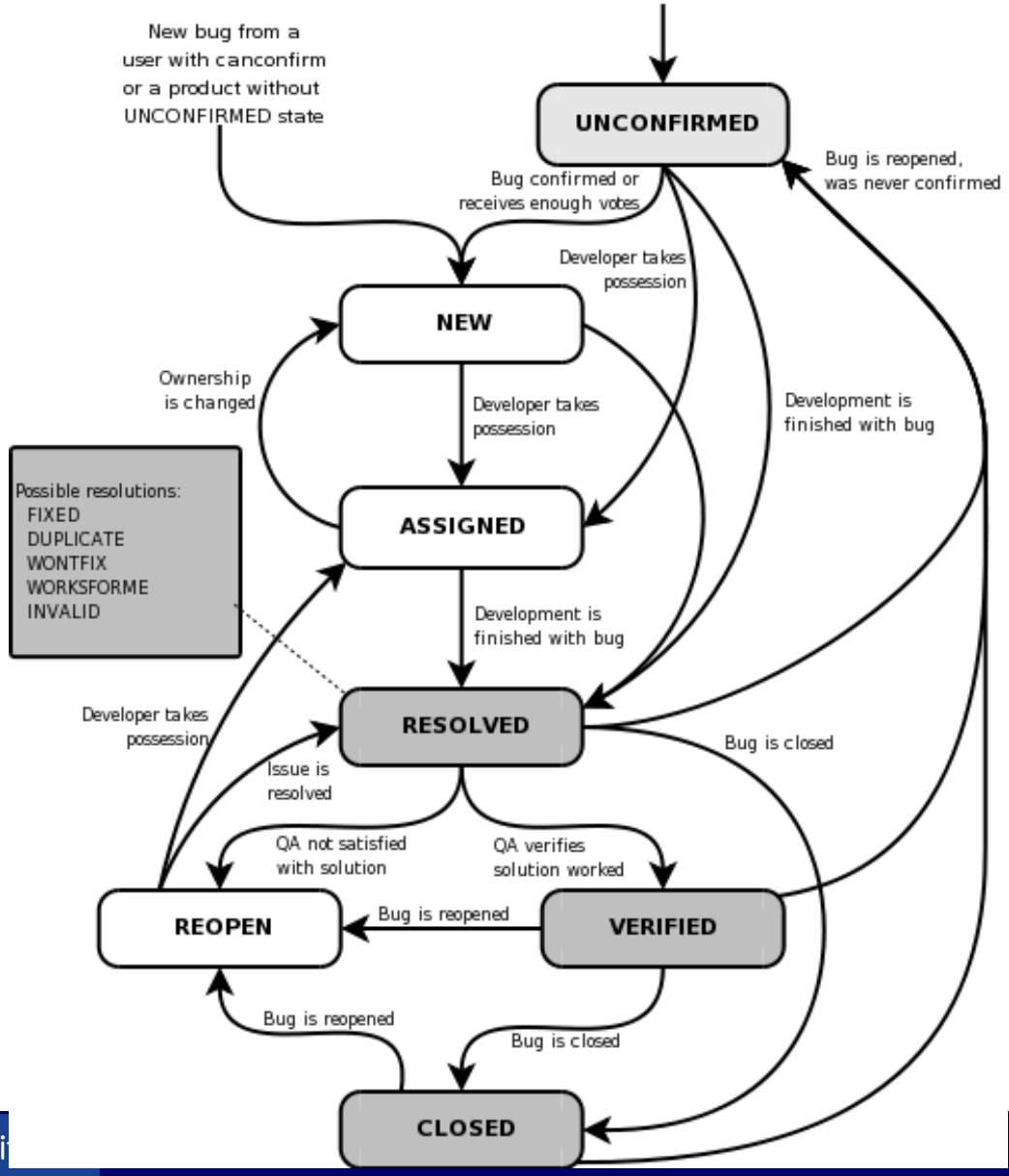
- 한 줄 길이는 **80**자가 넘지 않도록

```
if ( ((blah[i] < 0 ) && (grr[j] > 234))
     || ((blah[j] == 3456) && (grr[i] <= 4444))
     || ((blah[i] > 10) && (grr[j] == 3333))
     )
{
     stmt1;
     stmt2;
} else {…}
```

목차

- 소프트웨어 품질에 관한 목표
- 결함 방지 방법론
- 코딩 스타일 표준화
- 버그 추적 시스템
- 유용한 도구
- 질의 응답

버그의 Life Cycle



버그 추적 시스템

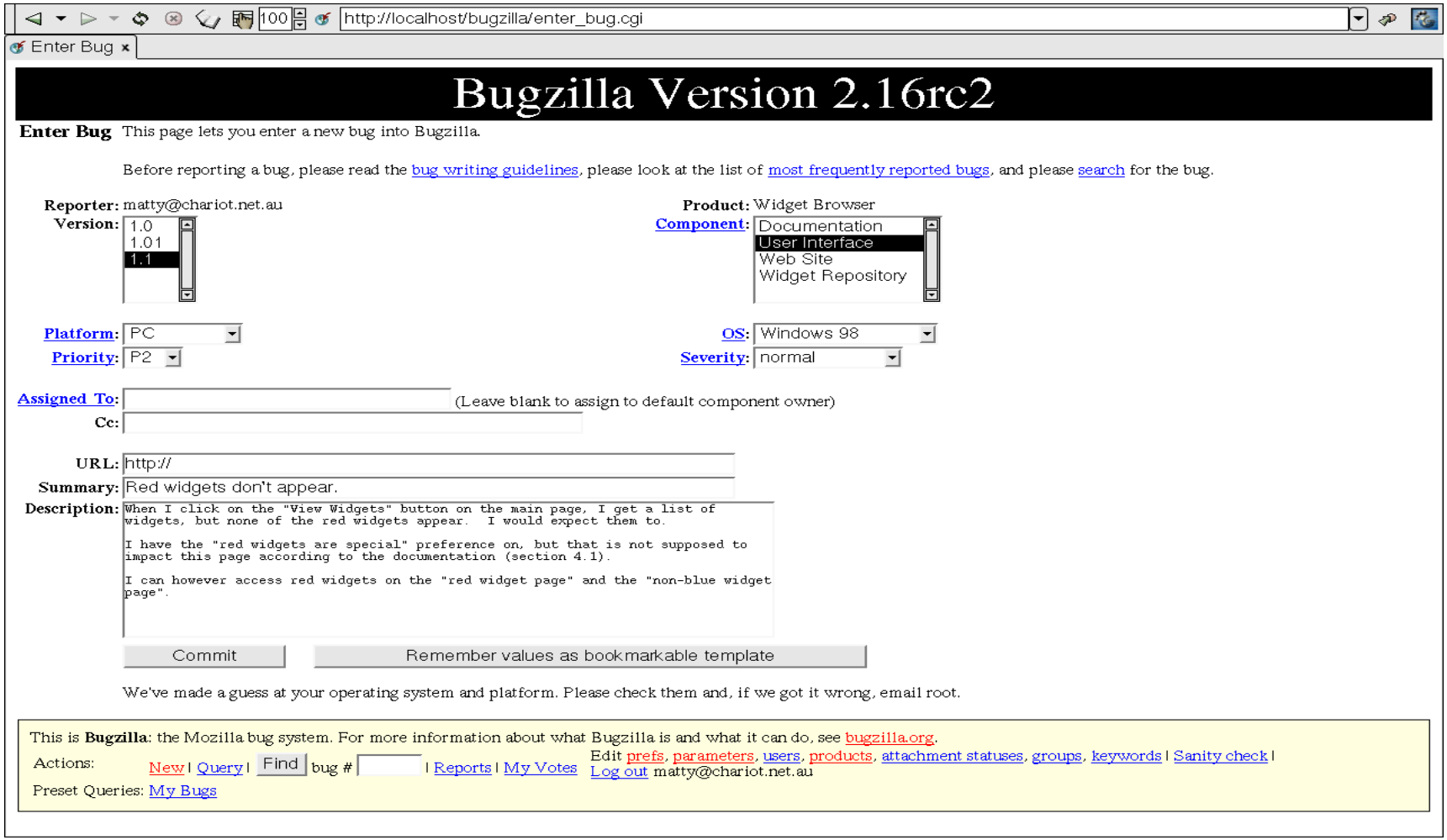
- 버그 추적 시스템
  - 버그를 보고하여 담당자에게 알리고 보고된 버그들의 처리상황을 추적할 수 있는 시스템
    - 웹서버, 메일서버, 소스코드 관리도구와 연동
  - 일반적으로 버그 추적 뿐 아니라 다양 한 용도로 사용될 수 있다.
    - Issue tracking, 개발 및 수정 사항

예) Bugzilla, Mantis

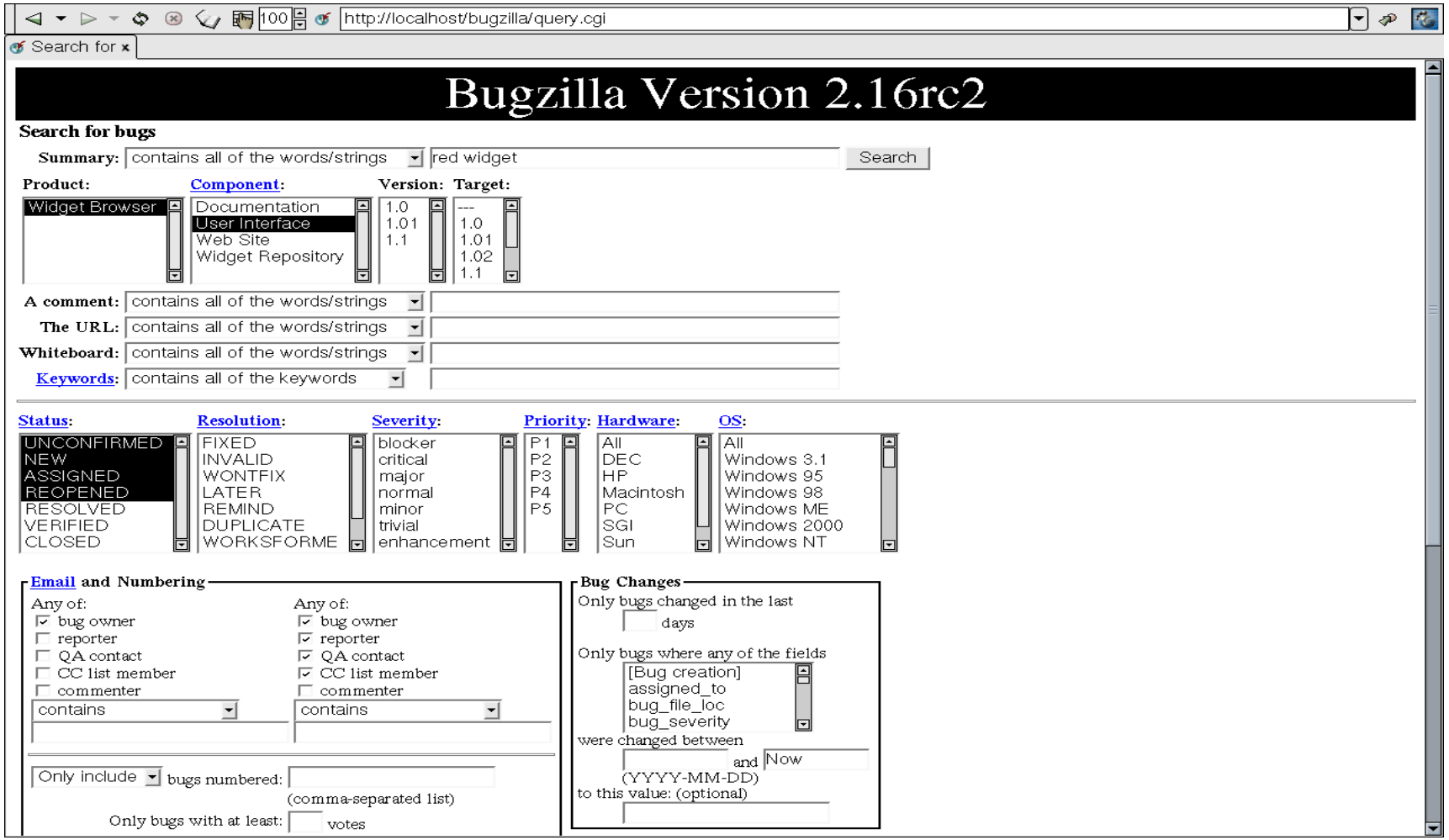
버그 추적 시스템의 기능

- 버그의 종류 및 내용, **test case**, 수정기록, 해결 여부 등에 대한 데이터베이스
  - 버그 등록, 현재 처리 상황 관리
- **Communication** 지원
  - 수정 담당자 지정,
  - Communication
- 소프트웨어 관리
  - 소스코드 수정 내용 관리
  - Patch 관리
  - Manage quality assurance (QA)

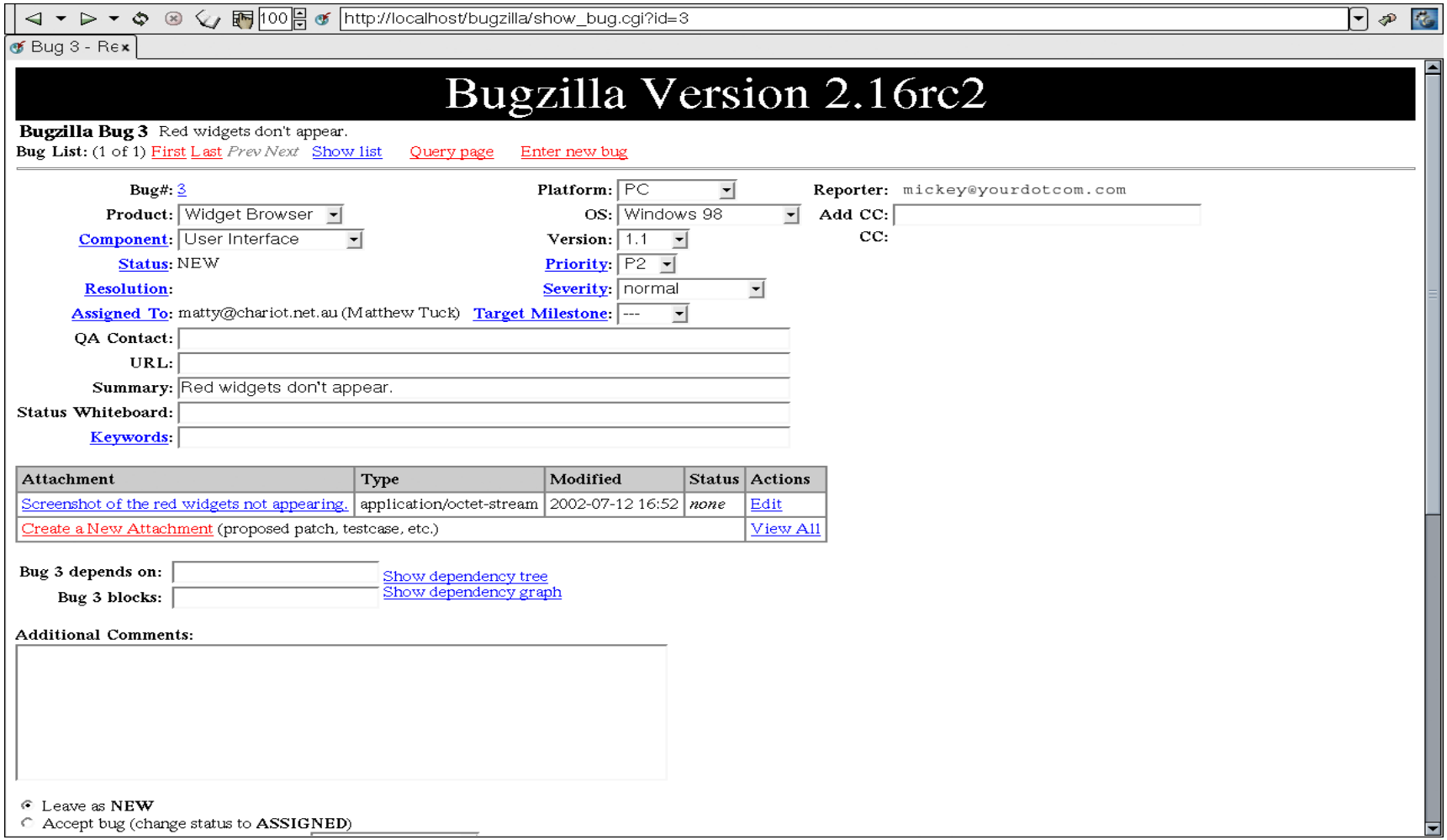
Bugzilla Screenshot: 버그 등록



Bugzilla Screenshot: 버그 Query



Bugzilla Screenshot: 버그 Query



목차

- 소프트웨어 품질에 관한 목표
- 결함 방지 방법론
- 코딩 스타일 표준화
- 버그 추적 시스템
- **유용한 도구**
- 질의 응답

유용한 도구: 개발자용

- **정적 코드 체크 (static code checker)**
  - 소스코드 중 버그, 또는 실수로 의심할 만한 부분을 자동 검색

  Ex) Splint, Flawfinder, McCabe, …

- **Debugger / dynamic checker**
  - VC++ IDE, Debugging Tools for Windows, Bound Checker, Purifier, …

- **Coverage checker**
  - 테스트 시 코드의 각 부분이 실제로 실행되었는지 검사

  Ex) TrueCoverage, PureCoverage, …

유용한 도구: 조직 및 시스템용

- 소스 코드 관리 시스템
  - 개발자의 중대한 실수로부터 프로젝트 소스를 보호할 수 있음

  Ex) CVS, Visual Source Safe, ClearCase, RCS, …

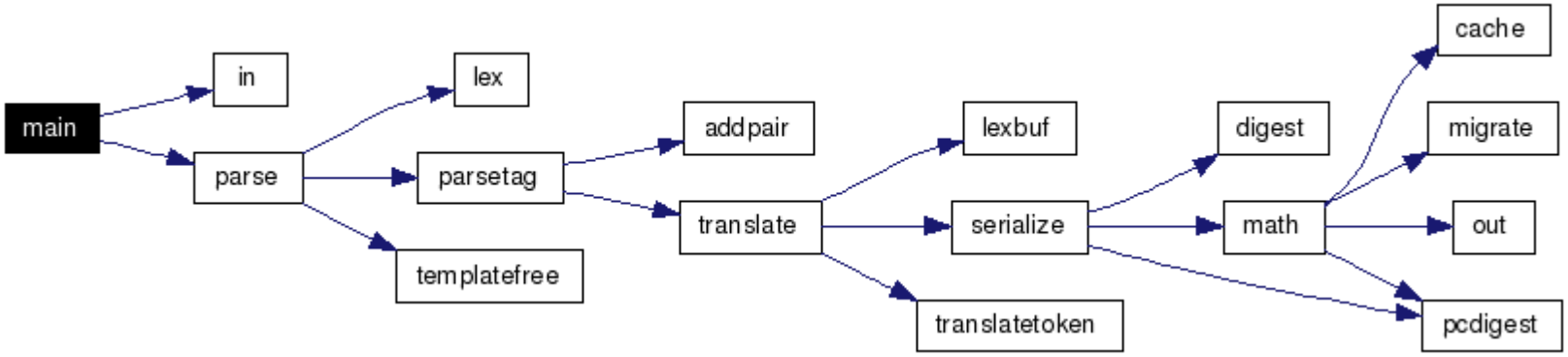
- 버그 추적 시스템
  - 발견된 버그 등록, 담당자 배정, 수정 확인, 수정 기록 및 통계 등

  Ex) Bugzilla, Mantis, …

유용한 도구: 기타

- 매뉴얼 생성기
  - 간단한 주석으로부터 자동 문서화
  - 함수간 call graph 자동 생성

Ex) Doxygen, …



질의 & 응답

감사합니다!