NETSEC-KR 2011

Java Application의 Top 5 Secure Coding Tips

2009.6~2012.6

2011. 4. 27

㈜소프트4소프트

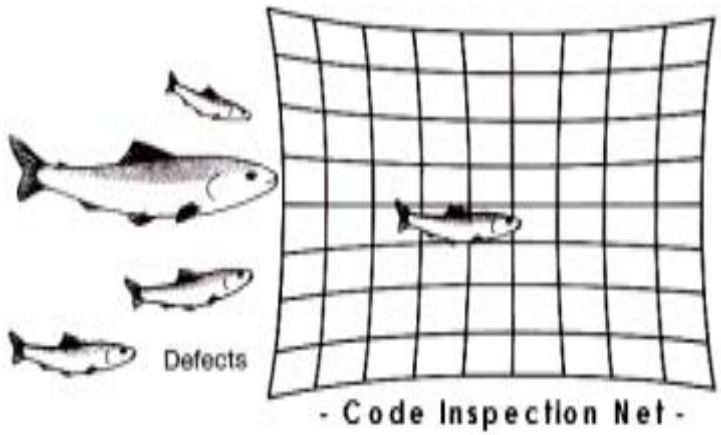
www.soft4soft.com
02-553-9464

목 차

1 코드 검사 및 코딩 표준

◆ 코드검사 목적 및 활동

- 소스 코드의 결함 탐지**(Prevention)** 및 진단**(Detection)**
  - 코딩 스타일/결함, 설계결함, 런-타임결함, 보안결함, 표준준수**(MISRA-C/C++)** 등
  - 프로그램의 정확성, 안정성 및 보안성 등 보장 향상
- 자체 개발/운영 표준 코딩 가이드
  - 시스템**(업무)** 이해 및 코딩 수준의 상향 평준화 **–** 초급/외주개발자
  - 개발 단축 및 **QA** 비용, 시간 절약 **– Review**/프로그램 검수
- 고객 만족도 향상 및 향후 유지보수 용이

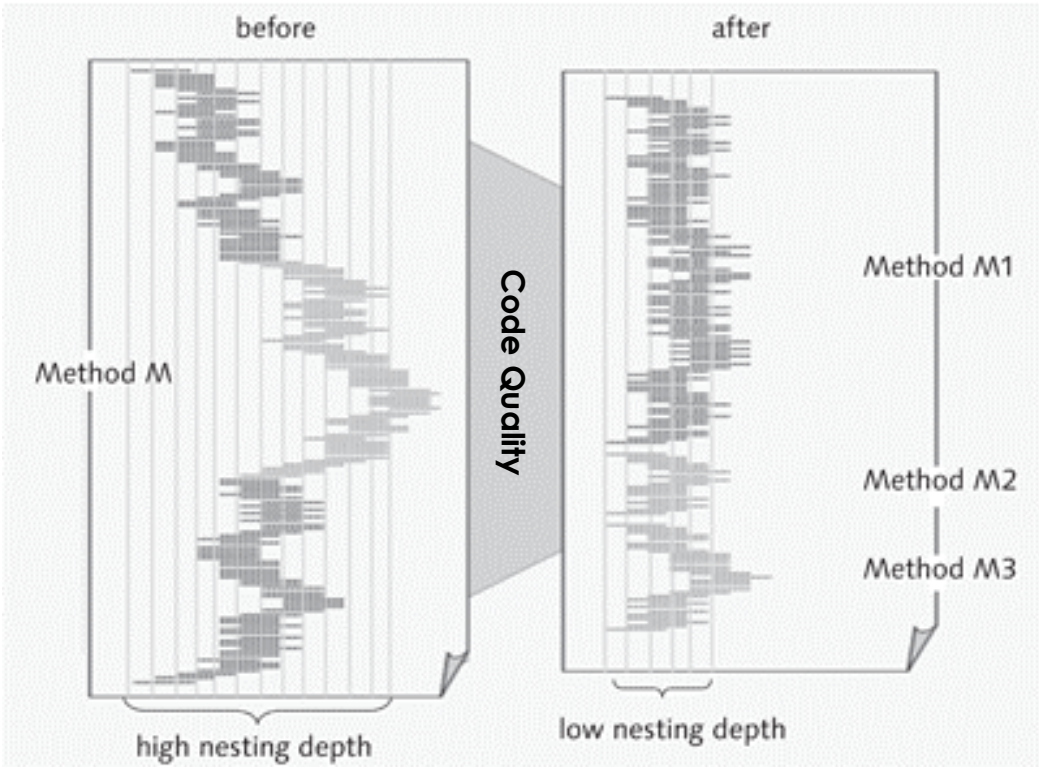


테스팅 이전에 **50-90%** 가량의 결함을 제거함으로써, 개발 공정의 **10%~30%**, 테스팅 비용 및 공정의 **5-10**배, 그리고 유지보수 비용의 **2/3**를 절감할 수 있다 - ***Software Inspection***

1 코드 검사 및 코딩 표준

◆ 코딩 표준

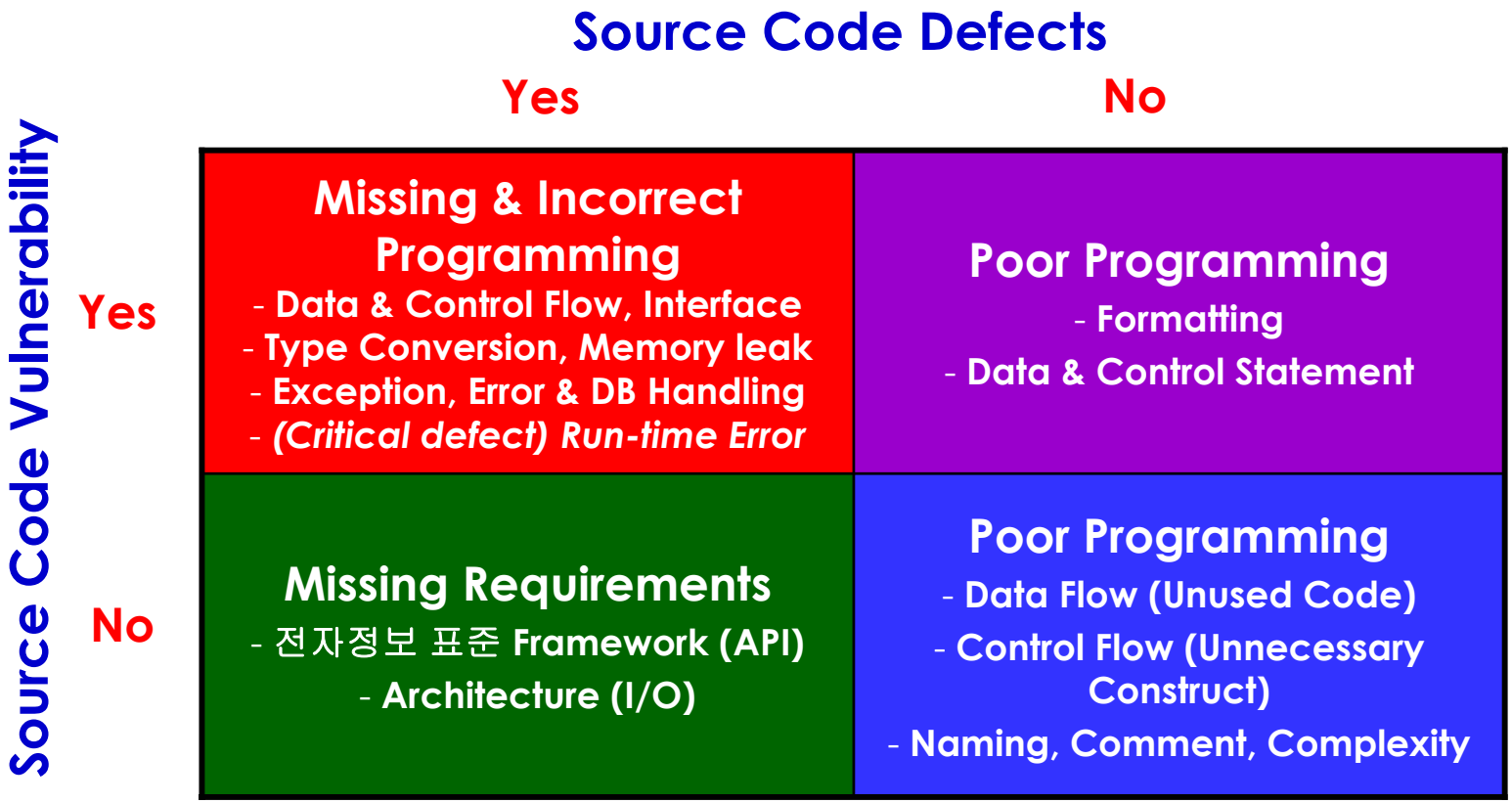
- ➢ 개발 명세 및 코딩 가이드 이해 – **Code Defect Prevention**
  - ○ 개발자 오용, 실수, 오해 최소화, 디버깅 작업 최소화
  - ○ **Ex) API, Utility API, Framework API** 사용 표준화
- ➢ 코드 구조화 – **Program Simple & Small**
  - ○ 코드 이해성 향상 및 유지보수 비용 절감



2 SW 보안 취약점

◆ 코드 취약점 유형

- (가독성, 유지보수성) **Poor Programming,**
- (정확성, 안전성) **Missing&Incorrect Programming**
- (신뢰성) **Missing Requirements**



2 SW 보안 취약점

◆ 코드 취약점 심각도 분류

Java Syntax	Severity		
	Low	Medium	High
Declarations and Initialization	O		
Expressions	O	O	
Numeric Types and Operations	O	O	
Object Orientation	O	O	O
Methods	O	O	
Exceptions and Error Handling	O	O	
Locking/ Concurrency	O	O	
Thread APIs	O		
Input Output	O	O	
Input Validation and Data Sanitization	O	O	O
Application Programming Interfaces		O	
Platform Security		O	O
Runtime Environment			O
Serialization	O	O	O
Miscellaneous	O	O	O

Severity: (1) low (denial-of-service attack, abnormal termination)
(2) medium (data integrity violation, unintentional information disclosure)
(3) high (run arbitrary code, privilege escalation)

2.1 코딩 표준을 통한 보안 취약점 예방

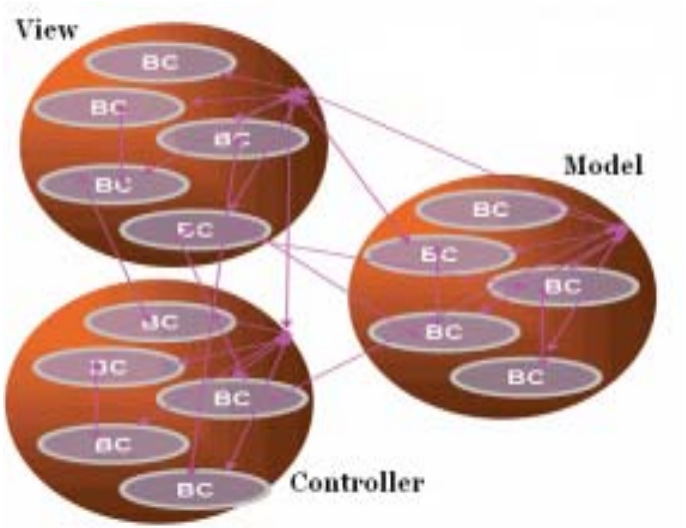
◆ Try-catch-finally Coding Standards

➢ (정의) **try, catch** 문에서 예외의 발생 유무에 관계 없이 반드시 **finally** 문이 실행함

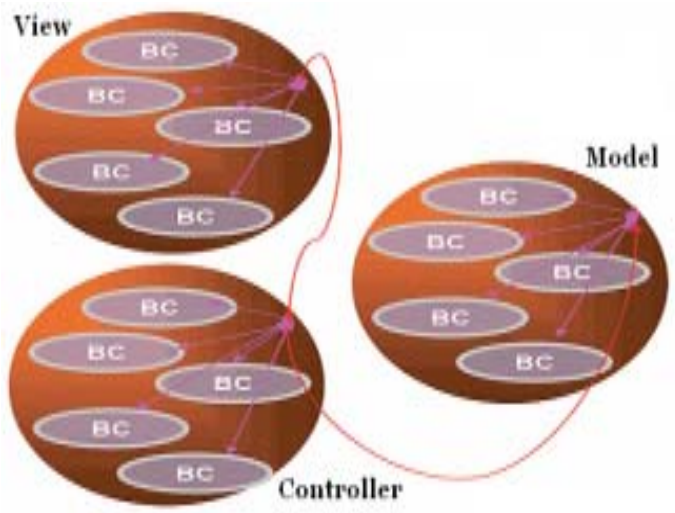
Java Syntax / Rules	Empty Block	Catch Block	Finally Block
try { *body-code;* } catch (*exception-classname variable-name*){ *error-handler-code;* } finally { *close-code;* }	•(1)Format "try-catch-finally" Syntax •(2)Format "try-catch-finally" Statement	•(1)Don't use different Exception type •(2)Checked Exception type "SQLException(DataAccessException), Exception" •(3)Log type compliance by standard(Framework API or Architecture)	•(1)Do not use "return/Throw" syntax •(2)Resource release type by standard : (API, Utility API, or Framework API) •(3)Reuse after non-free resource
Code Defect	**Formatting Convention**	**Control Error, Architecture Compliance**	**Control Flow Error, Resource leak, Architecture Compliance**
Security Cause	**Ignore Exception (=Empty Catch)**	**Unhandled Exception, Nullpointer Exception**	**Exception loss (=Return Inside Finally)**
Security Vulnerability		**Denial of service attack Data integrity Violation**	**Denial of service attack**

2.2 전자정부 표준 Framework을 통한 보안 취약점 예방

- **Architecture(Framework)의 Component 및 Pattern 준수 검증**
  - 각 **Component**들의 동작 분배 및 개념적 통일성 등 가이드 활용
    - **(Java) Component I/F, Hierarchy Layer, DB(SQL)/Exception/Error(Log) Handling** 등 준수
  - 결함 예방
    - 성능, 보안성, 계산 오류, **API** 등의 치명적인 결함 예방 **(**프로그램 오류 최소화**)**
    - **SW** 시스템의 신뢰성 및 유지보수성 효과 **(**코드 일관성**)**
  - **Component I/F Examples: MVC Model** 준수 검사



코딩 표준 부재(미준수)

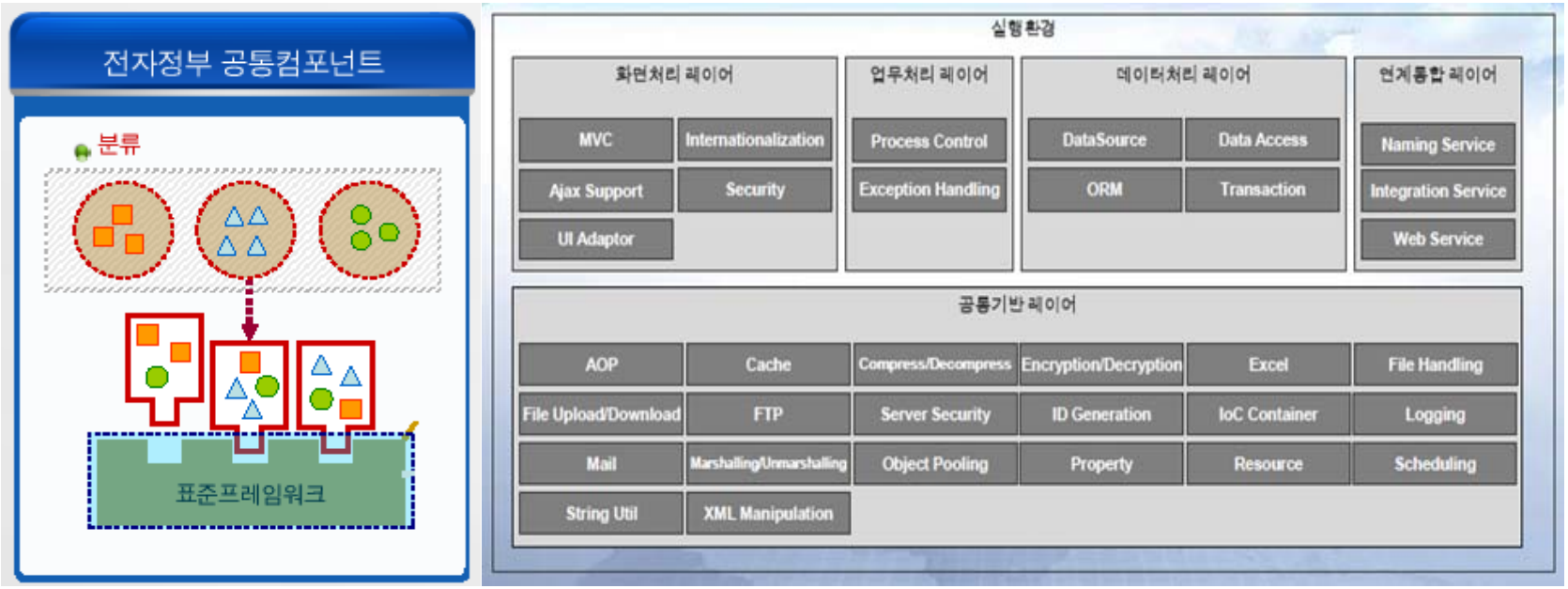


코딩 표준 준수

2.2 전자정부 표준 Framework을 통한 보안 취약점 예방

◆ 전자정부 표준 개발 **Framework** 기능의 보안 취약점 예방 효과

실행환경의 서비스 그룹(서비스)	보안 취약점 예방 종류
화면처리 **Layer (Security)**	SQL/Command Injection, Cross Site Scripting (XSS), Cross-Site Request Forgery (CSRF)
업무처리 **Layer (Exception Handling)**	Information Exposure Through an Error Message
데이터처리 **Layer (Data Access/Source)**	SQL Injection , DB information Exposure, DB Resource Leak
공통기반 **Layer (File Handling, Server Security, Encryption/Decryption)**	File Resource Leak, Broken Authentication and Session Management, Encryption of Sensitive Data



< 전자정부 개발 **Framework** 환경 중 실행환경: **4** 환경, **13**서비스그룹, **54** 서비스 >

3.1 코딩 서식의 취약점 및 예방

◆ Coding Formatting - Declaration

정의	하나의 선언에서 여러 변수들을 선언하지 말 것 (1) 다른 유형의 변수들, (2) 초기화 및 초기화되지 않은 변수의 혼합
위험	변수들의 초기값들의 혼란을 초래
예방	하나의 선언에 하나의 변수 선언
결과	denial-of-service attack

Noncompliant Code

```
int  i, J = 1;
```

Compliant Code

```
int  i = 1;
int  J = 1;

or

int  I=1, J = 1;
```

프로그래머 또는 검토자가 i와 j 모두 1로 초기화 됐다는 실수를 범할 수 있음

3.1 코딩 서식의 취약점 및 예방

◆ Coding Formatting – Expression

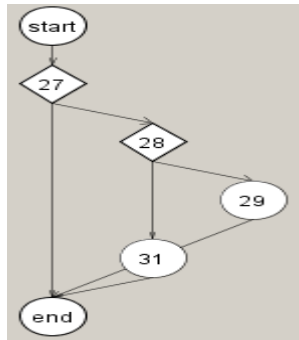
정의	제어 문장에 중괄호({}) 사용
위험	로직 오류
예방	항상 중괄호({}) 사용하여 애매성 제거
결과	denial-of-service attack

Noncompliant Code

```
int privileges;

if (invalid_login())
   if (allow_guests())
      privileges = GUEST;
else
   privileges = ADMINISTRATOR;

or
if (invalid_login())
  login = 0;
else
  System.out.println("Login is valid\n"); // debugging line added
  login = 1; // this line always gets executed, regardless of a valid login!
```



Compliant Code

```
int privileges;

if (invalid_login()) {
   if (allow_guests()) {
      privileges = GUEST;
   }
} else {
     privileges = ADMINISTRATOR;
}
```

권한이 없는 사용자가 권리자 권한을 얻을 수 있는 취약점

3.2 코드 결함의 취약점 및 예방


◆ Code Defects – Exception Behavior

정의	finally 블록에서 종료하지 말 것
위험	로직 오류
예방	finally 블록에서 return, break, continue, throw의 키워드를 사용하지 말 것
결과	denial-of-service attack

Noncompliant Code

```
class TryFinally {
  private static boolean doLogic() {
    try {
      throw new IllegalStateException();
    } finally {
      System.out.println("Uncaught Exception");
      return true;
    }
  }

  public static void main(String[] args) {
    doLogic();
  }
}
```

Compliant Code

```
class TryFinally {
  private static boolean doLogic() {
    try {
      throw new IllegalStateException();
    } finally {
      System.out.println("Caught Exception");
    }
    // Any return statements must go here; applicable
     only when exception is thrown conditionally
  }
  public static void main(String[] args) {
    doLogic();
  }
}
```

(1) try 블록에서 발생되는 예외가 무시됨, **(2) Method**의 **return** 값이 **finally** 블록의 **return** 값으로 대체됨

3.2 코드 결함의 취약점 및 예방

◆ Code Defects – Return Value Check

정의	Method return 값 무시하지 말 것
위험	Method return 값 무시는 예기치 않은 프로그램 동작 발생
예방	Method return 값(또는 에러 조건) 검사
결과	data integrity violation

Noncompliant Code

```
public class Ignore {
  public static void main(String[] args) {
    String original = "insecure";
    original.replace( 'i', '9' );
    System.out.println(original);
  }
}
```

Compliant Code

```
public class DoNotIgnore {
  public static void main(String[] args) {
    String original = "insecure";
    original = original.replace( 'i', '9' );
    System.out.println(original);
  }
}
```

String.replace() method의 return 값 무시로, original string의 Update 실패

3.2 코드 결함의 취약점 및 예방

◆ Code Defects – Null Pointer Dereference

정의	Null pointer dereference하지 말 것 (역참조는 Function call, 변수 읽기/쓰기, 배열 접근에서 가능)
위험	프로그램 종료
예방	return 값 검사, 변수 또는 객체가 non-null인지 확인
결과	denial-of-service attack

Noncompliant Code

```
public static int cardinality(Object obj, final Collection col) {
  int count = 0;
  Iterator it = col.iterator();
  while (it.hasNext()) {
    Object elt = it.next();
    if ((null == obj && null == elt) || obj.equals(elt)) {  // null
pointer dereference
      count++;
    }
  }
  return count;
}
or
String cmd = System.getProperty("cmd");
cmd = cmd.trim(); // null point dereference
```

Compliant Code

```
public static int cardinality(Object obj, final
    Collection col) {
  int count = 0;
  Iterator it = col.iterator();
  while (it.hasNext()) {
    Object elt = it.next();
    if ((null == obj && null == elt) ||
        (null != obj && obj.equals(elt))) {
      count++;
    }
  }
  return count;
}
```

만약 공격자가 프로그램 환경을 제어하여 **cmd**를 미정의 한다면, **trim()** 호출시 **NULL pointer exception** 발생

3.2 코드 결함의 취약점 및 예방

◆ Code Defects – Resource Leak

정의	Stream, Connection과 같은 자원은 사용 후 반드시 종료
위험	많은 파일 또는 DB 연결이 Open되어 자원 고갈, 성능 저하, 정보 노출 등의 문제점 발생 (Garbage Collection 시작되기 전에)
예방	Close Method로 모든 자원을 해제 (API, Utility API, Framework API)
결과	denial-of-service attack, unintentional information disclosure

Noncompliant Code

```
try {
    con = con.getConnection;
}
catch (Exception e) {
    log(e)
}
finally {
}
```

Compliant Code

```
try {
    con = con.getConnection;
}
catch (SQLException sqle) {
    log(e)
}
finally {
    if (con != null) try { rs.close(); } catch (Exception e) {}
}
```

프로그램 규모와 사용 가능한 메모리에 따라, **Heap Space**가 고갈될 때 **"OutOfMemoryError"**발생하여, 프로그램 정지 결과를 초래

4 결론

- A coding standard helps you:
  - avoid undefined usage
  - avoid unspecified usage
  - avoid implementation-defined usage
  - guard against *compiler errors*
  - guard against common *programmer error*
  - limit program complexity
  - establish an objective basis for code review

You can't manage what you can't measure

Q & A