ICSE MET’17 Keynote

Metamorphic Testing: Beyond Testing Numerical Computations

Dr. Zhi Quan (George) Zhou
Associate Professor
University of Wollongong
Australia
zhiquan@uow.edu.au
Why this topic?

• Observation: MT is often illustrated using mathematical functions, e.g.
  ◦ \( \sin(x) = \sin(x+360) \)
  ◦ \( \sin(-x) = -\sin(x) \)
  ◦ …

• This is because mathematical functions have well known properties that everyone understands.
Why this topic?

Misunderstanding: Is MT just for numerical programs?
Why this topic?

The fact:

Presentation outline

- MT in security testing
  - A motivating example: the Heartbleed bug
  - Detecting obfuscator bugs
- MT for online search
- MT beyond verification
Presentation outline

- **MT in security testing**
  - A motivating example: the Heartbleed bug
  - Detecting obfuscator bugs
- **MT for online search**
- **MT beyond verification**
MT in Security Testing
A motivating example: the Heartbleed bug

- Under the Heartbeat protocol, the client sends a message and the message length to the server, and the server sends back the same message.
MT in Security Testing
A motivating example: the Heartbleed bug

Server, if you are still there, send me “banana” (6 letters)
MT in Security Testing
A motivating example: the Heartbleed bug
MT in Security Testing

A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an
arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

### 4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

- The specification (RFC 6520)
  (TLS and DTLS Heartbeat Extension)

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessageType::payload_length];
    opaque padding[payload_length];
} HeartbeatMessage;
The bug: when this protocol was implemented in OpenSSL, the programmer forgot to include any bounds-checking code.

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing
A motivating example: the Heartbleed bug

Server, if you are still there, send me “banana” (600 letters)
600 characters in server buffer starting from “b”. They can be anything.

banana. User George set password to “abcdefg”. Server master key is set to “0987654321” ...
MT in Security Testing

A motivating example: the Heartbleed bug

- Difficulty in detecting the bug
  - No crash
  - No hang
  - Undetectable with fuzz testing alone.
MT in Security Testing
A motivating example: the Heartbleed bug

• How MT can help?
  ◦ To identify MRs, the tester will ask:

  “What if I change some of the parameter values?”
What if I change type to a different value?

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```c
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```
What if I change \texttt{payload\_length} to a different value?

\textbf{This MR will generate failure test cases!}

4. Heartbeat Request and Response Messages
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

\begin{verbatim}
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
\end{verbatim}
What if I change two or more parameters?

MT can guide the tester to think beyond the normal range of input values (negative testing).

```c
uint16 payload_length;
opaque payload[HeartbeatMessage.payload_length];
opaque padding[padding_length];
} HeartbeatMessage;
```
MT in Security Testing

Related work: MT in penetration test

An Inferential Metamorphic Testing Approach to Reduce False Positives in SQLIV Penetration Test

Lei Liu¹, Guoxin Su², Jing Xu¹, Biao Zhang¹, Jiehui Kang¹, Sihan Xu¹, Peng Li¹ and Guannan Si³
¹College of Computer and Control Engineering, Nankai University, Tianjin, China
²School of Computing and Information Technology, University of Wollongong, Australia
³School of Information Science and Electrical Engineering, Shandong Jiaotong University, Jinan, China

Abstract—SQL Injection Vulnerability (SQLIV) has been the top-ranked threat to the Web security consistently for many years. Penetration tests, which are a most widely adopted technique to detect SQLIV, are usually affected by testing inaccuracy. This problem is even worse in inference-based, blind penetration tests for online Web sites, where Web page variations (such as those caused by inbuilt dynamic modules or user interactions) may lead to a large number of False Positives (FP). We present a novel approach called Inferential Metamorphic Testing (IMT) to reduce FP in SQLIV penetration tests. First, we define the notion of Inferential Metamorphic Relations (IMR), which is inherited from Mutational Metamorphic Testing (MMT). Second, we Web structures (such as a search engine), back-end programs of a Web page may exhibit behaviors that are similar to SQLIV. These dynamic variations of Web page contents actually disable the traditional test oracles to some extent, making the FP problem as an oracle problem essentially.

Many works have been done on improving the detecting capabilities of SQLIV penetration tests. Some focus on combining static and dynamic testing approaches that usually use static code analysis or server proxy to increase the testing capability of SQLIV penetration tests [6][7]. But, the need to access the source code or setting proxy by altering server programs, which are often infeasible for legacy systems or third-party outsourcing limits the applicability of these
Presentation outline

• MT in security testing
  ◦ A motivating example: the Heartbleed bug
  ◦ Detecting obfuscator bugs
• MT for online search
• MT beyond verification
Detecting obfuscator bugs
A pilot study

- Critical security infrastructure
  - Transform an original program into an equivalent program
    - less readable
    - to prevent attackers from analyzing and understanding the original code.
Detecting obfuscator bugs
A pilot study

- As important as compilers
- But difficult to test
- A surprising observation:
  - Compiler testing: a lot of research
  - Obfuscator testing: almost nothing
- Our work: first to report real bugs.
Detecting obfuscator bugs

Obfuscators under test

- Cobfusc
  - open source Linux utility that makes a C source file unreadable, but compilable.

Detecting obfuscator bugs

Obfuscators under test

---

Ubuntu Manpage: cobfusc — make a C source file unreadable but compilable - Mozilla Firefox

Ubuntu Manpage: cobfusc —

---

zesty (1) cobfusc.1.gz
Provided by: cutils_1.6-4_amd64

NAME

cobfusc — make a C source file unreadable but compilable

SYNOPSIS

cobfusc [-AabdemntxV] [-c case] [-f suffix] [-g file] [-i idobf]
[-z file] [file ...]
Detecting obfuscator bugs

Obfuscators under test

- Stunnix CXX-Obfus
  - commercial C/C++ obfuscator
  - users include many Fortune 500 companies

http://stunnix.com/prod/cxxo
Detecting obfuscator bugs

Obfuscators under test
Detecting obfuscator bugs

Obfuscators under test

- Tigress
  - C obfuscator developed at the University of Arizona that supports novel defenses against both static and dynamic reverse engineering.

The Tigress C Diversifier/Obfuscator
Detecting obfuscator bugs

Obfuscators under test

- Obfuscator-LLVM
  - open source tool in LLVM compilation suite.
  - Given a C source program, Obfuscator-LLVM outputs obfuscated and compiled binary code.
  - LLVM users include Adobe Systems, Apple, Intel, and Sony.

https://github.com/obfuscator-llvm/obfuscator/wiki
Detecting obfuscator bugs

Obfuscators under test

Obfuscator-LLVM is a project initiated in June 2010 by the information security group of the University of Applied Sciences and Arts Western Switzerland of Yverdon-les-Bains (HEIG-VD).

The aim of this project is to provide an open-source fork of the LLVM compilation suite able to provide increased software security through code obfuscation and tamper-proofing. As we currently mostly work at the Intermediate Representation (IR) level, our tool is compatible with all programming languages (C, C++, Objective-C, Ada and Fortran) and target platforms (x86, x86-64, PowerPC, PowerPC-64, ARM, Thumb, SPARC, Alpha, CellSPU, MIPS, MSP430, SystemZ, and XCore).
Detecting obfuscator bugs

The metamorphic relations (MRs)

- \( \text{MR}_1 \)

\[
\begin{align*}
Q & \Rightarrow \text{obfuscator} \Rightarrow O(Q) \\
& \Rightarrow \text{compiler} \Rightarrow C(O(Q))
\end{align*}
\]

\[
\begin{align*}
P & \Rightarrow \text{obfuscator} \Rightarrow O(P) \\
& \Rightarrow \text{compiler} \Rightarrow C(O(P))
\end{align*}
\]

Same outputs for the same inputs
Detecting obfuscator bugs
The metamorphic relations (MRs)

- **MR$_1$**
  
  How do we generate equivalent (P, Q)?
  
  MR$_{1.1}$: by a separate tool
  MR$_{1.2}$: by the obfuscator.
Detecting obfuscator bugs

The metamorphic relations (MRs)

- **MR\textsubscript{2}**
  - For the same input program, the obfuscator should generate (behaviourally) equivalent programs regardless of when the obfuscator is run.
Detecting obfuscator bugs

The metamorphic relations (MRs)

- **MR\textsubscript{3}**
  - Looks at obfuscated source code without compiling it.
  - Checks whether obfuscation rules have been applied consistently each time the obfuscator runs.
Detecting obfuscator bugs

Tigress

\[
\begin{align*}
\text{if } (i > j) & \quad \text{if } ((\text{int})(i > (\text{long})j + 116) - 116)) \{ \\
\quad i -= 10; & \quad i = (i - (10L + 116)) + 116; \\
\text{else} & \quad } \text{else} \{ \\
\quad i += 10; & \quad i = (i + (10L + 116)) - 116; \\
\}\end{align*}
\]
Detecting obfuscator bugs

Tigress

\[
\begin{align*}
\text{if } (i > j) \quad & \text{either true or false, i.e. 1 or 0.} \\
\text{\quad i -= 10;} \\
\text{else} \\
\text{\quad i += 10;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } ((\text{int })(i > (\text{long } j + 116) - 116)) \quad & \text{either true or false, i.e. 1 or 0.} \\
\text{\quad i = (i - (10L + 116)) + 116;} \\
\text{\quad } & \quad \text{else} \quad \text{\{ \\
\text{\quad \quad \quad i = (i + (10L + 116)) - 116; \\
\text{\quad \quad \quad \}} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } (i <= j) \quad & \text{either true or false, i.e. 1 or 0.} \\
\text{\quad i += 10;} \\
\text{else} \\
\text{\quad i -= 10;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } ((\text{int })(i <= (\text{long } j + 116) - 116)) \quad & \text{either true or false, i.e. 1 or 0.} \\
\text{\quad i = (i + (10L + 116)) - 116;} \\
\text{\quad } & \quad \text{else} \quad \text{\{ \\
\text{\quad \quad \quad i = (i - (10L + 116)) + 116; \\
\text{\quad \quad \quad \}} \\
\end{align*}
\]
Detecting obfuscator bugs

\textbf{Tigress}

\begin{align*}
\text{if } \ (i > j) & \quad \text{ then } i := 10; \\
\text{else } & \quad i := 10; \\
\end{align*}

\(P_1\)

\begin{align*}
\text{if } \ ((\text{int} \ (i > (\text{long} \ j + 116) - 116)) \ {\{ \\
\quad i := (i - (10L + 116)) + 116; \\
\}} \text{ else } \{ \\
\quad i := (i + (10L + 116)) - 116; \\
\}\}
\end{align*}

\(O(P_1)\)

\begin{align*}
\text{if } \ (i \leq j) & \quad \text{ then } i := 10; \\
\text{else } & \quad i := 10; \\
\end{align*}

\(P_2\)

\begin{align*}
\text{if } \ ((\text{int} \ (i \leq (\text{long} \ j + 116) - 116)) \ {\{ \\
\quad i := (i + (10L + 116)) - 116; \\
\}} \text{ else } \{ \\
\quad i := (i - (10L + 116)) + 116; \\
\}\}
\end{align*}

\(O(P_2)\)
Detecting obfuscator bugs

Tigress

\[
\begin{align*}
\text{if } (i > j) & \quad \text{true} \\
 & \quad i -= 10; \\
\text{else} & \\
 & \quad i += 10; \\
\end{align*}
\]

\[
\begin{align*}
\text{if } (i <= j) & \\
 & \quad i += 10; \\
\text{else} & \\
 & \quad i -= 10; \\
\end{align*}
\]

“else” branch is unreachable

“else” branch is unreachable
Any MT test case \((i, j)\) will detect a failure because \(i-10 \neq i+10\).

\[
\begin{align*}
\text{if } (i > j) & \quad i -= 10; \\
\text{else} & \quad i += 10; \\
\end{align*}
\]

\[
\begin{align*}
\text{if } (i \leq j) & \quad i += 10; \\
\text{else} & \quad i -= 10; \\
\end{align*}
\]
Conventional testing without MT can detect the failure only when $i \leq j$ (hence not guaranteed).
In comparison, MT guarantees detection of the failure in this example, because $O(P_1)$ and $O(P_2)$ always output different values for $i$. 

```
if (i > j)
  i -= 10;
else
  i += 10;
```

\[ i = i - 10 \]

\[ i = i + 10 \]
Detecting obfuscator bugs

Cobfusc

MR_{1.2}

P: \texttt{int k = 20;}
   \texttt{\//Rz5Wq3OCvuqsA30uaEY0EvC95AI}n

O(O(O(P))):
\texttt{int k = ((5*(1*1+0)+2)*((2*(1*1+0)+0)*1*(1*1+0)+0)+0)+(3*(2*1+0)+0)); \//}

Rz5Wq3OCvuqsA30uaEY0EvC95AI}n

\textbf{Can’t be compiled.}
Detecting obfuscator bugs
Obfuscator-LLVM

$\textbf{MR}_2$ : When an obfuscator runs at different times for the same program, the behaviour of the output programs should be equivalent.
Detecting obfuscator bugs

Obfuscator-LLVM

1 $ clang -mllvm -bcf -mllvm -boguscf-loop=3 PBP.c
2 $ a.out 10000022
3 10000022
4 $ clang -mllvm -bcf -mllvm -boguscf-loop=3 PBP.c
5 $ a.out 10000022
6 14195494
7 $ clang -mllvm -bcf -mllvm -boguscf-loop=3 PBP.c
8 $ a.out 10000022
9 10000022
Detecting obfuscator bugs

**Obfuscator-LLVM**

```
1 $ clang PBP.c
2 $ a.out 10000022
3 14195494
4 $ clang PBP.c
5 $ a.out 10000022
6 14195494
7 $ clang PBP.c
8 $ a.out 10000022
9 14195494
```
Detecting obfuscator bugs

Stunnix CXX-Obfus

MR₃ : checking that obfuscated source files based on the same input source file are consistent.
Detecting obfuscator bugs

Stunnix CXX-Obfus

1 #include <stdio.h>
2 int j = 1908;
3 int k = 1662;
4 int m = 1734;
5 int n = 468;
6 int p = 1046;
7 int q = 613;
Detecting obfuscator bugs

Stunnix CXX-Obfus

```c
#include <stdio.h>
int j = (0x1cc8+2138-0x1dae);
int k = (0x734+890-0x430);
int m = (0x9e7+132-0x3a5);
int n = (0x1e1+3746-0x1eaf);
int p = 1046;
int q = (0x30b+5768-0x172e);
```

(b)

```c
#include <stdio.h>
int j = (0x1e12+3135-0x22dd);
int k = (0xcb8+260-0x73e);
int m = (0x1da8+1116-0x1b3e);
int n = (0x239b+1251-0x26aa);
int p = (0x90f+4654-0x1727);
int q = (0x6d1+6323-0x1d1f);
```

(c)
Relevant publication

Presentation outline

• MT in security testing
  ◦ A motivating example: the Heartbleed bug
  ◦ Detecting obfuscator bugs
• MT for online search
• MT beyond verification
MT for Online Search
Related publications

- Ongoing projects to report more interesting findings …
MT for Online Search

Significance

- Post-industrial society / knowledge society / information society

Key: to find the *information*. 
MT for Online Search

Significance

- Online search / information retrieval
  - The main activity to discover information
    - cf. “browsing”
  - Very difficult to test.
MT for Online Search Applications
MT for Online Search Applications
MT for Online Search Applications
MT for Online Search Applications

IEEE Xplore®
Digital Library

Search 4,212,058 items

Enter Search Term

Basic Search  Author Search  Publication Search

Advanced Search  Other Search Options

Tickets.com - A Ticket Exploration Engine - Sports, Concerts, Theater, Arts & More - Mozilla Firefox

Music  Sports  Arts & Theater  Family

ProVenue  Love

Search by team, performer, event or venue...
MT for Online Search
MT for Online Search
Applications

Location based service / local search

MT for Online Search

Traditional evaluation methods

- Precision \( \frac{|C|}{|B|} \) expensive
- Recall \( \frac{|C|}{|A|} \) impossible

Difficulties:
1. the sheer volume of online data
2. how to decide “relevance”?
3. how to assess ranking quality?
MT for Online Search

How to identify MRs?

- A top-down approach
MT for Online Search

MR examples

- First identify some General MRs, e.g.:

  If A implies B then
  * results for A $\subseteq$ results for B, and
  * $| \text{results for A} | \leq | \text{results for B} |$.

A general MR can generate many concrete MRs.
a general MR: If A implies B then results for A ⊆ results for B.

a concrete MR: filtering by location

Example: www.yellowpages.com.au
a general MR: If A implies B then results for A \( \subseteq \) results for B.

a concrete MR: filtering by location

(a)

(b)
A general MR: If A implies B then results for A \( \subseteq \) results for B.

Concrete MRs: using logical operations such as AND, OR, EXCLUDE:

results for \((A_1 \text{ AND } A_2)\) \(\subseteq\) results for \(A_1\)
results for \((A_1 - A_2)\) \(\subseteq\) results for \(A_1\)
results for \((A_1)\) \(\subseteq\) results for \((A_1 \text{ OR } A_2)\)
a general MR: If A implies B then results for A \( \subseteq \) results for B.

concrete MRs: using logical operators such as AND, OR, EXCLUDE:

- results for \((A_1 \text{ AND } A_2)\) \(\subseteq\) results for \(A_1\)
- results for \((A_1 - A_2)\) \(\subseteq\) results for \(A_1\)
- results for \(A_1\) \(\subseteq\) results for \((A_1 \text{ OR } A_2)\)

\(A_1, A_2\) can be any search criteria, e.g. query terms and filters
**a general MR:** If A implies B then results for A $\subseteq$ results for B.

**concrete MRs:** using logical operators such as AND, OR, EXCLUDE:
- results for $(A_1 \text{ AND } A_2) \subseteq$ results for $A_1$
- results for $(A_1 - A_2) \subseteq$ results for $A_1$
- results for $(A_1) \subseteq$ results for $(A_1 \text{ OR } A_2)$

$A_1, A_2$ can be any search criteria, e.g. query terms and filters
A general MR: If A implies B then results for A $\subseteq$ results for B.

Concrete MRs: using logical operators such as AND, OR, EXCLUDE:

results for \((A_1 \text{ AND } A_2)\) $\subseteq$ results for \(A_1\)

results for \((A_1 - A_2)\) $\subseteq$ results for \(A_1\)

results for \((A_1)\) $\subseteq$ results for \((A_1 \text{ OR } A_2)\)

A_1, A_2 can be any search criteria, e.g. query terms and filters.
concrete MRs: using logical operators such as AND, OR, EXCLUDE:
results for \((A_1 \text{ AND } A_2)\) ⊆ results for \(A_1\)
results for \((A_1 - A_2)\) ⊆ results for \(A_1\)
results for \((A_1)\) ⊆ results for \((A_1 \text{ OR } A_2)\)

\(A_1, A_2\) can be any search criteria, e.g. query terms and filters.
A1, A2 can be any search criteria, e.g. query terms and filters.

concrete MRs: using logical operators such as AND, OR, EXCLUDE results for (A1 AND A2) ⊆ results for A1
results for (A1 - A2) ⊆ results for A1
results for (A1) ⊆ results for (A1 OR A2)
MT for Online Search

Find search results that include the following:

GLIF 5Y4W

All of these terms
Any of these terms
This exact phrase
None of these terms

Add to search

Words or phrases can be combined with these operators:

AND (intersection of sets)
OR (union of sets)
NOT (complement of a set)

This search did not find any results containing (GLIF OR 5Y4W).
MT for Online Search
Validity must be carefully considered

- Because online search may only return approximate results.
  - We cannot use simple mathematics such as:
    - If $A=1$, $B=2$ then
    - $A+B=3$. 
MT for Online Search
Validity must be carefully considered

- **Dynamic** nature of online data?
  - All tests must be repeatable.
MT for Online Search

Validity must be carefully considered

- Relationship between users’ information needs and logical consistency (MRs)?
  - Users’ perceived quality of ranking decreases if they find strong inconsistencies in search results.
MT for Online Search

Ranking quality

- Most users are more concerned about the first few items.
$RS_0$ (results for \textless term\textgreater )

<table>
<thead>
<tr>
<th>HTML$_1$</th>
<th>HTML$_2$</th>
<th>PDF$_1$</th>
<th>PDF$_2$</th>
<th>TXT$_1$</th>
<th>TXT$_2$</th>
<th>TXT$_3$</th>
<th>TXT$_4$</th>
<th>..........</th>
<th>TXT$_{20}$</th>
</tr>
</thead>
</table>

$RS_1$

$RS_2$ (results for \textless\textless term\textgreater\textless\textless filetype:txt\textgreater\textless\textgreater )

| TXT$_a$ | TXT$_b$ | TXT$_c$ | TXT$_d$ | .......... | TXT$_t$ |
MT for Online Search

**Ranking quality**

- Studied different file types
  - HTML, PDF, TXT
  - Sensitive / vulnerable to hyperlink structure / topology of Web components
  - Needs improvement for less structured files
    - How about multimedia search?

- Further research topic
  - Any bias in search results?
Useful for detecting security vulnerabilities

Undetectable with traditional approaches.
Presentation outline

- MT in security testing
  - A motivating example: the Heartbleed bug
  - Detecting obfuscator bugs
- MT for online search
- MT beyond verification
MT beyond verification

- Software testing can be performed for different purposes, with verification and validation being the core tasks.
- Testing is also used for other types of quality assessment beyond V&V, such as usability and scalability.
MT beyond verification

- Meeting the challenges
  - lack of specifications
  - lack of oracles
  
  when verifying, validating, and assessing large and complex software systems.

- User-oriented approach
Figure adapted from: Mauro Pezzè and Michal Young, *Software Testing and Analysis: Process, Principles and Techniques*
Note: User manuals are not “specifications” --- “an adequate guide to building a product that will fulfill its goals”.
Difficulties in user validation

- A large body of software systems come without specifications, or with incomplete specifications, e.g.:
  - online systems
  - open source development
  - poor software evolution
  - web services
Difficulties in user validation

- The lack of knowledge about the software design and specifications, coupled with the oracle problem, makes user validation extremely difficult.
  - It is hard for users to decide if, and to what degree, the systems are appropriate for their needs.
Difficulties in user validation

- The lack of knowledge about the software design and specifications, coupled with the oracle problem, makes user validation extremely difficult.

  - It is hard for users to decide, to what degree, the systems are appropriate for their needs.

Solution: define MRs from the user’s perspective.

Beyond traditional “verification” and “fault detection”.

Z.Q. (George) Zhou, University of Wollongong
MT beyond verification
From verification to validation

"phrase"

Bing

Returns results that contain the specified phrase, exactly.

Example

"foo"
MT beyond verification
From verification to validation

Isaac Newton - Wikipedia, the free encyclopedia
Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727) was an English mathematician, physicist, astronomer, theologian, and author who is widely regarded as one of the most influential scientists of all time. His contributions to mathematics, physics, and astronomy revolutionized the scientific revolution and continue to influence science and philosophy today.

Isaac Newton - Biography - Philosophical Dictionary
Explore the history and discoveries of Isaac Newton, and optics and his groundbreaking laws of motion, at www.biology.com/people/isaac-newton-9422656

Issack Newton Profiles | Facebook
View the profiles of people named Issack Newton and others you may know...
MT beyond verification
From V&V to quality assessment

- A crucial deficiency in Bing’s functional completeness, which refers to the degree to which the set of functions covers all the specified tasks and user objectives.
ISO/IEC 25010 Product Quality Model and Quality in Use Model

<table>
<thead>
<tr>
<th>Product Quality Model</th>
<th>Product Quality Model (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional suitability</strong></td>
<td><strong>Maintainability</strong></td>
</tr>
<tr>
<td>* Functional completeness</td>
<td>Modularity</td>
</tr>
<tr>
<td>* Functional correctness</td>
<td>Reusability</td>
</tr>
<tr>
<td>Functional appropriateness</td>
<td>Analysability</td>
</tr>
<tr>
<td><strong>Performance efficiency</strong></td>
<td>Modifiability</td>
</tr>
<tr>
<td>Time behaviour</td>
<td>Testability</td>
</tr>
<tr>
<td>Resource utilization</td>
<td>Portability</td>
</tr>
<tr>
<td>* Capacity</td>
<td>Adaptability</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Installability</td>
</tr>
<tr>
<td>Co-existence</td>
<td>Replaceability</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td><strong>Quality in Use Model</strong></td>
</tr>
<tr>
<td>Appropriateness recognizability</td>
<td><strong>Effectiveness</strong></td>
</tr>
<tr>
<td>Learnability</td>
<td>Efficiency</td>
</tr>
<tr>
<td>* Operability</td>
<td>Satisfaction</td>
</tr>
<tr>
<td>* User error protection</td>
<td>Usefulness</td>
</tr>
<tr>
<td>User interface aesthetics</td>
<td>Trust</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Pleasure</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>Comfort</td>
</tr>
<tr>
<td>Maturity</td>
<td>Freedom from risk</td>
</tr>
<tr>
<td>Availability</td>
<td>Economic risk mitigation</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Health and safety risk mitigation</td>
</tr>
<tr>
<td>Recoverability</td>
<td>Environmental risk mitigation</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Context coverage</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>* Context completeness</td>
</tr>
<tr>
<td>Integrity</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td></td>
</tr>
<tr>
<td>Accountability</td>
<td></td>
</tr>
<tr>
<td>Authenticity</td>
<td></td>
</tr>
</tbody>
</table>
ISO/IEC 25010 Product Quality Model and Quality in Use Model

<table>
<thead>
<tr>
<th>Functional suitability</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Functional completeness</td>
</tr>
<tr>
<td>* Functional correctness</td>
</tr>
<tr>
<td>Functional appropriateness</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time behaviour</td>
</tr>
<tr>
<td>Resource utilization</td>
</tr>
<tr>
<td>* Capacity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-existence</td>
</tr>
<tr>
<td>Interoperability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appropriateness recognizability</td>
</tr>
<tr>
<td>Learnability</td>
</tr>
<tr>
<td>* Operability</td>
</tr>
<tr>
<td>* User error protection</td>
</tr>
<tr>
<td>User interface aesthetics</td>
</tr>
<tr>
<td>Accessibility</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Maturity</td>
</tr>
<tr>
<td>Availability</td>
</tr>
<tr>
<td>Fault tolerance</td>
</tr>
<tr>
<td>Recoverability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
</tr>
<tr>
<td>Integrity</td>
</tr>
<tr>
<td>Non-repudiation</td>
</tr>
<tr>
<td>Accountability</td>
</tr>
<tr>
<td>Authenticity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modularity</td>
</tr>
<tr>
<td>Reusability</td>
</tr>
<tr>
<td>Analysability</td>
</tr>
<tr>
<td>Modifiability</td>
</tr>
<tr>
<td>Testability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptability</td>
</tr>
<tr>
<td>Installability</td>
</tr>
<tr>
<td>Replaceability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality in Use Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Effectiveness</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usefulness</td>
</tr>
<tr>
<td>Trust</td>
</tr>
<tr>
<td>Pleasure</td>
</tr>
<tr>
<td>Comfort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Economic risk mitigation</td>
</tr>
<tr>
<td>Health and safety risk mitigation</td>
</tr>
<tr>
<td>Environmental risk mitigation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Freedom from risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context coverage</td>
</tr>
<tr>
<td>* Context completeness</td>
</tr>
<tr>
<td>Flexibility</td>
</tr>
</tbody>
</table>
ISO/IEC 25010 Product Quality Model and Quality in Use Model

<table>
<thead>
<tr>
<th>Product Quality Model</th>
<th>Product Quality Model (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional suitability</strong></td>
<td><strong>Maintainability</strong></td>
</tr>
<tr>
<td>* Functional completeness</td>
<td>Modularity</td>
</tr>
<tr>
<td>* Functional correctness</td>
<td>Reusability</td>
</tr>
<tr>
<td>Functional appropriateness</td>
<td>Analysability</td>
</tr>
<tr>
<td><strong>Performance efficiency</strong></td>
<td>Modifiability</td>
</tr>
<tr>
<td>Time behaviour</td>
<td>Testability</td>
</tr>
<tr>
<td>Resource utilization</td>
<td>Portability</td>
</tr>
<tr>
<td>* Capacity</td>
<td>Adaptability</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Installability</td>
</tr>
<tr>
<td>Co-existence</td>
<td>Replaceability</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td></td>
</tr>
<tr>
<td>Appropriateness recognizability</td>
<td></td>
</tr>
<tr>
<td>Learnability</td>
<td></td>
</tr>
<tr>
<td>* Operability</td>
<td></td>
</tr>
<tr>
<td>* User error protection</td>
<td></td>
</tr>
<tr>
<td>User interface aesthetics</td>
<td></td>
</tr>
<tr>
<td>Accessibility</td>
<td></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td></td>
</tr>
<tr>
<td>* Maturity</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td></td>
</tr>
<tr>
<td>Recoverability</td>
<td></td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Context coverage</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>* Context completeness</td>
</tr>
<tr>
<td>Integrity</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td></td>
</tr>
<tr>
<td>Accountability</td>
<td></td>
</tr>
<tr>
<td>Authenticity</td>
<td></td>
</tr>
</tbody>
</table>
Functional correctness: more examples

- Search for a paper in the ACM digital library:

  “In black and white: an integrated approach to class-level testing of object-oriented programs”
Functional correctness: more examples

(a)

(www) ACM Digital Library

No results were found.

(b) Edit the query directly, or use the form below

No results were found.

(c) In black and white: an integrated approach to class-level test
Functional correctness: more examples

Oracle on the fly: Shoot first and then point to the target.
Functional correctness: more examples

Seoul traffic

25 RESULTS

Seoul - Wikipedia, the en.wikipedia.org/wiki/Seoul

traffic Seoul

No results found for traffic Seoul.
Becampicillin" "Aspirin" "Flecainide"

百度为您找到相关结果2个

"Flecainide" "Aspirin" "Becampicillin"
MT beyond verification

Measuring software performance

Performance efficiency: MT revealed scalability problems of the SUTs.
MT beyond verification

Measuring software performance: scalability

Performance degradation when searching large domains.

steeper
MT beyond verification
Measuring software performance: scalability

Performance degradation when searching large domains.
**Reliability**: degree to which a system performs specified functions under specified conditions for a specified period of time.

We performed MT under different operational profiles and quantified the test results on an hourly basis.
MT beyond verification
Measuring reliability
MT beyond verification
Measuring reliability
MT beyond verification
Measuring reliability
MT beyond verification
Measuring reliability
MT beyond verification
Measuring reliability

Z.Q. (George) Zhou, University of Wollongong
Operability: “easy to operate and control.”

A lot of search operators were tested in MT.

ISO/IEC 25010 Product Quality Model and Quality in Use Model

<table>
<thead>
<tr>
<th>Product Quality Model</th>
<th>Product Quality Model (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operability</strong></td>
<td><strong>Maintainability</strong></td>
</tr>
<tr>
<td>&quot;easy to operate and control.&quot;</td>
<td>Modularity</td>
</tr>
<tr>
<td></td>
<td>Reusability</td>
</tr>
<tr>
<td></td>
<td>Analysability</td>
</tr>
<tr>
<td></td>
<td>Modifiability</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td><strong>Portability</strong></td>
</tr>
<tr>
<td>Co-existence</td>
<td>Adaptability</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Installability</td>
</tr>
<tr>
<td></td>
<td>Replaceability</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td><strong>Quality in Use Model</strong></td>
</tr>
<tr>
<td>Appropriateness recognizability</td>
<td><strong>Efficiency</strong></td>
</tr>
<tr>
<td>Learnability</td>
<td>Usefulness</td>
</tr>
<tr>
<td>Operability</td>
<td>Trust</td>
</tr>
<tr>
<td>* User error protection</td>
<td>Pleasure</td>
</tr>
<tr>
<td>User interface aesthetics</td>
<td>Comfort</td>
</tr>
<tr>
<td>Accessibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td><strong>Freedom from risk</strong></td>
</tr>
<tr>
<td>* Maturity</td>
<td>Economic risk mitigation</td>
</tr>
<tr>
<td>Availability</td>
<td>Health and safety risk mitigation</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Environmental risk mitigation</td>
</tr>
<tr>
<td>Recoverability</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Context coverage</strong></td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>* Context completeness</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
</tr>
<tr>
<td>Non-repudiation</td>
<td></td>
</tr>
<tr>
<td>Accountability</td>
<td></td>
</tr>
<tr>
<td>Authenticity</td>
<td></td>
</tr>
</tbody>
</table>
MT beyond verification

Measuring usability

User error protection: e.g. even if users didn't issue their queries by following the "optimal order."

<table>
<thead>
<tr>
<th>Product Quality Model</th>
<th>Product Quality Model (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional suitability</td>
<td>Maintainability</td>
</tr>
<tr>
<td>* Functional completeness</td>
<td>Modularity</td>
</tr>
<tr>
<td>* Functional correctness</td>
<td>Reusability</td>
</tr>
<tr>
<td>Functional appropriateness</td>
<td>Analysability</td>
</tr>
<tr>
<td>Performance efficiency</td>
<td>Modifiability</td>
</tr>
<tr>
<td>Time behaviour</td>
<td>Testability</td>
</tr>
<tr>
<td>Resource utilization</td>
<td>Portability</td>
</tr>
<tr>
<td>* Capacity</td>
<td>Adaptability</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Installability</td>
</tr>
<tr>
<td>Co-existence</td>
<td>Replaceability</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Usability</td>
</tr>
<tr>
<td></td>
<td>Appropriateness recognizability</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td></td>
<td>* Operability</td>
</tr>
<tr>
<td></td>
<td>User error protection</td>
</tr>
<tr>
<td></td>
<td>User interface aesthetics</td>
</tr>
<tr>
<td></td>
<td>Accessibility</td>
</tr>
<tr>
<td>Reliability</td>
<td>Quality in Use Model</td>
</tr>
<tr>
<td>* Maturity</td>
<td>effectiveness</td>
</tr>
<tr>
<td>Availability</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Satisfaction</td>
</tr>
<tr>
<td>Recoverability</td>
<td>Usefulness</td>
</tr>
<tr>
<td>Security</td>
<td>Trust</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Pleasure</td>
</tr>
<tr>
<td>Integrity</td>
<td>Comfort</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td>Freedom from risk</td>
</tr>
<tr>
<td>Accountability</td>
<td>Economic risk mitigation</td>
</tr>
<tr>
<td>Authenticity</td>
<td>Health and safety risk mitigation</td>
</tr>
<tr>
<td></td>
<td>Environmental risk mitigation</td>
</tr>
<tr>
<td></td>
<td>Context coverage</td>
</tr>
<tr>
<td></td>
<td>* Context completeness</td>
</tr>
<tr>
<td></td>
<td>Flexibility</td>
</tr>
</tbody>
</table>
MT beyond verification

Quality in Use Model: characterizes the impact that the product has on its stakeholders.

Effectiveness: “accuracy and completeness with which users achieve specified goals.”
Context completeness: A system "can be used ... in all the specified contexts of use."

We tested with different operational profiles e.g. different languages, query types, and domains to search.
Ongoing Research

• Location-based search
• GPS navigation software
• eCommerce websites
Questions are welcome