Chapter 6: Software Construction for Robustness
6.1 Robustness and Correctness
健壮性与正确性

May 8, 2020

Outline

- **What are Robustness and Correctness?**

- **How to measure Robustness and Correctness?**

- **Objectives of this chapter**

第六章：进入软件构造最关键的质量特性
——健壮性和正确性。

本节在1-2节的基础上，重申了Robustness and Correctness的重要性，澄清了二者之间的差异，并指明了在软件构造中处理二者的典型技术（防御式编程、异常处理、测试、调试等）

# 1 What are Robustness & Correctness?

Robustness 健壮性

- **Robustness:** "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" (IEEE Std 610.12-1990) 健壮性：系统在不正常输入或不正常外部环境下仍能够表现正常的程度

- **Robust programming 面向健壮性的编程**
  - A style of programming that focuses on handling unexpected termination and unexpected actions. 处理未期望的行为和错误终止
  - It requires code to handle these terminations and actions gracefully by displaying accurate and unambiguous error messages. 即使终止执行，也要准确/无歧义的向用户展示全面的错误信息
  - These error messages allow the user to more easily debug the program. 错误信息有助于进行debug

Robustness principle (Postel's Law)

- **Paranoia (偏执狂) –** A programmer assumes users are out to break their code, and assumes that his own written code may fail or work incorrectly. 总是假定用户恶意、假定自己的代码可能失败

- **Stupidity -** The programmer assumes users will try incorrect, bogus and malformed inputs. 把用户想象成白痴，可能输入任何东西
  - As a consequence, the programmer returns to the user an unambiguous, intuitive error message that does not require looking up error codes.
  - The error message should try to be as accurate as possible without being misleading to the user, so that the problem can be fixed with ease. 返回给用户的错误提示信息要详细、准确、无歧义

- **Robustness principle (Postel's Law):** 对别人宽容点，对自己狠一点
  - Be conservative in what you do; be liberal in what you accept from others.
  - "Be conservative in what you send, be liberal in what you accept"

对自己的代码要保守，对用户的行为要开放

Robustness principle (Postel's Law)

Jon Postel

6 Aug 1943 – 16 Oct 1998
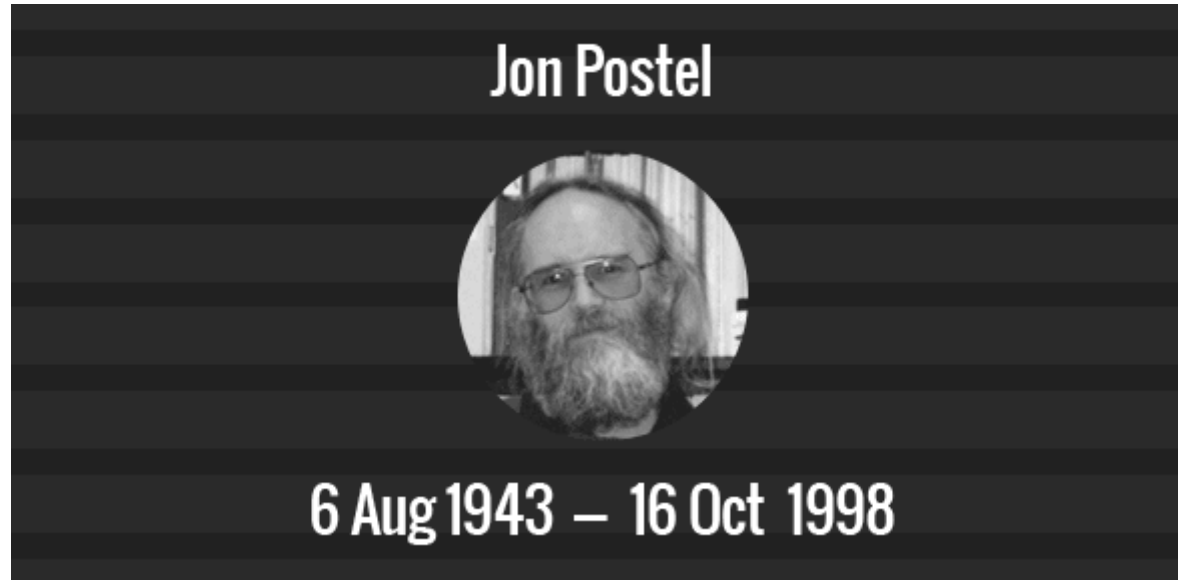
Be conservative in what you do, be liberal in what you accept from others
(often reworded as "Be conservative in what you send, be liberal in what you accept").

Principles of robust programming

- **Dangerous implements -** Users should not gain access to libraries, data structures, or pointers to data structures. 封闭实现细节，限定用户的恶意行为

  - This information should be hidden from the user so that the user doesn't accidentally modify them and introduce a bug in the code.

  - When such interfaces are correctly built, users use them without finding loopholes to modify the interface.

  - The user therefore focuses solely on his or her own code.

- **Can't happen -** Code is modified and may introduce a possibility that an "impossible" case occurs. 考虑极端情况，没有"不可能"

  - Impossible cases are therefore assumed to be highly unlikely instead.

  - The developer thinks about how to handle the case that is highly unlikely, and implements the handling accordingly.

Correctness 正确性

- **Correctness** is defined as the software's ability to perform according to its specification. **正确性：程序按照spec加以执行的能力，是最重要的质量指标！**

- **Robustness vs. correctness: at opposite ends of the scale.**
  - *Correctness* means never returning an inaccurate result; no result is better than an inaccurate result. 正确性：永不给用户错误的结果
  - *Robustness* means always trying to do something that will allow the software to keep operating, even if that leads to results that are inaccurate sometimes. 健壮性：尽可能保持软件运行而不是总是退出

- **Robustness adds built-in tolerance for common and non-critical mistakes, while correctness throws an error when it encounters anything less than perfect input.**
  - 正确性倾向于直接报错(error)，健壮性则倾向于容错(fault-tolerance)

Comparison of Robustness and Correctness

Problem	Robust approach	Correct approach
A rogue web browser that adds trailing whitespace to HTTP headers. 浏览器发出包含空格的URL	Strip whitespace, process request as normal.	Return HTTP 400 Bad Request error status to client.
A video file with corrupt frames. 视频文件有坏帧	Skip over corrupt area to next playable section.	Stop playback, raise "Corrupt video file" error.
A config file with lines commented out using the wrong character. 配置文件使用了非法字符	Internally recognize most common comment prefixes, ignore them.	Terminate on startup with "bad configuration" error.
A user who enters dates in a strange format. 奇怪格式的日期输入	Try parsing the string against a number of different date formats. Render the correct format back to the user.	Invalid date error.

"No Dashes Or Spaces"

- Credit card numbers are always printed and read aloud in groups of four digits.

- Computers are pretty good at text processing, so wasting the user's time by forcing them to retype their credit card numbers in strange formats is pure laziness on behalf of developer.

- In Google Maps, you can enter just about *anything* in the search box and it will figure out a street address.

- 健壮性：避免给用户太大压力，帮助用户承担一些麻烦

Pay My Bill | Return to Bill Summary

✓ Payment Type   ② Payment Details

Payment Frequency: AutoPayment

Step 2: Enter payment information

Method of Payment: Credit Card

Name on Card:

Please enter the name on your card.

Card Number: 4111 1111 1111 1111

Please enter your card number.

Cardholder name   **Expiration date**   **Security code**

Cardholder name   CVC

Please enter numbers only.

What's this?

❶ **Credit card number**   **Billing ZIP code**

4444 4444 4444 4   VISA MasterCard AMERICAN EXPRESS DISCOVER   ZIP

Comparison of Robustness and Correctness

- **Robustness makes life easier for users and third-party developers.**
  - By building in a bit of well thought-out flexibility, it's going to grant a second chance for users with clients that aren't quite compliant, instead of kicking them out cold.

- **Correctness makes life easier for your developers.**
  - Instead of bogging down checking/fixing parameters and working around strange edge cases, they can focus on the one single model where all assumptions are guaranteed.
  - Any states outside the main success path can thus be ignored — producing code that is briefer, easier to understand, and easier to maintain.

| 健壮性： 让用户变得更容易：出错也可以容忍，程序内部已有容错机制 | 正确性： 让开发者变得更容易：用户输入错误，直接结束。 （不满足precondition的调用） |

Comparison of Robustness and Correctness

> Internally, seek correctness; Externally, seek robustness.

- **Externally and Internally:**

  - **External interfaces (UI, input files, configuration, API etc)** exist primarily to serve users and third parties. Make them robust, and as accommodating as possible, with the expectation that people will input garbage.

  - An application's **internal model** (i.e. domain model) should be as simple as possible, and always be in a 100% valid state. Use invariants and assertions to make safe assumptions, and just throw a big fat exception whenever you encounter anything that isn't right.

  - Protect the internal model from external interfaces with an anti-corruption layer which maps and corrects invalid input where possible, before passing it to the internal model.在内外部之间做好隔离，防止"错误"扩散

- **Make your external interfaces robust, and make your internal model correct** 对外的接口，倾向于健壮；对内的实现，倾向于正确

  - If you ignore users' needs, no one will want to use your software.

  - If you ignore programmers' needs, there won't *be* any software.

Comparison of Robustness and Correctness

- **Safety critical applications tend to favor correctness to robustness.**

  – It is better to return no result than to return a wrong result.

- **Consumer applications tend to favor robustness to correctness.**

  – Any result what so ever is usually better than the software shutting down.

- **Reliability (可靠性)**. The ability of a system to perform its required functions under stated conditions whenever required—having a long mean time between failures.
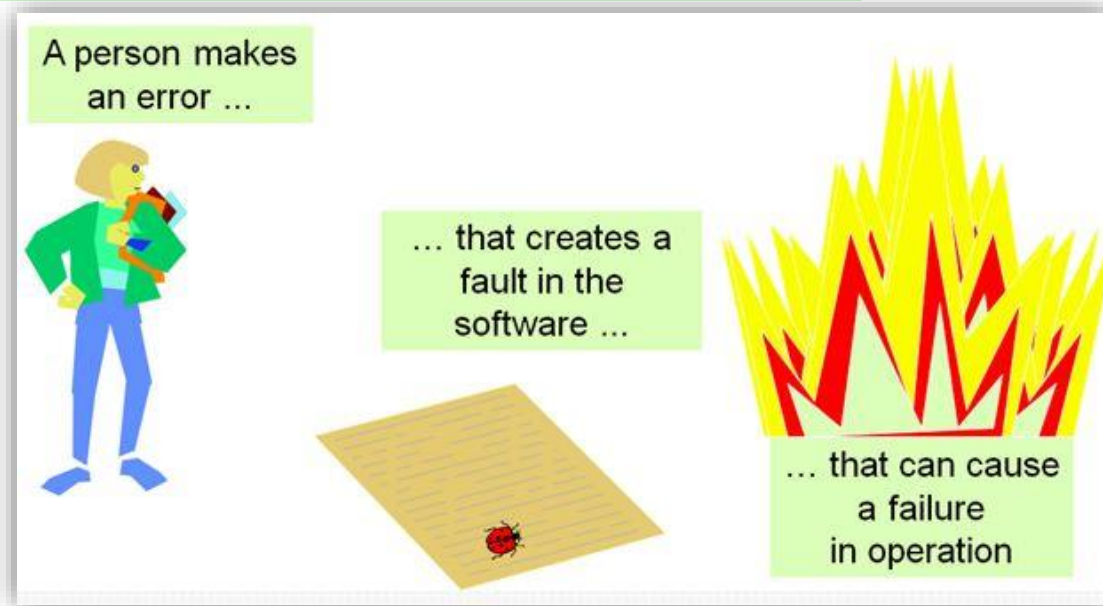
- **Reliability = Robustness + Correctness**

Terms to denote software woes

- **An error is a wrong decision made during the development of a software system.** **(error ≈ mistake)** 程序员犯的错误

- **A defect is a property of a software system that may cause the system to depart from its intended behavior.** 缺陷，**bug**的根源

- **A fault**: wrong or missing function in the code. **(defect ≈ fault, bug)**

- **A failure** is the manifestation of a fault during execution, is the event of a software system departing from its intended behavior during one of its executions. 失效，运行时的外在表现

- 因果关系：**error → defect/fault/bug → failure**

- 程序员犯错导致软件存在缺陷，导致软件运行时失效

Error → Fault → Failure

- **The error: write "+" to "-"**

- **The fault/defect/bug : the wrong result of statement.**

- **The execution will show a failure.**

```
value1 = 5;
value2 = 3;
ans = value1 - value2;
printf("The addition of 5 + 3 = %d.", ans);
```

A person makes an error …

… that creates a fault in the software …

… that can cause a failure in operation

- **A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code.**

- **If this defect is executed, in certain situations the system will produce wrong results, causing a failure.**

Steps for improving robustness and correctness

- **Step 0: To program code with robustness and correctness objectives using assertions, defensive programing, code review, formal validation, etc**

- **Step 1: To observe failure symptoms (Memory dump, stack traces, execution logs, testing)**

- **Step 2: To identify potential fault (bug localization, debug)**

- **Step 3: To fix errors (code revision)**

2 How to measure robustness and correctness?

Mean time between failures (MTBF) 外部观察角度

- **Mean time between failures (MTBF，平均失效间隔时间)** is the predicted elapsed time between inherent failures of a system during operation.

  - MTBF is calculated as the arithmetic mean (average) time between failures of a system.

- **The definition of MTBF depends on the definition of what is considered a system failure.**

  - For complex, repairable systems, failures are considered to be those out of design conditions which place the system out of service and into a state for repair.

  - Failures which occur that can be left or maintained in an unrepaired condition, and do not place the system out of service, are not considered failures under this definition.

Mean time between failures (MTBF)

- **Mean time between failures (MTBF)** describes the expected time between two failures **for a repairable system**, while **mean time to failure (MTTF)** denotes the expected time to failure for a **non-repairable system**.

Time Between Failures

Time to Repair

Time to Failure

System Failure

Resume Normal Operations

System Failure
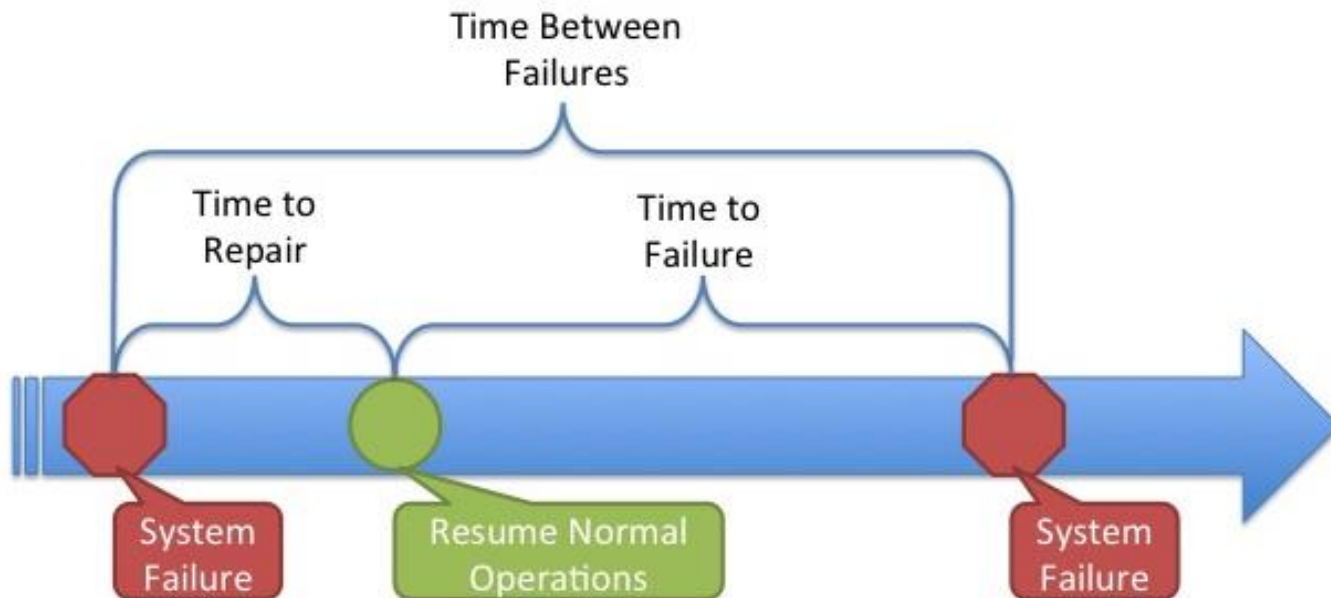
Residual defect rates 内部观察角度（间接）

- **Residual defect rates 残余缺陷率 refers to "bugs left over after the software has shipped" per KLOC: 每千行代码中遗留的bug的数量**
  - **1 - 10 defects/kloc: Typical industry software.**
  - **0.1 - 1 defects/kloc: High-quality validation. The Java libraries might achieve this level of correctness.**
  - **0.01 - 0.1 defects/kloc: The very best, safety-critical validation. NASA and companies like Praxis can achieve this level.**

- **This can be discouraging for large systems.**
  - For example, if you have shipped a million lines of typical industry source code (1 defect/kloc), it means you missed 1000 bugs!

Recall Maintainability (Chapter 6)

For a given problem, Let:

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

Halstead Volume: a composite metric based on the number of (distinct) operators and operands in source code.

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \dfrac{\eta_1}{2} \times \dfrac{N_2}{\eta_2}$
- Effort: $E = D \times V$

The difficulty measure is related to the difficulty of the program to write or understand

The effort measure translates into actual coding time using the following relation,

- Time required to program: $T = \dfrac{E}{18}$ seconds

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation.

- Number of delivered bugs : $B = \dfrac{E^{\frac{2}{3}}}{3000}$ or, more recently, $B = \dfrac{V}{3000}$ is accepted
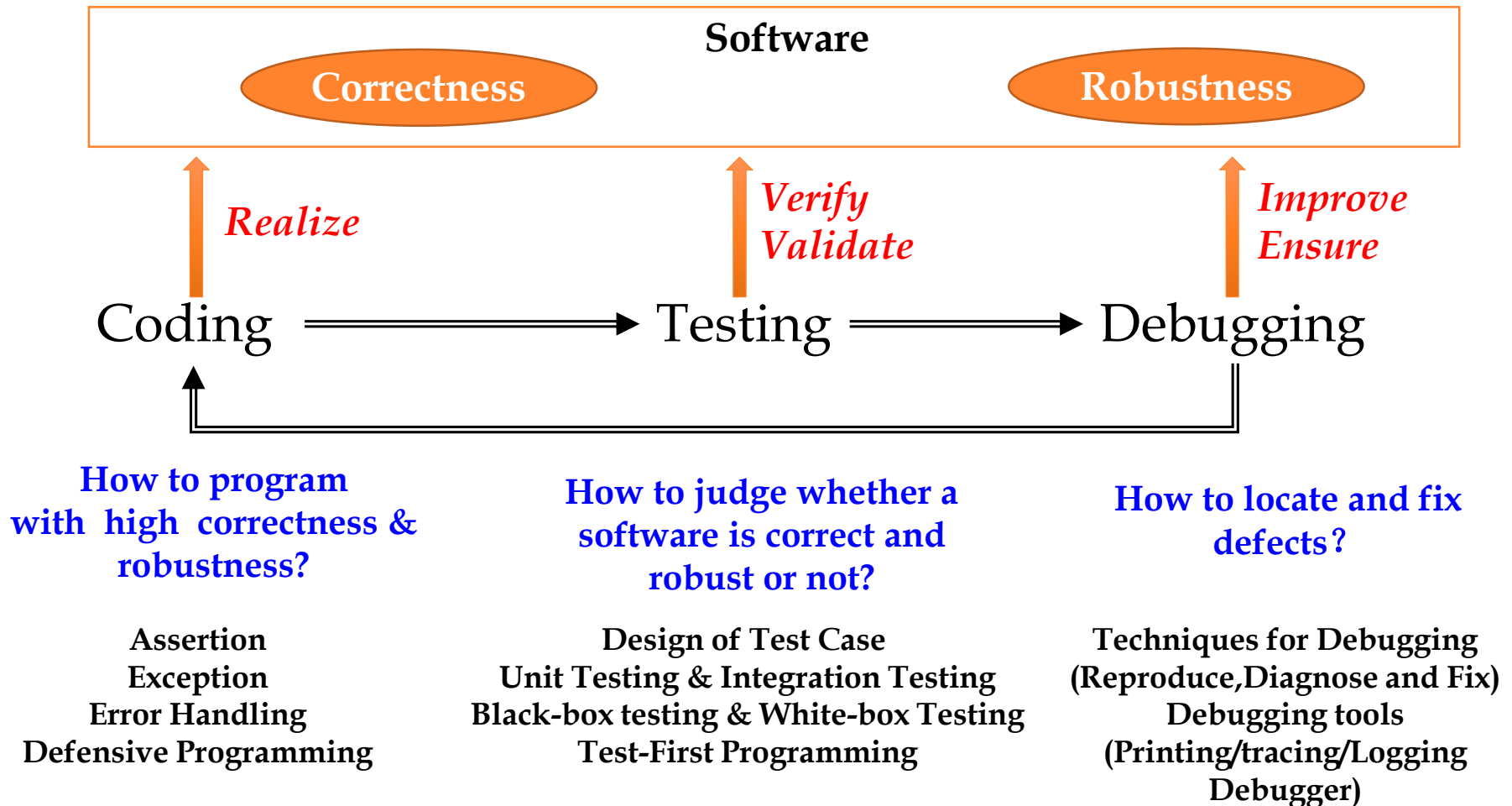
3 Objectives of this chapter

Objectives of this chapter

Software

Correctness

Robustness

Realize

Verify Validate

Improve Ensure

Coding ⟶ Testing ⟶ Debugging

How to program with high correctness & robustness?

How to judge whether a software is correct and robust or not?

How to locate and fix defects？

Assertion
Exception
Error Handling
Defensive Programming

Design of Test Case
Unit Testing & Integration Testing
Black-box testing & White-box Testing
Test-First Programming

Techniques for Debugging
(Reproduce,Diagnose and Fix)
Debugging tools
(Printing/tracing/Logging
Debugger)

The end

May 8, 2020