Is it a Bug?

Lessons learnt from developing safety critical software

Rodney Parkin
IV&V Australia

Overview

- Lessons learnt from developing safety-critical software
- How this relates to software in a more general context
- The role of testers and the conflicts they encounter
- An extended "bug" classification with examples of use

Lessons learnt from developing safety-critical software

- *Safety* is not the same thing as *Reliability*
- The causes of *unsafe* behaviour include not only:
  - Malfunction (ie failure to operate as specified)

… but also …

  - Human error
  - Operation outside intended conditions
  - Unintended environmental interactions
  - Inherently unsafe specifications

Lessons learnt from developing safety-critical software

- Defining "safe" can be difficult
- "Safety" is a property of systems not software
- Public expectations for safety are far greater than realistically achievable reliability

Some examples …

- Implantable defibrillator does not shock when battery low – *an inherently unsafe spec*
- Vital signs monitor allows "alarms disabled" to be saved as startup default – *a poor response to human error*
- Implantable defibrillator shuts down on memory errors – *a poor response to hardware error*
- Implantable defibrillator shuts down on "incidental" interrogation – *a poor response to operation outside the intended environment*

How the lessons learnt can be related to software in a more general context

- *Safety* is not the same thing as *Reliability*
- Causes of *unsafe* behaviour include not only:
  - Malfunction (ie failure to operate as specified)

… as also …

  - Human error
  - Operation outside intended conditions
  - Unintended environmental interactions
  - Inherently unsafe specifications

- *"Operates desirably"* is not the same as "*Operates as specified"*
- Causes of *undesirable* behaviour include not only:
  - Malfunction (ie the traditional "bug")

… but also …

  - Bad response to user error
  - Bad response to operation outside intended conditions
  - Bad environmental inter-dependencies
  - Inherently bad specifications

How the lessons learnt can be related to software in a more general context

- Defining "safe" can be difficult
- "Safety" is a property of systems not software
- Public expectations for safety are far greater than realistically achievable reliability
- Fully defining the desired behaviour can be difficult
- "Desirable" behaviour is a property of systems as a whole, not just particular software
- User expectations extend well beyond just performing the specified functionality

Fully defining the desired behaviour can be difficult …

- Specifications are never complete – there are usually *implied* requirements:
  - The system doesn't mislead its users
  - The system doesn't impose gratuitous restrictions on its users
  - Screens and printouts aren't visually corrupted
  - Screens and printouts don't have spelling mistakes
- Specifications often have implications which are not initially obvious
  - Bad interaction between functions
  - Missing functionality
  - "Just not right"

Bugs can be related to bad, missing, or implied requirements

"Desirable" behaviour is a property of systems as a whole, not just particular software …

- "*Good*" software should not make unnecessary assumptions about its environment
  - Performance and availability of hardware
  - How other parts of the system are configured
- "*Good*" software should not unnecessarily affect its environment
  - Change configuration settings of other software
  - Affect operation of other software
  - Have additional unintended output

Bugs can be related to poor interaction with the rest of the system

User expectations extend well beyond just performing the specified functionality …

- The software should respond gracefully to error
  - User error
  - Hardware error
- The software should respond gracefully when operated outside its intended environment or conditions
  - Degrade rather than fail
  - Warn the user of reduced capability
- The software should fail gracefully
  - Visibly and transparently
  - With no unnecessary consequential damage

Bugs can be related to poor response to error, misuse, or failure

The role of testers

- Functional testing is supposed to find "bugs" – that is, failures to operate as specified
- However, it often uncovers undesirable characteristics that are not specifically related to the spec
- When these are reported, they are often dismissed as "not a bug", and outside the role of the tester
- Testers need a better way to categorise these problems so that they are not dismissed

"It is like that by design"

"The users will never do that"

"It worked in my environment"

"You haven't configured it correctly"

"Read the user manual"

A classification of "bugs"

- Non-conformance to spec
  - Written specification (the traditional "bug")
  - Implied specification
- Specification problem
  - Undesirable characteristics
  - Missing functionality
- Undesirable system interaction
  - Sensitivity to system environment
  - Impact on system environment
- Undesirable response to error
  - User error
  - Hardware error
- Undesirable response to misuse
  - Unintended use scenarios
  - Incorrect configuration
- Undesirable response to system failure

Some examples …

- A bug reporting system is designed for *intranet* use, but has no effective security when installed in the same way on the *internet*
  - *"You weren't supposed to do that"*

  Undesirable response to misuse (incorrect configuration)
- A web-based site accepts credit card payments protected by SSL, but transactions are then forwarded by email (without security)
  - *"that's what it was designed to do"*

  Spec problem (undesirable characteristics)

Some examples …

- A multi-user operating system leaves all open files truncated to zero length after a crash (even unchanged files)
  - *"it won't happen in practice"*

  Undesirable response to system failure
- A test system for some prototype hardware tests a data bus by reading and writing back static data, but the test passes even when the data bus is not physically connected
  - *"it's not possible"*

  Undesirable response to error (hardware error)

Some examples …

- An application's installation script automatically upgrades Java to a newer version, but this stops other applications from working
  - *"the problem is in the other applications"*

  Undesirable system interaction (impact on system environment)
- An electronic parts vendor provides a catalog on CD, you have to install it on your system but it does not have an uninstall
  - *"you don't need to uninstall it"*

  Specification problem (missing functionality)

Some examples …

- A web-based application requiring login expects the user to navigate via links on each page, but using the browser "back" button logs the user out of the web session
  - *"the user shouldn't do that"*

  Undesirable response to error (user error)
- A Windows application assumes only one copy is running at a time, but fails with user switching under Windows XP
  - *"it works in my environment"*

  Undesirable system interaction (sensitivity to system environment)

So … Is it a Bug?

- Many "bugs" found by functional testing are *NOT* spec violations, and are often dismissed as not functional test issues
- However they can be due to:
  - bad, missing, or implied requirements
  - bad interaction with the rest of the system
  - poor response to error, misuse, or failure

Report them as such!