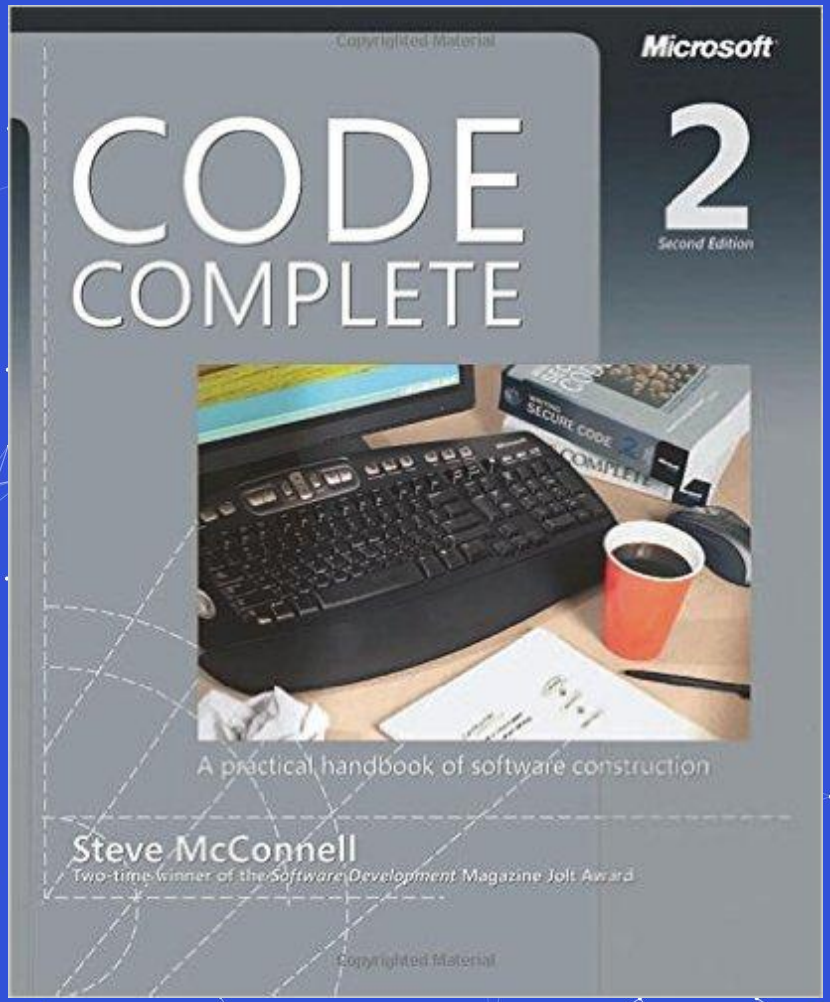
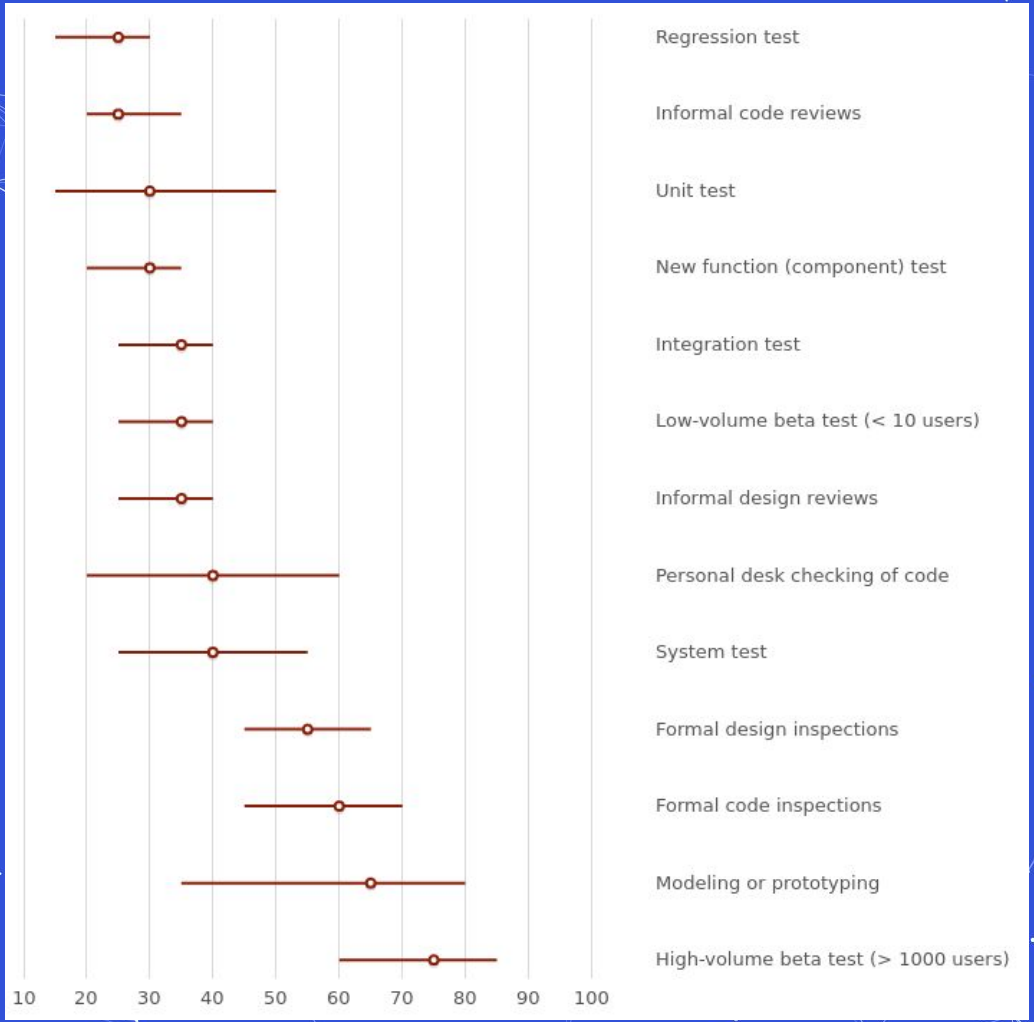
Battling against blackhats

Ethereum European Developers Conference
Feb 2017 - Joseph Chow

“… formal design and code inspections rank as the most effective methods of defect removal yet discovered … [it] can top 85%, about twice those of any form of testing.” Capers Jones, Applied Software Measurement, 3rd Ed. McGraw Hill 2008


Copyrighted Material
Microsoft
CODE
COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award
Copyrighted Material


Regression test
Informal code reviews
Unit test
New function (component) test
Integration test
Low-volume beta test (< 10 users)
Informal design reviews
Personal desk checking of code
System test
Formal design inspections
Formal code inspections
Modeling or prototyping
High-volume beta test (> 1000 users)
10
20
30
40
50
60
70
80
90
100

When software is decentralized, who owns its security?

ethereum

- Community
- Software will either be defended by the community or crumble at the hands of blackhat attackers
- Need to recruit troops and as much of the community to join the battle against blackhats

Outline of this talk

ethereum

1. Specs
2. Tests (suite)
3. Rollout plans
4. Smart contract making an external to an untrusted contract

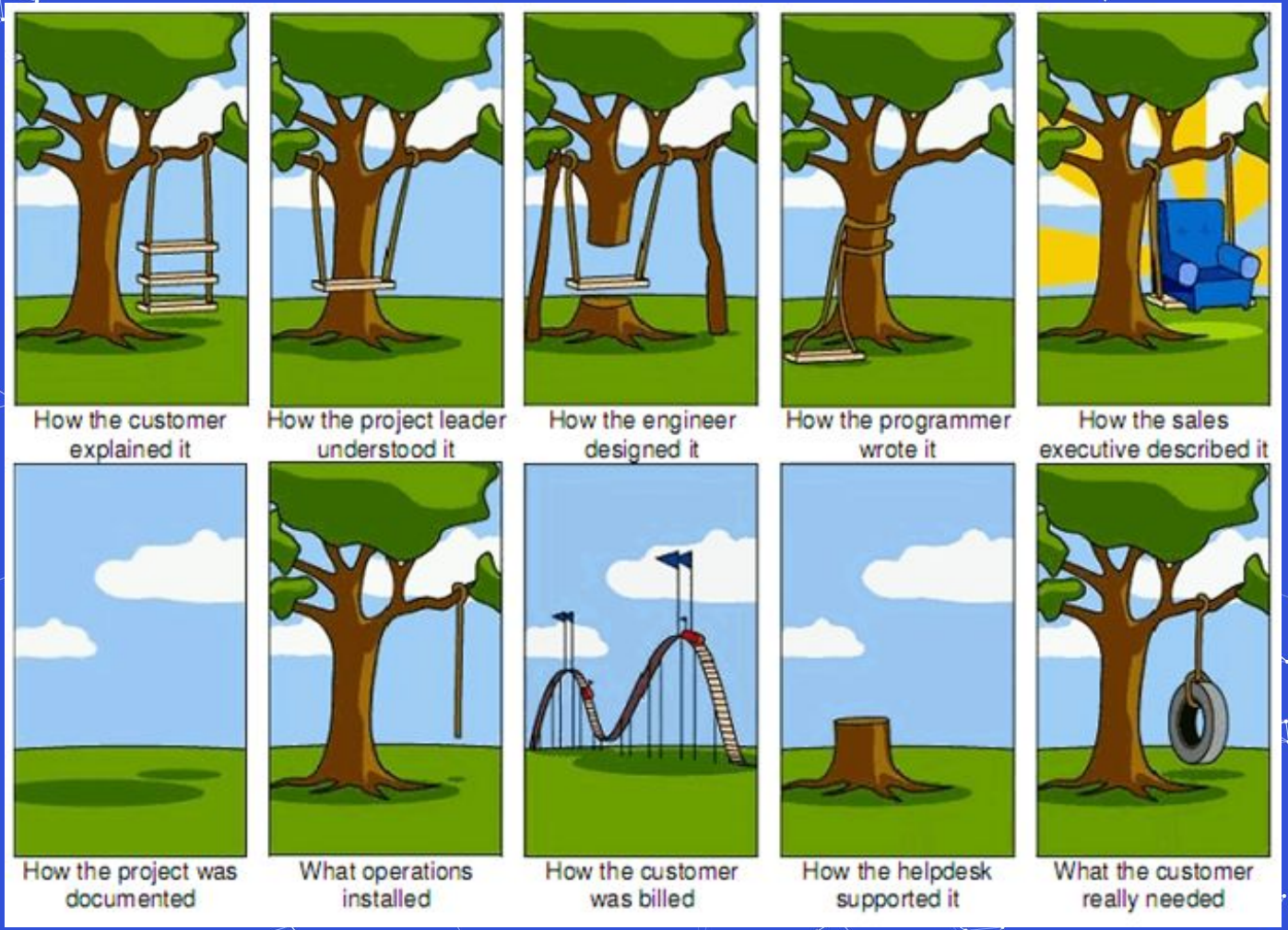
Community resource: for the community, by the community

ethereum

https://github.com/ConsenSys/smart-contract-best-practices

Feel free to submit a pull request, for anything:

- Fix a typo, or example
- Add a link to a community blog post (even
- your own), or other related security info
- Write a new section


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Specs

- Paint the picture of the software for our community and troops
  - No one builds a chair with 3 legs
- Communicate assumptions, expectations, our understanding of our code
- Reduce the effort for community and troops to understand the code
- Help recruit troops
- Bugs in specs are the cheapest to fix
  - Push quality upstream
- Tools like formal verification will only be as helpful as the quality of the specs

Specs shine the light for more troops to join the battle and defence

ethereum

- Blackhats don't often lurk in the daylight of the obvious
- Blackhats dwell in the darkness of subtleties, complexities, corner cases
  - Examples: old state bloat attacks; mispriced opcodes; reentrancy

Tests (suite)

- Tests codify specs
- Tests can be an easy way for troops to understand code
- A gap in tests can lead to a bug
- Priceless for catching regression bugs caused by refactoring
- Write as many tests as you can and focus on design and code inspections

Rollout (deployment) plans

ethereum

- When will specs be public?
- Timeframe for public feedback
- Timeframe for fixing, updating, and improving specs
- Testnets
- Alpha test
- Beta test
- Bounty programs
- A production system needs baking time in production

External Calls

- Avoid calls to untrusted contracts as much as you can
  - Untrusted basically means a contract you've not written

- Assume untrusted contracts are malicious
- Avoid untrustedContract.doSomething()
- Avoid address.call()
  - Avoid address.delegatecall(), address.callcode()
- **After any untrusted call, assume that the state of your contract has been manipulated**

External Calls - Example

```
contract Victim {
  // state
  int x = 2;
  uint private y = 1;

  function foo() {
    x--;
    msg.sender.call.value(10)();
    // x, y is now unknown
  }

  function g() { x++; }
  function h() internal { y++; }
  function bar() {
    if (x%2 == 0) h();
  }

}
```

“recursive” reentrancy

reentrancy

```
contract Untrusted {
  function() {  // fallback function
    v = Victim(msg.sender);
    v.foo();
    v.g();
    v.bar();
  }
}
```

Unknown unknowns and other tips

- Solidity compiler
  - Compiler bugs: unknown unknowns
  - Especially if you're close to launch:
    - Be careful of the latest cutting-edge version
    - Be careful of the optimizer
    - Be careful of "esoteric" Solidity features
- Reuse code, EIP 190: Package Management, Zeppelin-Solidity
- Be aware of blockchain properties

Conclusion

ethereum

Write as many tests and focus on design and code inspections

Specs help recruit troops, and shine light on blackhats

Roll out carefully and allow time for troops to help and code to bake

Pay very close attention to untrusted contracts

https://github.com/ConsenSys/smart-contract-best-practices

Community resource open for all contributions

References

- Applied Software Measurement, Capers Jones, 3rd Ed. McGraw Hill 2008
- Code Complete, Steve McConnell, 2nd Ed. Microsoft Press 2004
- https://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code
- https://www.cs.umd.edu/~basili/publications/technical/T46%20Pt%201%20of%202.pdf
- http://insights.cermacademy.com/2013/03/10-preventing-software-failure-c-capers-jones
- https://github.com/ConsenSys/smart-contract-best-practices

Appendix

General Philosophy

ethereum

- Prepare for failure
  - This is not defeat, but admitting unknown unknowns
- Roll out carefully
  - A production system needs baking time in production
  - Testnets, beta on mainnet, then production mainnet
- Keep contracts simple
- Stay up to date
  - Bibliography at https://github.com/ConsenSys/smart-contract-best-practices Includes community bloggers, Twitter, Reddit...
- Be aware of blockchain properties

Prepare for failure example (from SingularDTV)

ethereum

```
uint fundBalance;

function checkInvariants() constant internal {

    if (this.balance < fundBalance) throw;

}

function emergencyCall() external noEther {

    if (this.balance < fundBalance) {

        if (this.balance > 0 && !workshop.send(this.balance)) {

            throw;

        }

    }

}
```

Use send(), avoid call.value()()

ethereum

- // good

  if(!someAddress.send(100)) { ... // Some failure code }

- // bad

  if(!someAddress.call.value(100)()) { ... // Some failure code }

- send() is safe because attacker only gets 2,300 gas: only enough to log an event
- call.value()() passes along virtually all gas to the attacker's fallback function

Handle errors in raw calls

ethereum

- Raw calls do not progagate exceptions
  - address.send(), address.call(), (delegatecall and callcode) return false if they fail
- Unlike ExternalContract(address).doSomething() which will throw if doSomething() throws
- // good

  if(!someAddress.send(100)) { ... // Some failure code }
- // bad

  someAddress.send(100); // an “unchecked send”

Keep fallback functions simple

ethereum

- Receiving Ether from a .send(), fallback function only gets 2,300 gas: can only log an event
  - function() { LogDepositReceived(msg.sender); }
- Use a proper function if more gas is required
  - function deposit() external { balances[msg.sender] += msg.value; }

- // bad, uses more than 2,300 gas. Breaks senders that use send() instead of call.value()()

function() { balances[msg.sender] += msg.value; }

Denial of Service

- Unexpected throw; the block gas limit; unbounded arrays; misunderstanding gas refunds.
- // INSECURE

```
contract Auction {
    address currentLeader;
    uint highestBid;
    function bid() {
        if (msg.value <= highestBid) { throw; }
        if (!currentLeader.send(highestBid)) { throw; } // Refund the old leader, and throw if it fails
        currentLeader = msg.sender;
        highestBid = msg.value;
    }
}
```

- A currentLeader that refuses payment will permanently be the leader.
- Throw can't be removed otherwise Call Depth Attack. Solution: favor "pull" over "push"

Favor “pull” over “push” for external calls

ethereum

```
// good
contract auction {
    address highestBidder;
    uint highestBid;
    mapping(address => uint) refunds;

    function bid() external {
        if (msg.value < highestBid) throw;

        if (highestBidder != 0) {
            refunds[highestBidder] += highestBid; // record
the refund that this user can claim
        }

        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    function withdrawRefund() external {
        uint refund = refunds[msg.sender];
        refunds[msg.sender] = 0;
        if (!msg.sender.send(refund)) {
            refunds[msg.sender] = refund; // reverting state
because send failed
        }
    }
}
```

More information

ethereum

"Pull" over "push" for external calls (and payments)

Denial of Service against contracts

Reentrancy and race conditions, and many more

https://github.com/ConsenSys/smart-contract-best-practices

Feel free to submit a pull request, for anything:

- Fix a typo, or example
- Add a link to a community blog post (even your own), or other related security info
- Write a new section