







Testing Ethereum Apps

- Does what it's supposed to do
- Doesn't do what it's NOT supposed to do
 - Boundary conditions
- Fulfills requirements



Testing Ethereum Apps

- Start thinking about testing BEFORE creating your app!
 - Design software tests while you define requirements
- Don't skimp on testing
 - Code deployed to the blockchain is immutable
 - Only way to fix bugs is to stop using that code



Live Public Blockchain

- Mainnet
- Pros
 - No simulations

 - Transactions happen in real-time
 Mining activity reflects how timing will actually work
- Cons
 - Slower transactions
 - Costs moneyImmutable
- Don't test on mainnet!



Public Test Blockchain

- Ex: Ropsten
- Pros
 - Sharing an environment, like the live blockchainVery similar to mainnet

 - Mining activity
 - Consensus mechanism and timing might be different, though
 - No cost
- Cons
 - Slow
 - Mining activity isn't exactly like mainnet



Local Test Blockchain

- Private
- You control all aspects
- Ex: Ganache
- Pros
 - Free
 - Fast
 - Can auto-mine
 - Easy to reset
- Cons

 - No mining activityNot interactive environment



Ethereum Testing Strategy

- First, test on private test blockchain
- Then, test on public test blockchain
- Finally, test on mainnet



6 Steps to Testing Your dApp

- 1. Write smart contract code and test cases
- 2. Compile code
- 3. Deploy code to a blockchain
- 4. Run test cases
- 5. Identify failure causes (bugs) and propose changes to address failures
- 6. Return to step 1 and repeat process until all failures are addressed







- Smart Contract Data Code
 - Code can be downloaded from accompanying files, sourceCode > supplyChainApp > contracts
 - If you have been following along with the project, you will already have imported the code into your Truffle directory







Solidity Smart Contract Options

- Command-line interface (Solidity)
 - Tedious
 - Direct access to invoke any function
 - Doesn't work for batches
- Solidity smart contracts
 - Test other smart contracts
- JavaScript



Testing Using the Command Line

- Simple tests
 - Invoke a function one at a time
- Quick and flexible
- Good for one-time tests



Testing Code Operation

- Overflows and underflows
 - Check that numbers aren't larger or smaller than allowed
- Valid return values
 - Check that each function returns the correct values for the caller
 - If calculations are made, check that the answer is correct
- Boundary conditions
 - Code can handle data that meets or exceeds expected limits



Testing Code Operation

- Iteration limits
 - Test that each loop iterates correctly
- Input and output data formats
 - Data provided in unexpected format can still be handled
- Input and output data validation
 - Invalid characters are sanitized or rejected



Steps for Command-Line Testing

- 1. Get smart contract address from Truffle
- 2. Invoke smart contract's functions and examine return values







- Smart Contract Data Code
 - The test.txt document can be copied from the following slides or found in the accompanying files, sourceCode > supplyChainApp > test > test.txt
 - Be sure to replace the A-G addresses with your Ganache addresses at the beginning and throughout the text file. If you use a text editor, you can do a Find and Replace to speed up this step.



Accounts:

/* Be sure to copy the addresses for A-G from your own Ganache addresses at the beginning and throughout this file.
If you use a text editor, you can do a Find and Replace to speed up this step.
*/

A-0: 0x8858d98eC700363a2A1D9308c7312653d186f9B0
B-1: 0xd295d0BF5Fb583219CB7b8AB1a3F3f5E218D0442
C-2: 0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84
D-3: 0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002
E-4: 0x7776756fbA7e1bF1883D97D89D72C5b0510b189e
F-5: 0xD00e57997d5002423234d5C651CeA86f0e14E8FA
G-6: 0xAE2bCafCb611820359Ae72907b23543EcB15DC41



```
supplyChain.deployed().then(function(instance) {return instance });
// Create 3 manufacturer participants (A, B, C)
supply Chain. deployed (). then (function (instance) \ \{return instance.add Participant ("A", "passA", "0x8858d98eC700363a2A1D9308c7312653d186f9B0", "Manufacturer") \ \});
supplyChain.deployed().then(function(instance) {return instance.addParticipant("B","passB","0xd295d0BF5Fb583219CB7b8AB1a3F3f5E218D0442","Supplier") });
supplyChain.deployed().then(function(instance) {return instance.addParticipant("C","passC","0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84","Consumer") });
// Create 2 supplier participants (D, E)
supplyChain.deployed().then(function(instance) {return instance.addParticipant("D","passD","0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002","Supplier") });
supplyChain.deployed().then(function(instance) {return instance.addParticipant("E","passE","0x7776756fbA7e1bF1883D97D89D72C5b0510b189e","Supplier") });
// Create 2 consumer participants (F, G)
supplyChain.deployed().then(function(instance) {return instance.addParticipant("F","passF","0xD00e57997d5002423234d5C651CeA86f0e14E8FA","Consumer") });
supply Chain. deployed (). then (function (instance) \ \{return \ instance. add Participant ("G", "passG", "0xAE2bCafCb611820359Ae72907b23543EcB15DC41", "Consumer") \ \});
// Get participant details
supplyChain.deployed().then(function(instance) {return instance.getParticipant(0)});
supply Chain.deployed().then(function(instance) \ \{return\ instance.getParticipant(1)\});
supply Chain.deployed (). then (function (instance) \ \{return\ instance.getParticipant (2)\});
supplyChain.deployed().then(function(instance) {return instance.getParticipant(3)});
supplyChain.deployed().then(function(instance) {return instance.getParticipant(4)});
supplyChain.deployed().then(function(instance) {return instance.getParticipant(5)});
supplyChain.deployed().then(function(instance) {return instance.getParticipant(6)});
```



```
// Create 6 products 100, 101 (owned by A), 200, 201 (owned by B), 300, 301 (owned C)

supplyChain.deployed().then(function(instance) {return instance.addProduct(0, "ABC", "100", "123", 11) });

supplyChain.deployed().then(function(instance) {return instance.addProduct(0, "DEF", "101", "456", 12) });

supplyChain.deployed().then(function(instance) {return instance.addProduct(1, "GHI", "200", "789", 13, {from: "0xd295d0BF5Fb58319CB7b8AB1a3F3F5E218D0442"}) });

supplyChain.deployed().then(function(instance) {return instance.addProduct(1, "JKL", "201", "135", 14, {from: "0xd295d0BF5Fb583219CB7b8AB1a3F3F5E218D0442"}) });

supplyChain.deployed().then(function(instance) {return instance.addProduct(2, "MNO", "300", "357", 15, {from: "0x9c4c246bca58D3b821bFfdbdB88D60EBE2727E84"}) });

supplyChain.deployed().then(function(instance) {return instance.addProduct(2, "PQR", "301", "759", 16, {from: "0x9c4c246bca58D3b821bFfdbdB88D60EBE2727E84"}) });

// Get product details

supplyChain.deployed().then(function(instance) {return instance.getProduct(0) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(1) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(2) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(3) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(4) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(4) });

supplyChain.deployed().then(function(instance) {return instance.getProduct(4) });
```



```
// Move products along supply chain: Manufacturer=> Supplier=> Supplier=> Consumer
supplyChain.deployed().then(function(instance) {return instance.newOwner(0, 3, 0, {from: "0x8858d98eC700363a2A1D9308c7312653d186f9B0"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(1, 3, 3, {from: "0xd295d0BF5Fb583219CB7b8AB1a3F3f5E218D0442"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(2, 3, 4, {from: "0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(0, 3, 1, {from: "0x8858d98eC700363a2A1D9308c7312653d186f9B0"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(2, 4, 5, {from: "0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(1, 4, 2, {from: "0xd295d0BF5Fb583219CB7b8AB1a3F3f5E218D0442"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(3, 6, 4, {from: "0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002"}) });
supplyChain.deployed().then(function(instance) {return instance.newOwner(3, 4, 1, {from: "0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002"}) });
supply Chain. deployed (). then (function (instance) \ \{return \ instance.new Owner (3, 4, 3, \{from: "0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002"\}) \ \});
supply Chain. deployed (). then (function (instance) \ \{return \ instance.new Owner (4, 5, 2, \{from: "0x7776756fbA7e1bF1883D97D89D72C5b0510b189e"\}) \ \});
supply Chain. deployed (). then (function (instance) \ \{return \ instance. new Owner (3, 4, 0, \{from: "0x4c538EbFF3a7b70c0FAad645B90D8d6A55B48002"\}) \ \});
supply Chain. deployed (). then (function (instance) \ \{return \ instance. new Owner (4, 6, 0, \{from: "0x7776756fbA7e1bF1883D97D89D72C5b0510b189e"\}) \ \});
supply Chain. deployed (). then (function (instance) \ \{return instance.new Owner (4, 5, 3, \{from: "0x7776756fbA7e1bF1883D97D89D72C5b0510b189e"\}) \ \});
supplyChain.deployed().then(function(instance) {return instance.getProvenance(0) });
{\tt supplyChain.deployed().then(function(instance) \{return\ instance.getProvenance(1)\ \});}
{\tt supplyChain.deployed().then(function(instance) \{return\ instance.getProvenance(2)\ \});}
supplyChain.deployed().then(function(instance) {return instance.getProvenance(3) });
supplyChain.deployed().then(function(instance) {return instance.getProvenance(4) });
supplyChain.deployed().then(function(instance) {return instance.getProvenance(5) });
```



```
//
///
function addParticipant(string name, string pass, address pAdd, string pType) public returns (uint)
function getParticipant(uint p_id) public view returns (string,address,string)
function addProduct(uint own_id, string modelNumber, string partNumber, string serialNumber, uint productCost) public returns (uint)
function getProduct(uint prod_id) public view returns (string,string,string,uint,address,uint)

function newOwner(uint user1_id ,uint user2_id, uint prod_id) onlyOwner(prod_id) public returns(bool)
/* function getProductRegistrationHistory(uint prod_id) public returns (registration[]) */
function getOwnership(uint reg_id) public view returns (uint,uint,address,uint)
/* function getRegistraionList(uint prod_id) public returns (uint) */
function authenticateParticipant(uint uid ,string uname ,string pass ,string utype) public view returns (bool)
```







- Smart Contract Data Code
 - The test.txt document can be copied from the previous slides or found in the accompanying files, sourceCode > supplyChainApp > test > test.txt
 - Be sure to replace the A-G addresses with your Ganache addresses at the beginning and throughout the text file. If you use a text editor, you can do a Find and Replace to speed up this step.

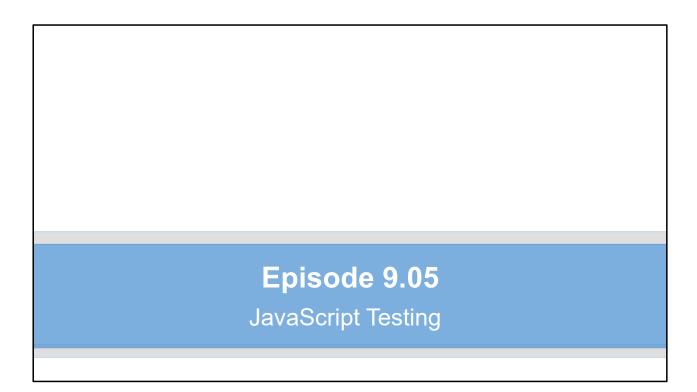






- Smart Contract Data Code
 - The test.txt document can be copied from the previous slides or found in the accompanying files, sourceCode > supplyChainApp > test > test.txt
 - Be sure to replace the A-G addresses with your Ganache addresses at the beginning and throughout the text file. If you use a text editor, you can do a Find and Replace to speed up this step.







JavaScript Testing

- Pros:
 - Access blockchain data and run functions
 - Run test cases each time you change your smart contract
 - Test data to check for expected values
 - Run all tests with one statement
- Cons:
 - More complex

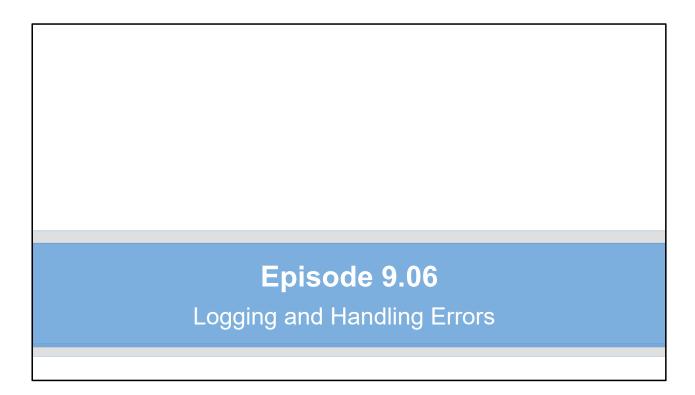


```
var SupplyChain = artifacts.require('./SupplyChain.sol');
contract('SupplyChain', async accounts => {
   it("should create a Participant", async () => {
   let instance = await SupplyChain.deployed();
   let participantId = await instance.addParticipants(0);
   assert.equal(participantId = await instance.addParticipants(0);
   assert.equal(participantId = await instance.addParticipants(0);
   assert.equal(participantId = await instance.addParticipants(0);
   assert.equal(participantId = await instance.addParticipant("B","passB","0xd295d0BF5Fb583219CB7b8ABB1a3F3f5E218D0442","Supplier");
   participant = await instance.addParticipant("B","passB","0xd295d0BF5Fb583219CB7b8ABB1a3F3f5E218D0442","Supplier");
   participant = await instance.addParticipant("C","passC","0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84","Consumer");
   assert.equal(participantDetails," async () => {
    let instance = await SupplyChain.deployed();
    let participantDetails = await instance.getParticipant(0);
   assert.equal(participantDetails, async () => {
    let instance = await SupplyChain.deployed();
    participantDetails = await instance.getParticipant(1);
   assert.equal(participantDetails, async () => {
    let instance = await SupplyChain.deployed();
    participantDetails = await instance.getParticipant(2);
    assert.equal(participantDetails, async () => {
    let instance = await instance.getParticipant(2);
    assert.equal(participantDetails, async () => {
    let instance = await instance.getParticipant(2);
    assert.equal(participantDetails, async () => {
    let
```



- supply_chain.js Code
 - The supply_chain.js code can be copied from the previous slide or found in the accompanying files, sourceCode > supplyChainApp > test > supply_chain.js
 - You can open this document with any text editor, like Notepad in Windows or TextEdit in macOS







Blockchain Error Reporting

- Solidity doesn't track error messages in the standard way
- Solidity doesn't put any output messages in log files
- Why no logging capability?
 - Every node would have to store the log files in the blockchain!



Blockchain Error Handling

- Be explicit in your code
- When something bad happens, return to calling context
 - The client/caller



Handling Errors in Solidity

- Detecting errors and responding to them relies on good communication
 - Caller and smart contract
- Smart contract must consider:
 - Error handling
 - Communicating the error back to the caller
- Smart contracts can pass messages
 - revert()
 - •require()



basicMath.sol Code

```
pragma solidity >=0.4.21 <0.6.0;

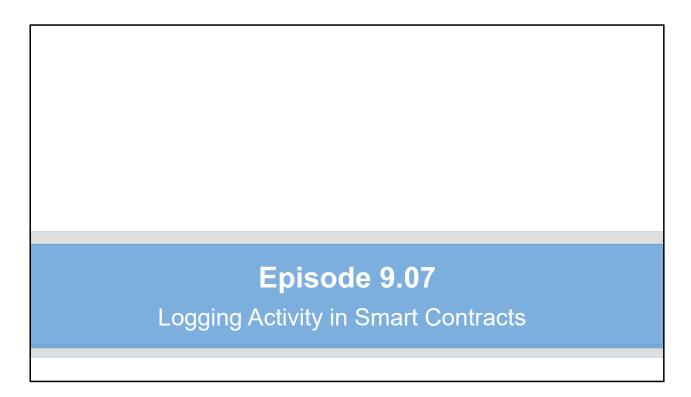
contract basicMath {
   uint256 constant private MAX_UINT256 = 2**256 - 1;

   function add(uint256 _numberA, uint256 _numberB) public pure returns(uint256) {
      return _numberA + _numberB;
   }
}</pre>
```



- basicMath.sol Code
 - The basicMath.sol code can be copied from the previous slide or found in the accompanying files, sourceCode > supplyChainApp > contracts > basicMath.sol







Logging Activity in Smart Contracts

- Solidity events
 - Notify the client that something has happened
 - Stored in the blockchain
 - Costs less
 - Can be indexed for easy lookup
 - Useful for logging activity



- SupplyChain.sol Code
 - Code can be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > SupplyChain.sol
 - If you have been following along with the project, you will already have imported the code into your Truffle directory







Fixing Bugs in a dApp

- Syntax errors
 - Compiler will find these
 - Fix: compile all code and fix highlighted errors
- Semantics error
 - Can cause silent bugs
 - Fix: extensive testing

