Overview

• Last Class
  – More Design Patterns
  – More Software Process

• This Class
  – Scrum Review
  – Testing

• Next Class
  – eXtreme Programming review
  – Use cases
Study Review Final

• Silver Bullet Accidental vs Essential difficulties
• Design Patterns, understand the description, context, consequences (good and bad) for at least one design pattern
• XP Extreme Practices—12 key practices
  – Programmer
  – Management
  – Customer
• Tool chains—be able to name a couple of tools and describe what they do
• Scrum, roles, burndown chart, applicability
• Use cases, writing scenarios, generating graphs
• Testing, QA
  – quality metrics,
  – Common errors
  – Equivalence partitioning, black-box/white-box
  – Definitions of error, fault, failure
Whiteboard Exercise (silver bullet)

• What is a silver bullet?
• How does it relate to software?
• What does it mean to have no silver bullet?
• Who coined the term?
• When might we expect a silver bullet in software?
Silver Bullets

• What is a silver bullet?
  – Any straightforward solution perceived to have extreme effectiveness.

• How does it relate to software?
  – Is there some great technological discovery yet to be invented that will take away all the inherent problems with building software?

• What does it mean to have no silver bullet?
  – "there is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."
  – “We can’t expect to see twofold gain in two years.”
Silver Bullets

• Who coined the term?
  – Fred Brooks, 1986; author of the Mythical Man Month; No Silver Bullet Refired, 1995

• When might we expect a silver bullet in software?
  – An order of magnitude over 40 years might be achievable with several techniques in tandem

• Summary: There is no magic cure for the “software crisis”
Software Tools & Methods

SCRUM REVIEW
Scrum

• Derived from the rugby term “scrum”
  – Despite appearances, is an organized test of strength and skill
• Work is done in sprints (iterations) that form releases
• Key Roles: Scrum Master and Product Owner (On-Site Customer)
• Key Practices: Daily stand-up meeting, time-boxing, and burn-down chart
Scrum Schematic
Scrum Development Process

Roles
- **Product Owner**: Set priorities
- **ScrumMaster**: Manage process, remove blocks
- **Team**: Develop product
- **Stakeholders**: Observe & advise

Key Artifacts
- **Product Backlog**
  - List of requirements & issues
  - Owned by Product Owner
  - Anybody can add to it
  - Only Product Owner prioritizes
- **Sprint Goal**
  - One-sentence summary
  - Declared by Product Owner
  - Accepted by team
- **Sprint Backlog**
  - List of tasks
  - Owned by team
  - Only team modifies it
- **Blocks List**
  - List of blocks & unmade decisions
  - Owned by ScrumMaster
  - Updated daily
- **Increment**
  - Version of the product
  - Shippable functionality (tested, documented, etc.)

Key Meetings
- **Sprint Planning Meeting**
  - Hosted by ScrumMaster; 1-1 day
  - In Product Backlog, existing product, business & technology conditions
    1. Select highest priority items in Product Backlog; declare Sprint Goal
    2. Team turns selected items into Sprint Backlog
    3. Out: Sprint Goal, Sprint Backlog
- **Daily Scrum**
  - Hosted by ScrumMaster
  - Attended by all, but Stakeholders don’t speak
  - Same time every day
  - Team updates Sprint Backlog; ScrumMaster updates Blocks List
- **Sprint Review Meeting**
  - Hosted by ScrumMaster
  - Attended by all
  - Informal, 4-hour, informational
  - Team demos Increment
  - All discuss
  - Hold retrospective
  - Announce next Sprint Planning Meeting

Development Process

Copyright 2004, William C. Wake, William.Wake@acm.org, www.xp123.com
Free for non-commercial use. 1-25-04
Tricks to a successful Sprint Project

• Have a formal kickoff with a fixed time, eg. 10 – 30 days for first increment
• Use a tools for each of the documents
• Be the note-taker yourself
• Listen carefully to what everyone says
  – Pick out and write down the details no matter how small
  – It’s okay to have some participants not do anything due to other projects or priorities on some days
  – Let everyone have their say
• Pass out paperwork and cross out tasks, blocks, and goals in the meeting
  – Include crossed out tasks in handouts
Scrum Case Study
Scrum Case Study

• Legacy project and staff paralyzed by over-specification, infighting and dependencies
• Continued regular development on overall product
• Assembled series of goals, one issue at a time: performance, platform change, live updates
• Ran a sprint project for each goal with different teams
• Created a competitive atmosphere
• Keep to a standard work week < 50 hours, no weekends
Scrum Pushbacks

• Programmer pushback
  – “This is stupid”
    • Nothing more enjoyable than seeing developers transform in the middle of a project and get excited about what they are doing and having a sense of ownership and accomplishment
  – Resent visibility into work habits
    • Very uncomfortable for some staff, to the point of resigning, but need to factor that in when scoping the project
  – “My manager will go to the meeting for me”
    • Need the people actually doing the work—flat organization
  – “These are just like my daily meetings”
    • Productive vs Unproductive, the key is the structure and supporting documents and tasks
Scrum Pushbacks

• Product and Business pushback
  – “How do you keep track of how far along the project is?”
    • It’s a 30 day project, when we’re 15 days in, it’ll be halfway done
  – “How do you know if they’ll meet their deadlines?”
    • On day 30, we’ll be able to see what goals have been completed and which haven’t
  – “How do we grant them an extension?”
    • No extensions—it’s pass or fail
  – “What if they fail? Won’t that hurt the business?”
    • The scope of the project is such that if it succeeds, the business succeeds, if it fails, we factored in the risk and we try something else
  – “I want to be in the meetings.”
    • No. Pigs and Chickens. My meeting, my scope, my process model.
Scrum Burndown

• Useful to also predict
  – Stability of software
  – Resource planning
  – Estimated completion date
  – Problem areas
Whiteboard Exercise

What week will the sprint backlog be completed?

When will the Sprint project end?

Sprint Backlog items vs. Week

Deadline, end of week 9
SOFTWARE TESTING
Definitions

• Error
  – A human action that produces an incorrect result

• Fault
  – Manifestation of an error
  – The result of an error is a fault in the code.

• Failure
  – Observable consequences of a fault or faults

• A failure may be caused by more than one fault and a fault may cause different failures

• It’s theoretically impossible to eliminate all faults from all but the most trivial software programs
  – Complete and exhaustive testing is intractable
Quality Assurance Activities

• Verification
  – Check product against specification
  – Building the system right

• Validation
  – Check product against world (stakeholder expectations)
  – Building the right system

• van Vliet considers all quality assurance activities as testing

• Quality Control (QC) validating physical and environmental constraints and preparations before providing to customer
Testing

• Waterfall model show testing as an activity or box
  – In practice, testing is performed constantly
• There has never been a project where there was too much testing.
  – Products always ship with some defects
• Test cases are a valuable resource
  – Should be managed like code
Whiteboard Exercise (testing)

• Name and describe four types of testing.

• What is the difference between black box and white box testing?
Whiteboard Exercise (testing)

Name and describe four types of testing.

- **Unit testing**- testing parts, e.g. classes
- **Regression**- Testing to ensure that previous functionality still works (after new code has been added)
- **Integration**- Testing two or more components together
- **System**- Testing the whole she-bang
- **Beta**- Pre-release testing with actual users
- **Alpha**- Pre-release testing within the company with developers and customers
Whiteboard Exercise (testing)

• White box testing
  – Glass box, clear box, transparent box, etc.
  – Designs test cases based on internal structure
  – Programming skills to test all possible execution paths through the software
  – Exhaustive method to trigger all possible outputs or output classes
  – Tests based on implementation, so if implementation changes, test cases need to change too
Whiteboard Exercise (testing)

• Black box testing
  – External view of the system as a black box
  – Can be functional or non-functional tests
  – Uses valid and invalid test data
  – Success is measured by correct output
  – Good for uncovering unimplemented parts of specification
  – No guarantee that all paths through code are tested
  – Good candidate for automated testing
Testing Techniques

• Manual Test Techniques
  – Reading
  – Walkthroughs and Inspections
  – Stepwise Abstraction
• Scenario-Based Evaluation
• Correctness Proofs
• Coverage-Based Techniques
• Fault-Based Techniques
• Error-Based Techniques
Test Design Techniques

- Equivalence partitioning
- Boundary value analysis
- Decision table testing
- Pairwise testing
- State transition tables
- Use case testing
- Cross-functional testing
Smoke Testing

• The first test after repairs to provide some assurance that the fix didn’t break other parts of the system
• Make sure that changes didn’t cause catastrophic errors
  – Plumbing—fixing a pipe results in undo water pressure causing a leak in a different place
• Typically done by developers before build is released to testers
• Basic, automated test to validate code changes
  – “Shallow and wide” that touches all areas
Fuzz Testing

• A software technique that provides invalid, unexpected, or random data to the inputs of a program
• File formats and network protocols most common targets of fuzz testing
  – Buffer overflow errors
  – Best place to apply is across trust boundaries
• Can use anything,
  – environment variables,
  – mouse and keyboard events,
  – random sequence of API calls
Stress Testing

• A measure of the stability or robustness of a software system by running the system beyond normal operational capacity, often to the breaking point, to observe the results

• Capacity planning

• Concurrency & concurrent user activities

• Example: 30 day stress test that measured software and hardware behavior with max events per second input
Boundary Testing

• Zero-One-Infinity
• Test case data is based on extremes of the input domain
• Maximum and minimum and just inside/outside the boundaries
• Similar to equivalence partitioning, but solely focuses on “corner cases”
Acceptance Testing

• A black box testing technique performed on a system prior to or as part of the delivery
  – A “ceremonial” handoff process to the customer, i.e. site or field testing
  – QA department accepting a specific build from engineering
  – Handoff of major subsystem to an integration partner
  – Any transfer of ownership

• All or nothing testing technique

• Covers specific items of interest, e.g. latest changes, customer interested, etc.
Fault Seeding

• “Bedbugging”
• Technique where opposing teams plant bugs into software to validate testing process
  – Insert number of known faults to monitor rate of detection
  – Good predictor for how many unknown faults remain
• May do through code inspection
• Programmers/testers may be alerted to known number of bugs or hints to areas
Test Automation

- Use of software in to control the execution of test cases
- GUI automation tools
- Rational Functional Tester, storyboarding
- Good way to automate exploratory automation, but with more coverage
- Code automation tools, e.g. JUnit

```java
import org.junit.*;

public class MultiplicationTest {
    /** Test whether $3 \times 2 = 6$, according to the JVM. */
    @Test
    public void testMultiplication() {
        Assert.assertEquals("Multiplication", 6, 3 * 2);
    }
}
```
Error-Based Techniques

• Certain kinds of problems are known to be difficult
  – Lead to common errors
  – Go after these errors
Common Time and Date Errors

- Y2K
- mm/dd/yyyy (the US) vs dd/mm/yyyy (rest of the world
- 24 hour vs. 12 hour clock
- Formatting
  - Day 36 of a month?
  - Hour 27 of a day?
- Leap Years
Where do Bugs Hide?

• Memory
  – Be careful about memory leaks in Java.
  – The job of the garbage collector is to find objects that are no longer needed by an application and to remove them when they can no longer be accessed or referenced.
  – The key point to remember is that an object is only counted as being unused when it is no longer referenced.
  – If your program is getting a java.lang.OutOfMemoryError after executing for a while, a memory leak is highly likely.

Runtime Heap Summary
Where do Bugs Hide?

• Preventing memory leaks
  – Collection classes, such as **hashables and vectors**, are common places to find the cause of a memory leak. Specially if the class has been declared **static** and exists for the life of the application.
  – Another common problem occurs when you register a class as an **event listener** without bothering to unregister when the class is no longer needed.
  – Member variables of a class that point to other classes simply **need to be set to null at the appropriate time**.
Where do Bugs Hide?

- **Threads / Concurrency**
  - When you get different results on the same input
  - Testing and debugging multithreaded programs is difficult because concurrency hazards do not manifest themselves uniformly or reliably.

- **3:47.01** User 1 initiates tx to pay electricity online
- **3:47.03** User 2 deposits $5,000
- **3:47.05** User 1 finishes tx to pay electricity online $10

- Select in DB
- Account number: 123
- OldBalance: $30
- Update in DB
- Account number: 123
- OldBalance: $30
- Balance: $5,030
- Update in DB
- Account number: 123
- OldBalance: $30
- Balance: $20

- Recommendation: Verify that the information did not change in DB before updating it
Where do Bugs Hide?

- Escaping characters when converting between encoding schemes
  - HTML <-> Java <-> Database

Java - HTML

```java
// HTML Special Chars
if (c == '\n')
    sb.append("\n");
else if (c == '\t')
    sb.append("\t");
else if (c == '\r')
    sb.append("\r");
else if (c == ':')
    sb.append(":");
else if (c == ';')
    sb.append(";");
else if (c == ' ') // Handle Spaces
    sb.append(" ");
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '&')
    sb.append("&");
else if (c == '<')
    sb.append("<");
else if (c == '>')
    sb.append(">");
else if (c == '/')
    sb.append("/");
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\')
    sb.append("\"\"); // Escape Quotation Marks
else if (c == '\"')
```
Security Bugs

SELECT * FROM users
WHERE username = 'NAME'
and password = 'PASSWORD'

Name: a
Password: ' OR 't'='t

SELECT * FROM users
WHERE
username = 'a' and password = ''
OR 't'='t'
Security Bugs

SELECT * FROM users
WHERE username = 'NAME'
and password = 'PASSWORD'

Name: a
Password: 'DROP TABLE users;--'

SELECT * FROM users
WHERE username = 'a' and password = '';
DROP TABLE users;
--'

© Susan Elliott Sim, 2009  Week 9, Slide 44
Coverage-Based Techniques

• Coverage is expressed in terms of how much of the software work product has been covered by testing
  – Percentage of statements, paths, branches, etc.
  – Percentage of requirements

• Control-Flow Coverage

• Data-Flow Coverage

• Both are based on turning the work product into a graph
How Do We Choose Test Cases?

public boolean isValidMonth(int num) {

    if (num >= 1 && num <= 12) {
        return true;
    }
    return false;
}

• Can we test this function for $-2^{31}$ to $2^{31}-1$?
• Is there any difference between `isValidMonth(13)`, `isValidMonth(100)`, `isValidMonth(1000)`?
Equivalence partitioning

- Input data and output results often fall into different classes where all members of a class are related e.g. positive numbers, negative numbers, strings without blanks, etc.
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition- input and outputs lie within partitions
Example: Age of Customers

<table>
<thead>
<tr>
<th>Class</th>
<th>Representative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>-5</td>
</tr>
<tr>
<td>0-12</td>
<td>6</td>
</tr>
<tr>
<td>13-19</td>
<td>15</td>
</tr>
<tr>
<td>20-35</td>
<td>30</td>
</tr>
<tr>
<td>36-120</td>
<td>60</td>
</tr>
<tr>
<td>High</td>
<td>160</td>
</tr>
</tbody>
</table>
Equivalence partitioning

Each equivalence partition is shown as an ellipse.

Input equivalence partitions
Sets of data where all the set members should be processed in equivalent way

System

Output equivalence partitions
Program outputs that have common characteristics

Invalid inputs  Valid inputs

Outputs
Equivalence partitions

Program accepts 4 to 10 inputs which are five-digit integers greater than 10,000

Possible test input values

© Susan Elliott Sim, 2009

Week 9, Slide 50
Search routine specification

procedure Search (Key : ELEM ; T: ELEM_ARRAY;
     Found : out BOOLEAN; L: out ELEM_INDEX) ;

Pre-condition
The array has at least one element
T’FIRST <= T’LAST

Post-condition
The element is found and is referenced by L
( Found and T (L) = Key)

or
The element is not in the array
( not Found and
not (exists i, T’FIRST >= i <= T’LAST, T (i) = Key ))
Search routine - input partitions

• Inputs where the key element is a member of the array
• Inputs where the key element is not a member of the array
• Inputs where a pre-condition does not hold
Search routine - input partitions

<table>
<thead>
<tr>
<th>Array</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single value</td>
<td>In sequence</td>
</tr>
<tr>
<td>Single value</td>
<td>Not in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>First element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Last element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Middle element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Not in sequence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input sequence (T)</th>
<th>Key (Key)</th>
<th>Output (Found, L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
<td>true, 1</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>false, ??</td>
</tr>
<tr>
<td>17, 29, 21, 23</td>
<td>17</td>
<td>true, 1</td>
</tr>
<tr>
<td>41, 18, 9, 31, 30, 16, 45</td>
<td>45</td>
<td>true, 7</td>
</tr>
<tr>
<td>17, 18, 21, 23, 29, 41, 38</td>
<td>23</td>
<td>true, 4</td>
</tr>
<tr>
<td>21, 23, 29, 33, 38</td>
<td>25</td>
<td>false, ??</td>
</tr>
</tbody>
</table>