Lecture 6
Software Quality Measurements
Some materials are based on Fenton’s book

Copyright © Yijun Yu, 2005
Last lecture and tutorial …

Software Refactoring

• We showed the use of refactoring techniques on understanding software, improving its maintainability
• We explained the relationship between refactoring, tuning and restructuring
• Any questions related to design patterns and refactoring so far?
• ……
• The result of such improvements can be measured quantitatively
Today …

On Software Quality Measurements

1. What are measurements?
2. Quality attributes and their metrics
   – Performance metrics
   – Complexity metrics
3. How do you use these numbers?
   – Statistic Analysis to gain understanding on projects
   – Management: Monitoring the evolution of software development
4. Summary

References
1. What are measurements?

• A relation of the real world is “reflected” in that of the math world
  – If A is taller than B, B is taller than C, then A is taller than C

• Preserve the relations in your metrics

• Software measurements
  – Software size?
    LOC
    LOC – comments
    LOC in Python vs. LOC in Fortran?
2. Quality that matters

• Company A beats company B, because of which reason do you think?
  (1) A deliver more features than B
  (2) A has larger market share
  (3) A deliver software with fewer bugs
  (4) A is cheaper

• Killer applications
  – Browser
  – Chips
  – Desktop
  – Operating System
  – Database Systems

• Andy Grove’s story in his book “Only paranoid can survive”
A few more remarks

• Producing quality products has been identified as a key factor in the long term success (i.e. profitability) of organizations
• Quality doesn't happen by chance
• Quality control must be embedded into the process.
• The quality movement
What is software quality?

- Software quality is defined as
  - Conformance to explicitly stated functional [correctness] and non-functional requirements [performance, security, maintainability, usability, etc.]
    *i.e. Build the software described in the system Requirements and Specifications*
  - Conformance to explicitly documented development standards, *i.e. Build the software the right way*
  - Conformance to implicit characteristics that are expected of all professionally developed software, *i.e. Build software that meets the expectations of a reasonable person*: in law this is called the principle of merchantability
Managing Software Quality

1. Define what *quality* means for large software systems
2. Measure Quality of a complete or partial system
3. Devise actions to improve quality of the software
   - Process improvements
     • Process Performance improvements => Product Productivity improvements
   - Product improvements
4. Monitor Quality during development
   - Software Quality Assurance - a team devoted to encouraging and enforcing quality standards
Some quality attributes and metrics

- Performance
- Reliability
- Correctness
- Maintainability
- Security
- Interoperability
- Usability
- Extensibility
- Reusability
- -ilities …

- Time, Space
- MTBF
- # Bugs / Size
- Size, Structureness
- Counter analysis
- Integration
- …
- …
- …
- …
2.1 Performance

It is h/w bound, but can be improved by s/w

- Moore’s Law = 2x speedup every 18 months
- Software improvement for most cases are also possible (algorithms, optimizing compiler)
- It is sometimes more expensive to apply hardware improvements, sometimes more expensive to apply software improvements
- Advice: study the bottlenecks in your program using a profiler
  - parallelism
  - locality
2.1.1 Moore’s law (Intel)

Itanium 2 processor: 410,000,000 transistors
Cell processor: 234,000,000 transistors

http://www.intel.com/research/silicon/mooreslaw.htm
2.1.2 Performance metrics

- Time, in relation to the input size
  - CPU cycles, in relation to the input size
  - Cache misses, in relation to the input size
  - Network delay, system perf.
  - Network throughput, system perf.

- Space, in relation to the input size
  - Workload (memory footprint size), in relation to the input size
  - Network traffic, in relation to the input size
2.2 Software Complexity

- Software code base has increasing complexity – Lehman’s Law #2.
- As a result, the code is harder to maintain.
- This is the central theme of Software Engineering.
- Well-understood complexity metrics
  - McCabe complexity
  - Halstead complexity
- Advices: refactoring or restructuring
2.2.1 Lehman’s law on software complexity
2.2.2 Complexity metrics

• Source size or compiled size
  – Lines of code (LOC)
  – McCabe's complexity
    \[ |V| + |E| - 2 \]
    for a control flow graph \( G=(V, E) \).
  – Halstead's Software Science metrics
    \[ (N_1 + N_2) \log (n_1 + n_2) \]
    \( N_1 = \) operands, \( N_2 = \) operators
    \( n_1 = \) unique operands, \( n_2 = \) unique operators

• OO Software Metrics
  – Cohesion metrics in Packages, Classes, Methods
  – Coupling metrics in Packages, Classes, Methods
3. How to use them in software development process?

1. Modeling
   Quality soft-goal: intention [topic] taxonomy

2. Satisfiable?
   Claim soft-goal: prioritization (bottleneck), metrics, constraints, rationale

3. Selection
   Claim soft-goal: refactoring steps + label propagation

4. Refactoring

5. Releasing
A toy example

• Matrix Multiplication
  
  real*8 A(512,512), B(512,512), C(512,512)
  
  do i = 1 , M
  
    do j = 1, L
    
      do k = 1, N
      
        C(i,k) = C(i,k) + A(i,j) * B(j,k)

• Quality goal: "speedup the program 20x without sacrificing the code complexity 4x"
Some restructuring examples
Loop unrolling

real*8 A(512,512),B(512,512),C(512,512)
do i = 1 , M
  do j = 1 , L
    do k = 1 , N, 4
      C(i,k) = C(i,k) + A(i,j) * B(j,k)
      C(i,k+1) = C(i,k+1) + A(i,j) * B(j,k+1)
      C(i,k+2) = C(i,k+2) + A(i,j) * B(j,k+2)
      C(i,k+3) = C(i,k+3) + A(i,j) * B(j,k+3)
Some restructuring examples
Loop tiling

\[
\begin{align*}
&\text{do } i = 1, M, B1 \\
&\text{do } j = 1, L, B2 \\
&\quad \text{do } k = 1, N, B3 \\
&\quad \quad \text{do } ib = i, \min(i+B1, M) \\
&\quad \quad \quad \text{do } jb = j, \min(j+B2, L) \\
&\quad \quad \quad \quad \text{do } kb = k, \min(k+B3, N) \\
&\quad \quad \quad \quad \quad C(ib,kb) = C(ib,kb)+A(ib,jb) \times B(jb,kb)
\end{align*}
\]
Some restructuring examples

Loop interchanging

real*8 A(512,512), B(512,512), C(512,512)
do k = 1, N
  do j = 1, L
    do i = 1, M
      C(i,k) = C(i,k) + A(i,j) * B(j,k)
Some restructuring examples

Array padding

real*8 A(515,515), B(515,515), C(515,515)
do k = 1, N
do j = 1, L
do i = 1, M
    C(i,k) = C(i,k) + A(i,j) * B(j,k)
Problem

• Given the bunch of possible restructuring, which one is applicable, which one is profitable and which one is disastrous?
• How to represent and reuse the knowledge in many different applications?
• How to apply the knowledge to a new domain?
• Answer: Qualitatively and quantitatively reasoning
3.1 Qualitative reasoning
Decomposition of the performance soft-goal
Claim softgoal influences selections

Other quality softgoal favors prioritizations

Claim softgoal reflect constraints
Considering another quality soft-goal alters prioritizations and selections
Labelling propagations verify the choices
Some remarks

• Each operationalization (thick nodes) is a restructuring (transformation) technique

• They contribute differently to their parent goals. If you do not have the subject (input), these rules generally encode the experiences

• You must collect data to quantitatively fine-tune the goal model
3.2 Quantitative reasoning

- When multiple criteria is concerned, the pareto curve defines the "optimal" solutions
Data collection

Experiment environment

• Hardware: Intel 1.2GHz Pentium 4 processor, with L1 cache (size=8KB, line=64 bytes, associativity=4), L2 cache (size=512KB, line=32 bytes, associativity=8).

• Tools: Datrix for measuring code complexity, VTune for measuring performance through hardware counters
Metrics

- Time index = clockticks(t(p)) / clockticks(p)
- Complexity index = complexity(t(p))/complexity(p) where complexity(p) = 
  v(g) ratio + length ratio + volume ratio
- ratio = (metric − metric_{\text{min}}) / (metric_{\text{max}} - metric_{\text{min}})
- V(G) metric = e − n + 2
  length metric = (N_1 + N_2)
  Volume metric = (N_1 + N_2) \log_2 (n_1 + n_2)
- e is the number of edges, n is the number of nodes in the control flow graph
- N_1 = number of operators
- N_2 = number of operands
- n_1 = number of unique operators
- n_2 = number of unique operands
## Data gathered

<table>
<thead>
<tr>
<th>R</th>
<th>time (sec.)</th>
<th>CPI ($10^6$)</th>
<th>L1 ($10^6$)</th>
<th>L2 ($10^6$)</th>
<th>V (G)</th>
<th>length</th>
<th>volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>63.91</td>
<td>64.9</td>
<td>257.9</td>
<td>185.5</td>
<td>4</td>
<td>96</td>
<td>462</td>
</tr>
<tr>
<td>2</td>
<td>19.06</td>
<td>20.4</td>
<td>78.6</td>
<td>71.8</td>
<td>4</td>
<td>235</td>
<td>1164</td>
</tr>
<tr>
<td>3</td>
<td>4.92</td>
<td>3.36</td>
<td>307.8</td>
<td>1.7</td>
<td>7</td>
<td>185</td>
<td>964</td>
</tr>
<tr>
<td>4</td>
<td>1.54</td>
<td>1.33</td>
<td>129.1</td>
<td>47.8</td>
<td>4</td>
<td>96</td>
<td>462</td>
</tr>
<tr>
<td>5</td>
<td>5.45</td>
<td>6.30</td>
<td>265.6</td>
<td>12.5</td>
<td>4</td>
<td>96</td>
<td>462</td>
</tr>
<tr>
<td>6</td>
<td>1.11</td>
<td>1.23</td>
<td>123.9</td>
<td>44.8</td>
<td>4</td>
<td>96</td>
<td>462</td>
</tr>
<tr>
<td>7</td>
<td>3.30</td>
<td>4.28</td>
<td>324.1</td>
<td>2.1</td>
<td>7</td>
<td>312</td>
<td>1682</td>
</tr>
<tr>
<td>8</td>
<td>0.89</td>
<td>0.89</td>
<td>81.3</td>
<td>3.0</td>
<td>7</td>
<td>312</td>
<td>1682</td>
</tr>
</tbody>
</table>

...
The multi-objective decision making process

Multiple goal metrics changed by transformations
A real example

- Header restructuring project
- Considered one more metric: functionalities
- The experience show that using a new algorithm can dramatically improve the performance (Moore’s law)
- Also refactoring techniques when applied can reduce the complexity (Lehman’s law)
Header restructuring metrics

- Complexity increases when new functionality introduced
- Performance improved when tuning is applied
- Refactoring is applied to control the complexity
- Productivity is concerned to provide larger coverage

**HEADER RESTRUCTURING PROJECT**
- Lasted one year and a half
- 4 Milestones based on GCC 3.4.0
- Works on 7.2 MLOC

- Another round of performance tuning
- New algorithm used
- IBM component

An example of evolution through measured attributes
Your exercise

• Monitor the evolution of your software product by measuring its metrics
  – Statically:
    complexity metrics: LOC, Halstead, McCabe
  – Dynamically:
    Performance metrics: time (clockticks, #instructions), space (cache misses, L1 instruction, L1 data, L2 cache, etc., memory footprint)

• Decide on which is the urgent non-functional task
4. Summary

• The concepts of software measurements
• How to measure some quality metrics
• You need to know your software and manage it by numbers
• Through these numbers, you will know/improve your own capability too
Further readings

• Y. Yu et al. “Software refactoring guided by softgoals”, *REFACE workshop in conjunction with WCRE’03.*
What’s next …

• A Tutorial on software measuring tools
  – How to measure performance?
  – How to measure code complexity?
  – How to measure your code in Eclipse?