CISC327 - Software Quality Assurance

Lecture 25
Software Metrics
Exam 3...

• ...pretty good?
Exam 3...

• Q1(i): I was looking for
  – “yes, it is systematic”
  – or “yes, it is systematic when...”
  – details of at least one kind of mutation

• Q1(ii): at least one of:
  – it changes the files, not the tests
  – it tests the adequacy (completeness) of a test suite rather than the code itself
Exam 3...

• Q2
  – Cost to fix an error during coding
    = 5x–10x cost during requirements
  – Cost to fix an error during operation
    = 40x–1000x (only good choice on exam: 100x)

• Both right 3 marks,
  one right with decent explanation 2 marks,
  one right 1 mark
Exam 3...

• Q3
  - Regression tests:
    • functionality tests—same kind of tests we’ve been working with (e.g. on A1)
    • failure tests—not reproducing old bugs
    • operational tests—real production runs
Exam 3...

• Q4
  – As you may have guessed, I had (too much) fun making up this question
  – I should have just dropped the “Aspect of formal inspections” part and asked:
    how were Bogosys’s inspections inconsistent with formal Fagan code inspections
Exam 3...

• Q4
  – As you may have guessed, I had (too much) fun making up this question
  – I was really hoping someone would just write “WTF” in 7-cm letters across the page (though such an answer would probably not have received full marks)
  – Median of 5/5 on this question
  – Most frequent mistake: not showing that you understood that 5,000 lines per hour is outrageous (I took off 1 mark)
Exam 3...

• Q5
  – (i): What is refactoring not allowed to change about the code?
    • Functionality, behaviour, input/output behaviour
    • “Requirements”, “specifications”
      Not the best answer because “requirements” are not necessarily part of the code
  – (ii): Refactor common code in payout()
    • “not known to be zero” ≠ “known not to be zero”
    • One input (x in lines 4–6, y in lines 9–11) is known to be zero, the other is not known to be zero
Exam 3...

• Q5
  – (iii): Code smell for payout()—entertaining:
    • “Long method” (it’s C so it’s not a method, but okay)

    “No, payout does not have this smell, it’s only 18 lines”

    “Yes, payout has this smell, there is duplicated code”

    “Yes, payout has this smell, because lines 13–16 can be rewritten”
Exam 3...

• Q5
  – (iii): Code smell for payout()—entertaining:
    • “Bad method/variable names”

    “No, payout does not have this smell, x and y are good names, payout is descriptive”

    “Yes, payout has this smell, x and y are not descriptive” (so...what names would you choose? to me, this is an example of code having a smell but being just fine)
Exam 3...

• Q5
  – (iii): Code smell for payout()—entertaining:
    • “Useless comments”
      “No, payout has no comments”
      “No, payout has a very good comment”
Exam 3...

• Q5
  – (iii): Code smell for payout()—entertaining:
    • “Missing comments”

    “No, payout has a long specification”

    “Yes, payout needs more comments”
Exam 3...

• Q5
  – (iii): Code smell for payout()—entertaining:
    • “Switch statements instead of polymorphism”
      
      “No, payout has no switch statements”

    • also it’s in C, so there is no OO polymorphism
      (there are function pointers, though)
The Rest of the Course

• This week (approximately):
  – Software Metrics

• Then:
  – Security
  – Programming Languages and Software Quality
Introduction to Software Metrics

• Today we begin looking at measurement of software quality using software metrics
  – What are software quality metrics?
  – Some basic measurement theory
  – Sample reliability metrics
Software Quality Metrics

• Applying Measurement to Software
  – Software metrics are measurable properties of software systems, their development and use
  – Wide range of different measures:
    • properties of the software product itself
    • the process of producing and maintaining it
    • its source code, design, tests, etc.
  – Examples:

<table>
<thead>
<tr>
<th>Number of failures</th>
<th>Number of lines of code per programmer per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lines of code</td>
<td>Number of failures per 1,000 lines of code</td>
</tr>
<tr>
<td>Time to build</td>
<td>Number of decisions per 1,000 lines of code</td>
</tr>
</tbody>
</table>
What are Metrics Good For?

- **Reliability and Quality Control**
  - Metrics help us to predict and control the quality of our software
  - **Example**: By measuring relative effectiveness of defect detection and removal of various testing or inspection methods, we can choose the best one for our software products
What are Metrics Good For?

- **Cost Estimation and Productivity Improvement**
  - Metrics help us predict effort to produce or maintain our software, and to improve our scheduling and productivity.
  - **Example:** By measuring code production using different languages or tools, we can choose those that give the best results.
What are Metrics Good For?

• **Quality Improvement**
  
  – Metrics help us to improve code quality and maintainability
  
  – **Example**: By measuring complexity of our program code, we can identify sections of code most likely to fail or difficult to maintain
Kinds of Metrics

• Three Basic Kinds
  – There are three kinds of software quality metrics: product metrics, process metrics and project metrics

• Product Metrics
  – Product metrics are those that describe the internal and external characteristics of the product itself
  – **Examples:** size, complexity, features, performance, reliability, quality level
  – **Most** common software metrics are of this kind
  – Product metrics apply regardless of software process
    • but the measurements may be influenced by process!
Kinds of Metrics

• Process Metrics
  – Process metrics measure the process of software development and maintenance, to improve it
  – Examples: effectiveness of defect removal during development, pattern of defect arrival during testing, response time for fix

• Project Metrics
  – Project metrics are those that describe the project characteristics
  – Examples: number of developers, development cost, schedule, productivity
If You Want to Know, Measure...

— “When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science”

Lord Kelvin
Measurement Basics

• ... But Make Sure You Know What You Are Measuring

   “In truth, a good case could be made that if your knowledge is meager and unsatisfactory, the last thing in the world you should do is make measurements. The chance is negligible that you will measure the right things accidentally.”

George Miller
Measurement Basics

• Definition of Measurement
  – **Measurement** is the process of empirical **objective** assignment of numbers to **entities**, to characterize an **attribute**

• Huh?
  – **Entity** = an object or event, such as a source program
  – **Attribute** = a feature or property of an entity, such as the size of the program
  – **Objective** = based on a well-defined **rule** whose results are **repeatable**, such as counting the number of source lines in the program
Measurement Basics

• Definition of Measurement
  – **Measurement** is the process of empirical **objective** assignment of numbers to **entities**, to characterize an attribute

• In Other Words...
  – Each **entity** is given a **number**, which tells you about its attribute
  – **Example**: Each source program has a **source line count**, which tells you about its **size**
# Measurement Basics

- **Example Measurements**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Age</td>
<td>Years since birth</td>
</tr>
<tr>
<td>Person</td>
<td>Age</td>
<td>Months since last birthday</td>
</tr>
<tr>
<td>Source code</td>
<td>Length</td>
<td># Lines of Code (LOC)</td>
</tr>
<tr>
<td>Source code</td>
<td>Length</td>
<td># Executable Statements</td>
</tr>
<tr>
<td>Testing process</td>
<td>Duration</td>
<td>Time in hours from start to finish</td>
</tr>
<tr>
<td>Tester</td>
<td>Efficiency</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Testing process</td>
<td>Fault frequency</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Source code</td>
<td>Quality</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Operating system</td>
<td>Reliability</td>
<td>Mean time to failure / rate of failure occurrence</td>
</tr>
</tbody>
</table>
Measurement Basics

• **Common Mistakes in Software Measurement**
  
  – It's easy to make mistakes in choosing what or how to measure software characteristics
  
  – To avoid mistakes, **stick to the definition of measurement**

1. You must specify **both** an **entity** and an **attribute**, not just one or the other

   • **Example**: Don't just say you are measuring a **program**, say what **property** of the program you are measuring
   
   • **Example**: Don't just say you are measuring the **size** of the software, say what **artifact** of the software you are measuring the size of (e.g., source code)

   • Common bad habit outside software engineering
Measurement Basics

• Common Mistakes in Software Measurement

2. You must define the entity precisely
   • **Example**: Don't just say program, say program source code

3. You must have a good intuitive understanding of the attribute before you propose a measure for it
   • **Example**: We have good evidence that size is related to number of source lines
Measurement Basics

• Common Mistakes in Software Measurement
  – It is a mistake to propose a measure if there is no clear consensus on what attribute it is characterizing
    • Example: Number of defects per KLOC (1000 lines of code) - characterizes quality of code, or quality of testing?
  – It is a mistake to redefine an attribute to fit an existing measure
    • Example: If we've measured # defects found this month, don't mistake that as an indicator of code quality
Kinds and Uses of Software Measurement

• Kinds of Measurement
  – Two distinct kinds of measurement
    • Direct and indirect measurement

• Uses of Measurement
  – Two distinct uses for measurement
    • Assessment (the way things are now)
    • Prediction (the way things are likely to be in the future)
  – Measurement for prediction requires a prediction system
Direct Measurement

• **Some Direct Software Measures**
  
  – **Direct** measures are numbers that can be derived directly from the entity without other information
  
  – **Examples:**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Measurement Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of source code</td>
<td>Measured by number of lines</td>
</tr>
<tr>
<td>Duration of testing process</td>
<td>Measured by elapsed hours</td>
</tr>
<tr>
<td>Number of defects discovered during the testing process</td>
<td>Measured by counting defects</td>
</tr>
<tr>
<td>Effort of a programmer on a project</td>
<td>Measured by person-months worked</td>
</tr>
</tbody>
</table>
## Indirect Measurement

- **Some Indirect Software Measures**
  - **Indirect** measures are numbers that are derived by combining two or more direct measures to characterize an attribute
  
  - **Examples:** (x divided by y)
    
    | Measure                        | Formula                                             |
    |-------------------------------|-----------------------------------------------------|
    | Programmer productivity       | Lines of code produced                              |
    |                               | Person-months of effort                             |
    | Program defect density        | Number of defects                                   |
    |                               | Length of source code                               |
    | Requirements stability        | Original number of requirements                      |
    |                               | Total number of requirements                        |
    | Test effectiveness ratio      | Number of items covered                              |
    |                               | Total number of items                               |
Predictive Measurement

• Prediction Systems
  – Measurement for prediction requires a prediction system, consisting of:
    1. A mathematical model
       Example: $E = a S^b$, where $E$ is the effort to be predicted, $S$ is the estimated size in lines of code, and $a$ and $b$ are constants; if $b = 2$, the model says effort is quadratic in LOC
    2. A procedure for determining the model parameters
       Example: Analyze past project data to determine $a$ and $b$
    3. A procedure for interpreting the results
       Example: Use Bayesian probability analysis to determine the likelihood that our prediction is accurate within 10%
Reliability Metrics

• **Probability of failure on demand (POFOD)**
  - The *probability* that a demand for service from a system will result in a system failure
  - POFOD = 0.001 means that there is a 1/1,000 chance that a failure will occur when a demand is made

• **Rate of occurrence of failures (ROCOF)**
  - The probable *number* of failures likely to be observed in a certain time period (e.g., one hour)
  - Reciprocal of ROCOF is the mean time between failures (MTBF)
    • If ROCOF is two failures/hour, MTBF = 30 min.
Reliability Metrics

• Availability
  – The ability of a system to deliver services when requested, or the probability that a system will be operational when a demand is made for service
  – Availability of 0.9999 means that, on average, the system will be available for 99.99% of operating time

<table>
<thead>
<tr>
<th>Availability</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>The system is available for 90% of the time. In a 24 hour period, the system will be unavailable for 144 minutes</td>
</tr>
<tr>
<td>0.99</td>
<td>In a 24-hour period, the system is unavailable for 14.4 minutes</td>
</tr>
<tr>
<td>0.999</td>
<td>The system is unavailable for 84 seconds in a 24-hour period</td>
</tr>
<tr>
<td>0.9999</td>
<td>The system is unavailable for 8.4 seconds in a 24-hour period, or roughly one minute per week</td>
</tr>
</tbody>
</table>
Software Measurement

• Software metrics help us understand the technical process that is used to develop a software product
  – The process is measured to be improved
  – The product is measured to increase its quality

• But..
  – Measuring software projects is controversial
  – It is not yet clear which are the appropriate metrics for a software project, or whether people, processes, or products can be compared using metrics
Summary

• **Metrics and Measurement**
  – Measurement is about characterizing the attributes of entities
    • Can be direct or indirect
    • Can be for either assessment or prediction

• **References**
  – Sommerville Ch. 23

• **Assignment #4**
  – due Wednesday