SSW-540: Fundamentals of Software Engineering

Software Measurement

Dr. Richard Ens
School of Systems and Enterprises

Python Pointers

Did you know that you can do double assignments in Python?

```
a, b = 1, 2
print a, b        #This will print 1 2
```

This is really useful if you need to swap the values of two variables. You would need a temp variable in most languages, but not Python.

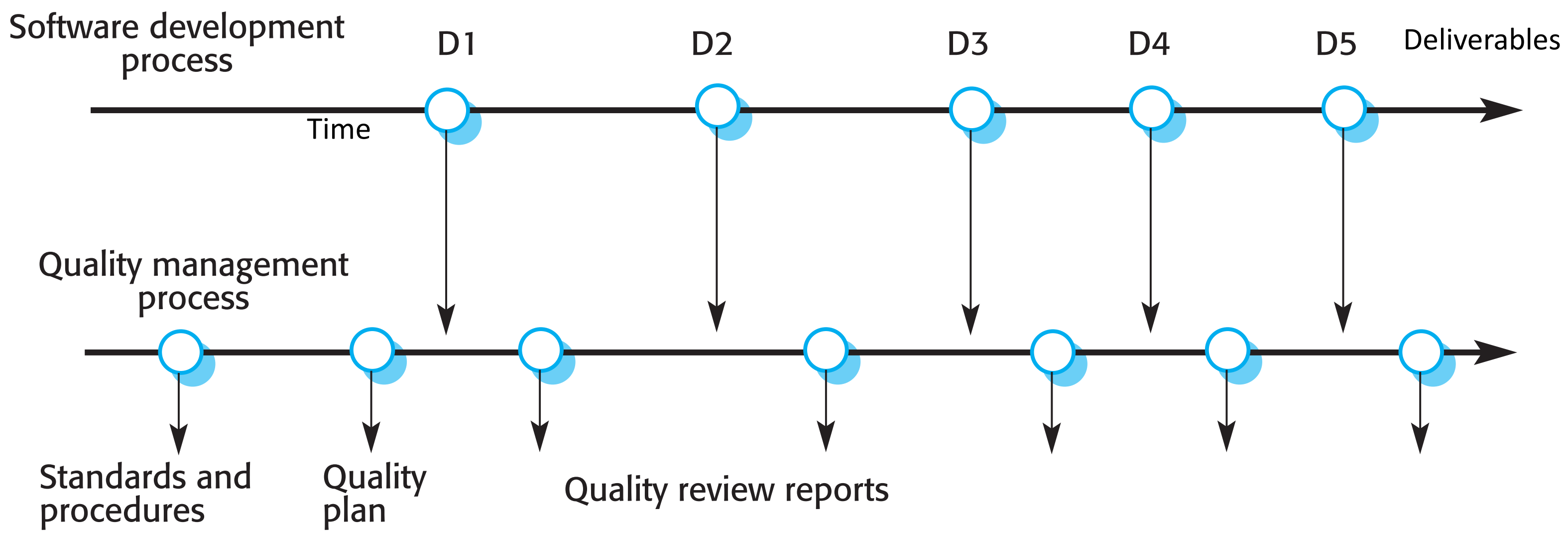
```
a, b = 1, 2
print a, b       #This will print 1 2
a, b = b, a
print a, b       # This will print 2 1
```

But there are some gotcha's! See the 6 minute Professor Rowland video in Module 8 "*A cool Python Assignment Statement*" for more information.

Software measurement is all about quality management

- Quality management tries to ensure that the required level of quality is achieved in a software product, given budget & time constraints.
- Quality software must
  - have a minimum number of defects.
  - reach the required standards of maintainability, reliability, portability, etc.
- Quality management:
  - At the organizational level, requires establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the project level, it requires
    - applying and checking that planned specific quality processes have been followed.
    - establishing and following a quality plan for a project.

Quality management and software development



- Quality management spans the entire software development process.

Software fitness for purpose

- ✧ Has the software been properly tested?
- ✧ Is the software sufficiently dependable to be put into use?
- ✧ Is the performance of the software acceptable for normal use?
- ✧ Does the software meet security guidelines and requirements?
- ✧ Is the software usable?
- ✧ Is the software well-structured, understandable and maintainable?
- ✧ Have programming and documentation standards been followed in the development process?

How do we know the answers to these questions? **We measure**!

Measuring software

- What can we measure?
- Are inspections worth the cost?
- What percent of requirements defects are found in requirements inspections?
- If you are using agile methods, is pair-programming improving your software or your productivity?
- What is the impact of test-driven development on productivity?
- What percent of projects are completed on time? ...are completed on budget? …are completed at all?
- On average, how big are project budget overruns?

Software measurement

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programs, many software development organizations still don't make systematic use of software measurement.
- There are few established standards in this area.

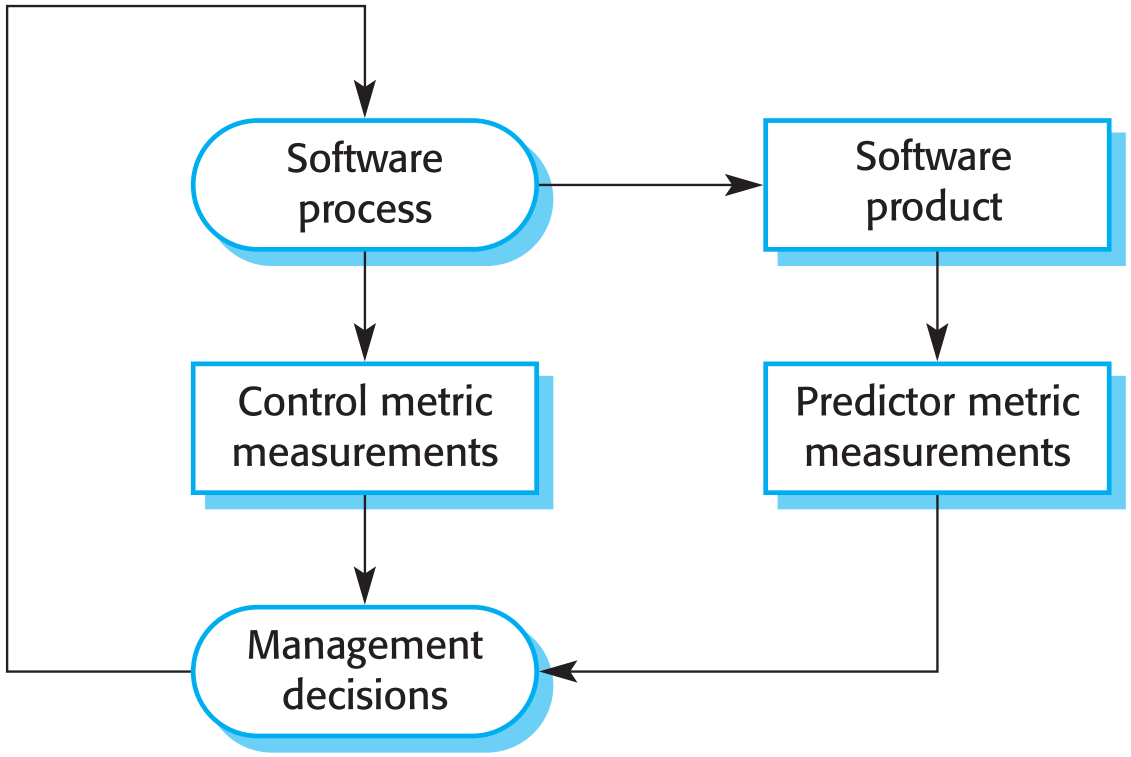
Software metrics

- Software metrics refers to any type of measurement that relates to a software system, process or related documentation.
  - E.g., lines of code in a program, the Fog readability index, number of person-days required to develop a component.
- Metrics allow the software and the software process to be quantified.
- Metrics may be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Types of process metric

- *The time taken for a particular process to be completed*
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- *The resources required for a particular process*
  - Resources might include total effort in person-days, travel costs or computer resources.
- *The number of occurrences of a particular event*
  - Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, the number of defect reports in a delivered system and the average number of lines of code modified in response to a requirements change.

Predictor and control measurements



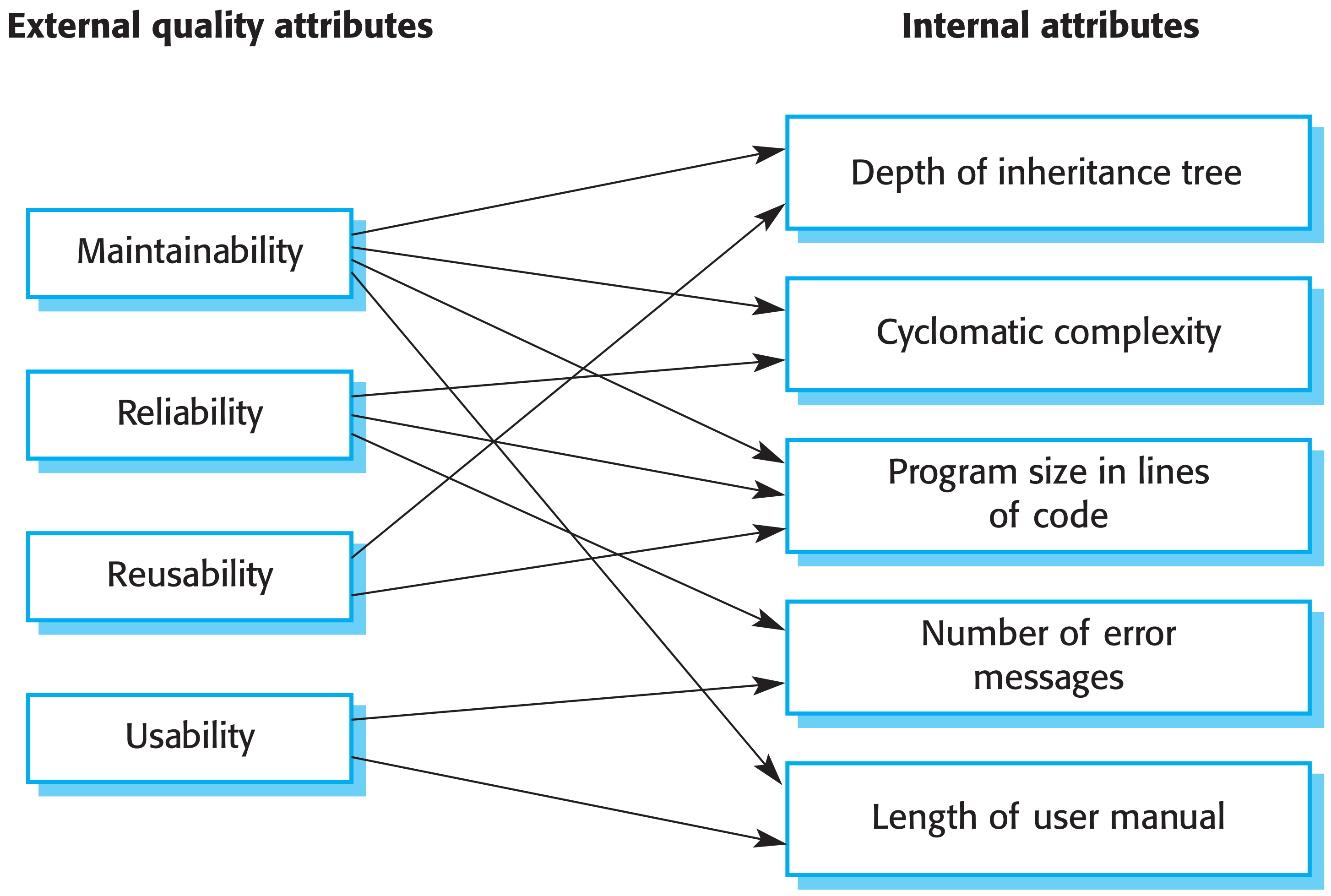
Use of measurements

- Can assign a value to system quality attributes
  - By measuring the characteristics of system components, such as LOC (lines of code) and complexity attributes, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
- Can identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain defects because the complexity makes them harder to understand.

Metrics assumptions

- A software property can be measured accurately.
- A relationship exists between what we can measure and what we really want to know.
  - We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been or can be formalized and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Relationships between internal and external software



Problems with measurement in industry

- ✧ Quantifying the return on investment of introducing an organizational metrics program has proved to be difficult.
- ✧ There are no standards for software metrics or standardized processes for measurement and analysis.
- ✧ In many companies, software processes are not standardized and are poorly defined and controlled.
- ✧ Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS, or through more agile procedures.
- ✧ Introducing measurement adds additional overhead to processes.

Empirical software engineering

✧ Software measurement and metrics are the basis of empirical software engineering.

✧ This is a research area in which experiments on software systems and the collection of data about real projects has been used to form and validate hypotheses about software engineering methods and techniques.

✧ Research on empirical software engineering has not had a significant impact on software engineering practice.

✧ It is difficult to relate generic research to a project that is different from the research study.

Product metrics

- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.

Dynamic and static metrics

- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Static software product metrics

Software metric	Description
Fan-in/Fan-out What if Fan-in = 0?	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.

Static software product metrics

Software metric	**Description**
Cyclomatic complexity	This is a measure of the control complexity of a program, measured by the number of paths in a control flow graph. This control complexity may be related to program understandability, the therefore to maintainability.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite

Object-oriented metric	**Description**
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

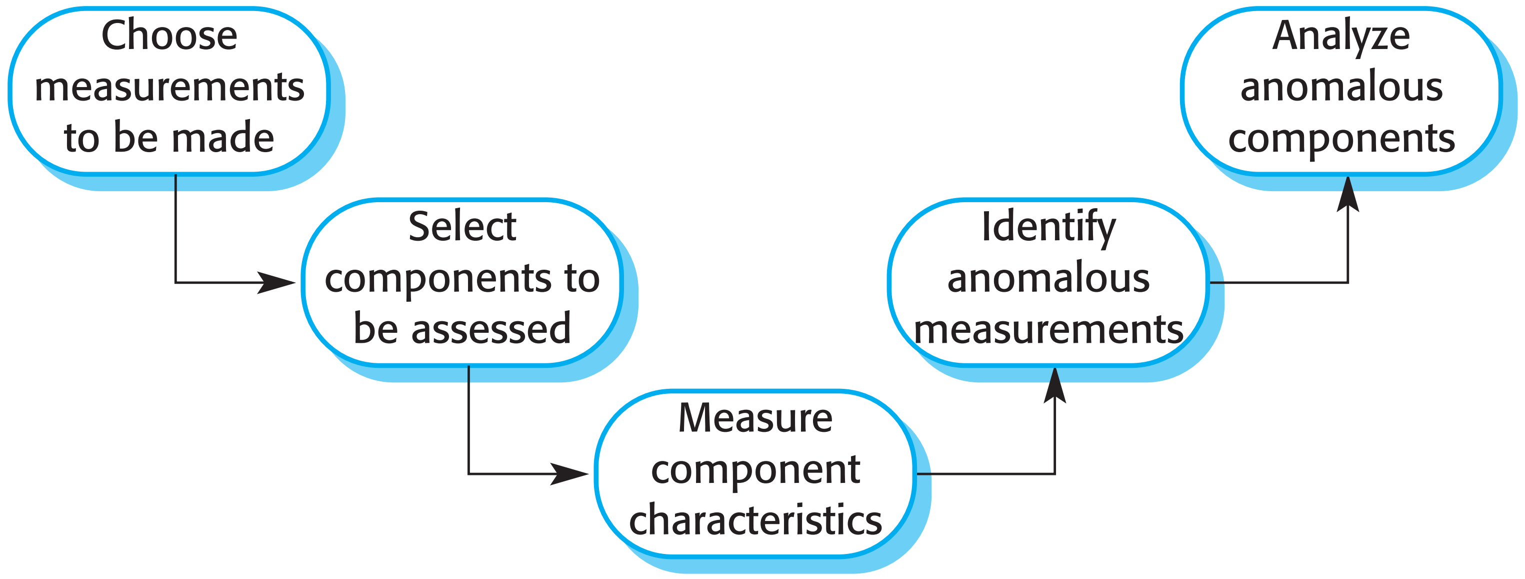
The CK object-oriented metrics suite

Object-oriented metric	Description
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
Lack of cohesion in methods (LCOM)	LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It might not provide any additional, useful information over and above that provided by other metrics.

Software component analysis

- System components can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.
- Some development organizations set inspection thresholds on such metrics (e.g., any components with cyclomatic complexity greater than 10 require a formal review).

The process of product measurement

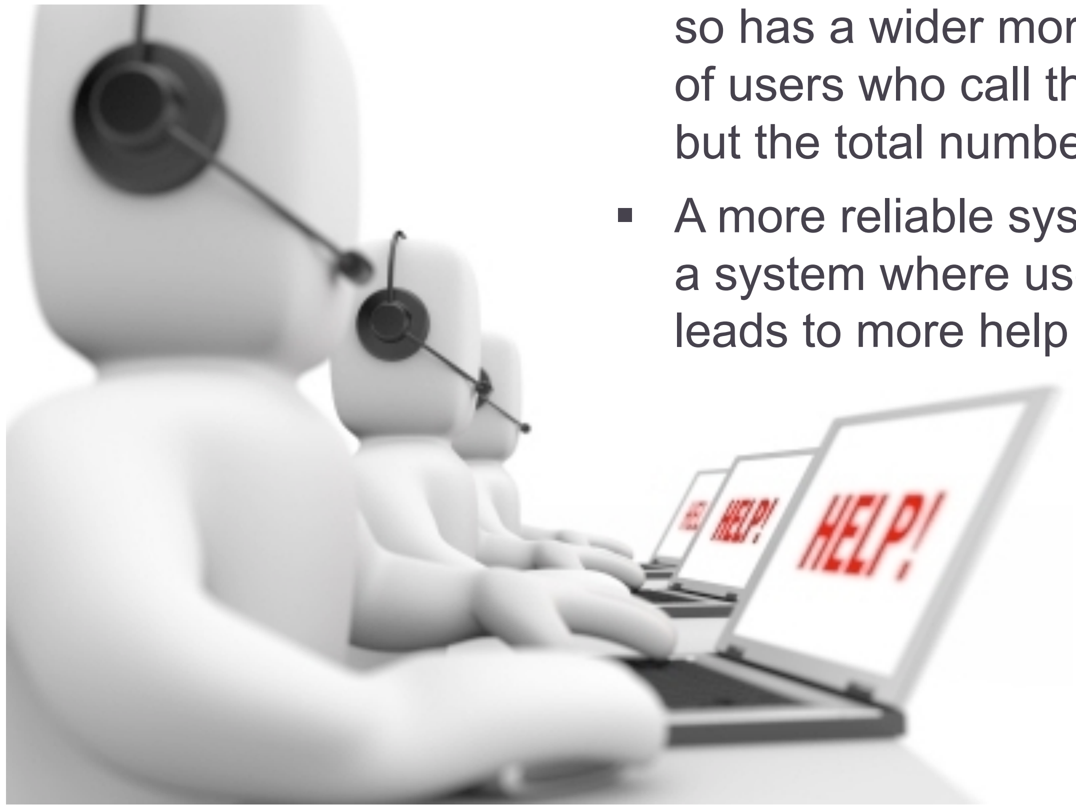

Choose measurements to be made
Select components to be assessed
Measure component characteristics
Identify anomalous measurements
Analyze anomalous components

Measurement ambiguity

- When you collect quantitative data about software and software processes, you have to analyze that data to understand its meaning.
- It is easy to misinterpret data and to make inferences that are incorrect.
- You cannot simply look at the data on its own. You must also consider the context where the data is collected.
- Often the very act of collecting data makes the data collected less meaningful. (E.g., the Hawthorne studies.)

Measurement surprises

- Reducing the number of faults in a program can lead to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total number may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.



Software context

- ✧ Processes and products that are being measured are not insulated from their environment.
- ✧ The business environment is constantly changing and it is impossible to avoid changes to work practice just because they may make comparisons of data invalid.
- ✧ Data about human activities cannot always be taken at face value. The reasons why a measured value changes are often ambiguous. These reasons must be investigated in detail before drawing conclusions from any measurements that have been made.

Software analytics

- ✧ Software analytics refers to analytics specific to software systems and related software development processes.
- ✧ Analytics aim to describe, predict and improve software development, maintenance and management of complex software systems.
- ✧ Analytics empower individuals and teams engaged in software development to gain and share insight from their data to make better decisions.

Software analytics enablers

- ✧ The automated collection of user data by software product companies when their product is used.
  - If the software fails, information about the failure and the state of the system can be sent over the Internet from the user’s computer to servers run by the product developer.
- ✧ The use of open source software available on platforms such as Sourceforge and GitHub and open source repositories of software engineering data.
  - The source code of open source software is available for automated analysis and this can sometimes be linked with data in the open source repository.

Analytics tool use

- Tools should be easy to use as managers are unlikely to have experience with analysis.
- Tools should run quickly and produce concise outputs rather than large volumes of information.
- Tools should make many measurements using as many parameters as possible. It is impossible to predict in advance what insights might emerge.
- Tools should be interactive and allow managers and developers to explore the analyses.

Status of software analytics

- Software analytics is still immature and it is too early to say what effect it will have.
- Not only are there general problems of ‘big data’ processing, our knowledge depends on collected data from large companies.
  - This is primarily from generic software products and it is unclear if the tools and techniques appropriate for these products can also be used with custom software.
- Small companies are unlikely to invest in the data collection systems that are required for automated analysis so may not be able to use software analytics.

Six areas of software metrics

1. Code
   - Static
   - Dynamic
2. Productivity of programmers and others on the project
3. Design
4. Testing
5. Maintainability
6. Management
   - Cost
   - Duration, time
   - Staffing

Why do we care about software size?

- Size is one of the 5 core metrics behind software estimation:
  - Schedule (duration)
  - Effort (cost)
  - Quality (defects)
  - Productivity
  - Size (scope)
- Without software size, it's hard to estimate:
  - How long a software project will take
  - How much it will cost
  - How many people we will need
  - How many defects we can expect to find during testing
  - How productive we are likely to be

There's a non-linear relationship between size and schedule, effort (cost) and defects.

Estimating size is easy, right?

- ✧ No!
- ✧ During each phase of the life cycle, we can *progressively elaborate* our estimate of size.
- ✧ There are two distinct sizes to estimate:
  - Functional size – how much software functionality will be delivered to the end user – the size end users care about.
  - Technical size – how much software logic is needed to create the needed functionality – the size developers focus on.

The most common sizing methods

- Order of magnitude (T-shirt) sizing
- Functional size (normalized to *Function Points*, then to *Implementation Units*)
  - Functional and Business Requirements
  - User Stories
  - Use Cases
  - Function Points (ISO Standards: IFPUG MARK-II, COSMIC, FISMA, NESMA)
- Technical size (normalized to *Implementation Units*)
  - Objects and Business Process Configurations
  - Technical Components
  - Source Code Files
  - SLOC Counts

Note progression across development phases.

How we decide what to measure

- ✧ Decide who is the customer for the metrics
- ✧ What are their (business/software) goals?
- ✧ What data/metrics can, when collected, show whether or not those goals have been met?
- ✧ As projects progress, the goals of some customers change, so…
- ✧ Other data/metrics may be needed.

Example

- The metrics customer is the product manager
- One of her/his goals is releasing, in a timely manner, a quality project
- Possible metrics needed early in development:
  - # of baselined requirements
  - # of modules projected, unit tested, integrated
- Possible metrics late in development:
  - Rate of defect discovery per system version
  - # of defects in most recent system version (defect density)
  - Rate of defect correction (speed with which defects are fixed)
  - Burn down rate (often used in Agile to measure progress and estimate when development will complete)

Some measurement caveats

- Software engineering data is often *dirty*
  - E.g., # of hours worked (did you count the time you told your colleague about your weekend?), or measuring velocity in Agile (based on number of features, story points, requirements or backlog items implemented per time-box)
  - Focus on (approximate) accuracy rather than precision
- Measuring can have adverse impact
  - What management attends to usually improves at the expense of what management neglects to view
- Measurement programs are overhead
  - Cost always must be weighed against benefit

Classroom Activity – within your EMSS project teams

Activity:

Projects run on measurements. Discuss and agree on the measurements you will use during (and perhaps after) your EMSS development. Using the web, investigate the availability of tools, open source and proprietary, that can assist in taking and analyzing the measurements you need.

Submission to complete the Canvas Assignment EMSS Week 8:

Submit your measurement plan, including the tools you have identified.