Hyperion Development

Task: Quality Assurance

www.hyperiondev.com

Introduction

Welcome to the Quality Assurance Task!

In this task, we show you how to ensure that the software you develop is high-quality.

You don't have to take our courses alone! This course has been designed to be taken with an online mentor that marks your submitted code by hand and supports you to achieve your career goals on a daily basis.

To access this mentor support simply navigate to www.hyperiondev.com/support.

Introduction to Quality Management

Ever since the 1960s, when the first large-scale software systems were developed, problems with software quality were discovered. These problems continued to plague software engineering throughout the 20th century. The software that was delivered during this time was slow, unreliable, difficult to maintain and hard to reuse. This undesirable situation ultimately led to the development of formal techniques of software quality management. These software quality management techniques were developed from the methods used in the manufacturing industry and have lead to significant improvements in the level of software quality.

For a software system, software quality management has three principal concerns:

1. Quality management is concerned with establishing a framework of organisational processes and standards that will lead to high-quality software at the organisational level. The software development processes to be used and standards that should apply to the software and related documentation, including the system requirements, design, and code, should be defined by the quality management team.

2. Quality management is concerned with the application of specific quality processes, checking that these processes have been followed, and ensuring that the outputs of the project conform to the standards applicable to it at the project level.

3. At the project level, quality management is also concerned with establishing a quality plan for a project. This plan should set out the quality goals for the project and define what processes and standards are to be used.

In the manufacturing industry, the term 'quality assurance' is widely used. Quality assurance (QA) is the is the definition of processes and standards that should hopefully lead to high-quality products. Different companies in the software industry however, define quality assurance differently. Quality assurance can simply mean the definition of procedures, processes, and standards that are aimed at leading to high-quality software, or, it can also include all configuration management, verification, and validation activities that are applied after a product has been handed over by a development team. In this task, we shall include verification, validation and the processes of checking that quality procedures have been properly applied in the definition of quality assurance.

In most companies, the Quality Assurance team is responsible for managing the release testing process. In other words, they manage the testing of the software before it is

released to customers. This team is responsible for checking that there are system tests for all the requirements and that proper records are maintained of the testing process.

Quality management provides an independent check on the software development process. Project deliverables are checked to ensure that they are consistent with organisational standards and goals during the quality management process. The QA team and the development team should be independent of one another. This allows them to take an objective view of the software and to report on software quality without being influenced by software development issues.

The process of developing a quality plan for a project is known as quality planning. In the quality plan, the desired software qualities should be set. The quality plan should also describe how these software qualities are to be assessed. For a particular system, the quality plan therefore defines what is considered "high-quality" software. This plan ensures that software engineers do not make different or conflicting assumptions which product attributes reflect the most important quality characteristics.

In his book on software management, Humphrey (1989) outlined the structure of a quality plan as follows:

- Product introduction: The product, its intended market, and the quality expectations for the product are defined.
- Product plans: The release dates and responsibilities for the product, as well as plans for distribution and product servicing are defined.
- Process descriptions: The development and service processes and standards that should be used for product development and management are described.
- Quality goals: The quality goals and plans for the product are defined. This includes identification and justification of critical product quality attributes.
- Risks and risk management: The key risks that might affect product quality and the actions to be taken to address these risks are defined.

You should try to keep quality plans as short as possible as it is less likely that people would read a document that is too long.

Software Quality

Unlike products in the manufacturing industry, the assessment of software quality is a subjective process. The quality management team has to use their judgment to decide if an acceptable level of quality has been achieved or not. This is because:

- It is difficult to write a complete and unambiguous software specification. Requirements may be interpreted differently by software developers and customers. It can therefore be almost impossible to determine whether or not software conforms to its specification.

- Requirements from many different classes of stakeholders are integrated into specifications. These requirements are most often a compromise and may not include the requirements of all stakeholder groups. The stakeholders that are excluded might therefore see the system as being of poor quality even though all the agreed upon requirements have been implemented.

- Certain quality characteristics cannot be measured directly. For example, maintainability cannot be measured and therefore cannot be specified in an unambiguous way.

In order to determine whether or not software is of a high-quality, the quality management team has to answer questions about the system's characteristics. Examples of these questions include:

- In the development process, of the software, have programming and documentation standards been followed?

- Has the software been tested properly?

- Is the software dependable enough to be put into use?

- Is the performance of the software acceptable when used normally?

- Is the usability of the software acceptable?

- Is the software understandable and structured well?

It is generally assumed that the system will be tested against its requirements. These tests determine whether or not the system delivers the required functionality. The QA team should therefore review the tests that have been developed and ensure that they have been properly carried out by examining the test records.

The subjectiveness of a software system is based largely on its non-functional characteristics. Users will often find other ways to do what they want to do if the software's functionality is not what is expected. If the software is unreliable or too slow, however, it is almost impossible for users to achieve their goals. Software quality is therefore not just about whether the software functionality has been correctly implemented, but also depends on non-functional system attributes. Some important software quality attributes are:

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability

Obviously, it is not possible any system to be optimized for all of these attributes. By improving robustness, for example, you might lose some performance. The most important quality attributes for the software that is being developed should therefore be defined in the quality plan. The quality plan should also define the quality assessment process, which is an agreed way of assessing whether some quality, such as maintainability or robustness, for example, is present in the product.

Software Standards

The definition or selection of standards that should apply to the software development process or software product is an important part of quality assurance. Tools and methods to support the use of these standards may also be chosen as part of this QA process. Project-specific processes then have to be defined to monitor the use of the standards and check that they have been followed once the standards have been selected for use.

Software standards are important because:

1. They are based on knowledge, which is acquired after a great deal of trial and error, about the best or most appropriate practice for the company. By building this knowledge into a standard, the company is able to reuse experience and avoid mistakes that have been made previously.

2. They provide a framework for defining what 'quality' means in a particular setting. By using standards you establish a basis for deciding if a required level of quality has been achieved.

3. They ensure continuity when a task might have been started by one person has to be taken up and continued by someone else. They ensure that within an organisation, all software engineers adopt the same practices and it is therefore little learning is required when starting new work.

There are two types of software engineering standard that are related to one another:

1. Product standards: These standards apply to the software product being developed. This includes:
    a. document standards (E.g. the structure of requirements documents)
    b. documentation standards (E.g. a standard comment header for a class definition)
    c. coding standards, which define how a programming language should be used.

2. Process standards: These standards define the processes that should be followed during software development. These standards may include definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes.

Standards have to deliver value. This value is in the form of increased product quality. Standards that are expensive to apply in terms of time and effort and that only lead to small improvements in quality are a waste of time. You should ensure that product standards are designed to be applied and checked in a cost-effective way, and process standards include the definition of processes that check that product standards have been followed.

Developing international standards is normally a long, drawn-out process where those who are interested in the standard meet, develop drafts that are commented on and finally agree on the standard. The production of standards is supported by national and international bodies such as the U.S. DoD, ANSI, BSI, NATO, and the IEEE. These standards are general and can be applied across a range of many different projects.

National and international standards have been developed for software engineering terminology, programming languages, notations, procedures for acquiring and writing software requirements, quality assurance procedures, and software verification and validation processes (IEEE, 2003). For safety and security-critical systems, more specialized standards, such as IEC 61508 (IEC, 1998), have also been developed.

When developing standards for a company or organisation, quality management teams should base their standards on national or international standards. The quality assurance team should draw up a standards "handbook" using international standards as a starting point. This handbook should define the standards that are needed by their organisation.

Different development processes are needed for different software, so standards need to be adaptable. According to individual circumstances, each project manager should have the authority to modify process standards. It is important to ensure, however, that when changes are made, they do not lead to a loss of product quality. This can affect a company's relationship with its customers and may lead to increased project costs.

The problems of inappropriate standards can be avoided by careful quality planning early in the project. The project manager and the quality manager should decide which standards should be used without change, which should be modified, and which should be ignored. New standards may also have to be created in response to customer or project requirements.

The ISO 9001 Standards Framework

ISO 9000 is an international set of standards that can be used in the development of quality management systems in all industries. The most general of these standards is called ISO 9001. ISO 9001 applies to organisations that design, develop, and maintain products, including software. The ISO 9001 standard is a framework for developing software standards and not a standard itself. It sets out general quality principles, describes quality processes in general, and lays out the organisational standards and procedures that should be defined. These should be documented in a company's quality manual.

The ISO 9001 standard is oriented around nine core processes. These core processes are shown in the table below:

Product Delivery Processes	Supporting Processes
Business Acquisition	Business Management
Design and Development	Supplier Management
Test	Inventory Management
Production and Delivery	Configuration Management
Service and Support	

A company must document how its processes relate to these core processes if it wishes to be ISO 9001 conformant. It must also define and maintain records that show that the organisational processes that have been defined have been followed. The ISO 9001 standard does not define or prescribe the specific quality processes that an organisation should use.

Some customers demand that software development companies are ISO 9001 certified. An ISO 9001 certification assures customers that the company has an approved quality management system in place. A company's quality management processes and process documentation are examined by independent accreditation authorities who then decide if they are ISO 9001 compliant. If they are, the company is then ISO 9001 certified.

People sometimes think that an ISO 9001 certification means that the quality of the software produced by certified companies will be better than software produced by uncertified companies. However, this is not always the case. It is not guaranteed that ISO 9001 certified companies use the best software development practices or that their processes lead to high-quality software. The ISO 9001 standard simply ensures that the organisation has quality management procedures in place and these procedures are followed.

Companies that use agile development methods are generally not very concerned with ISO 9001 certification. This is because agile methods focus mostly on the code being developed and less on documentation. They therefore have little in common with the formal quality processes that are discussed in ISO 9001.

Reviews and Inspections

Reviews and inspections are activities that check the quality of the deliverables of a project. This involves examining the software, its documentation and process records to find errors and omissions. It also involves checking if quality standards have been

followed. Reviews and inspections are done alongside program testing as part of the software verification and validation process.

With reviews, the software and its associated documentation are examined by a group of people. The group of people look for potential problems in the software and non-conformance with standards. They then make informed judgments about the level of quality of a system or project deliverable. These assessments can then be used by project managers to make planning decisions and allocate resources to the development process.

Quality reviews are based on documents that are produced during the software development process. You may review software specifications, designs, code, process models, test plans, configuration management procedures, process standards, and user manuals. During the review, consistency and completeness of the documents or code should be checked and you should make sure quality standards have been followed.

As part of the quality management process, the conclusions of the review should be formally recorded. If the reviewer discovers any problems, their findings should be passed to the whoever is responsible for correcting errors or omissions in the software.

Reviews and Inspections are not intended to assess how people in the development team perform. Rather, its purpose is to improve software quality. Unlike component testing which is a more private process, reviewing is a public process to discover errors. Mistakes that an individual makes are therefore revealed to the whole development team. Project managers have to be sensitive to individual concerns to ensure that all developers engage in the review process constructively. A working culture that provides support without blame when errors are discovered should be developed.

The Review Process

The review process normally has three phases:

1. Pre-review activities: These essential initial activities are concerned with review planning and review preparation. Review planning generally involves setting up a review team, arranging a time and place for the review, and distributing the documents to be reviewed. Review preparation involves the review team meeting up to get an overview of the software to be reviewed. Each review team member reads and gains an understanding of the software, documents and relevant standards. They then work to find errors, omissions, and departures from standards individually.

2. The review meeting: The creator of the document or program being reviewed should give a run-through of the document or program to the review team during the review meeting. The review should only be two hours at most. One member of the review team should chair the review and another member should record all review decisions and actions to be taken formally. The chair is responsible for ensuring that all written comments which are provided by the reviewers are considered. A record of comments and actions agreed during the review should be signed by the chair.

3. Post-review activities: The issues and problems raised during the review must be addressed once the review meeting has finished. This can involve fixing software bugs, refactoring software so that it conforms to quality standards, or rewriting documents. A management review might also be necessary to decide if more resources should be made available to certain problems. The review chair also checks that the review comments have all been taken into account after the changes have been made. A further review will sometimes also be required to check that the changes made cover all of the previous review comments.

Review teams normally have three or four principal reviewers. One member of the team should be a senior designer who should be responsible for making significant technical decisions. Other project members, such as the designers of related subsystems might be invited to contribute to the review by the principal reviewers. These project members should concentrate on reviewing sections of the document that affect their work rather than the whole document. The review team might also decide to circulate the document and ask for written comments from a wide range of project members. The project manager does not need to be involved in the review unless there are problems that require changes to the project plan.

The review process relies on all members of a development team being in the same location and available for a team meeting. This is not always the case however. Project teams can be distributed across countries or even continents so it is not always possible for the entire team to meet in the same room. Document editing tools may be used to support the review process in such cases. These tools allow team members to annotate the document or software source code with comments no matter where they are. These comments can then be approved or rejected by other team members.

In agile software development, the review process is often informal. For example, in extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member. Extreme programming mostly relies on individuals taking the initiative to improve and refactor code. Standards compliance issues are not normally considered as Agile approaches are not usually standards-driven.

Since agile methods lack formal quality procedures, there can be problems in using agile approaches in companies that have developed detailed quality management procedures. Quality reviews are used within a plan-driven development process since they can often slow down the pace of software development. Reviews can be planned and other work scheduled in parallel with them in a plan-driven process. In agile approaches this is impractical since these methods focus solely on code development.

Program Inspections

With program inspections, team members collaborate to find bugs in the program that is being developed. Program inspections do not require the program to be executed so they complement testing.

Program inspections involve a team of people who carefully review the program source code line by line. The team members should come from many different backgrounds. They try to discover defects and problems and then describe these at an inspection meeting. These defects may be logical errors, anomalies in the code that might indicate an erroneous condition or features that have been omitted from the code. A checklist of common programming errors is often used to focus the search for bugs during an inspection. This checklist may be based on examples from books or from knowledge of common defects. Different checklists are used for different programming languages because each language has its own characteristic errors.

Many software development companies are reluctant to use inspections even though they are cost-effective. Some software engineers with experience of program testing are unwilling to accept that inspections can be more effective for defect detection than testing. Inspections require additional costs during design and development so managers may be suspicious.

Agile processes don't normally use formal inspection processes. They rather rely on team members cooperating to check each other's code. In extreme programming, pair programming is an effective substitute for inspection as it is a continual inspection process. With pair programming, two people look at every line of code and check it before it is accepted.

Software Measurement and Metrics

Software measurement is concerned with determining a numeric value for an attribute of a software component, system, or process. You might be able to draw conclusions about

the quality of software by comparing these values to each other and to the organisational standards.

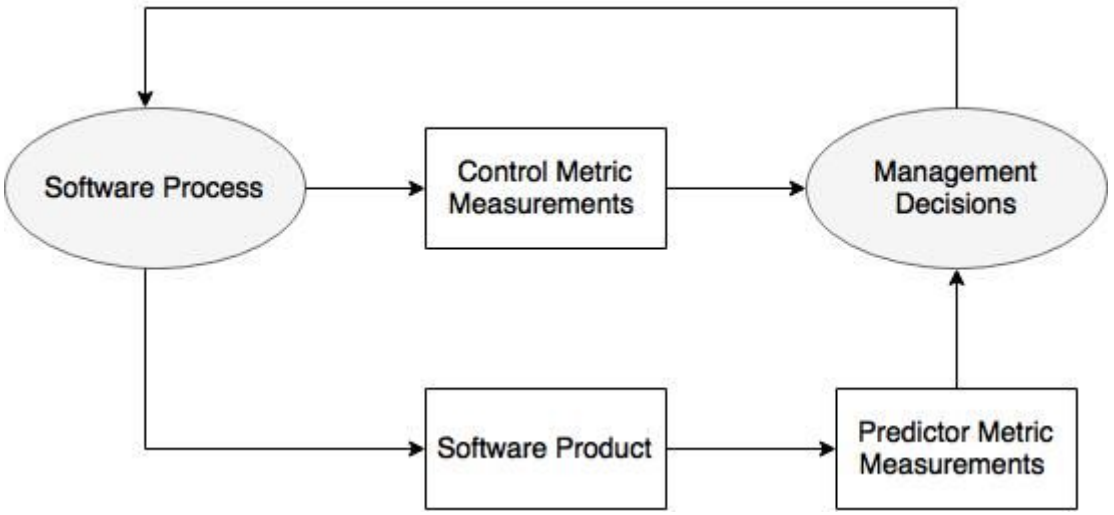
Let's say for example that an organisation would like to introduce a new software-testing tool. As a baseline for assessing the effectiveness of the tool, you can record the number of software defects discovered at a given time. You then repeat this process after using the tool for some time. If more defects are found in the same amount of time, after the tool has been introduced, you may decide that it provides useful support for the software validation process.

Using measurement in place of reviews to make judgments about software quality is the long-term goal of software measurement. Ideally, a system could be assessed using a range of metrics and, a value for the quality of the system could be determined from these measurements. The software could then be approved without review if it reached a required level of quality. The measurement tools could also highlight areas of the software that need to be improved. The ideal situation, however, has yet to become a reality.

A characteristic of a software system, system documentation, or development process that can be measured is known as a software metric. The number of lines of code in a software system, the number of reported faults in a software product that has been delivered, and the number of days required to develop a component of a system are all examples of metrics.

There are two types of metrics; control metrics or predictor metrics. Control metrics support process management, and predictor metrics help to predict characteristics of the software. Control metrics are related to software processes, while predictor metrics are associated with the software product itself. Examples of control metrics are; the average effort and the time required to repair defects that have been reported. Examples of predictor metrics are the cyclomatic complexity of a program, the average length of identifiers in a program and the number of attributes and methods associated with classes in a design.

As shown in the diagram below, control and predictor metrics may both influence management decisions.



Predictor and Control Metrics

Managers use process measurements to decide if process changes should be made. They use predictor metrics to help estimate the effort required to make software changes.

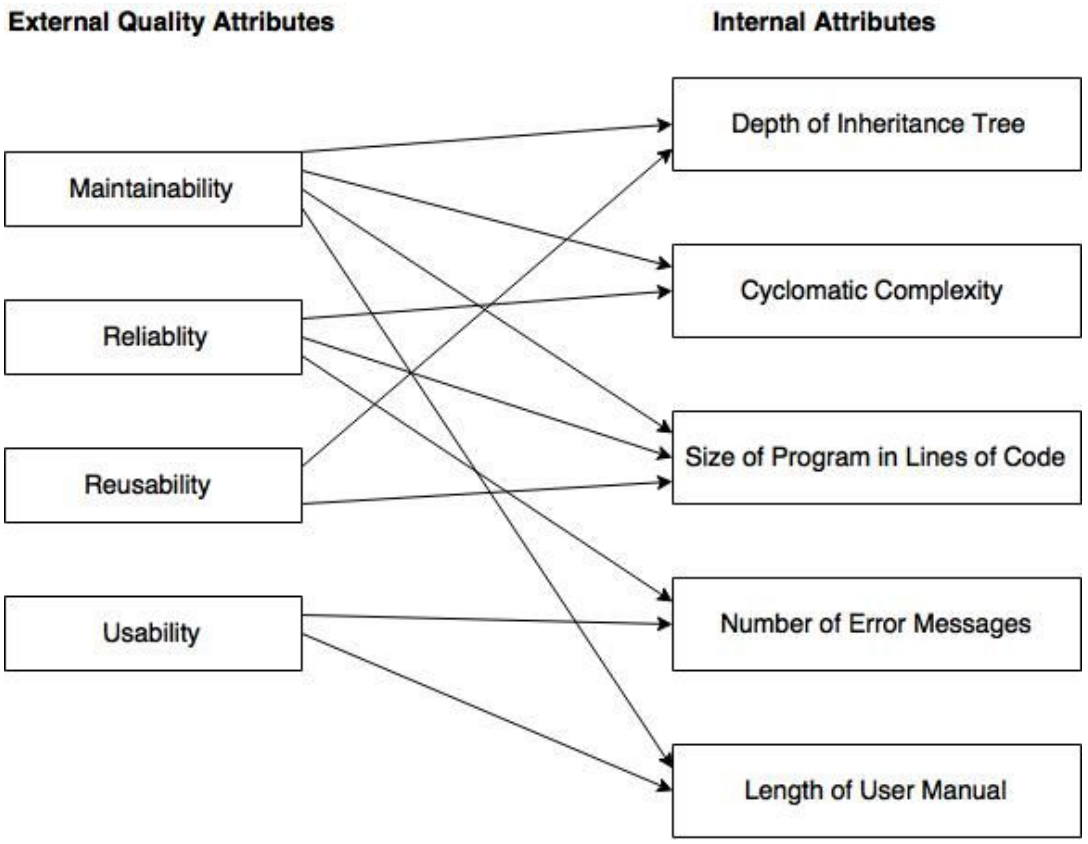
Measurements of a software system can be used in two ways:

1. Assigning a value to system quality attributes: you can assess system quality attributes (E.g. maintainability) by measuring the characteristics of system components and combining these measurements.

2. Identifying the system components whose quality is not up to standard: individual components with characteristics that deviate from normal can be identified by measurements. You can measure components to discover the one with the highest complexity for example. Those components with the highest complexity are the most likely to contain bugs because the complexity makes them harder to understand.

It is difficult to make direct measurements of many of the software quality attributes that we have previously listed. Quality attributes such as maintainability, understandability, and usability cannot be measured objectively as they are external attributes that relate to how developers and users experience the software and are affected by subjective factors, such as user experience and education. You have to measure some internal attributes such as the size and complexity, of the software and assume that these are related to the quality characteristics that you are concerned with in order to make a judgment about these attributes.

The diagram below shows a few external software quality attributes and internal attributes that could be related to them:



The relationship between external and internal attributes

Three conditions must hold if you wish to use the measure of the internal attribute as a predictor of the external software characteristic:

1. The internal attribute must be measured accurately.

2. The value of the quality attribute must be related to the value of the attribute that can be measured.

3. This relationship between the internal and external attributes must be understood, validated, and expressed in terms of a formula or model.

Software tools that analyze the source code of the software are used to measure internal software attributes.

Product Metrics

Predictor metrics that are used to measure internal attributes of a software system are known as product metrics. Some examples of product metrics are the system size, measured in lines of code, or the number of methods associated with each class. Software characteristics that can be easily measured, however, do not have a clear and consistent relationship with quality attributes such as understandability and maintainability. Depending on the development processes, technologies used and the type of system that is being developed the relationship may vary.

There are two classes of product metrics:

1. Dynamic metrics: These are collected by measurements made of a program in execution. They can be collected during system testing or after the system has gone into use. Examples of dynamic metrics are the number of bug reports and the time taken to complete a computation.

2. Static metrics: These collected by measurements made of representations of the system. Representations of the system can be the design, program, or documentation. Examples of static metrics are the code size and the average length of identifiers used.

Efficiency and reliability of a program can be assessed using dynamic metrics, while complexity, understandability, and maintainability of a software system can be assessed using static methods.

The relationship between dynamic metrics and software quality characteristics is usually clear. It is easy to measure the execution time required for functions for example. It is also easy to determine the time required to start up a system. These dynamic metrics relate directly to the system's efficiency.

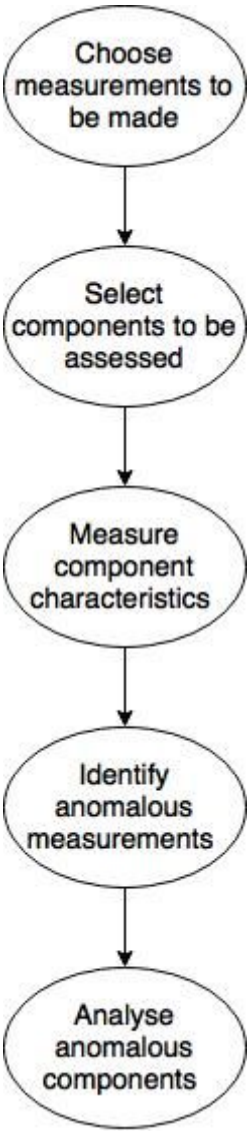
Static metrics, on the other hand, have an indirect relationship with quality attributes. Program size and control complexity seem to be the most reliable predictors of understandability, system complexity, and maintainability. The table below shows some static metrics:

Software metric	Description
Fan-in/Fan-out	Fan-in is the number of methods that call another method (say Y). Fan-out is the number of methods that are called by method Y. A high fan-in value means that Y is tightly related to the rest of the

	design and changes to Y will have a domino effect. A high fan-out value suggests that the complexity of Y may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	A measure of the size of the program. The larger the size of the program, the more complex and error-prone it is likely to be.
Cyclomatic complexity	A measure of the control complexity of a program. This may be related to program understandability.
Length of identifiers	A measure of the average length of identifiers in a program. This includes variables names, class names, method names, etc.). The longer the identifiers, the more likely they are to be meaningful. This relates to the understandability of the program.
Depth of conditional nesting	A measure of the depth of nesting of if-statements in a program. If statements that are nested too deeply are hard to understand and can be error-prone.
Fog index	A measure of the average length of words and sentences in documents. The higher the value of the Fog index the more difficult a document is to understand.

Software Component Analysis

The process of product measurement may be part of a software quality assessment process is shown below:



The process of product measurement

Each component in a system can be analysed separately using a range of metrics. The values of these metrics may then be compared for different components and even with historical measurement data collected on previous projects. Measurements, which deviate significantly from normal, can imply that there are problems with the quality of these components.

In the component measurement process the key stages are:

1. Choose measurements to be made: formulate the questions that the measurement is intended to answer and define the measurements required to

answer these questions.

2. Select components to be assessed: metric values for all of the components in a software system might not need to be assessed. You can sometimes select a representative selection of components for measurement. This selection can allow you to make an overall assessment of system quality. You might alternatively wish to focus on the core components of the system that are in constant use.

3. Measure component characteristics: The components that are selected are measured and the associated metric values computed. This involves processing the component representation using an automated data collection tool.

4. Identify anomalous measurements: The component measurements are then compared with each other and to previous measurements that have been recorded in a measurement database. Abnormally high or low values for each metric should be noted. These Abnormal values suggest that there could be problems with the component exhibiting these values.

5. Analyze anomalous components: The components that have abnormal values for your chosen metrics should then be examined to decide whether or not these abnormally high or low metric values mean that the quality of the component is compromised. There may be some other reason for the abnormal value, however, so this may not mean that there are component quality problems.

Collected data should always be maintained and records should be kept of all projects even when data has not been used during a particular project. You can then make comparisons of software quality across projects and validate the relations between internal component attributes and quality characteristics, once a sufficiently large measurement database has been established.

Measurement Ambiguity

Once quantitative data about software and software processes have been collected, it must be analysed in order to understand its meaning. Data can easily be interpreted in many different ways and it is easy to make inferences that are incorrect. You must consider the context where the data is collected and not simply look at the data on its own.

Compulsory Task

Answer the following:

- Explain why a high-quality software process should lead to high-quality software products. Discuss possible problems with this system of quality management.

- Briefly describe possible standards that might be used for:
    - The use of control constructs in Java
    - Reports which might be submitted for a term project in a university
    - The process of purchasing and installing a new computer

- Explain why program inspections are an effective technique for discovering errors in a program. What types of error are unlikely to be discovered through inspections?

- Explain why design metrics are, by themselves, an inadequate method of predicting design quality.

- Why it is difficult to validate the relationships between internal product attributes, such as cyclomatic complexity and external attributes, such as maintainability.

SHARE YOUR THOUGHTS WITH US

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes. Think the content of this task, or this course as a whole, can be improved or think we've done a good job? Click here to share your thoughts anonymously.