

SCM

Software Configuration Management

- **Definition:**
 - the art of identifying, organizing, and controlling modifications to the software being built by a programming team.
- **Goal**
 - maximize productivity by minimizing mistakes (Babich, 1986).
- **It is an umbrella activity that is applied throughout the software process.**
- **Because change can occur at any time, SCM activities are developed to:**
 - identify change
 - control change
 - ensure that change is being properly implemented
 - report changes to others who may have an interest.

Software configuration management vs software support

- software support and software configuration management are different
- **Software support**
 - a set of software engineering activities that occur after software has been delivered to the customer and put into operation.
- **Software configuration management**
 - a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

Source of change

- There are four fundamental sources of change:
 - New business or market conditions dictate changes in product requirements or business rules.
 - New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
 - Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
 - Budgetary or scheduling constraints cause a redefinition of the system or product.

SCM's important concepts

- Software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software.
- SCM can be viewed as a software quality assurance activity that is applied throughout the software process.
- There are important concepts related to change management in SCM:
 - Software configuration items
 - Baselines

Manajemen Konfigurasi Perangkat Lunak

- **Definisi:**
 - seni mengidentifikasi, mengatur, dan mengendalikan modifikasi perangkat lunak yang sedang dibangun oleh tim pemrograman.
- **Sasaran**
 - memaksimalkan produktivitas dengan meminimalkan kesalahan (Babich, 1986).
- **Ini adalah aktivitas payung yang diterapkan di seluruh proses perangkat lunak.**
- **Karena perubahan dapat terjadi kapan saja, kegiatan SCM dikembangkan untuk:**
 - mengidentifikasi perubahan
 - perubahan kontrol
 - memastikan bahwa perubahan diterapkan dengan benar
 - melaporkan perubahan kepada orang lain yang mungkin berkepentingan.

Manajemen konfigurasi perangkat lunak

vs dukungan perangkat lunak

- dukungan perangkat lunak dan manajemen konfigurasi perangkat lunak berbeda
- **Dukungan perangkat lunak**
 - sekumpulan aktivitas rekayasa perangkat lunak yang terjadi setelah perangkat lunak dikirimkan ke pelanggan dan dioperasikan.
- **Manajemen konfigurasi perangkat lunak**
 - serangkaian aktivitas pelacakan dan pengendalian yang dimulai ketika proyek rekayasa perangkat lunak dimulai dan berakhir hanya ketika perangkat lunak tidak beroperasi.

Sumber perubahan

- Ada empat sumber perubahan mendasar:
 - Kondisi bisnis atau pasar baru mendikte perubahan dalam persyaratan produk atau aturan bisnis.
 - Kebutuhan pelanggan baru menuntut modifikasi data yang dihasilkan oleh sistem informasi, fungsionalitas yang diberikan oleh produk, atau layanan yang diberikan oleh sistem berbasis komputer.
 - Reorganisasi atau pertumbuhan/perampungan bisnis menyebabkan perubahan prioritas proyek atau struktur tim rekayasa perangkat lunak.
 - Keterbatasan anggaran atau penjadwalan menyebabkan pendefinisian ulang sistem atau produk.

Konsep penting SCM

- Manajemen konfigurasi perangkat lunak adalah serangkaian aktivitas yang telah dikembangkan untuk mengelola perubahan sepanjang siklus hidup perangkat lunak komputer.
- SCM dapat dilihat sebagai aktivitas jaminan kualitas perangkat lunak yang diterapkan di seluruh proses perangkat lunak.
- Ada konsep penting terkait manajemen perubahan di SCM:
 - Item konfigurasi perangkat lunak
 - Garis Dasar

Software configuration items

- The output of the software process can be divided into three broad categories:
 - computer programs (both source level and executable forms)
 - documents that describe the computer programs (targeted at both technical practitioners and users)
 - data (contained within the program or external to it).
- The items that comprise all information produced as part of the software process are collectively called a *software configuration*.
- As the software process progresses, the number of *software configuration items* (SCIs) grows rapidly.

Software configuration items (1)

- We have already defined a software configuration item as information that is created as part of the software engineering process.
- In the extreme, a SCI could be considered to be a single section of a large specification or one test case in a large suite of tests.
- More realistically, an SCI can be: a document, or a entire suite of test cases, or a named program component (e.g., a C++ function or an Java package).

Software configuration items (2)

- SCIs are organized to form *configuration objects* that may be cataloged in the project database with a single name. A configuration object has a name, attributes, and is "connected" to other objects by relationships.
- The relationship can be compositional (single-headed arrow) or interrelationship (double-headed arrows).
- If a change were made to the source code object, the interrelationships enable a software engineer to determine what other objects (and SCIs) might be affected.

Baselines (1)

- A *baseline* is a software configuration management concept that helps us to control change without seriously impeding justifiable change.
- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

Item konfigurasi perangkat lunak

- Keluaran dari proses perangkat lunak dapat dibagi menjadi tiga kategori besar:
 - program komputer (tingkat sumber dan bentuk yang dapat dieksekusi)
 - dokumen yang menjelaskan program komputer (ditargetkan pada praktisi teknis dan pengguna)
 - data (terdapat di dalam program atau di luarnya).
- Item yang terdiri dari semua informasi yang dihasilkan sebagai bagian dari proses perangkat lunak secara kolektif disebut a *konfigurasi perangkat lunak*.
- Seiring berjalannya proses perangkat lunak, jumlah *item konfigurasi perangkat lunak*(SCI) tumbuh dengan cepat.

Item konfigurasi perangkat lunak (1)

- Kami telah mendefinisikan item konfigurasi perangkat lunak sebagai informasi yang dibuat sebagai bagian dari proses rekayasa perangkat lunak.
- Secara ekstrim, SCI dapat dianggap sebagai satu bagian dari spesifikasi besar atau satu kasus uji dalam rangkaian pengujian yang besar.
- Secara lebih realistik, SCI dapat berupa: dokumen, atau seluruh rangkaian kasus uji, atau komponen program bernama (misalnya, fungsi C++ atau paket Java).

Item konfigurasi perangkat lunak (2)

- SCI diatur untuk membentuk *objek konfigurasi* yang dapat dikatalogkan dalam database proyek dengan satu nama. Objek konfigurasi memiliki nama, atribut, dan "terhubung" ke objek lain melalui hubungan.
- Hubungan tersebut dapat berupa komposisi (panah berkepala tunggal) atau hubungan timbal balik (panah berkepala dua).
- Jika perubahan dibuat pada objek kode sumber, hubungan timbal balik memungkinkan insinyur perangkat lunak untuk menentukan objek lain (dan SCI) apa yang mungkin terpengaruh.

Dasar (1)

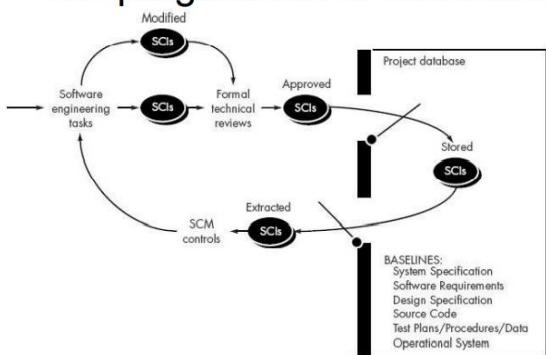
- SEBAHAGIAR dasar adalah konsep manajemen konfigurasi perangkat lunak yang membantu kita mengontrol perubahan tanpa secara serius menghambat perubahan yang dapat dibenarkan.
- IEEE (IEEE Std. No. 610.12-1990) mendefinisikan baseline sebagai:

Spesifikasi atau produk yang telah ditinjau dan disepakati secara formal, yang selanjutnya berfungsi sebagai dasar untuk pengembangan lebih lanjut, dan yang dapat diubah hanya melalui prosedur pengendalian perubahan formal.

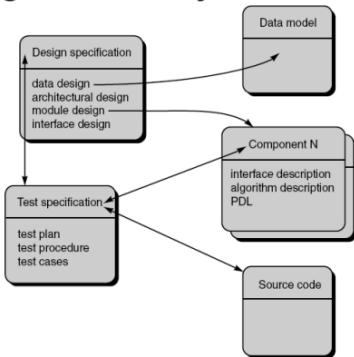
Baselines (2)

- Before a software configuration item becomes a baseline, change may be made quickly and informally.
- However, once a baseline is established, changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.
- In the context of software engineering, a baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review.

The progression of baselines



Configuration objects: example



The SCM process

- Followings are questions to discuss about SCM:
 - How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
 - How does an organization control changes before and after software is released to a customer?
 - Who has responsibility for approving and ranking changes?
 - How can we ensure that changes have been made properly?
 - What mechanism is used to appraise others of changes that are made?
- These questions lead us to the definition of five SCM tasks: *identification, version control, change control, configuration auditing, and reporting*.

Dasar (2)

- Sebelum item konfigurasi perangkat lunak menjadi baseline, perubahan dapat dilakukan dengan cepat dan informal.
- Namun, begitu baseline ditetapkan, perubahan dapat dilakukan, tetapi prosedur formal yang spesifik harus diterapkan untuk mengevaluasi dan memverifikasi setiap perubahan.
- Dalam konteks rekayasa perangkat lunak, baseline adalah tonggak dalam pengembangan perangkat lunak yang ditandai dengan pengiriman satu atau lebih item konfigurasi perangkat lunak dan persetujuan SCI ini diperoleh melalui tinjauan teknis formal.

Proses SCM

- Berikut ini adalah pertanyaan-pertanyaan untuk didiskusikan tentang SCM:
 - Bagaimana organisasi mengidentifikasi dan mengelola banyak versi program yang ada (dan dokumentasinya) dengan cara yang memungkinkan perubahan diakomodasi secara efisien?
 - Bagaimana organisasi mengontrol perubahan sebelum dan setelah perangkat lunak dirilis ke pelanggan?
 - Siapa yang bertanggung jawab untuk menyetujui dan memberi peringkat perubahan?
 - Bagaimana kita dapat memastikan bahwa perubahan telah dilakukan dengan benar?
 - Mekanisme apa yang digunakan untuk menilai orang lain atas perubahan yang dibuat?
- Pertanyaan-pertanyaan ini membawa kita pada definisi lima tugas SCM: *identifikasi, kontrol versi, kontrol perubahan, audit konfigurasi, dan pelaporan*.

(1) Identification of objects in the software configuration

- To control and manage software configuration items, each must be separately named and then organized using an object-oriented approach.
- The following terms are important in identification of object:
 - Basic objects and aggregate objects
 - Features
 - Relationships
 - Object evolution

(1) Identifikasi objek dalam konfigurasi perangkat lunak

- Untuk mengontrol dan mengelola item konfigurasi perangkat lunak, masing-masing harus diberi nama secara terpisah dan kemudian diatur menggunakan pendekatan berorientasi objek.
- Istilah-istilah berikut ini penting dalam identifikasi objek:
 - Objek dasar dan objek agregat
 - Fitur
 - Hubungan
 - Evolusi objek

Basic objects and aggregate objects

- Two types of objects can be identified (Choi & Scacchi, 1989): **basic objects** and **aggregate objects**.
- A basic object is a "unit of text" that has been created by a software engineer during analysis, design, code, or test. E.g., a basic object might be a section of a requirements specification, a source listing for a component, or a suite of test cases that are used to exercise the code.
- An aggregate object is a collection of basic objects and other aggregate objects. E.g. **Design Specification**. Conceptually, it can be viewed as a named (identified) list of pointers that specify basic objects such as **data model** and **component N**.

Features

- Each object has a set of distinct features that identify it uniquely: a **name**, a **description**, a **list of resources**, and a **"realization"**.
- **The object name** is a character string that identifies the object unambiguously.
- **The object description** is a list of data items that identify the SCI type (e.g., document, program, data) represented by the object, a project identifier, change and/or version information.
- **Resources** are "entities that are provided, processed, referenced or otherwise required by the object." E.g., data types, specific functions, or even variable names may be considered to be object resources.
- **The realization** is a pointer to the "unit of text" for a basic object and null for an aggregate object.

Relationship (1)

- Configuration object identification must also consider the relationships that exist between named objects. An object can be identified as **<part-of>** or compositional of an aggregate object. The relationship **<part-of>** defines a hierarchy of objects. For example, using the simple notation:
*E-R diagram 1.4 <part-of> data model;
data model <part-of> design specification;*
.....

Relationship (2)

- The relationship can also be interrelationship (cross structural relationship). E.g.
*data model <interrelated> data flow model;
data model <interrelated> test case class m;*
- In the first case, the interrelationship is between a composite object, while the second relationship is between an aggregate object (**data model**) and a basic object (**test case class m**).

Objek dasar dan objek agregat

- Dua jenis objek dapat diidentifikasi (Choi & Scacchi, 1989): **objek dasar** dan **objek agregat**.
- Objek dasar adalah "unit teks" yang telah dibuat oleh insinyur perangkat lunak selama analisis, desain, kode, atau pengujian. Misalnya, objek dasar mungkin merupakan bagian dari spesifikasi persyaratan, daftar sumber untuk komponen, atau rangkaian kasus uji yang digunakan untuk menjalankan kode.
- Objek agregat adalah kumpulan objek dasar dan objek agregat lainnya. Misalnya **Spesifikasi desain**. Secara konseptual, ini dapat dilihat sebagai daftar penunjuk bernama (diidentifikasi) yang menentukan objek dasar seperti: **model data** dan **komponen N**.

Fitur

- Setiap objek memiliki serangkaian fitur berbeda yang mengidentifikasinya secara unik: nama, deskripsi, daftar sumber daya, dan sebuah "realisasi".
- **Nama objek** adalah string karakter yang mengidentifikasi objek dengan jelas.
- **Deskripsi objek** adalah daftar item data yang mengidentifikasi jenis SCI (misalnya, dokumen, program, data) diwakili oleh objek, pengidentifikasi proyek, perubahan dan/atau informasi versi.
- **Sumber daya** adalah "entitas yang disediakan, diproses, direferensikan atau diperlukan oleh objek." Misalnya, tipe data, fungsi tertentu, atau bahkan nama variabel dapat dianggap sebagai sumber daya objek.
- **Realisasi** adalah pointer ke "unit teks" untuk objek dasar dan null untuk objek agregat.

Hubungan (1)

- Identifikasi objek konfigurasi juga harus mempertimbangkan hubungan yang ada antara objek yang diberi nama. Sebuah objek dapat diidentifikasi sebagai **<part-of>** atau komposisi objek agregat. Hubungan **<part-of>** mendefinisikan hierarki objek. Misalnya, menggunakan notasi sederhana:

*Diagram ER 1.4 model data <bagian dari>;
model data <bagian-dari> spesifikasi desain;*

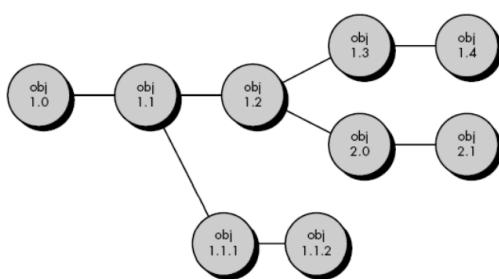
Hubungan (2)

- Hubungan tersebut juga dapat bersifat interrelationship (hubungan lintas struktural). Misalnya
*model data <saling terkait> model aliran data;
model data <interrelated> kelas kasus uji m;*
- Dalam kasus pertama, keterkaitannya adalah antara objek komposit, sedangkan yang kedua hubungan adalah antara objek agregat (**model data**) dan objek dasar (**kelas kasus uji m**).

Object evolution

- The identification scheme for software objects must recognize that objects evolve throughout the software process.
- Before an object is baselined, it may change many times, and even after a baseline has been established, changes may be quite frequent.
- It is possible to create an *evolution graph* (Gustavsson, 1989) for any object. The evolution graph describes the change history of an object

Evolution graph: example



(2) Version Control

- Combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- Each version of the software is a collection of SCIs (source code, documents, data), and each version may be composed of different variants.

(2) Version Control ... Example

- Consider a version of a simple program that is composed of entities 1, 2, 3, 4, and 5.
 - Entity 4 is used only when the software is implemented using color displays.
 - Entity 5 is used only when the software is implemented when monochrome displays are available.
- Therefore, two variants of the version can be defined:
 - (1) entities 1, 2, 3, and 4;
 - (2) entities 1, 2, 3, and 5.

Three Important Elements in Version Control

- Version: defined when major changes are made to one or more objects
- Entity: a collection of objects at the same revision level.
- Variant: a different collection of objects at the same revision level and therefore coexists in parallel with other variants
- The relationship between configuration objects and entities, variants and versions can be represented in a three-dimensional space called **object pool**

Evolusi objek

- Skema identifikasi untuk objek perangkat lunak harus mengenali bahwa objek berevolusi selama proses perangkat lunak.
- Sebelum sebuah objek dibuat garis dasar, itu dapat berubah berkali-kali, dan bahkan setelah garis dasar telah ditetapkan, perubahan mungkin cukup sering.
- Dimungkinkan untuk membuat *grafik evolusi* (Gustavsson, 1989) untuk objek apa pun. Grafik evolusi menggambarkan sejarah perubahan suatu objek

(2) Kontrol Versi

- Menggabungkan prosedur dan alat untuk mengelola berbagai versi objek konfigurasi yang dibuat selama proses perangkat lunak.
- Setiap versi perangkat lunak adalah kumpulan SCI (kode sumber, dokumen, data), dan setiap versi dapat terdiri dari varian yang berbeda.

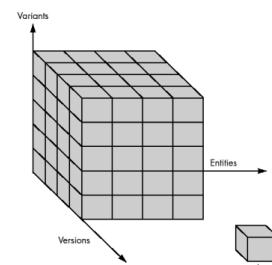
(2) Kontrol Versi ... Contoh

- Pertimbangkan versi program sederhana yang terdiri dari entitas 1, 2, 3, 4, dan 5.
 - Entitas 4 hanya digunakan ketika perangkat lunak diimplementasikan menggunakan tampilan berwarna.
 - Entitas 5 hanya digunakan saat perangkat lunak diimplementasikan saat tampilan monokrom tersedia.
- Oleh karena itu, dua varian versi dapat ditentukan:
 - (1) entitas 1, 2, 3, dan 4;
 - (2) entitas 1, 2, 3, dan 5.

Tiga Elemen Penting dalam Kontrol Versi

- Versi: didefinisikan ketika perubahan besar dilakukan pada satu atau lebih objek
- Entitas: kumpulan objek pada tingkat revisi yang sama.
- Varian: kumpulan objek yang berbeda pada tingkat revisi yang sama dan oleh karena itu hidup berdampingan secara paralel dengan varian lain
- Hubungan antara objek konfigurasi dan entitas, varian dan versi dapat direpresentasikan dalam ruang tiga dimensi **disebut kumpulan objek**

Object Pool



Version Control Approach

- A number of different automated approaches to version control have been proposed over the past decade.
- The primary difference in approaches
 - the sophistication of the attributes that are used to construct specific versions and variants of a system
 - the mechanics of the process for construction.

(3) Change Control

- Facts:
 - Too much change control, we create problems.
 - Too little, and we create other problems.
- For a large software engineering project, uncontrolled change rapidly leads to chaos.
- For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.

Systematic of Change Control Process (1)

- A change request is submitted and evaluated
 - to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.
- The results of the evaluation are presented as a change report, which is used by a change control authority (CCA)—a person or group who makes a final decision on the status and priority of the change.
- An engineering change order (ECO) is generated for each approved change.
- The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit.
- The object to be changed is "checked out" of the project database, the change is made, and appropriate SQA activities are applied.
- The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.

Access and Synchronization Control (1)

- The "check-in" and "check-out" process implements two important elements of change control—access control and synchronization control.
- Access control governs which software engineers have the authority to access and modify a particular configuration object.
- Synchronization control helps to ensure that parallel changes, performed by two different people, don't overwrite one another

Access and Synchronization Control (2)

- Based on an approved change request and ECO, a software engineer checks out a configuration object.
- An access control function ensures that the software engineer has authority to check out the object, and synchronization control locks the object in the project database so that no updates can be made to it until the currently checked out version has been replaced.
- Note that other copies can be checked-out, but other updates cannot be made. A copy of the baselined object, called the extracted version, is modified by the software engineer. After appropriate SQA and testing, the modified version of the object is checked in and the new baseline object is unlocked.

Pendekatan Kontrol Versi

- Sejumlah pendekatan otomatis yang berbeda untuk kontrol versi telah diusulkan selama dekade terakhir.
- Perbedaan utama dalam pendekatan
 - kecanggihan atribut yang digunakan untuk membangun versi dan varian tertentu dari suatu sistem
 - mekanisme proses konstruksi.

(3) Ubah Kontrol

- Fakta:
 - Terlalu banyak kontrol perubahan, kita menciptakan masalah.
 - Terlalu sedikit, dan kita menciptakan masalah lain.
- Untuk proyek rekayasa perangkat lunak yang besar, perubahan yang tidak terkendali dengan cepat menyebabkan kacauan.
- Untuk proyek semacam itu, kontrol perubahan menggabungkan prosedur manusia dan alat otomatis untuk menyediakan mekanisme untuk mengontrol perubahan.

Sistematis Proses Pengendalian Perubahan (1)

- Permintaan perubahan diajukan dan dievaluasi
 - untuk menilai manfaat teknis, potensi efek samping, dampak keseluruhan pada objek konfigurasi lain dan fungsi sistem, dan proyeksi biaya perubahan.
- Hasil evaluasi disajikan sebagai laporan perubahan, yang digunakan oleh change control authority (CCA)—seseorang atau kelompok yang membuat keputusan akhir tentang status dan prioritas perubahan.
- Perintah perubahan rekayasa (ECO) dibuat untuk setiap perubahan yang disetujui.
- ECO menjelaskan perubahan yang akan dilakukan, kendala yang harus dihormati, dan kriteria untuk tinjauan dan audit.
- Objek yang akan diubah "diperiksa" dari database proyek, perubahan dibuat, dan aktivitas SQA yang sesuai diterapkan.
- Objek tersebut kemudian "diperiksa" ke database dan mekanisme kontrol versi yang sesuai digunakan untuk membuat versi perangkat lunak berikutnya.

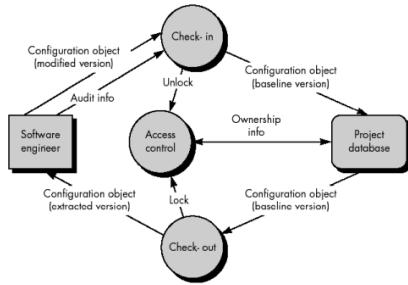
Kontrol Akses dan Sinkronisasi (1)

- Proses "check-in" dan "check-out" menerapkan dua elemen penting dari kontrol perubahan—kontrol akses dan kontrol sinkronisasi.
- Kontrol akses mengatur insinyur perangkat lunak mana yang memiliki wewenang untuk mengakses dan memodifikasi objek konfigurasi tertentu.
- Kontrol sinkronisasi membantu memastikan bahwa perubahan paralel, yang dilakukan oleh dua orang yang berbeda, tidak menimpa satu sama lain

Kontrol Akses dan Sinkronisasi (2)

- Berdasarkan permintaan perubahan dan ECO yang disetujui, insinyur perangkat lunak memeriksa objek konfigurasi.
- Fungsi kontrol akses memastikan bahwa perekayasa perangkat lunak memiliki wewenang untuk memeriksa objek, dan kontrol sinkronisasi mengunci objek dalam database proyek sehingga tidak ada pembaruan yang dapat dilakukan hingga versi yang saat ini diperiksa telah diganti.
- Perhatikan bahwa salinan lain dapat diperiksa, tetapi pembaruan lain tidak dapat dilakukan. Salinan objek baseline, yang disebut versi yang diekstraksi, dimodifikasi oleh insinyur perangkat lunak. Setelah SQA dan pengujian yang sesuai, versi objek yang dimodifikasi diperiksa dan objek dasar yang baru dibuka kuncinya.

Access and Synchronization Control (3)



(4) Configuration Audit

- Identification, version control, and change control help the software developer to maintain order in what would otherwise be a chaotic and fluid situation.
- How can we ensure that the change has been properly implemented?
- The answer is (1) formal technical reviews and (2) the software configuration audit.
- The formal technical review focuses on the technical correctness of the configuration object that has been modified.
- The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects.
- A formal technical review should be conducted for all but the most trivial changes.

Configuration Audit ... cont'd

- A software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review.
- The audit asks and answers the following questions:
 - Has the change specified in the ECO been made? Have any additional modifications been incorporated?
 - Has a formal technical review been conducted to assess technical correctness?
 - Has the software process been followed and have software engineering standards been properly applied?
 - Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
 - Have SCM procedures for noting the change, recording it, and reporting it been followed?
 - Have all related SCIs been properly updated?

(5) Status Reporting

- Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions: (1) What happened? (2) Who did it? (3) When did it happen? (4) What else will be affected?
- The flow of information for configuration status reporting (CSR):
 - Each time an SCI is assigned new or updated identification, a CSR entry is made.
 - Each time a change is approved by the CCA (i.e., an ECO is issued), a CSR entry is made.
 - Each time a configuration audit is conducted, the results are reported as part of the CSR task.
 - Output from CSR may be placed in an on-line database, so that software developers or maintainers can access change information by keyword category.

Status Reporting ... cont'd

- Configuration status reporting plays a vital role in the success of a large software development project.
- When many people are involved, it is likely that "the left hand not knowing what the right hand is doing" syndrome will occur.
- CSR helps to eliminate these problems by improving communication among all people involved.

(4) Audit Konfigurasi

- Identifikasi, kontrol versi, dan kontrol perubahan membantu pengembang perangkat lunak untuk menjaga ketertiban dalam situasi yang kacau dan lancar.
- Bagaimana kita dapat memastikan bahwa perubahan telah diterapkan dengan benar?
- Jawabannya adalah (1) tinjauan teknis formal dan (2) audit konfigurasi perangkat lunak.
- Tinjauan teknis formal berfokus pada kebenaran teknis dari objek konfigurasi yang telah dimodifikasi.
- Peninjau menilai SCI untuk menentukan konsistensi dengan SCI lain, kelalaian, atau potensi efek samping.
- Tinjauan teknis formal harus dilakukan untuk semua kecuali perubahan yang paling sepele.

Audit Konfigurasi ... lanjutan

- Audit konfigurasi perangkat lunak melengkapi tinjauan teknis formal dengan menilai objek konfigurasi untuk karakteristik yang umumnya tidak dipertimbangkan selama peninjauan.
- Audit menanyakan dan menjawab pertanyaan-pertanyaan berikut:
 - Apakah perubahan yang ditentukan dalam ECO telah dilakukan? Apakah ada modifikasi tambahan yang dimasukkan?
 - Apakah tinjauan teknis formal telah dilakukan untuk menilai kebenaran teknis?
 - Apakah proses perangkat lunak telah diikuti dan apakah standar rekayasa perangkat lunak telah diterapkan dengan benar?
 - Apakah perubahan telah "disorot" di SCI? Apakah tanggal perubahan dan penulis perubahan telah ditentukan? Apakah atribut dari objek konfigurasi mencerminkan perubahan?
 - Apakah prosedur SCM untuk mencatat perubahan, mencatat, dan melapkannya telah diikuti?
 - Apakah semua SCI terkait telah diperbarui dengan benar?

(5) Pelaporan Status

- Pelaporan status konfigurasi (kadang disebut akuntansi status) adalah tugas SCM yang menjawab pertanyaan berikut: (1) Apa yang terjadi? (2) Siapa yang melakukannya? (3) Kapan itu terjadi? (4) Apa lagi yang akan terpengaruh?
- Aliran informasi untuk pelaporan status konfigurasi (CSR):
 - Setiap kali SCI diberikan identifikasi baru atau yang diperbarui, entri CSR dibuat.
 - Setiap kali perubahan disetujui oleh CCA (yaitu, ECO dikeluarkan), entri CSR dibuat.
 - Setiap kali audit konfigurasi dilakukan, hasilnya dilaporkan sebagai bagian dari tugas CSR.
 - Output dari CSR dapat ditempatkan dalam database on-line, sehingga pengembang atau pengelola perangkat lunak dapat mengakses informasi perubahan berdasarkan kategori kata kunci.

Pelaporan Status ... lanjutan

- Pelaporan status konfigurasi memainkan peran penting dalam keberhasilan proyek pengembangan perangkat lunak yang besar.
- Ketika banyak orang terlibat, kemungkinan akan terjadi sindrom "tangan kiri tidak tahu apa yang dilakukan tangan kanan".
- CSR membantu menghilangkan masalah ini dengan meningkatkan komunikasi di antara semua orang yang terlibat.

Procedures and Work Instructions

Objectives

- Explain the contribution of procedures to software quality assurance
- Explain the difference between procedures and work instructions
- List the activities involved in maintaining an organization's procedures manual.

Topic Coverage

- The need for procedures and work instructions
- Procedures and procedure manuals
- Work instructions and work instruction manuals
- The organizational framework for preparing, implementing and updating procedures and work instructions.

Software Quality Infrastructure

What are typical infrastructure components?

- Procedures and work instruction
- Quality support devices like templates and checklists
- Staff SQA training and certification activities
- Preventive and corrective actions
- Software configuration management
- Documentation and quality records control.

The need for procedures and work instructions

- "Why should we use SQA procedures and work instructions?"
- "Wouldn't it be better if every professional relied on his own experience and performed his task the best way he knows?"
- "What are the benefits to the organization of forcing me to perform a task only in the way chosen by them?"

Tujuan

- Jelaskan kontribusi prosedur untuk jaminan kualitas perangkat lunak
- Jelaskan perbedaan antara prosedur dan instruksi kerja
- Buat daftar aktivitas yang terlibat dalam pemeliharaan manual prosedur organisasi.

Cakupan Topik

- Perlunya prosedur dan instruksi kerja
- Prosedur dan manual prosedur
- Instruksi kerja dan manual instruksi kerja
- Kerangka organisasi untuk mempersiapkan, melaksanakan dan memutakhirkkan prosedur dan instruksi kerja.

Infrastruktur Kualitas Perangkat Lunak

Apa saja komponen infrastruktur tipikal?

- Prosedur dan instruksi kerja
- Perangkat pendukung berkualitas seperti template dan daftar periksa
- Pelatihan staf SQA dan kegiatan sertifikasi
- Tindakan pencegahan dan perbaikan
- Manajemen konfigurasi perangkat lunak
- Dokumentasi dan kontrol catatan kualitas.

Kebutuhan akan prosedur dan pekerjaan instruksi

- "Mengapa kita harus menggunakan prosedur dan instruksi kerja SQA?"
- "Bukankah lebih baik jika setiap profesional mengandalkan pengalamannya sendiri dan melakukan tugasnya dengan cara terbaik yang dia tahu?"
- "Apa manfaatnya bagi organisasi dengan memaksa saya melakukan tugas hanya dengan cara yang mereka pilih?"

Prosedur dan Instruksi Kerja



Figure 14.1: A Conceptual hierarchy for development of procedures and work instructions
18/4/2022 AMS

Procedures and Work Instructions

- **Procedure:**
 - Detailed activities or processes to be performed according to a given method for the purpose of accomplishing a task
 - Tend to be universal within the organization → they are applied whenever the task is performed (irrespective of the person or the organizational context)
- **Work Instruction:** is a supplement to a procedure → providing explicit details that are suitable solely to the needs of one team/department/unit

Procedures and Work Instructions

- SQA procedures are required to conform to the organization's quality policy but also tend to conform to international or national SQA standards → ISO 9000-3 (ISO, 1997; ISO/IEC, 2001)

Procedures and Work Instructions

- Aiming at:
 - Performance of tasks, processes or activities in the most effective and efficient way without deviating from quality requirements.
 - Effective and efficient communication between the separate staffs involved in the development and maintenance of software systems.
 - Uniformity in performance, achieved by conformity with procedures and work instructions, reduces the misunderstandings that lead to software errors.
 - Simplified coordination between tasks and activities performed by the various bodies of the organization. Better coordination means fewer errors.

Procedures

- **The Five W's: issues resolved by procedures**
 - **What activities have to be performed?**
 - **How should each activity be performed?**
 - **When should the activity be performed?**
 - **Where should the activity be performed?**
 - **Who should perform the activity?**

‘’ Prosedur dan Instruksi Kerja

- **Prosedur:**
 - Aktivitas atau proses terperinci yang akan dilakukan sesuai dengan metode yang diberikan untuk tujuan menyelesaikan tugas
 - Cenderung universal di dalam organisasi – diterapkan setiap kali tugas dilakukan (terlepas dari orang atau konteks organisasi)
- **Instruksi kerja:** adalah suplemen untuk prosedur - memberikan perincian eksplisit yang hanya sesuai dengan kebutuhan satu tim/ departemen/unit

Prosedur dan Instruksi Kerja

- Prosedur SQA diperlukan agar sesuai dengan kebijakan mutu organisasi tetapi juga cenderung sesuai dengan standar SQA internasional atau nasional - ISO 9000-3 (ISO, 1997; ISO/IEC, 2001)

Prosedur dan Instruksi Kerja

Bertujuan untuk:

- Kinerja tugas, proses atau kegiatan dengan cara yang paling efektif dan efisien tanpa menyimpang dari persyaratan kualitas.
- Komunikasi yang efektif dan efisien antara staf terlibat yang terlibat dalam pengembangan dan pemeliharaan sistem perangkat lunak.
- Keseragaman kinerja, dicapai dengan kesesuaian dengan prosedur dan instruksi kerja, mengurangi kesalahan yang menyebabkan kesalahan perangkat lunak.
- Koordinasi yang disederhanakan antara tugas dan aktivitas yang dilakukan oleh berbagai badan organisasi. Koordinasi yang lebih baik berarti lebih sedikit kesalahan.

Prosedur

- **Lima W: masalah diselesaikan dengan prosedur**
 - **What kegiatan topi harus dilakukan?**
 - **Howharuskah setiap aktivitas dilakukan?**
 - **When kapan aktivitas itu harus dilakukan?**
 - **Where di sini haruskah aktivitas itu dilakukan?**
 - **Who harus melakukan aktivitas?**

Procedures Manual

- The collection of all SQA procedures is usually referred to as the *SQA procedures manual*. Usually contains:
 - Type of sw development and maintenance activities carried out by the organization
 - Range of activities belong to each activity type
 - Range of customers

Frame 14.2 Fixed table of contents for procedures

- 1 Introduction *
- 2 Purpose
- 3 Terms and abbreviations *
- 4 Applicable documents
- 5 Method
- 6 Quality records and documentation
- 7 Reporting and follow-up *
- 8 Responsibility for implementation *
- 9 List of appendices *
- Appendices *

* Sections included only if applicable

AMS

Work Instructions and Work Manuals

- Work instructions deal with *the application of procedures, adapted to the requirements of a specific project team, customer, or other relevant party.*

Frame 14.3 SQA work instructions subjects – examples
Departmental work instructions <ul style="list-style-type: none">■ Audit process for new software development subcontractors (supplier candidates)■ Priorities for handling corrective maintenance tasks■ Annual evaluation of software development subcontractors■ On-the-job instructions and follow-up for new team members■ Design documentation templates and their application■ C++ (or other language) programming instructions
Project management work instructions <ul style="list-style-type: none">■ Coordination and cooperation with the customer■ Weekly progress reporting by team leaders■ Special design report templates and their application in the project■ Follow-up of beta site reporting■ Monthly progress reporting to the customer■ 4/1999 coordination of installation and customer's team instructions

Updating Procedures

The motivation to update existing procedures is based, among other things, on the following:

- Technological changes in development tools, hardware, communication equipment, etc.
- Changes in the organization's areas of activity
- User proposals for improvement
- Analysis of failures as well as successes
- Proposals for improvements initiated by internal audit reports
- Learning from the experience of other organizations
- Experiences of the SQA team

Prosedur Manual

- Pengumpulan semua prosedur SQA biasanya disebut sebagai *manual prosedur SQA*. Biasanya berisi:
 - Jenis kegiatan pengembangan dan pemeliharaan yang dilakukan oleh organisasi
 - Rentang kegiatan milik setiap jenis kegiatan
 - Jangkauan pelanggan

Instruksi Kerja dan Kerja manual

- Kesepakatan instruksi kerja dengan *penerapan prosedur, disesuaikan dengan persyaratan tim proyek tertentu, pelanggan, atau pihak terkait lainnya.*

Memperbarui Prosedur

Motivasi untuk memperbarui prosedur yang ada antara lain didasarkan pada:

- Perubahan teknologi dalam alat pengembangan, perangkat keras, peralatan komunikasi, dll.
- Perubahan dalam area aktivitas organisasi
- Usulan pengguna untuk perbaikan
- Analisis kegagalan serta keberhasilan
- Usulan perbaikan yang diprakarsai oleh laporan audit internal
- Belajar dari pengalaman organisasi lain
- Pengalaman tim SQA

Procedures and Work Instructions

Definition

- (1) A quantitative measure of the degree to which an item possesses a given quality attribute.
- (2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

Definisi

- (1) Ukuran kuantitatif sejauh mana suatu item memiliki atribut kualitas tertentu.
- (2) Fungsi yang inputnya adalah data perangkat lunak dan outputnya berupa nilai numerik tunggal yang dapat diinterpretasikan sebagai tingkat di mana perangkat lunak memiliki atribut kualitas tertentu.

Objectives of quality measurement

- To facilitate management control as well as planning and execution of the appropriate managerial interventions. Achievement of this objective is based on calculation of metrics regarding:
 - Deviations of actual functional (quality) performance from planned performance
 - Deviations of actual timetable and budget performance from planned performance.
- To identify situations that require or enable development or maintenance process improvement in the form of preventive or corrective actions introduced throughout the organization. Achievement of this objective is based on:
 - Accumulation of metrics information regarding the performance of teams, units, etc.

Objective (2)

- The metrics are used for comparison of performance data with indicators, quantitative values such as:
 - Defined software quality standards
 - Quality targets set for organizations or individuals
 - Previous year's quality achievements
 - Previous project's quality achievements
 - Average quality levels achieved by other teams applying the same development tools in similar development environments
 - Average quality achievements of the organization
 - Industry practices for meeting quality requirements.

Software quality metrics – requirements

General requirements	Explanation
Relevant	Related to an attribute of substantial importance
Valid	Measures the required attribute
Reliable	Produces similar results when applied under similar conditions
Comprehensive	Applicable to a large variety of implementations and situations
Mutually exclusive	Does not measure attributes measured by other metrics

Software quality metrics – requirements

Operative requirements	Explanation
Easy and simple	The implementation of the metrics data collection is simple and is performed with minimal resources
Does not require independent data collection	Metrics data collection is integrated with other project data collection systems: employee attendance, wages, cost accounting, etc. In addition to its efficiency aspects, this requirement contributes to coordination of all information systems serving the organization
Immune to biased interventions by interested parties	In order to escape the expected results of the analysis of the metrics, it is expected that interested persons will try to change the data and, by doing so, improve their record. Such actions obviously bias the relevant metrics. Immunity (total or at least partial) is achieved mainly by choice of metrics and adequate procedures

Tujuan pengukuran kualitas

- Untuk memfasilitasi kontrol manajemen serta perencanaan dan pelaksanaan intervensi manajerial yang sesuai. Pencapaian tujuan ini didasarkan pada perhitungan metrik mengenai:
 - Penyimpangan kinerja fungsional (kualitas) aktual dari kinerja yang direncanakan
 - Penyimpangan jadwal aktual dan kinerja anggaran dari kinerja yang direncanakan.
- Untuk mengidentifikasi situasi yang memerlukan atau memungkinkan pengembangan atau perbaikan proses pemeliharaan dalam bentuk tindakan pencegahan atau korektif yang diperkenalkan di seluruh organisasi. Pencapaian tujuan ini didasarkan pada:
 - Akumulasi informasi metrik mengenai kinerja tim, unit, dll.

Tujuan (2)

- Metrik digunakan untuk perbandingan data kinerja dengan indikator, nilai kuantitatif seperti:
 - Standar kualitas perangkat lunak yang ditetapkan
 - Target kualitas yang ditetapkan untuk organisasi atau individu
 - Prestasi kualitas tahun sebelumnya
 - Pencapaian kualitas proyek sebelumnya
 - Tingkat kualitas rata-rata yang dicapai oleh tim lain yang menerapkan alat pengembangan yang sama di lingkungan pengembangan yang serupa
 - Pencapaian kualitas rata-rata organisasi
 - Praktik industri untuk memenuhi persyaratan kualitas.

Classification of software quality metric

- First Classification
 - Process Metrics
 - Product Metrics
- Second Classification
 - Quality
 - Timetable
 - Effectiveness
 - Productivity

Klasifikasi kualitas perangkat lunak metrik

- Klasifikasi Pertama
 - Metrik Proses
 - Metrik Produk
- Klasifikasi Kedua
 - Kualitas
 - Jadwal
 - Efektivitas
 - Produktifitas

Process Metrics

- Software process quality metrics
- Software process timetable metrics
- Software process productivity metrics.

Software process quality metrics

- Error density metrics
- Error severity metrics
- Error removal effectiveness metrics

Error density metrics

- Software volume measures.
 - Some density metrics use the number of lines of code while others apply function points.
- Errors counted measures.
 - Some relate to the number of errors and others to the weighted number of errors. Weighted measures that ascertain the severity of the errors are considered to provide more accurate evaluation of the error situation.
 - A common method applied to arrive at this measure is classification of the detected errors into severity classes, followed by weighting each class

NCE and WCE

Error severity class a	Calculation of NCE (number of errors) b		Calculation of WCE Weighted errors D = b × c
	Relative weight c		
Low severity	42	1	42
Medium severity	17	3	51
High severity	11	9	99
Total	70	—	192
NCE	70	—	—
WCE	—	—	192

Error density metric

Code	Name	Calculation formula
CED	Code Error Density	$CED = \frac{NCE}{KLOC}$
DED	Development Error Density	$DED = \frac{NDE}{KLOC}$
WCED	Weighted Code Error Density	$WCED = \frac{WCE}{KLOC}$
WDDED	Weighted Development Error Density	$WDDED = \frac{WDE}{KLOC}$
WCEF	Weighted Code Errors per Function point	$WCEF = \frac{WCE}{NFP}$
WDEF	Weighted Development Errors per Function point	$WDEF = \frac{WDE}{NFP}$

Error severity metrics

Code	Name	Calculation formula
ASCE	Average Severity of Code Errors	$ASCE = \frac{WCE}{NCE}$
ASDE	Average Severity of Development Errors	$ASDE = \frac{WDE}{NDE}$

Metrik Proses

- Metrik kualitas proses perangkat lunak
- Metrik jadwal proses perangkat lunak
- Metrik produktivitas proses perangkat lunak.

Metrik kualitas proses perangkat lunak

- Metrik kepadatan kesalahan
- Metrik keparahan kesalahan
- Metrik efektivitas penghapusan kesalahan

Metrik kepadatan kesalahan

- Ukuran volume perangkat lunak.
 - Beberapa metrik kepadatan menggunakan jumlah baris kode sementara yang lain menerapkan titik fungsi.
- Kesalahan menghitung tindakan.
 - Beberapa berhubungan dengan jumlah kesalahan dan lain-lain dengan jumlah kesalahan tertimbang. Langkah-langkah berbobot yang memastikan tingkat keparahan kesalahan dianggap memberikan evaluasi yang lebih akurat dari situasi kesalahan.
 - Metode umum yang diterapkan untuk mencapai ukuran ini adalah klasifikasi kesalahan yang terdeteksi ke dalam kelas keparahan, dilukti dengan pembobotan setiap kelas

Error removal effectiveness metrics

Code	Name	Calculation formula
DERE	Development Errors Removal Effectiveness	$DERE = \frac{NDE}{NDE + NYF}$
DWERE	Development Weighted Errors Removal Effectiveness	$DWERE = \frac{WDE}{WDE + WYF}$

Key:

■ NYF = number of software failures detected during a year of maintenance service.
 ■ WYF = weighted number of software failures detected during a year of maintenance service.

Software process timetable metric

Code	Name	Calculation formula
TTO	Time Table Observance	$TTO = \frac{MSOT}{MS}$
ADMC	Average Delay of Milestone Completion	$ADMC = \frac{TCDAM}{MS}$

Key:

■ MSOT = milestones completed on time.
 ■ MS = total number of milestones.
 ■ TCDAM = total Completion Delays (days, weeks, etc.) for All Milestones. To calculate this measure, delays reported for all relevant milestones are summed up. Milestones completed on time or before schedule are considered "0" delays. Some professionals refer to completion of milestones before schedule as "minus" delays. These are considered to balance the effect of accounted-for delays (we might call the latter "plus" delays). In these cases, the value of the ADMC may be lower than the value obtained according to the metric originally suggested.

Software process productivity metrics

Code	Name	Calculation formula
DevP	Development Productivity	$DevP = \frac{DevH}{KLOC}$
FDevP	Function point Development Productivity	$FDevP = \frac{DevH}{NFP}$
CR	Code Reuse	$CR = \frac{ReKLOC}{KLOC}$
DocRe	Documentation Reuse	$DocRe = \frac{ReDoc}{NDoc}$

Key:

- DevH = total working hours invested in the development of the software system.
- ReKLOC = number of thousands of reused lines of code.
- ReDoc = number of reused pages of documentation.
- NDoc = number of pages of documentation.

Product metrics

- Product metrics refer to the software system's operational phase – years of regular use of the software system by customer.
- Customer services are of two main types:
 - Help desk services (HD) – software support by instructing customers regarding the method of application of the software and solution of customer implementation problems.
 - Corrective maintenance services – correction of software failures identified by customers/users or detected by the customer service team prior to their discovery by customers.

Product metrics

- HD quality metrics
- HD productivity and effectiveness metrics
- Corrective maintenance quality metrics
- Corrective maintenance productivity and effectiveness metrics

HD quality metrics

- HD calls density metrics – the extent of customer requests for HD services as measured by the number of calls.
- Metrics of the severity of the HD issues raised.
- HD success metrics – the level of success in responding to these calls. A success is achieved by completing the required service within the time determined in the service contract

HD calls density metrics

Code	Name	Calculation formula
HDD	HD calls Density	$HDD = \frac{NHYC}{KLMC}$
WHDD	Weighted HD calls Density	$WHDD = \frac{WHYC}{KLMC}$
WHDF	Weighted HD calls per Function point	$WHDF = \frac{WHYC}{NMFP}$

Key:

- NHYC = number of HD calls during a year of service.
- KLMC = thousands of lines of maintained software code.
- WHYC = weighted HD calls received during one year of service.
- NMFP = number of function points to be maintained.

Metrik produk

- Metrik produk mengacu pada fase operasional sistem perangkat lunak – tahun penggunaan reguler sistem perangkat lunak oleh pelanggan.
- Layanan pelanggan terdiri dari dua jenis utama:
 - Help desk services (HD) – dukungan perangkat lunak dengan menginstruksikan pelanggan mengenai metode penerapan perangkat lunak dan solusi masalah implementasi pelanggan.
 - Layanan pemeliharaan korektif – koreksi kegagalan perangkat lunak yang diidentifikasi oleh pelanggan/pengguna atau terdeteksi oleh tim layanan pelanggan sebelum ditemukan oleh pelanggan.

Metrik produk

- Metrik kualitas HD
- Metrik produktivitas dan efektivitas HD
- Metrik kualitas pemeliharaan korektif
- Metrik produktivitas dan efektivitas pemeliharaan korektif

Metrik kualitas HD

- Metrik kepadatan panggilan HD – tingkat permintaan pelanggan untuk layanan HD yang diukur dengan jumlah panggilan.
- Metrik tingkat keparahan masalah HD yang diangkat.
- Metrik keberhasilan HD – tingkat keberhasilan dalam menanggapi panggilan ini. Keberhasilan dicapai dengan menyelesaikan layanan yang diperlukan dalam waktu yang ditentukan dalam kontrak layanan

HD productivity and effectiveness metrics

Code	Name	Calculation formula
HDP	HD Productivity	$HDP = \frac{HDYH}{KLMC}$
FHDP	Function point HD Productivity	$FHDP = \frac{HDYH}{NMFP}$

Key:

- HDYH = total yearly working hours invested in HD servicing of the software system.

■ KLMC and NMFP are as defined in Table 21.6.

Kualitas pemeliharaan korektif metrik

- Metrik kepadatan kegagalan sistem perangkat lunak – berurusan dengan tingkat permintaan untuk pemeliharaan korektif, berdasarkan catatan kegagalan yang diidentifikasi selama operasi reguler sistem perangkat lunak.
- Metrik keparahan kegagalan sistem perangkat lunak – menangani tingkat keparahan kegagalan sistem perangkat lunak yang ditangani oleh tim pemeliharaan korektif. (Keparahan Rata-rata Kegagalan Sistem Perangkat Lunak – ASSF = WYF/NYF -, mengacu pada kegagalan perangkat lunak yang terdeteksi selama periode satu tahun)
- Kegagalan metrik layanan pemeliharaan – menangani kasus di mana layanan pemeliharaan tidak dapat menyelesaikan koreksi kegagalan tepat waktu atau bahwa koreksi yang dilakukan gagal. Maintenance Repeated repair Failure (MRepF), didefinisikan sebagai RepYF/NYF, di mana RepYF adalah jumlah panggilan kegagalan perangkat lunak yang berulang (kegagalan layanan)
- Metrik ketersediaan sistem perangkat lunak – berurusan dengan tingkat gangguan yang disebabkan oleh pelanggan seperti yang disadari oleh periode waktu di mana layanan sistem perangkat lunak tidak tersedia atau hanya tersedia sebagian

Corrective maintenance quality metrics

- Software system failures density metrics – deal with the extent of demand for corrective maintenance, based on the records of failures identified during regular operation of the software system.
- Software system failures severity metrics – deal with the severity of software system failures attended to by the corrective maintenance team. (Average Severity of Software System Failures – ASSSF = WYF/NYF - , refers to software failures detected during a period of one year)
- Failures of maintenance services metrics – deal with cases where maintenance services were unable to complete the failure correction on time or that the correction performed failed. Maintenance Repeated repair Failure (MRepF), is defined as RepYF/NYF, where RepYF is the number of repeated software failure calls (service failures)
- Software system availability metrics – deal with the extent of disturbances caused to the customer as realized by periods of time where the services of the software system are unavailable or only partly available

Software system failures density metrics

Code	Name	Calculation formula
SSFD	Software System Failure Density	$SSFD = \frac{NYF}{KLMC}$
WSSFD	Weighted Software System Failure Density	$WSSFD = \frac{WYF}{KLMC}$
WSSFF	Weighted Software System Failures per Function point	$WSSFF = \frac{WYF}{NMFP}$

Key:
 ■ NYF = number of software failures detected during a year of maintenance service.
 ■ WYF = weighted number of yearly software failures detected during a year of maintenance service.
 ■ KLMC = thousands of lines of maintained software code.
 ■ NMFP = number of function points designated for the maintained software.

Software system availability metrics

Code	Name	Calculation formula
FA	Full Availability	$FA = \frac{NYSerH - NYFH}{NYSerH}$
VitA	Vital Availability	$VitA = \frac{NYSerH - NYVitFH}{NYSerH}$
TUA	Total Unavailability	$TUA = \frac{NYTH}{NYSerH}$

Key:
 ■ NYSerH = number of hours software system is in service during one year. For an office software system that is operating 50 hours per week for 52 weeks per year, NYSerH = 2600 (50×52). For a real-time software application that serves users 24 hours a day, NYSerH = 8760 (365×24).
 ■ NYFH = number of hours where at least one function is unavailable (failed) during one year, including total failure of the software system.
 ■ NYVitFH = number of hours when at least one vital function is unavailable (failed) during one year, including total failure of the software system.
 ■ NYTH = number of hours of total failure (all system functions failed) during one year.
 ■ NYFH \geq NYVitFH \geq NYTH.
 ■ 1 – TUA \geq VitA \geq FA.

Software corrective maintenance productivity and effectiveness metrics

Code	Name	Calculation formula
CMaiP	Corrective Maintenance Productivity	$CMaiP = \frac{CMaiYH}{KLMC}$
FCMP	Function point Corrective Maintenance Productivity	$FCMP = \frac{CMaiYH}{NMFP}$
CMaiE	Corrective Maintenance Effectiveness	$CMaiE = \frac{CMaiYH}{NYF}$

Key:
 ■ CMaiYH = total yearly working hours invested in the corrective maintenance of the software system.
 ■ KLMC = thousands of lines of maintained software code.
 ■ NMFP = number of function points designated for the maintained software.
 ■ NYF = number of software failures detected during a year of maintenance service.

Implementation of software quality metrics

Requires:

- Definition of software quality metrics – relevant and adequate for teams, departments, etc.
- Regular application by unit, etc.
- Statistical analysis of collected metrics data.
- Subsequent actions:
 - Changes in the organization and methods of software development and maintenance units and/or any other body that collected the metrics data
 - Change in metrics and metrics data collection
 - Application of data and data analysis to planning corrective actions for all the relevant units.

Implementasi kualitas perangkat lunak metrik

Memerlukan:

- Definisi metrik kualitas perangkat lunak – relevan dan memadai untuk tim, departemen, dll.
- Aplikasi reguler berdasarkan unit, dll.
- Analisis statistik dari data metrik yang dikumpulkan.
- Tindakan selanjutnya:
 - Perubahan dalam organisasi dan metode unit pengembangan dan pemeliharaan perangkat lunak dan/ atau badan lain yang mengumpulkan data metrik
 - Perubahan dalam metrik dan pengumpulan data metrik
 - Penerapan data dan analisis data untuk merencanakan tindakan korektif untuk semua unit terkait.

Definition of new software quality metrics

Four Stages Process

- Definition of attributes to be measured: software quality, development team productivity, etc.
- Definition of the metrics that measure the required attributes and confirmation of its adequacy in complying with the requirements listed.
- Determination of comparative target values based on standards, previous year's performance, etc. These values serve as indicators of whether the unit measured (a team or an individual or a portion of the software) complies with the characteristics demanded of a given attribute.
- Determination of metrics application processes:
 - Reporting method, including reporting process and frequency of reporting
 - Metrics data collection method.

Definisi metrik kualitas perangkat lunak baru

Proses Empat Tahap

- Definisi atribut yang akan diukur: kualitas perangkat lunak, produktivitas tim pengembangan, dll.
- Definisi metrik yang mengukur atribut yang diperlukan dan konfirmasi kecukupannya dalam memenuhi persyaratan yang tercantum.
- Penentuan nilai target komparatif berdasarkan standar, kinerja tahun sebelumnya, dll. Nilai-nilai ini berfungsi sebagai indikator apakah unit yang diukur (tim atau individu atau sebagian dari perangkat lunak) sesuai dengan karakteristik yang diminta dari atribut yang diberikan.
- Penentuan metrik proses aplikasi:
 - Metode pelaporan, termasuk proses pelaporan dan frekuensi pelaporan
 - Metode pengumpulan data metrik.

Application of the metrics – managerial aspects

- Assigning responsibility for reporting and metrics data collection.
- Instruction of the team regarding the new metrics.
- Follow-up includes:
 - Support for solving application problems and provision of supplementary information when needed.
 - Control of metrics reporting for completeness and accuracy.
 - Updates and changes of metrics definitions together with reporting and data collection methods according to past performance.

Example – Comparison of US and Japanese software industries

source: Cusumano (1991)

Name	Calculation formula
Mean productivity (similar to DevP, Table 21.5)	KNLOC WorkY
Failure density (similar to SSFD, Table 21.8)	NYF KNLOC
Code reuse (similar to CR, Table 21.5)	ReKNLOC KNLOC

Key:
■ KNLOC = thousands of non-comment lines of code.
■ WorkY = human work-years invested in the software development.
■ ReKNLOC = thousands of reused non-comment lines of code.

Example – Comparison of US and Japanese software industries

source: Cusumano (1991)

Software quality metrics	United States	Japan
Mean productivity	7290	12447
Failure density	4.44	1.96
Code reuse	9.71%	18.25%
N – (number of companies)	20	11

Statistical analysis of metrics data

Background:

- Are there significant differences between the HD teams' quality of service?
- Do the metrics results support the assumption that application of the new version of a development tool contributes significantly to software quality?
- Do the metrics results support the assumption that reorganization has contributed significantly to a team's productivity

Statistical analysis of metrics data

- Descriptive statistics, such as the mean, median and mode as well as use of graphic presentations such as histograms, cumulative distribution graphs, pie charts and control charts (showing also the indicator values) to illustrate the information to which they relate, enable us to quickly identify trends in the metrics values.
- Analytical statistics, To determine whether the observed changes in the metrics are meaningful, whatever the direction, the observed trends must be assessed for their significance. This is the role of analytic statistics (e.g., regression tests, analysis of variance, or more basic tests such as the T-test and Chi-square test).

Penerapan metrik – manajerial aspek

- Menetapkan tanggung jawab untuk pelaporan dan pengumpulan data metrik.
- Instruksi tim mengenai metrik baru.
- Tindak lanjut meliputi:
 - Dukungan untuk memecahkan masalah aplikasi dan penyediaan informasi tambahan bila diperlukan.
 - Kontrol pelaporan metrik untuk kelengkapan dan akurasi.
 - Pembaruan dan perubahan definisi metrik bersama dengan metode pelaporan dan pengumpulan data sesuai dengan kinerja sebelumnya.

Analisis statistik data metrik

Latar belakang:

- Apakah ada perbedaan yang signifikan antara kualitas layanan tim HD?
- Apakah hasil metrik mendukung asumsi bahwa penerapan versi baru alat pengembangan berkontribusi signifikan terhadap kualitas perangkat lunak?
- Apakah hasil metrik mendukung asumsi bahwa reorganisasi telah memberikan kontribusi yang signifikan terhadap produktivitas tim?

Analisis statistik data metrik

- Statistik deskriptif, seperti mean, median dan modus serta penggunaan presentasi grafis seperti histogram, grafik distribusi kumulatif, diagram lingkaran dan diagram kontrol (menunjukkan juga nilai indikator) untuk mengilustrasikan informasi yang terkait, memungkinkan kita untuk cepat mengidentifikasi tren dalam nilai metrik.
- Statistik analitik, Untuk menentukan apakah perubahan yang diamati dalam metrik itu bermakna, apa pun arahnya, tren yang diamati harus dinilai signifikansinya. Inilah peran statistik analitik (misalnya, uji regresi, analisis varians, atau uji yang lebih mendasar seperti uji T dan uji Chi-kuadrat).

Limitations of software metrics

- Budget constraints in allocating the necessary resources (manpower, funds, etc.) for development of a quality metrics system and its regular application.
- Human factors, especially opposition of employees to evaluation of their activities.
- Uncertainty regarding the data's validity, rooted in partial and biased reporting.

Batasan metrik perangkat lunak

- Keterbatasan anggaran dalam mengalokasikan sumber daya yang diperlukan (tenaga, dana, dll) untuk pengembangan sistem metrik kualitas dan aplikasi regulernya.
- Faktor manusia, terutama penentangan karyawan terhadap evaluasi kegiatan mereka.
- Ketidakpastian validitas data berakar pada pelaporan yang parsial dan bias.

Factors affecting development process parameters,

- Programming style: strongly affects software volume, where "wasteful" coding may double the volume of produced code (KLOC).
- Volume of documentation comments included in the code: affects volume of the code. The volume of comments is usually determined by the programming style (KLOC).
- Software complexity: complex modules require much more development time (per line of code) in comparison to simple modules. Complex modules also suffer from more defects than simple modules of similar size (KLOC, NCE).
- Percentage of reused code: the higher the percentage of reused code incorporated into the software developed, the greater the volume of code that can be produced per day as well as the lower the number of defects detected in reviews, testing and regular use (NDE, NCE).
- Professionalism and thoroughness of design review and software testing teams: affects the number of defects detected (NCE).
- Reporting style of the review and testing results: some teams produce concise reports that present the findings in a small number of items (small NCE), while others produce comprehensive reports, showing the same findings for a large number of items (large NDE and NCE).

Faktor-faktor yang mempengaruhi proses pembangunan parameter,

- Gaya pemrograman: sangat memengaruhi volume perangkat lunak, di mana pengkodean "boros" dapat menggandakan volume kode yang dihasilkan (KLOC).
- Volume komentar dokumentasi yang disertakan dalam kode: memengaruhi volume kode. Volume komentar biasanya ditentukan oleh gaya pemrograman (KLOC).
- Kompleksitas perangkat lunak modul kompleks membutuhkan lebih banyak waktu pengembangan (per baris kode) dibandingkan dengan modul sederhana. Modul kompleks juga mengalami lebih banyak cacat daripada modul sederhana dengan ukuran yang sama (KLOC, NCE).
- Persentase kode yang digunakan kembali: semakin tinggi persentase kode yang digunakan kembali yang dimasukkan ke dalam perangkat lunak yang dikembangkan, semakin besar volume kode yang dapat diproduksi per hari serta semakin rendah jumlah cacat yang terdeteksi dalam peninjauan, pengujian, dan penggunaan reguler (NDE, NCE).
- Profesionalisme dan ketelitian tim peninjau desain dan pengujian perangkat lunak: memengaruhi jumlah cacat yang terdeteksi (NCE).
- Gaya pelaporan hasil tinjauan dan pengujian: beberapa tim menghasilkan laporan singkat yang menyajikan temuan dalam sejumlah kecil item (NCE kecil), sementara yang lain menghasilkan laporan komprehensif, menunjukkan temuan yang sama untuk sejumlah besar item (NDE besar dan NCE).

Factors affecting the magnitude of the product (maintenance) parameters

- Quality of installed software and its documentation (determined by the quality of the development team as well as the review and testing teams): the lower the initial quality of the software, the greater the anticipated software failures identified and subsequent maintenance efforts (NYF, NHYC).
- Programming style and volume of documentation comments included in the code: as in the development stage, both strongly affect the volume of the software to be maintained, where wasteful coding and documentation may double the volume of code to be maintained (KLMC).
- Software complexity: complex modules require investment of many more maintenance resources per line of code than do simple modules, and suffer from more defects left undetected during the development stage (NYF).
- Percentage of reused code: the higher the percentage of reused code, the lower the number of defects detected in regular use as well as the fewer required corrective maintenance and HD efforts (NYF).
- Number of installations, size of the user population and level of applications in use: affect the number of HD calls as well as the number of defects detected by users during regular use (NHYC, NYF).

Faktor-faktor yang mempengaruhi besarnya parameter produk (pemeliharaan)

- Kualitas perangkat lunak yang diinstal dan dokumentasinya (ditentukan oleh kualitas tim pengembangan serta tim peninjau dan pengujian): semakin rendah kualitas awal perangkat lunak, semakin besar kegagalan perangkat lunak yang diantisipasi diidentifikasi dan upaya pemeliharaan selanjutnya (NYF, NHYC).
- Gaya pemrograman dan volume komentar dokumentasi yang disertakan dalam kode: seperti pada tahap pengembangan, keduanya sangat memengaruhi volume perangkat lunak yang akan dipelihara, di mana pengkodean dan dokumentasi yang boros dapat menggandakan volume kode yang harus dipertahankan (KLMC).
- Kompleksitas perangkat lunak: modul kompleks memerlukan investasi lebih banyak sumber daya pemeliharaan per baris kode daripada modul sederhana, dan mengalami lebih banyak cacat yang tidak terdeteksi selama tahap pengembangan (NYF).
- Persentase kode yang digunakan kembali: semakin tinggi persentase kode yang digunakan kembali, semakin rendah jumlah cacat yang terdeteksi dalam penggunaan reguler serta semakin sedikit pemeliharaan korektif dan upaya HD (NYF) yang diperlukan.
- Jumlah penginstalan, ukuran populasi pengguna, dan tingkat aplikasi yang digunakan: memengaruhi jumlah panggilan IID serta jumlah kerusakan yang terdeteksi oleh pengguna selama penggunaan reguler (NHYC, NYF).