北航计算机学院

School of Computer Science & Engineering

Software Architecture: Principles and Practices

Documenting Software Architectures

张莉 教授
北京航空航天大学
软件工程研究所
lily@buaa.edu.cn

Module Objectives

This module will familiarize participants with

- Architectural views and how they are used to document software architectures
- Various types of views, why they are useful, and when to use them
- Notations often used to describe architectures
- What a software architecture document should contain

Why Document an Architecture?

- Architecture serves as the blueprint for the system and the project that develops it.
  - It defines the work assignments.
  - It is the primary carrier of quality attributes
  - It is the best artifact to early analysis
  - It is the key to post-deployment maintenance and mining.
- This blueprint must be understand if it is to be used. It must be communicated if it is to be understand.
- Documentation speaks for the architect, today, tomorrow, and 20 years from now.

Architecture Documentation Also Helps the Architecture Process

- Documentation enables an artifact-driven approach to software design
  - Completing the artifact means we've completed the design task.
- Documentation enables the set of design decisions that must be made along the way to establishing/maintaining the architecture.
- Documentation also clarifies the line between architectural and non-architectural design decisions.
  - Non-architectural design is the term preferred over detailed design. Architectural decision can be quite detailed!
  - Architectural decision are those that affect the system's ability to deliver on its behavioral and quality goals.

So, How do You document a software Architecture?

In practice, the answer seems to be

- "use UML"
- "use JavaDoc"
- "What else do I need besides my Rose class diagrams?"
- "Draw boxes and lines"
- "Not very well"
- "We usually don't"
- "How do you document a *what*?"

Goals

The goals for this course are to

- Help you decide what information about an architecture should be captured
- Provide guidelines and notations for capturing the necessary information, and examples for it
- Answer this question:

  "How do you record an architecture so that others can successfully use it, maintain it, and build a system from it?"

Note that by "document" we do not necessarily mean information printed on paper

Questions

- What software documentation have you read was very good?
- What made it good?

- What software documentation have you read that was very bad?
- What made it bad?

Principles of Sound Documentation

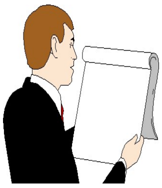
- Seven principles of sound documentation
  1. Write form the reader's point of view
  2. Avoid unnecessary repetition.
  3. Avoid ambiguity.
  4. Use a standard organization.
  5. Record your rationale.
  6. Keep documentation current, but not too current.
  7. Review documentation.

Principles of Sound Documentation

1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.

1. Write from the reader's point of view-1

- Determine who the readers are.
- Determine what readers will want to know
- Make the information easy to find.
- Your readers will appreciate your effort and be more likely to read your document.



1. Write from the reader's point of view-2

- Signs that the documentation was written for the writer's convenience include
  - "stream of consciousness" writing – Topics are presented in the random order that occurred to the writer.
  - "stream of execution" writing -- Topics are presented in which they occur in the computer.
  - Writer obviously makes too many assumptions about what the reader knows

Principles of Sound Documentation

1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.

Avoid Unnecessary Repetition

- Each kind of information should be recorded in only one place.
- This makes documents easier to use and easier to change.
- Repetition often confuses the reader, especially when information is repeated in slightly different ways. The reader is left to wonder
  - "Was the difference intentional? If so, why?"
  - If not, which way is correct?"

Principles of Sound Documentation

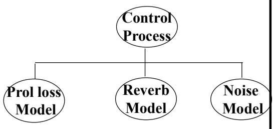
1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.

3. Avoid ambiguity（避免歧义）-1

- Documentation is for communicating information and ideas. If the reader misunderstands because of ambiguities, the documentation has faied.
- Precisely defined notations/languages help avoid ambiguity
- If you documentation uses a graphical language, ***always include a key***. The key should
  - Point to the language's formal definition, or
  - Give the meaning of each symbol (don't forget the lines!). If color or position is significant, define it.
- Make the key meaningful. Don't just say "element" and "relation". Different element/relation types should have different symbols.

3. Avoid ambiguity（避免歧义）-2

- Box-and-line diagrams are a common form of architectural notation.
- But what do they mean?



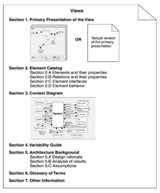
- If you use a box-and-line diagram, always define precisely what the boxes and lines mean.
- If you see a box-and-line diagram, ask the owner what it means. The result is usually entertaining.
- If you document contains known ambiguities, mark them so the can be resolved later.

Principles of Sound Documentation

1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.

4. Use a standard organization.-1

- Establish it, make sure that your documents follow, and make sure that readers know what it is.
- A standard organization
  - Help the reader navigate and find information.
  - Help the writer place information and measure the work left to be done.
  - Lets the writer record information as soon as it is known, in whatever order it is discovered
  - Embodies completeness rules and helps with validation



Views

Section 1. Primary Presentation of the View

OR

Textual version of the primary presentation

Section 2. Element Catalog
Section 2.A Elements and their properties
Section 2.B Relations and their properties
Section 2.C Element interfaces
Section 2.D Element behavior

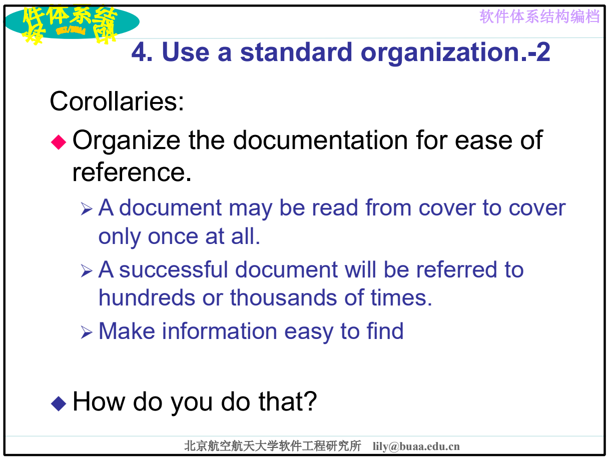
Section 3. Context Diagram

Section 4. Variability Guide

Section 5. Architecture Background
Section 5.A Design rationale
Section 5.B Analysis of results
Section 5.C Assumptions

Section 6. Glossary of Terms

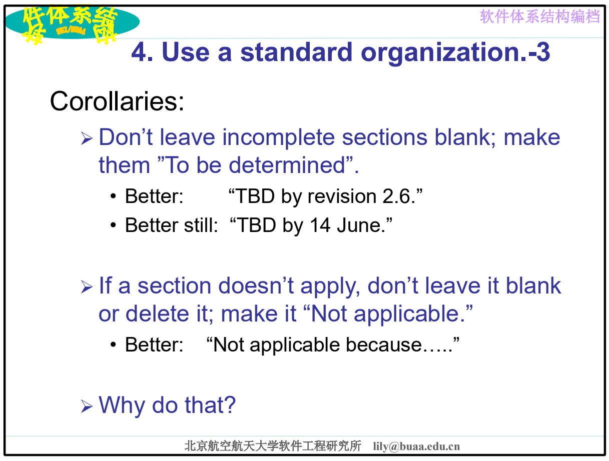
Section 7. Other Information



4. Use a standard organization.-2

Corollaries:

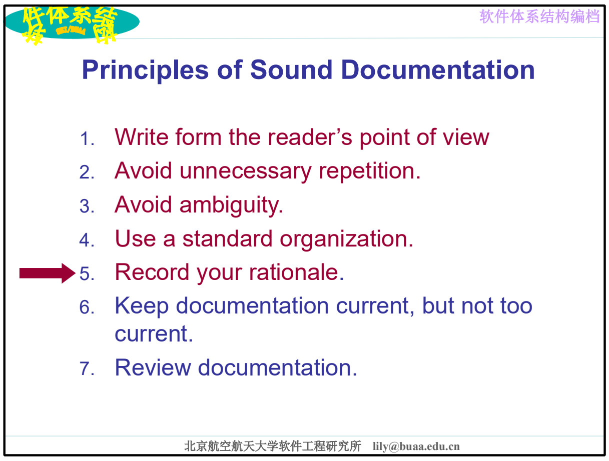
- Organize the documentation for ease of reference.
  - A document may be read from cover to cover only once at all.
  - A successful document will be referred to hundreds or thousands of times.
  - Make information easy to find
- How do you do that?



4. Use a standard organization.-3

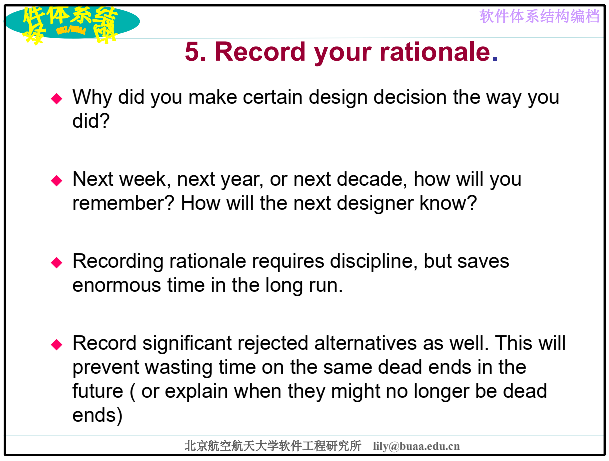
Corollaries:

- Don't leave incomplete sections blank; make them "To be determined".
  - Better: "TBD by revision 2.6."
  - Better still: "TBD by 14 June."
- If a section doesn't apply, don't leave it blank or delete it; make it "Not applicable."
  - Better: "Not applicable because….."
- Why do that?



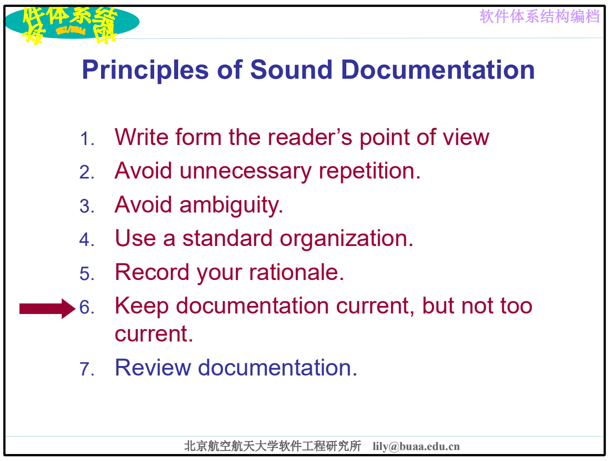
Principles of Sound Documentation

1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.



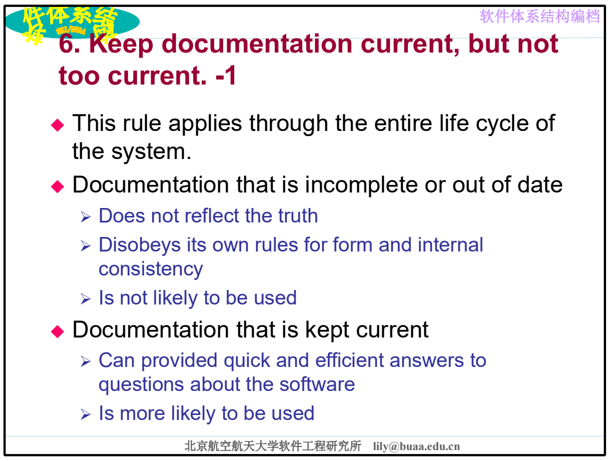
5. Record your rationale.

- Why did you make certain design decision the way you did?
- Next week, next year, or next decade, how will you remember? How will the next designer know?
- Recording rationale requires discipline, but saves enormous time in the long run.
- Record significant rejected alternatives as well. This will prevent wasting time on the same dead ends in the future (or explain when they might no longer be dead ends)



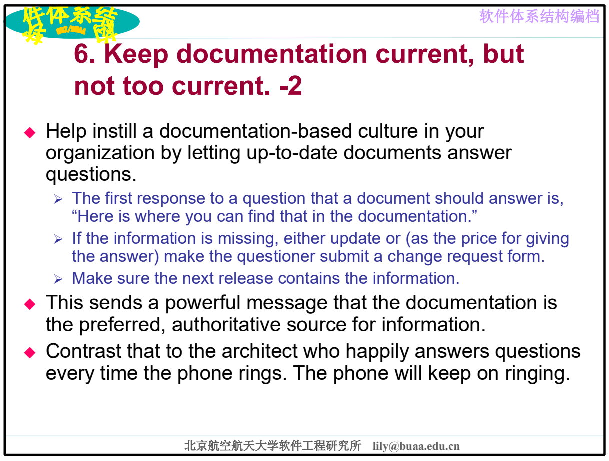
Principles of Sound Documentation

1. Write form the reader's point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.



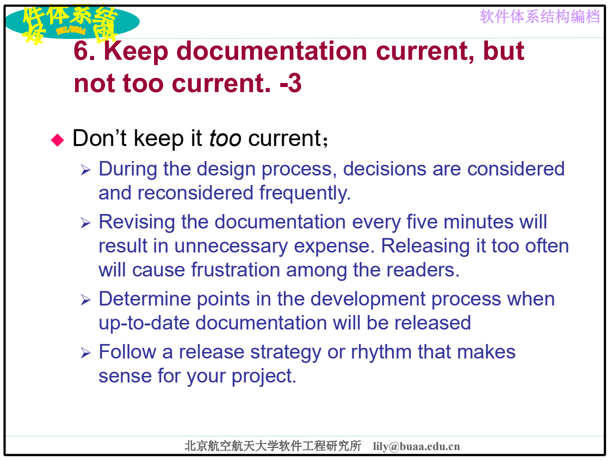
6. Keep documentation current, but not too current. -1

- This rule applies through the entire life cycle of the system.
- Documentation that is incomplete or out of date
  - Does not reflect the truth
  - Disobeys its own rules for form and internal consistency
  - Is not likely to be used
- Documentation that is kept current
  - Can provided quick and efficient answers to questions about the software
  - Is more likely to be used



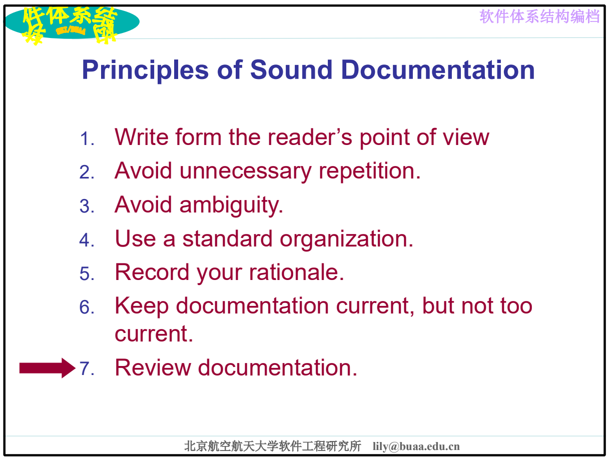
6. Keep documentation current, but not too current. -2

- Help instill a documentation-based culture in your organization by letting up-to-date documents answer questions.
  - The first response to a question that a document should answer is, “Here is where you can find that in the documentation.”
  - If the information is missing, either update or (as the price for giving the answer) make the questioner submit a change request form.
  - Make sure the next release contains the information.
- This sends a powerful message that the documentation is the preferred, authoritative source for information.
- Contrast that to the architect who happily answers questions every time the phone rings. The phone will keep on ringing.



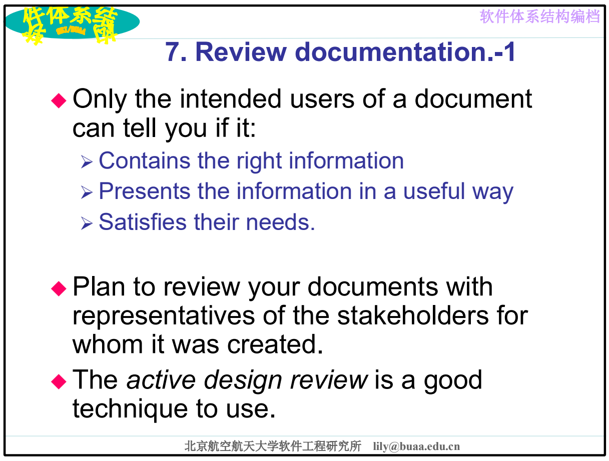
6. Keep documentation current, but not too current. -3

- Don’t keep it *too* current;
  - During the design process, decisions are considered and reconsidered frequently.
  - Revising the documentation every five minutes will result in unnecessary expense. Releasing it too often will cause frustration among the readers.
  - Determine points in the development process when up-to-date documentation will be released
  - Follow a release strategy or rhythm that makes sense for your project.



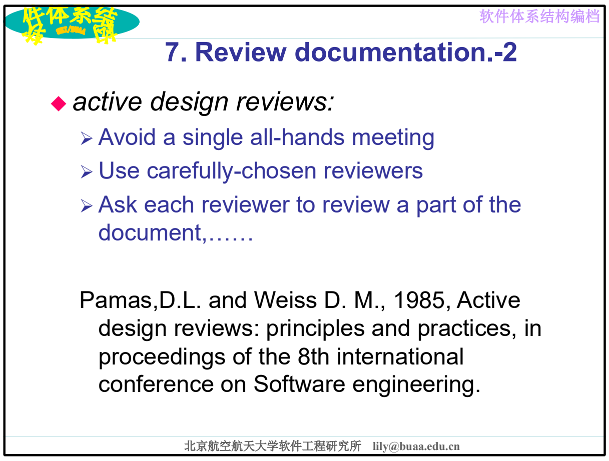
Principles of Sound Documentation

1. Write form the reader’s point of view
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record your rationale.
6. Keep documentation current, but not too current.
7. Review documentation.



7. Review documentation.-1

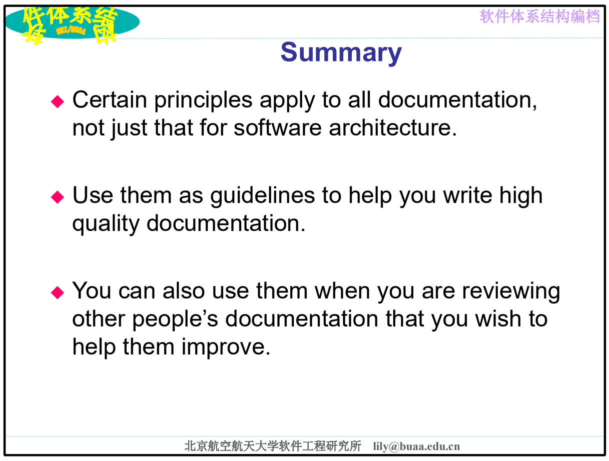
- Only the intended users of a document can tell you if it:
  - Contains the right information
  - Presents the information in a useful way
  - Satisfies their needs.
- Plan to review your documents with representatives of the stakeholders for whom it was created.
- The *active design review* is a good technique to use.



7. Review documentation.-2

- *active design reviews:*
  - Avoid a single all-hands meeting
  - Use carefully-chosen reviewers
  - Ask each reviewer to review a part of the document,……

Pamas,D.L. and Weiss D. M., 1985, Active design reviews: principles and practices, in proceedings of the 8th international conference on Software engineering.



Summary

- Certain principles apply to all documentation, not just that for software architecture.
- Use them as guidelines to help you write high quality documentation.
- You can also use them when you are reviewing other people’s documentation that you wish to help them improve.

Viewtypes, styles and views

- Views 视图
- Viewtypes: type of views 视图类型
- Styles 风格

Views

Recall that a ***view*** is a representation of a set of elements and the relations among them----equivalently, a representation of a structure found in the software.

A view constrains the types of elements, relations, and properties that are represented in that view; for example

- **A Layer view would show layers and the relations among them.**
- **A Client-Server view would show clients and servers and the relations among them.**

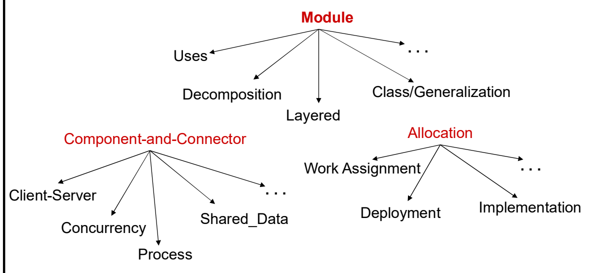
A view therefore shows some system elements but not all of them.

Views give us a way to envision architectures in different ways

View-Based Documentation

- All modern approaches to software architecture creation and documentation are based on views. A general principle for documenting a software architecture is
  - Documenting a software architecture is a matter of documenting the relevant views and then adding information that applies to more than one view.

But Which Views to Consider? - 1



Recall the examples of architectural structures we've seen before. Each one could be represented by a view.

But Which Views to Consider?-2

This “taxonomy” of structures reflects the fact that an architecture must consider the system in three ways:

1. How is it structured as a set of code units?
2. How is it structured as a set of elements that have runtime behavior and interactions.
3. How dose it related to non-software structures in its environment (e.g., hardware)?

But Which Views to Consider?-3

Since a view is a representation of a structure, these three kinds of structures lead to three kinds of views:

1. Module views shows elements that are units of implementation.

2. Components-and-Connector(c&c) views show elements that have runtime behavior and interactions.

3. Allocation views show how software structures are allocated to non-software structures (e.g., hardware).

Module Views

Views: Decomposition, Uses, Layers, Generalization…

Elements: modules, where a module is a code unit that implements some functionality

Relation: Relations among modules can be

- **A "is part of " B.** This defines a part-whole relation among modules.
- **A "depends on" B.** This defines a dependency relation among modules.
- **A "is a" B.** This defines specialization and generalization relations among modules.

What Are Module Views Used for?

- **Construction**: These views are the blueprints for the code. Modules are assigned to teams for implementation and are often the basis of for subsequent design (e.g., of interfaces).
- **Analysis**: Traceability and impact analysis rely on implementation units. Project management, budgeting, planning, and tracking often use modules.
- **Education**: A software developer can learn the development project's structure by understanding module views.

Notation for Module Views

- Informal:
  - box and line (nesting can represent "is part of")
  - textual outline
  - table
- Semi-formal:
  - UML class diagrams and package diagrams

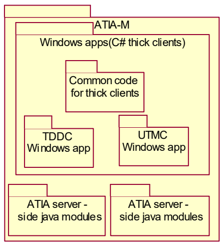
Five Styles in the Module Viewtype

- **Decomposition style**: documents how system responsibilities are partitoined across modules and how those modules are decomposed into submodules
- **Uses style**: tells the developer what other modules must exist for this portion of the system to work correctly.
- **Generalization style**: documents the "is a" relations among elements of the system.
- **Layered style**: documents the "allowed to use" relations among elements of the system.
- **Aspect style**: modules crossing concerns in object-oriented designs.

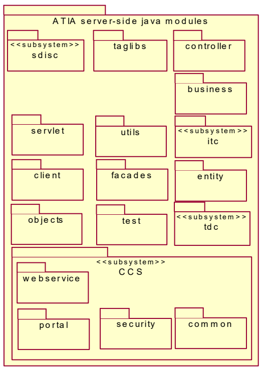
Decomposition style

- **Elements**: modules
- **Relations**:
  - "is part of" Criteria for decomposition vary:
    - Achievement of modifiability
    - Build vs. buy
    - Software product lines: common vs. unique parts
    - Developer's skills
- **Topology**: A child can have only one parent.
- **What it's for**:
  - Starting point for assigning responsibilities to modules as a prelude to subsequent, downstream design.
  - Change/impact analysis
  - Basis for work assignments
  - Basis for unit testing

Example 1: Module Decomposition View in UML


ATIA-M
Windows apps(C# thick clients)
Common code for thick clients
TDDC Windows app
UTMC Windows app
ATIA server - side java modules
ATIA server - side java modules

ATIA=Army Training Information Architecture



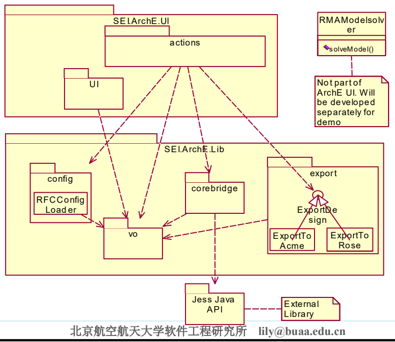
Example 2: Module Decomposition View (Textual Outline)

- **Behavior-Hiding Module**
  - **Function Driver Module**
    - Air Data Computer Module
    - Audible Signal Module
    - Computer Fail Signal Module
    - Doppler Radar Module
    - Flight Information Display Module
    - Forward Looking Radar Module
    - Heads-Up display Module
    - Inertial Measurement Set Module
    - Panel Module
    - Projected Map display set Module
    - Shipboard Inertial Nav.sys.Mod
    - Visual Indicator Module
    - Weapon Release Module
    - Ground Test Module
  - **Shared Services Module**
    - Mode Determination Module
    - Panel I/O Support Module
    - Shared Subroutine Module
    - Stage Director Module
    - System Value Module
- **Software Decision Module**
  - **Application Data Type Module**
    - Numeric data Type Module
    - State transition Event Mod.
  - **Data Banker Module**
    - Singular Values Module
    - Complex Event Module
  - **Filter Behavior Module**
  - **Physical Models Module**
    - Aircraft Motion Module
    - Earth Characteristics Module
    - Human Factors Module
    - Target Behavior Module
    - Weapons Behavior Module
  - **Software Utility Module**
    - Power-Up Initialization Module
    - Numerical Algorithms Module
  - **System Generation Module**
    - System Generation Parameter Mod.
    - Support Software Module

Uses Style

- **Elements**: modules
- **Relations**: uses, a specialization of "*depends on*"
  - A uses B if A depends on the presence of a correctly functioning B to satisfy its(A's) own requirements.
- **Topology**: no constraints(However, loops can cause problems with incremental system delivery.)
- **What it's for**:
  - Planning incremental development
  - System extensions
  - Debugging and testing
  - Gauging the effects of specific changes.

Example 3: Module "Uses" View in UML



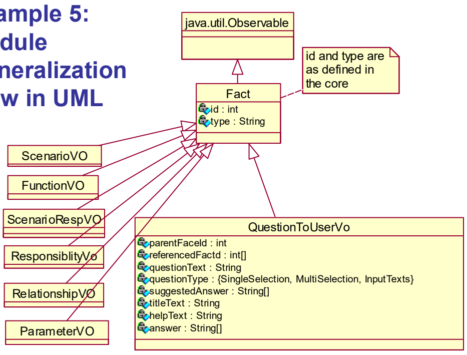
Example 4: Module "Uses" View (Table)

Module Name

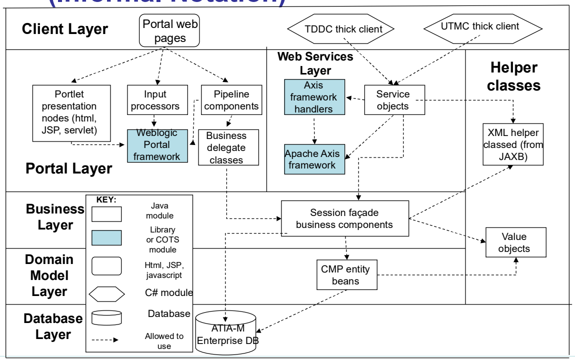
Module Name	A	B	C	D	E
A		√		√	
B			√		
C					
D					
E	√				

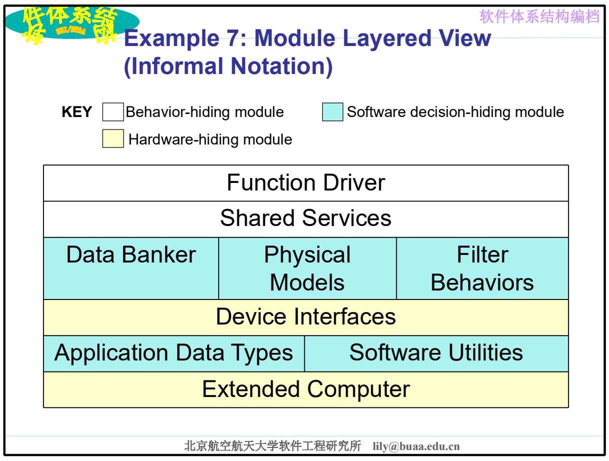
√ = Module in row uses module in column

Example 5: Module Generalization View in UML



Example 6: Module Layered View (Informal Notation)



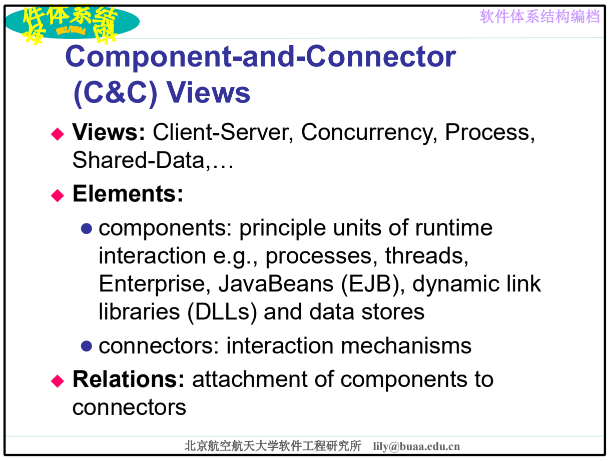


Example 7: Module Layered View (Informal Notation)

KEY □ Behavior-hiding module 🟦 Software decision-hiding module

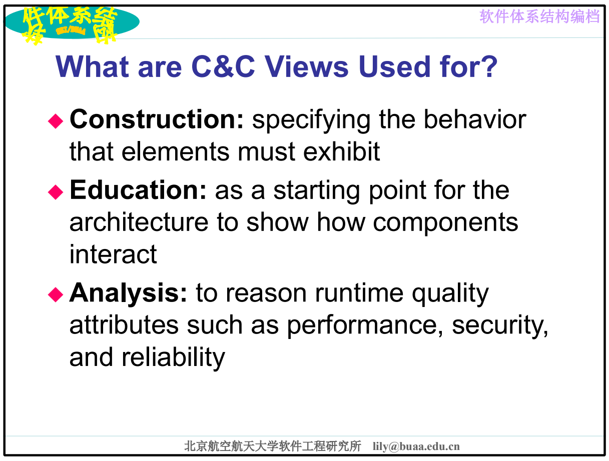
🟨 Hardware-hiding module

<table>
<tr><td colspan="6">Function Driver</td></tr>
<tr><td colspan="6">Shared Services</td></tr>
<tr><td colspan="2">Data Banker</td><td colspan="2">Physical Models</td><td colspan="2">Filter Behaviors</td></tr>
<tr><td colspan="6">Device Interfaces</td></tr>
<tr><td colspan="3">Application Data Types</td><td colspan="3">Software Utilities</td></tr>
<tr><td colspan="6">Extended Computer</td></tr>
</table>



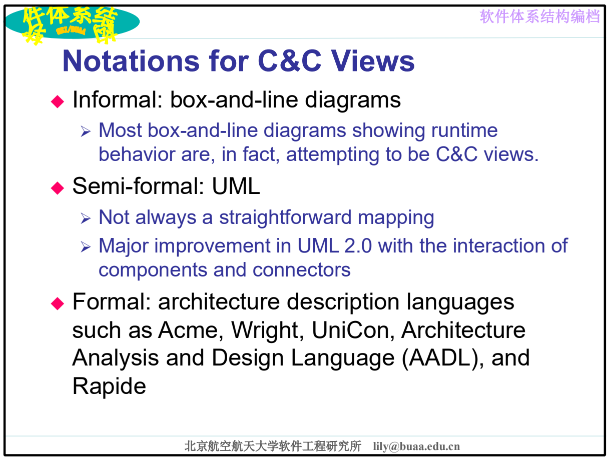
Component-and-Connector (C&C) Views

- **Views:** Client-Server, Concurrency, Process, Shared-Data,…
- **Elements:**
  - components: principle units of runtime interaction e.g., processes, threads, Enterprise, JavaBeans (EJB), dynamic link libraries (DLLs) and data stores
  - connectors: interaction mechanisms
- **Relations:** attachment of components to connectors



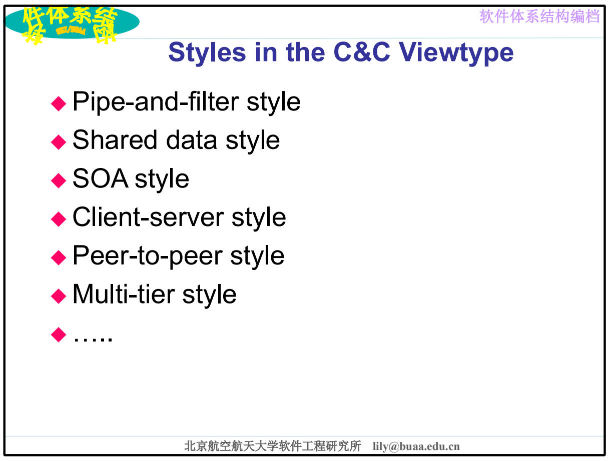
What are C&C Views Used for?

- **Construction:** specifying the behavior that elements must exhibit
- **Education:** as a starting point for the architecture to show how components interact
- **Analysis:** to reason runtime quality attributes such as performance, security, and reliability



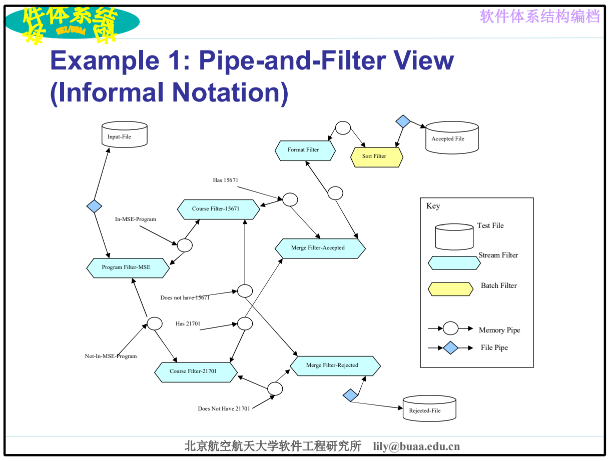
Notations for C&C Views

- Informal: box-and-line diagrams
  - Most box-and-line diagrams showing runtime behavior are, in fact, attempting to be C&C views.
- Semi-formal: UML
  - Not always a straightforward mapping
  - Major improvement in UML 2.0 with the interaction of components and connectors
- Formal: architecture description languages such as Acme, Wright, UniCon, Architecture Analysis and Design Language (AADL), and Rapide



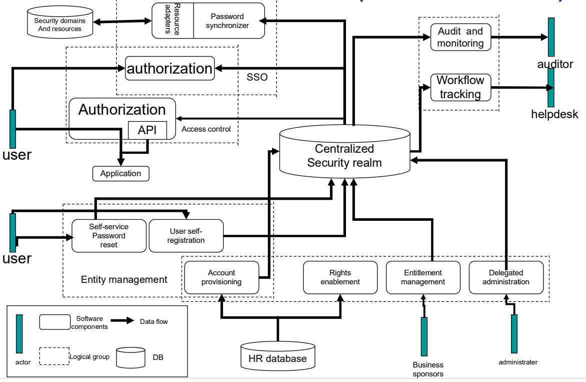
Styles in the C&C Viewtype

- Pipe-and-filter style
- Shared data style
- SOA style
- Client-server style
- Peer-to-peer style
- Multi-tier style
- …..

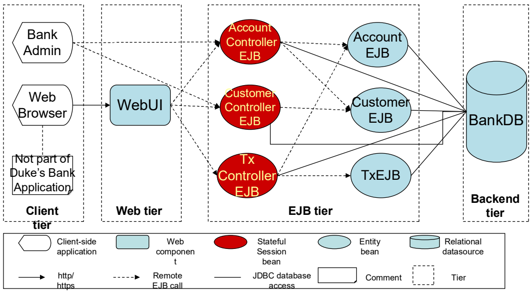


Example 1: Pipe-and-Filter View (Informal Notation)

Example 2: Shared-Data View (Informal Notation)



Example3: Multi-Tier View (Informal Notation)



Reconstructed from Duke's Bank Application---Sun J2EE 1.3 tutorial

Allocation Views

- **Views:** Development, Implementation, Work Assignment…
- **Elements:**
  - **Software elements (from module or C&C views)**
  - **Environment elements (e.g., files, processors, organization units)**
- **Relations:** Software elements are allocated or assigned to environment elements.

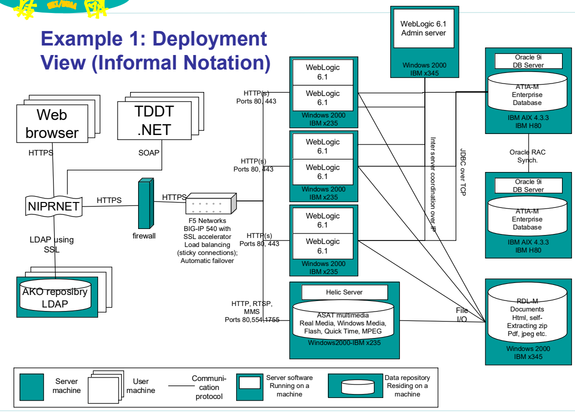
What Are Allocation Views Used for?

- **Communication:** These styles let everyone agree where the software "lives" during its environment, production and execution.
- **Analysis:** Analyzing purchasing options for execution and development environments. The Development style enables detailed performance modeling.
- **Education:** A Software developer can learn the team structure via the Work Assignment style. The Development style shows how the system works in its execution environment.

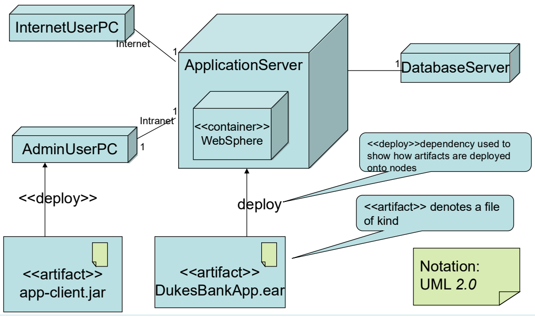
Styles of the Allocation Viewtype

- Deployment style
  - Allocates software elements to processing and communication nodes
  - Properties include those necessary to calculate (and achieve) performance and availability
- Implementation style
  - Allocates software elements to structures in the file system of the development or production environment
  - Properties include files and capacities.
- Work assignment style
  - Allocates software elements to organizational work units
  - Properties include skill sets.

Example 1: Deployment View (Informal Notation)


WebLogic 6.1 Admin server
Windows 2000 IBM x345
WebLogic 6.1
WebLogic 6.1
Windows 2000 IBM x235
Oracle 9i DB Server
ATIA-M Enterprise Database
IBM AIX 4.3.3 IBM H80
Web browser
TDDT .NET
HTTPS
SOAP
HTTP(s) Ports 80, 443
Inter server coordination over IP
JDBC over TCP
Oracle RAC Synch.
NIPRNET
HTTPS
firewall
HTTPS
F5 Networks BIG-IP 540 with SSL accelerator Load balancing (sticky connections); Automatic failover
LDAP using SSL
AKO reposibry LDAP
Helic Server
ASAT multimedia Real Media, Windows Media, Flash, Quick Time, MPEG
Windows2000-IBM x235
HTTP, RTSP, MMS Ports 80,554,1755
File I/O
RDL-M Documents Html, self-Extracting zip Pdf, jpeg etc.
Windows 2000 IBM x345
Server machine
User machine
Communi-cation protocol
Server software Running on a machine
Data repository Residing on a machine

Example 2: Deployment View in UML

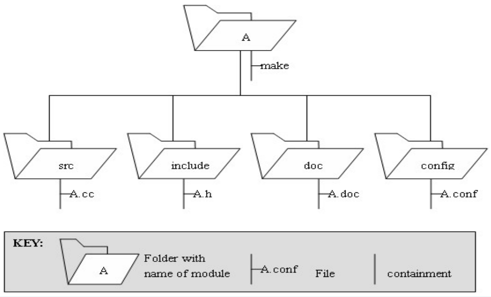


<<deploy>>dependency used to show how artifacts are deployed onto nodes

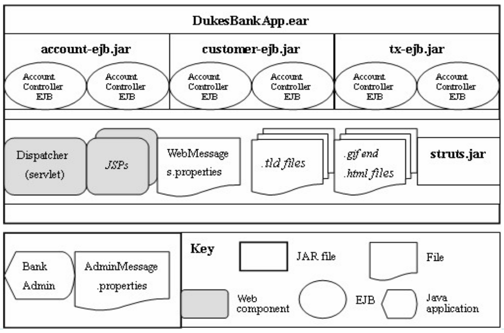
<<artifact>> denotes a file of kind

Notation: UML *2.0*

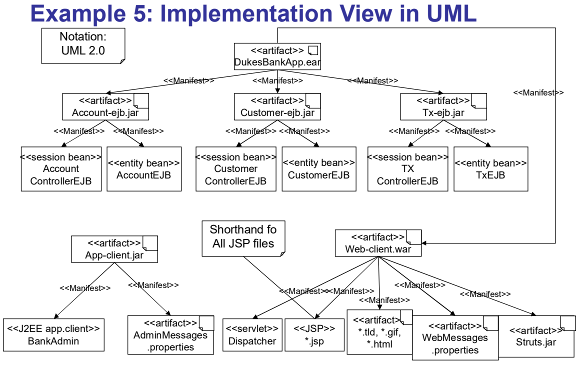
Example 3: Implementation View (Informal Notation)



Example 4: Implementation View (Informal Notation)



Example 5: Implementation View in UML


Notation: UML 2.0
<<artifact>> DukesBankApp.ear
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<artifact>> Account-ejb.jar
<<artifact>> Customer-ejb.jar
<<artifact>> Tx-ejb.jar
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<session bean>> Account ControllerEJB
<<entity bean>> AccountEJB
<<session bean>> Customer ControllerEJB
<<entity bean>> CustomerEJB
<<session bean>> TX ControllerEJB
<<entity bean>> TxEJB
<<artifact>> App-client.jar
Shorthand fo All JSP files
<<artifact>> Web-client.war
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<Manifest>>
<<J2EE app.client>> BankAdmin
<<artifact>> AdminMessages .properties
<<servlet>> Dispatcher
<<JSP>> *.jsp
<<artifact>> *.tld, *.gif, *.html
<<artifact>> WebMessages .properties
<<artifact>> Struts.jar

But Which Views should We Choose?

Which views are the most relevant? It depends on

- Who the stakeholders are
- How they will use the documentation

Four primary uses for architecture documentation are

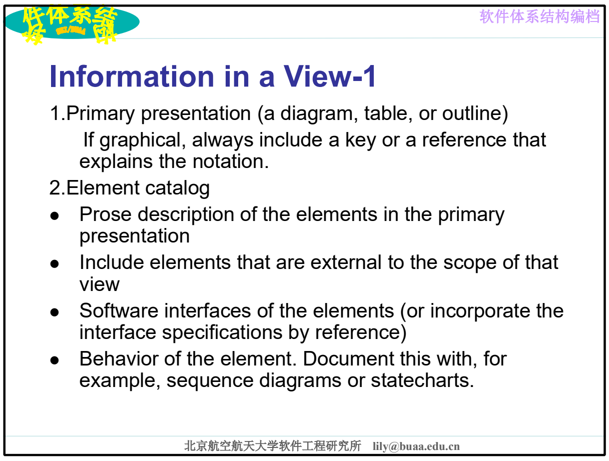
1.Construction - blueprints for the system
2.Education – introducing people to the project
3.Communication – among stakeholders
4.Analysis – assuring quality attributes

Choose the views that will best serve your stakeholders and their concerns. Work with them to make the best choice.

Software Architecture Documentation: More Than Just Diagrams

All we've seen so far are examples of diagrams. But diagrams----box-and-line drawings----are not enough!

What other information should be documented about a view?
What else should go in a documentation package beyond the views?



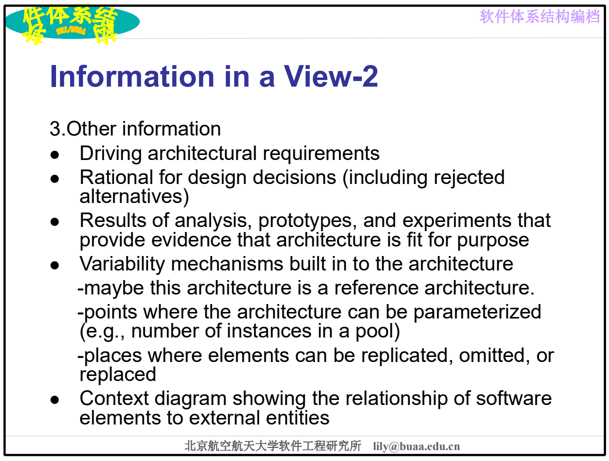
Information in a View-1

1.Primary presentation (a diagram, table, or outline)

If graphical, always include a key or a reference that explains the notation.

2.Element catalog

- Prose description of the elements in the primary presentation
- Include elements that are external to the scope of that view
- Software interfaces of the elements (or incorporate the interface specifications by reference)
- Behavior of the element. Document this with, for example, sequence diagrams or statecharts.



Information in a View-2

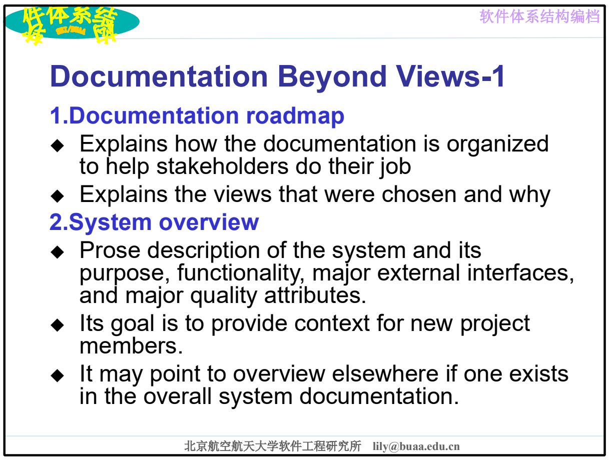
3.Other information

- Driving architectural requirements
- Rational for design decisions (including rejected alternatives)
- Results of analysis, prototypes, and experiments that provide evidence that architecture is fit for purpose
- Variability mechanisms built in to the architecture

  -maybe this architecture is a reference architecture.

  -points where the architecture can be parameterized (e.g., number of instances in a pool)

  -places where elements can be replicated, omitted, or replaced
- Context diagram showing the relationship of software elements to external entities



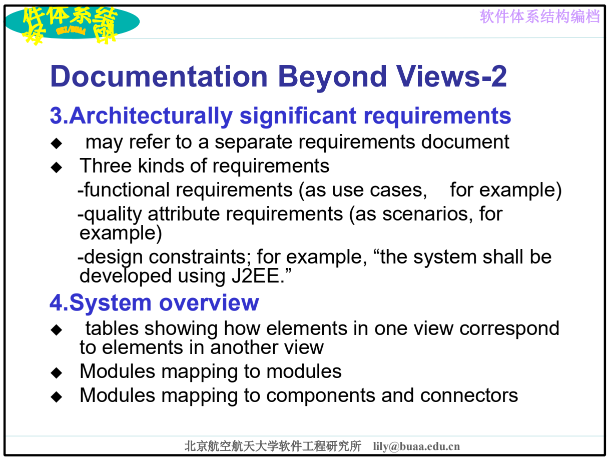
Documentation Beyond Views-1

1.Documentation roadmap

- Explains how the documentation is organized to help stakeholders do their job
- Explains the views that were chosen and why

2.System overview

- Prose description of the system and its purpose, functionality, major external interfaces, and major quality attributes.
- Its goal is to provide context for new project members.
- It may point to overview elsewhere if one exists in the overall system documentation.



Documentation Beyond Views-2

3.Architecturally significant requirements

- may refer to a separate requirements document
- Three kinds of requirements

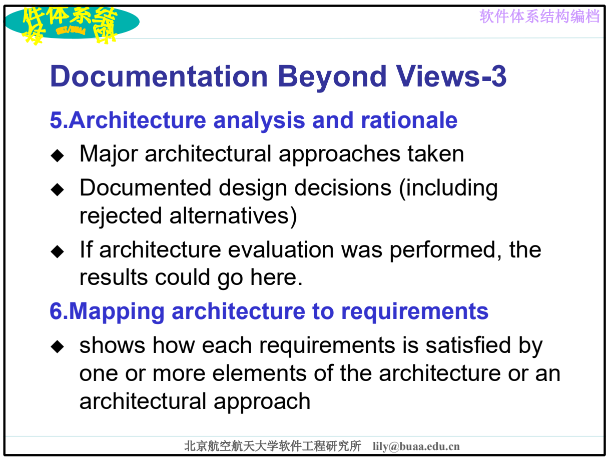
  -functional requirements (as use cases, for example)

  -quality attribute requirements (as scenarios, for example)

  -design constraints; for example, “the system shall be developed using J2EE.”

4.System overview

- tables showing how elements in one view correspond to elements in another view
- Modules mapping to modules
- Modules mapping to components and connectors



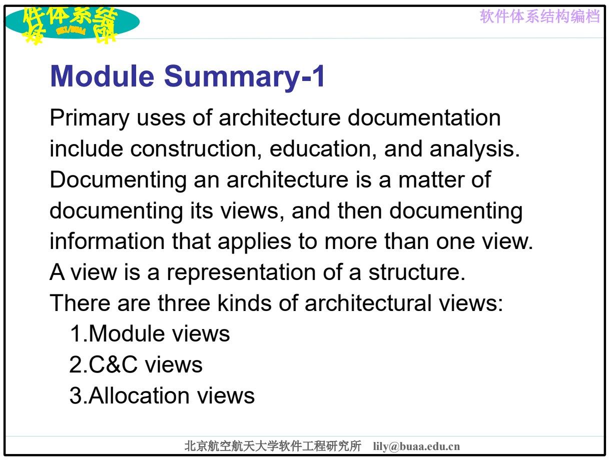
Documentation Beyond Views-3

5.Architecture analysis and rationale

- Major architectural approaches taken
- Documented design decisions (including rejected alternatives)
- If architecture evaluation was performed, the results could go here.

6.Mapping architecture to requirements

- shows how each requirements is satisfied by one or more elements of the architecture or an architectural approach



Module Summary-1

Primary uses of architecture documentation include construction, education, and analysis.

Documenting an architecture is a matter of documenting its views, and then documenting information that applies to more than one view.

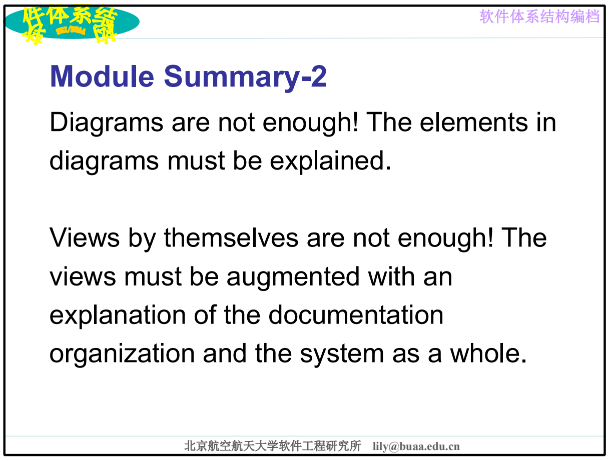
A view is a representation of a structure.

There are three kinds of architectural views:

1.Module views

2.C&C views

3.Allocation views



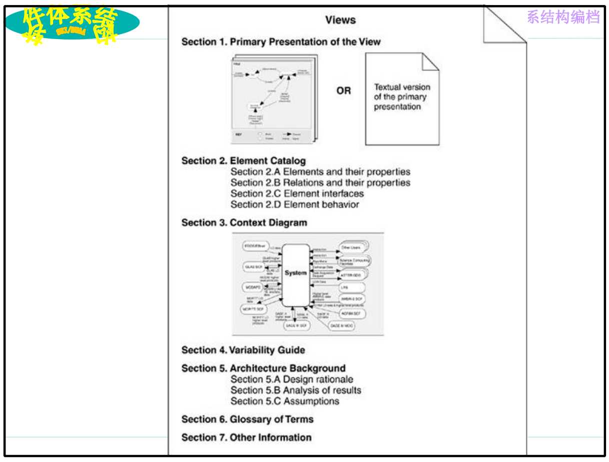
软件体系结构编档

Module Summary-2

Diagrams are not enough! The elements in diagrams must be explained.

Views by themselves are not enough! The views must be augmented with an explanation of the documentation organization and the system as a whole.

北京航空航天大学软件工程研究所 lily@buaa.edu.cn



系统结构编档

Views

Section 1. Primary Presentation of the View

OR

Textual version of the primary presentation

Section 2. Element Catalog
Section 2.A Elements and their properties
Section 2.B Relations and their properties
Section 2.C Element interfaces
Section 2.D Element behavior

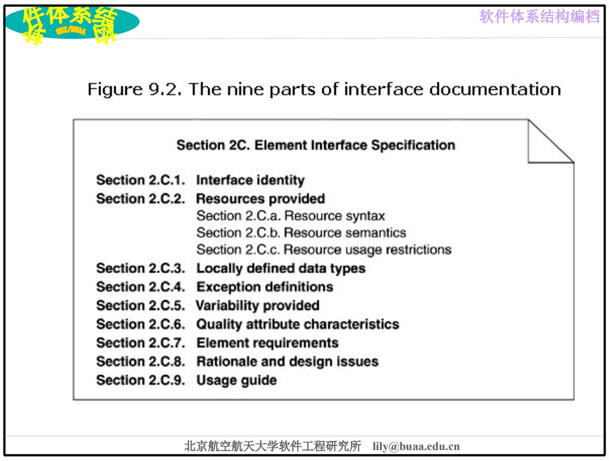
Section 3. Context Diagram

Section 4. Variability Guide

Section 5. Architecture Background
Section 5.A Design rationale
Section 5.B Analysis of results
Section 5.C Assumptions

Section 6. Glossary of Terms

Section 7. Other Information



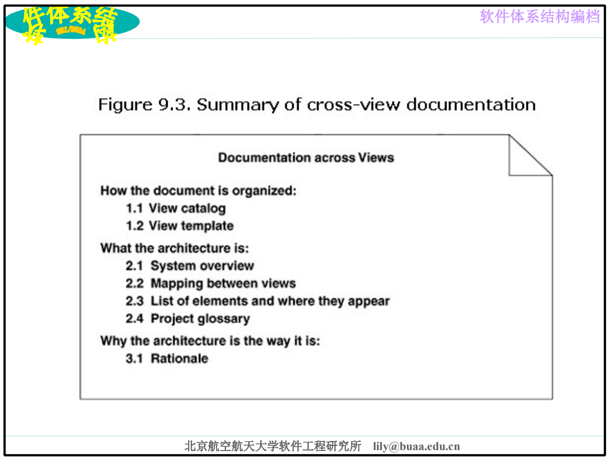
软件体系结构编档

Figure 9.2. The nine parts of interface documentation

Section 2C. Element Interface Specification

Section 2.C.1. Interface identity
Section 2.C.2. Resources provided
Section 2.C.a. Resource syntax
Section 2.C.b. Resource semantics
Section 2.C.c. Resource usage restrictions
Section 2.C.3. Locally defined data types
Section 2.C.4. Exception definitions
Section 2.C.5. Variability provided
Section 2.C.6. Quality attribute characteristics
Section 2.C.7. Element requirements
Section 2.C.8. Rationale and design issues
Section 2.C.9. Usage guide

北京航空航天大学软件工程研究所 lily@buaa.edu.cn



软件体系结构编档

Figure 9.3. Summary of cross-view documentation

Documentation across Views

How the document is organized:
1.1 View catalog
1.2 View template

What the architecture is:
2.1 System overview
2.2 Mapping between views
2.3 List of elements and where they appear
2.4 Project glossary

Why the architecture is the way it is:
3.1 Rationale

北京航空航天大学软件工程研究所 lily@buaa.edu.cn