NAVISION software

Describing UML Compatible Development Processes

«UML»'99, Fort Collins, Colorado, USA

Pavel Hruby
Navision Software a/s
ph@navision.com

Acknowledgement

I would like to thank professor B. Henderson-Sellers of the School of Computing Sciences, University of Technology, Sydney, Australia, for "shepherding" and for his useful suggestions and comments. I do, of course, take full responsibility for any omission or errors.

NAVISION software

Navision Software

Major provider of enterprise business solutions customized to meet the specific needs of mid-size companies all over the world.

Address: Frydenlunds Allé 6
2950 Vedbæk
Denmark

http://www.navision.com

I work at Systems Architecture at Navision Software. I am interested in software architectures and software development methods. Contact info is on the last slide.

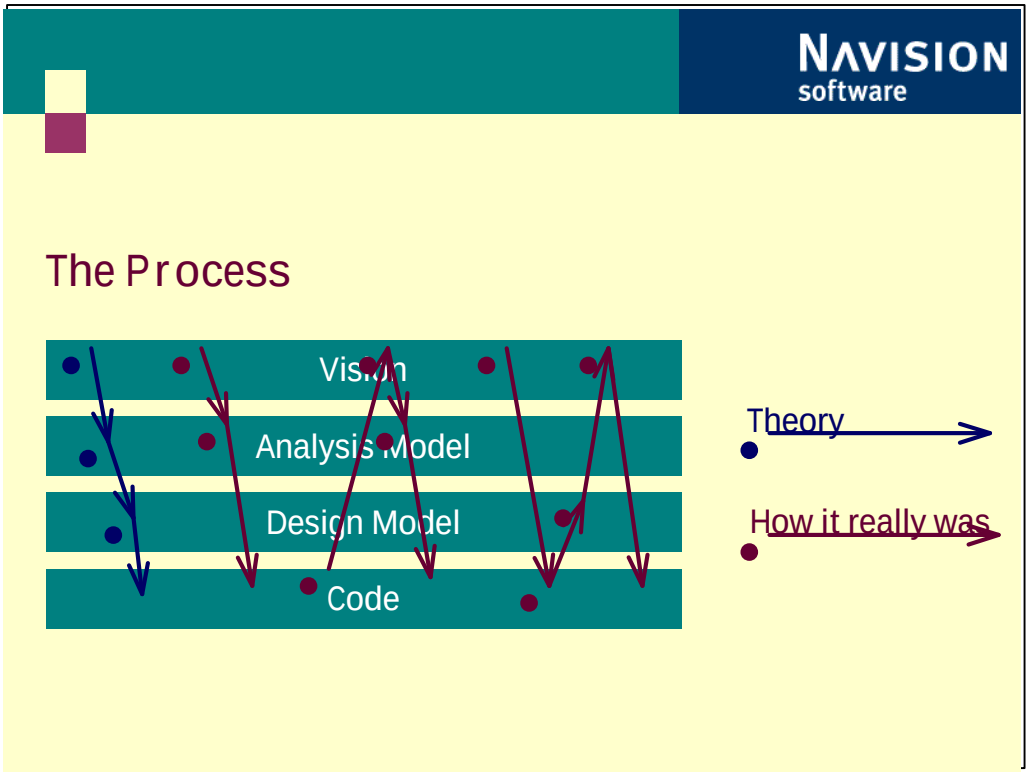
NAVISION software

UML is a Standard Notation, but ...

we also need a process...

- Fusion
- Applying Use Cases
- Microsoft Solutions Framework
- Catalysis
- and own processes and customizations

Managing complexity is a major issue in the specification of software development processes. Currently, over 50 object-oriented methodologies have been published that emphasize various aspects of software design issues and describe software development processes from various perspectives.



See the animation!

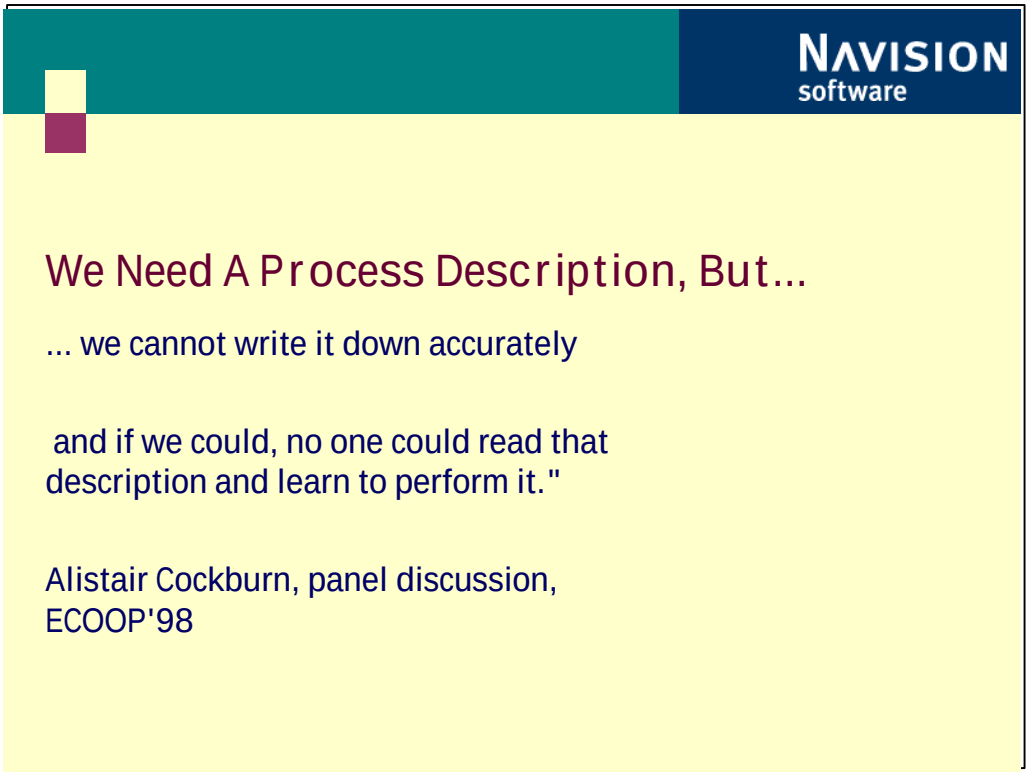
Almost all development processes suggest that you start with a vision or by gathering requirements, then proceed with an analysis model, then design and code. Incremental, spiral and other well-known process models are just variations of the same principle (roughly speaking, phases are shortened and repeated several times).

However, all projects I was “coaching” as a methodologist, were different.

In our first project, we started with a vision and then made analysis. But, after analysis we already had sufficient information to produce code. We found out that we had skipped the design phase.

In our second project we have got a code (prototype) at the very beginning. With the prototype in hand we formulated a product vision, then elaborated on analysis and jumped directly into coding.

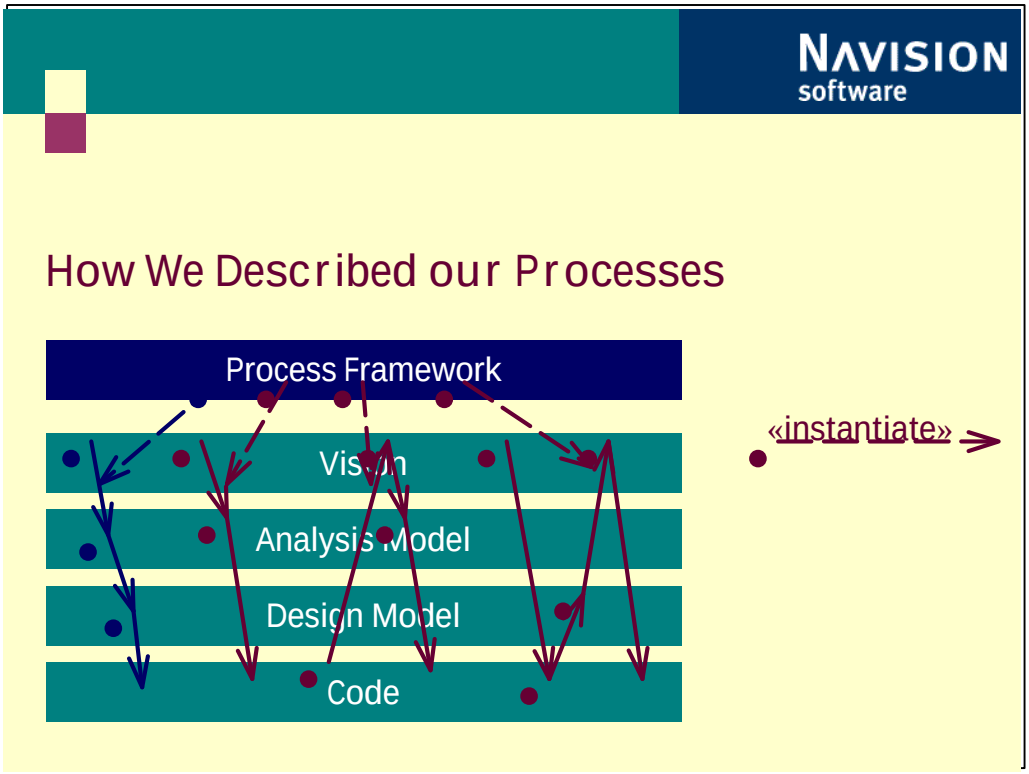
The third project started with a vision, then immediately proceeded with coding, the design of which was documented (and improved) later. After the design phase, we formulated a new vision directly proceeded to code.



The software development process, as actually performed, is so complex that we cannot write it down accurately, and if we could, no one could read that description and learn to perform it." (Alistair Cockburn, panel discussion, ECOOP'98)

In order to be useful, our software process description must capture the real processes (the red lines on the previous slide). The sequence: vision - analysis - design - code (the blue line on the previous slide) is just an illusion [1)], copied from one methodological book to another.

1) I know that *some* customers insist on a detailed specification up-front and force the waterfall process. According to my experience, these are in minority.



See the animation!

The rest of this presentation attempts to solve Alistair Cockburn's problem.

In our approach, instead of writing down a concrete process scenario, we specify a process framework that describes all allowable processes. Such a framework is abstract yet precise. To meet the demands of specific development problems, the framework is instantiated to create specific development processes.

This solution significantly simplifies the description of concrete development processes because the complexity is localized, in the abstract form, in the process framework.

NAVISION software

Have You Been to «UML»'98?

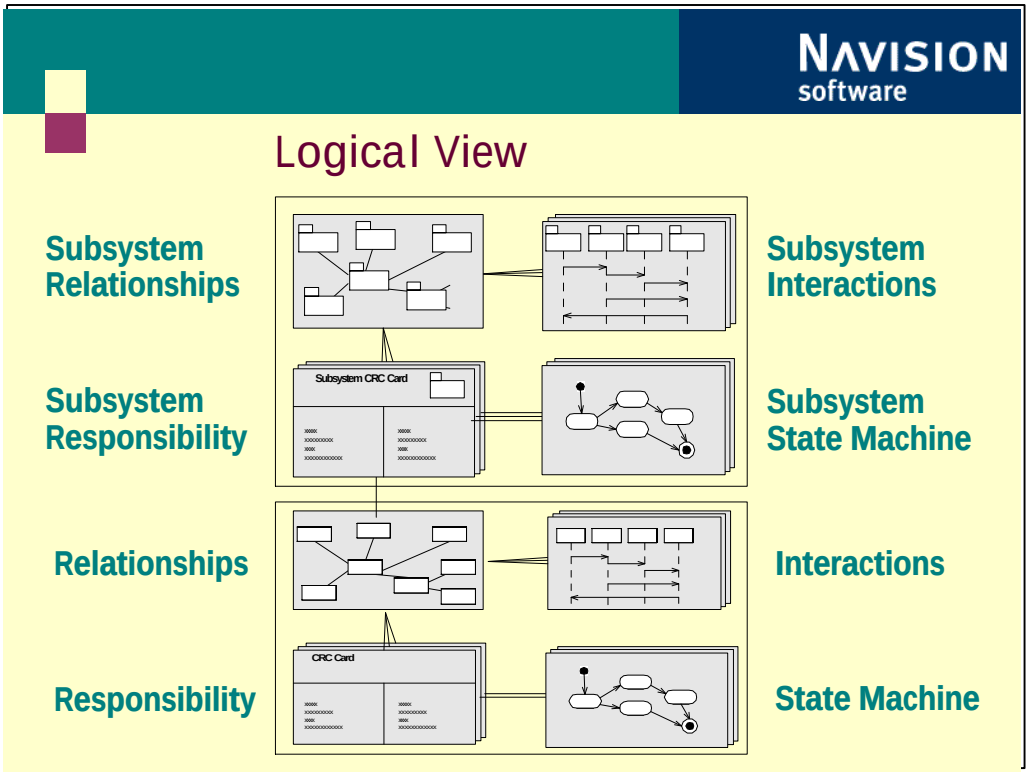
Pattern for Structuring UML Development Artifacts:

- brief summary is on the following slides...

This presentation, "Designing UML Based Design Processes," expands upon my earlier presentation, "Structuring UML Development Artifacts."

The following 15 slides summarize the results of the earlier presentation. However, if you would like a copy of the earlier presentation, please download it from http://www.navision.com/services/methodology/papers/UML_Mulhouse_P.pdf

Otherwise, continue to the slide 22 called "What about development processes?".

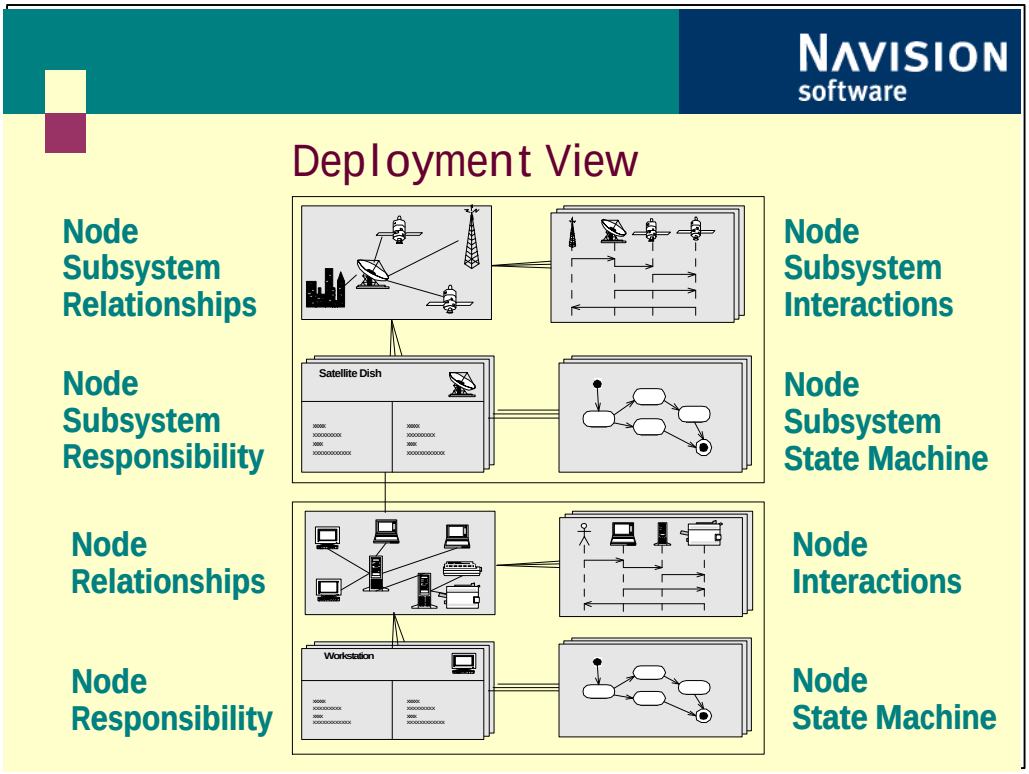


See the animation!

The logical design of a software subsystem can be fully described by the following design artifacts: object relationships, object interactions, responsibility and state machine of each object.

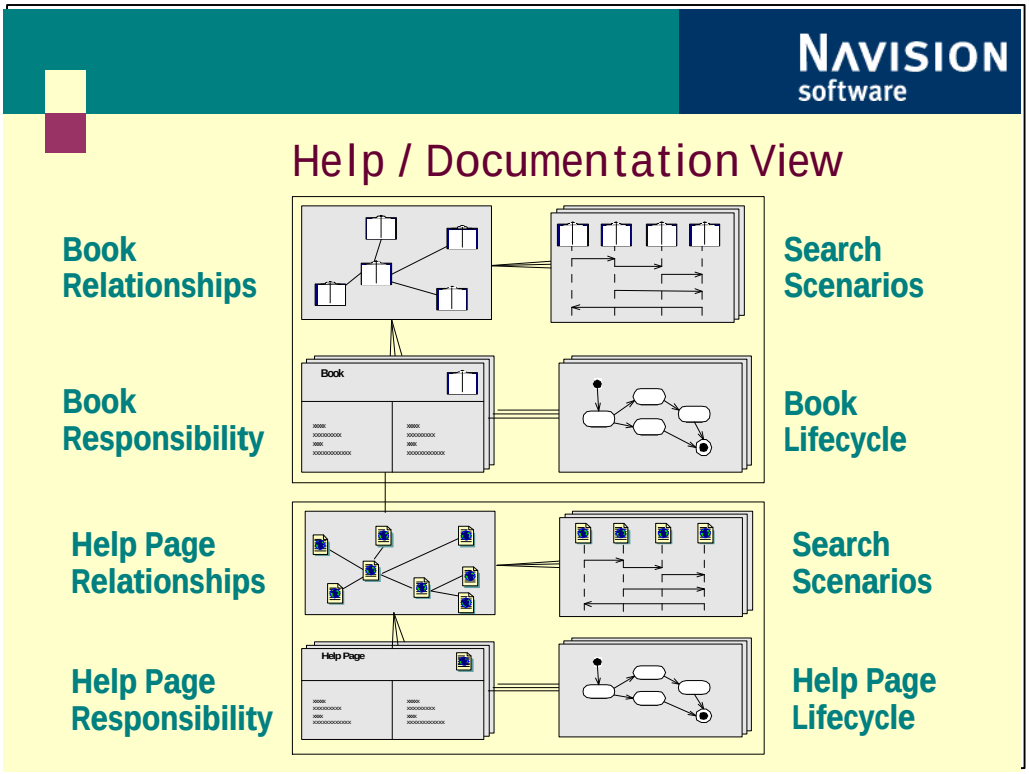
Together, these design artifacts describe the design of a subsystem. However, sometimes it is practical to describe subsystems at a higher level of abstraction. It is often sufficient to specify subsystem responsibility, abstract attributes that can be read and set, subsystem interfaces and abstract state machines.

Analogously, at the subsystem level of granularity we specify subsystem responsibility, relationships between subsystems, interactions between subsystems and subsystem state machines.



The physical design of a software system can be described by relationships between nodes, interactions between nodes, responsibility (purpose) of each node and node state machines (for example, a printer can be offline, online, busy and ready).

For large systems, it is sometimes necessary to describe physical subsystems (their responsibilities, interfaces, abstract attributes than can be read and set), relationships between physical subsystems, interactions between physical subsystems and abstract state machines of physical subsystems.

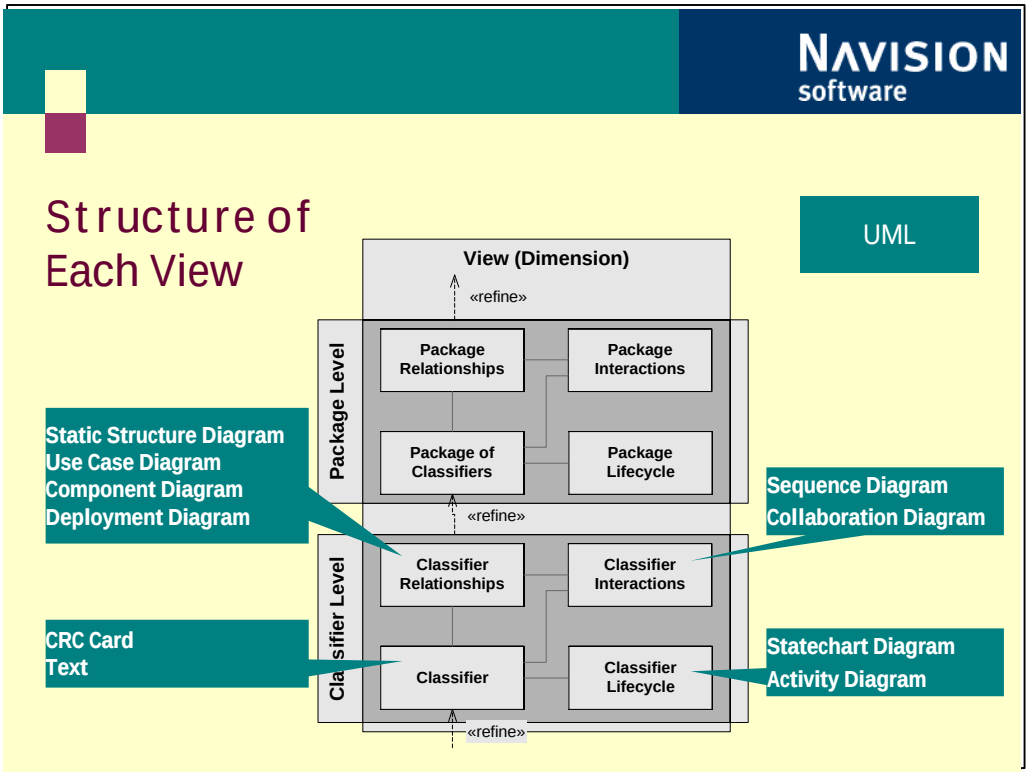


If an on-line help consists of a large number of help pages, simply providing a description of the help pages is sometimes not sufficient to get an overview.

To get an overview of an online help program, a designer specifies the relationships and links (navigability) between the help pages and typical searching scenarios. Some help pages have behavior; which can be specified as the help page lifecycle.

A designer defines the on line help subsystems, let's call them books, and describes the relationships between books, and the search scenarios that span different books.

The design of Internet sites can be documented in a similar manner.



Do you see the pattern on the previous slides?

Design artifacts specify the software system from various viewpoints and at various levels of granularity.

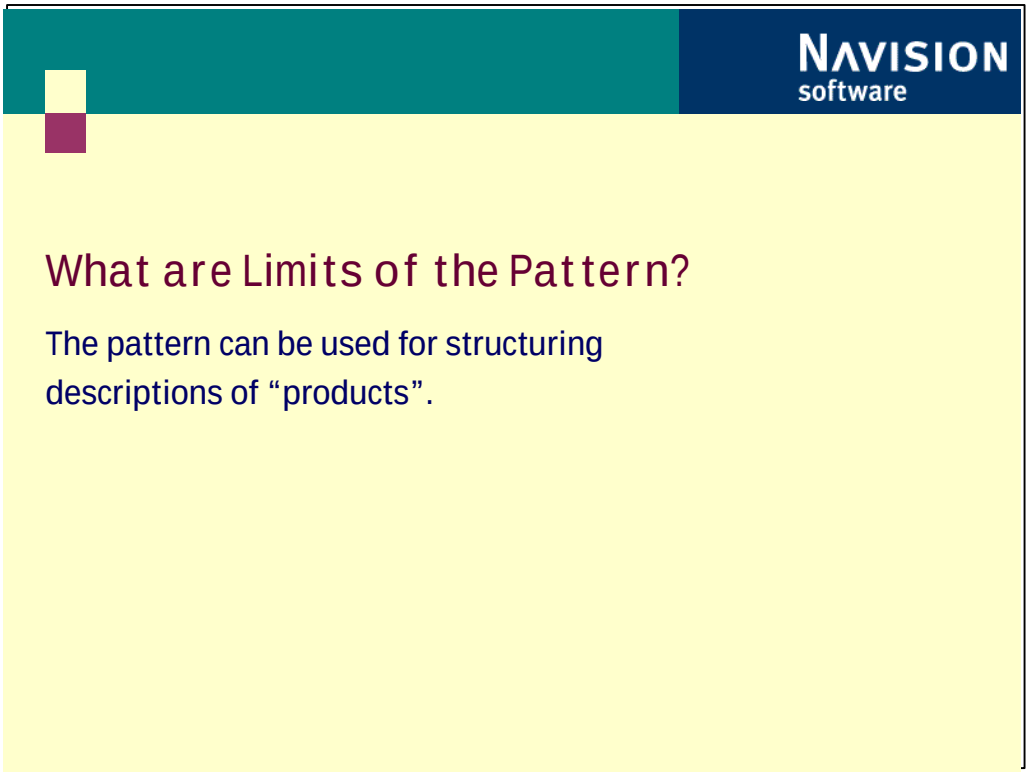
At each level of granularity and in each view, the software product can be described by four design artifacts:

- •static relationships between classifiers
- •dynamic interactions between classifiers
- •classifier responsibilities
- •classifier lifecycles

Each of these artifacts can be represented by UML diagrams or by text..

UML classifiers [1)] are class, interface, use case, node, subsystem, component and datatype.

1) At the time of writing this presentation (October 1999), collaboration is not a UML classifier (in UML 1.3). I think it should be.



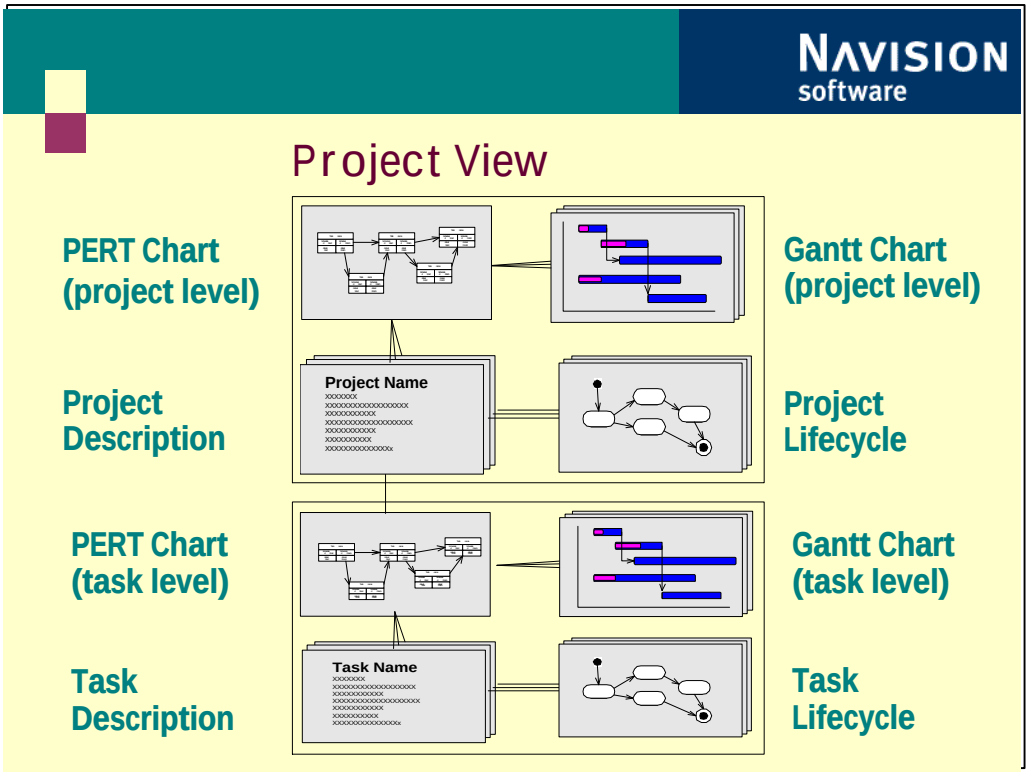
What are the limits of the pattern?

Can we structure the whole universe from various viewpoints and levels of granularity and describe each view and granularity level in four artifacts?

Well, I currently understand that the pattern can describe things that we, from our point of view, want to consider as a *product*.

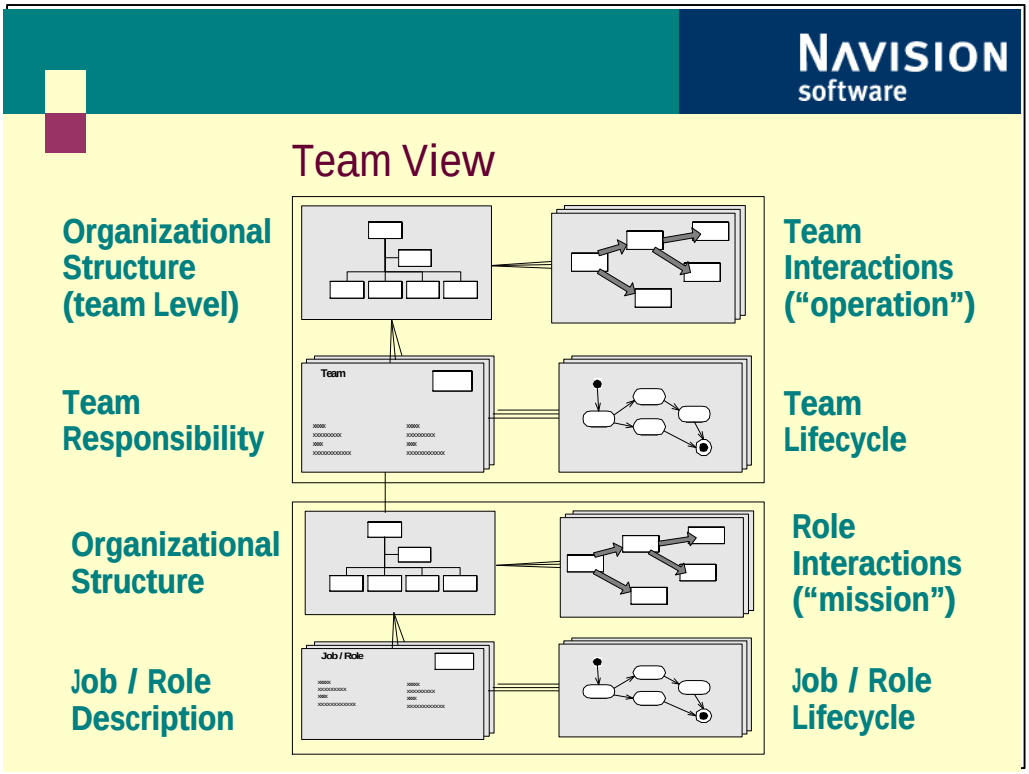
Every project generates a number of artifacts that we do not want to consider as a description of a software product. Examples of such artifacts are a glossary, minutes of meetings, reviews, comments and notes. These artifacts do not describe a *product*, and therefore they cannot be structured using the pattern. However, these artifacts can be related to the artifacts that are structured in the pattern. For example, the glossary is related to the artifact *system and* the minutes of meetings are related to the artifact *project*.

The previous slides discussed software development products, the next two slides discuss software management products.



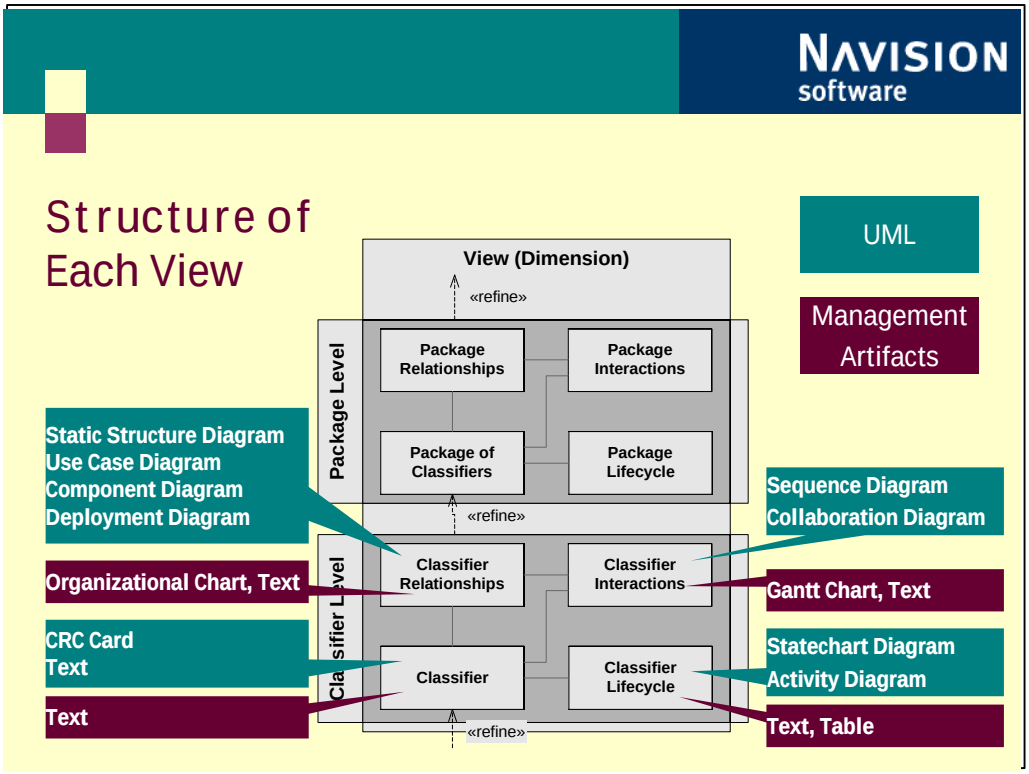
The pattern can be used to structure two kinds of management product: teams and projects.

This slide shows the *project* viewpoint. The artifacts *project* and *task* are project and task descriptions, typically represented by text. The artifact *project relationships* and *task relationships* specify static relationships between projects and tasks. These artifacts can be represented by a PERT chart. The PERT chart shows the task dependencies, which are the most important static relationships between tasks. The artifact *task interactions* specifies a project scenario in terms of starting and finishing tasks. Accordingly, the artifact *project interactions* specifies the project scenario in terms of starting and finishing projects. These artifacts are typically represented by Gantt charts, but they can also be represented by UML sequence diagrams. Gantt charts show the task constructors, which are the most important messages between tasks. UML sequence diagrams can show any kind of messages between tasks.



This slide shows the *team* viewpoint.

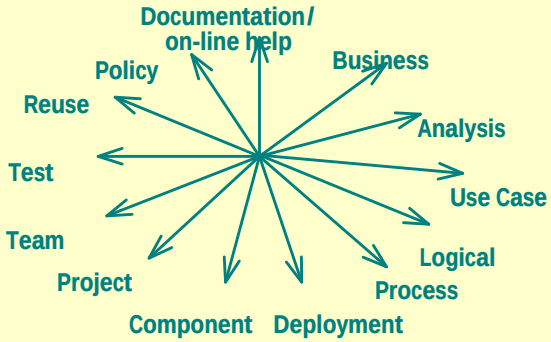
The artifacts *team relationships* and *role relationships* specify the organizational structure at two levels of granularity. The artifact *team* specifies the responsibility of the team and the artifact *role* specifies the role of the team member. Examples of roles are: developer, program manager, product manager, user education and logistics. The artifacts *team interactions* and *role interactions* specify scenarios - interactions between teams and team members - which are responses to various events.



Development artifacts are represented in UML, management artifact are represented by specific management diagrams and text.

NAVISION software

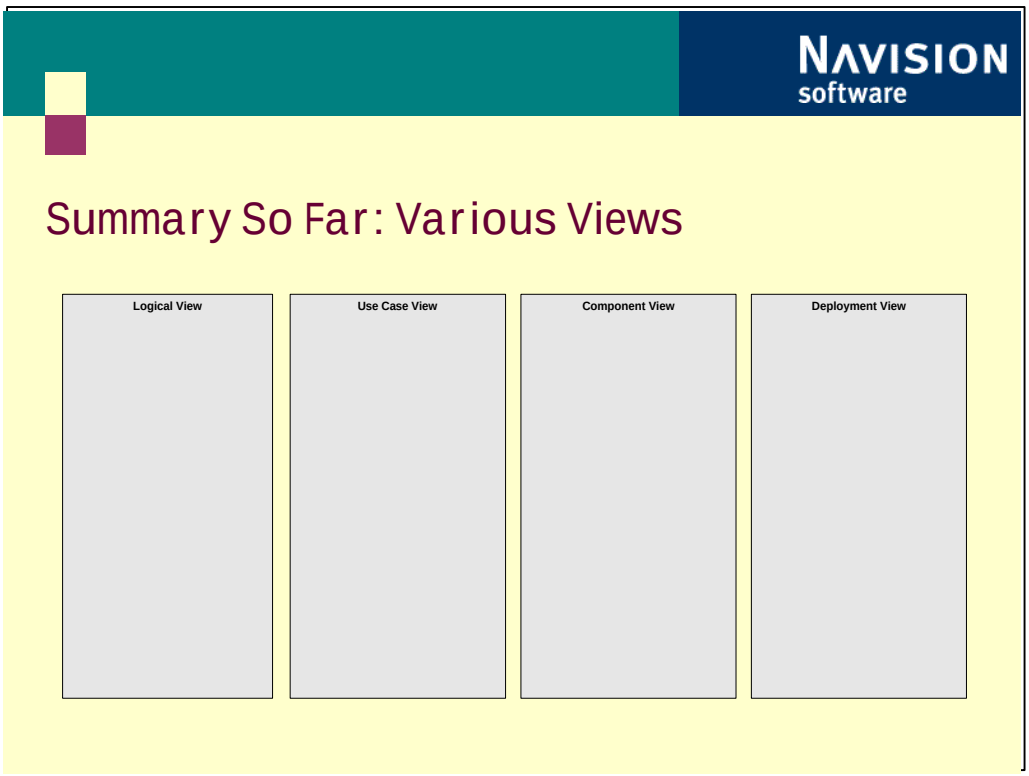
Orthogonal Concern Space



The pattern can be used in various views.

We have talked about the UML views, the project view and the team view. The use case view is discussed in the paper “Structuring UML Development Artifacts”.

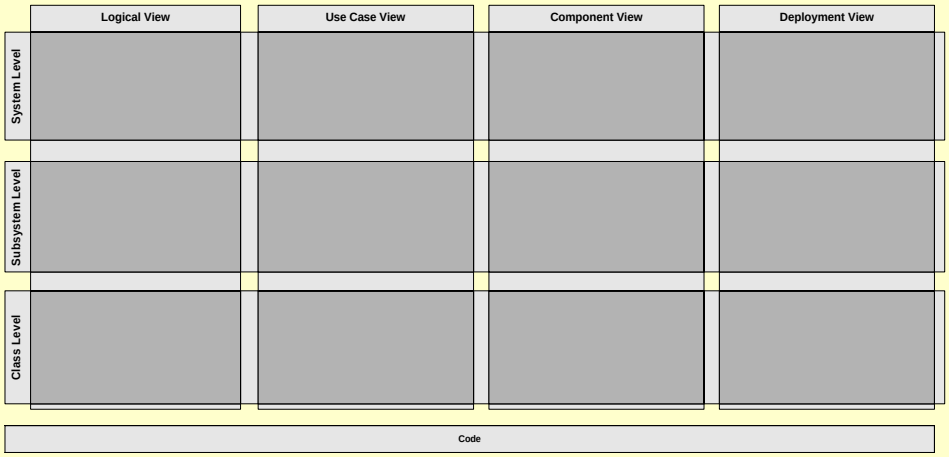
The benefit of this pattern is that it allows for adding (and removing) views in a consistent manner, in cases where developers want to specify something unusual or unexpected.



Examples of views are: the logical view, the use case view, the component view and the deployment view.

NAVISION software

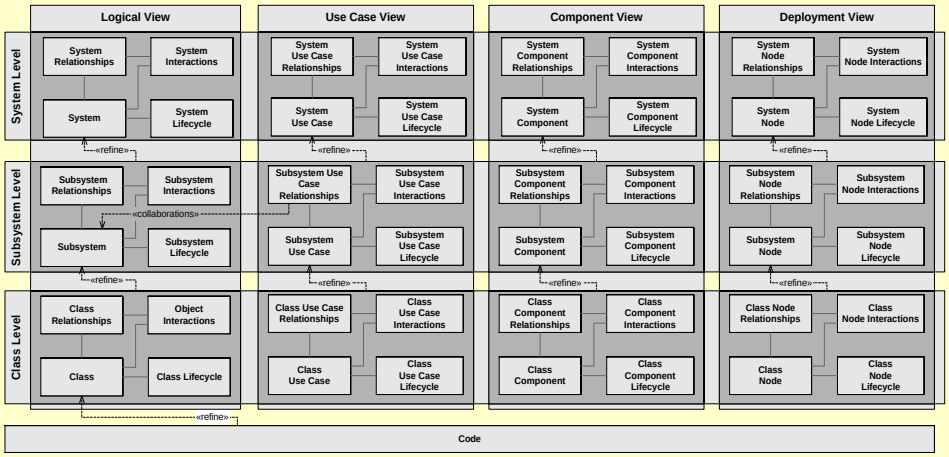
Summary So Far: Various Levels of Granularity



Examples of levels of granularity are: the system level, the subsystem level and the class level.

NAVISION software

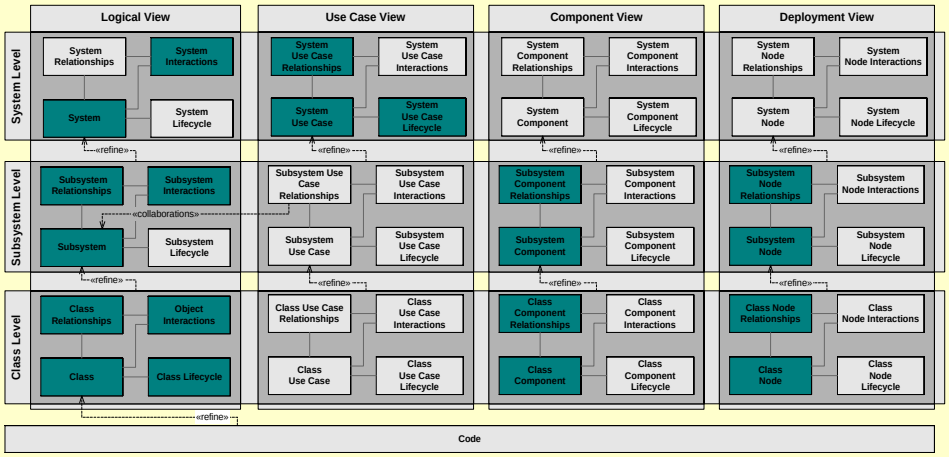
Summary So Far: Various Artifacts



At each level of abstraction and in each view, the software and management product can be described by four design artifacts: static relationships between classifiers, dynamic interactions between classifiers, classifier responsibilities and classifier lifecycles.

NAVISION software

Example: Rational Unified Process

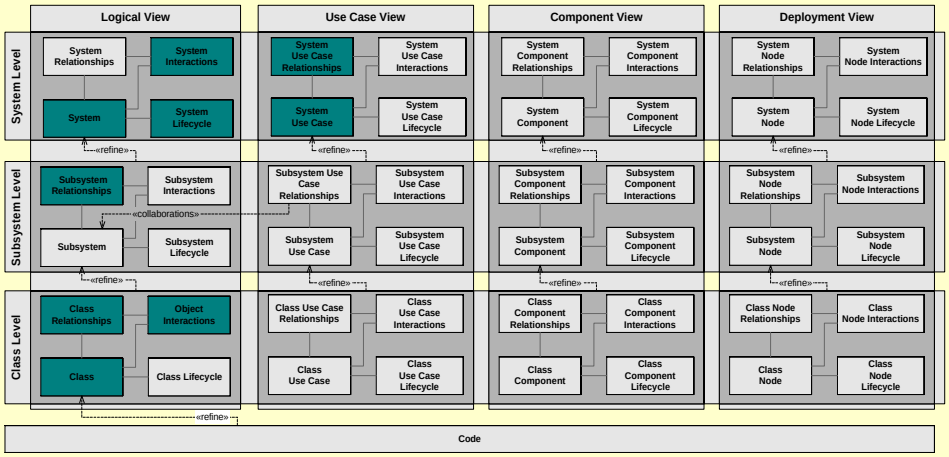


Processes of different development methods create different subsets of the software development artifacts identified in the previous section, because different methods focus on different aspects of software development.

The deliverables of Rational Unified Process are structured on use case, logical, deployment, implementation and process views, and tier, subsystem and class levels. Deployment and implementation views contain only component and node relationships and responsibilities. All interaction models are considered as a specific view called *process view*. The Rational Unified Process produces only use cases at the system level; the method does not produce any state models with the exception of the use case activity model and the class state model.

NAVISION software

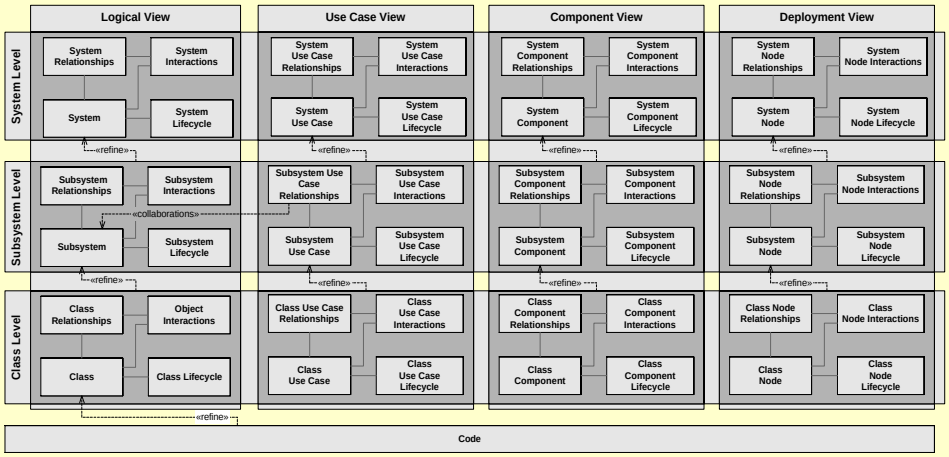
Example: Fusion Method (with Use Cases)



The Fusion method focuses on deliverables in the logical view at system, subsystem and class levels. At the system level, Fusion delivers the system model (*object model* in Fusion), the system interaction model (*scenario* in Fusion), the system (*operation model* in Fusion) and the system state model (*lifecycle model* in Fusion). At the subsystem level, Fusion delivers only the subsystem model (*system object model* in Fusion). At the class level, Fusion delivers the class model (*visibility graphs* and *inheritance graphs*), the object interaction model (*object interaction graphs*) and the class (*class descriptions* in Fusion). Fusion does not produce any state models except for the system state model (*lifecycle model* in Fusion). The new Fusion Engineering process (also known as Team Fusion) also produces use case model and use cases.

NAVISION software

What about Development Process?

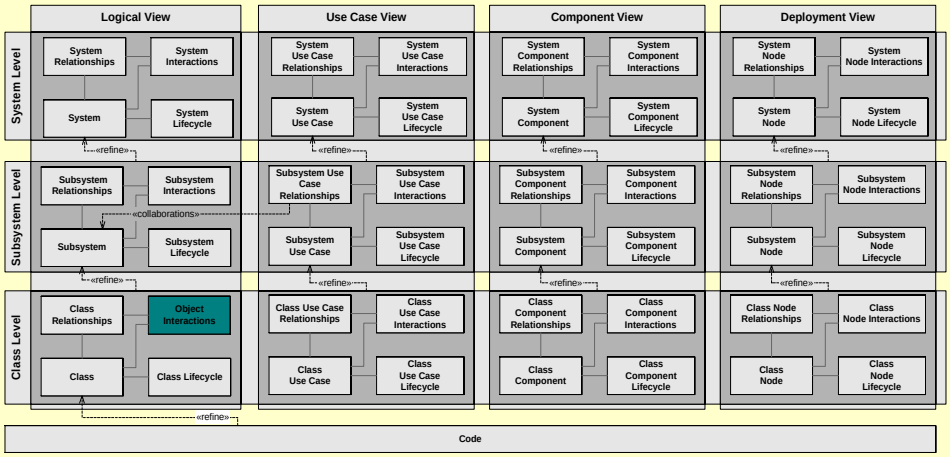


The previous section described the static structure of software development and management artifacts.

Software development artifacts can be created and completed in various orders depending on the features of various design methods.

NAVISION software

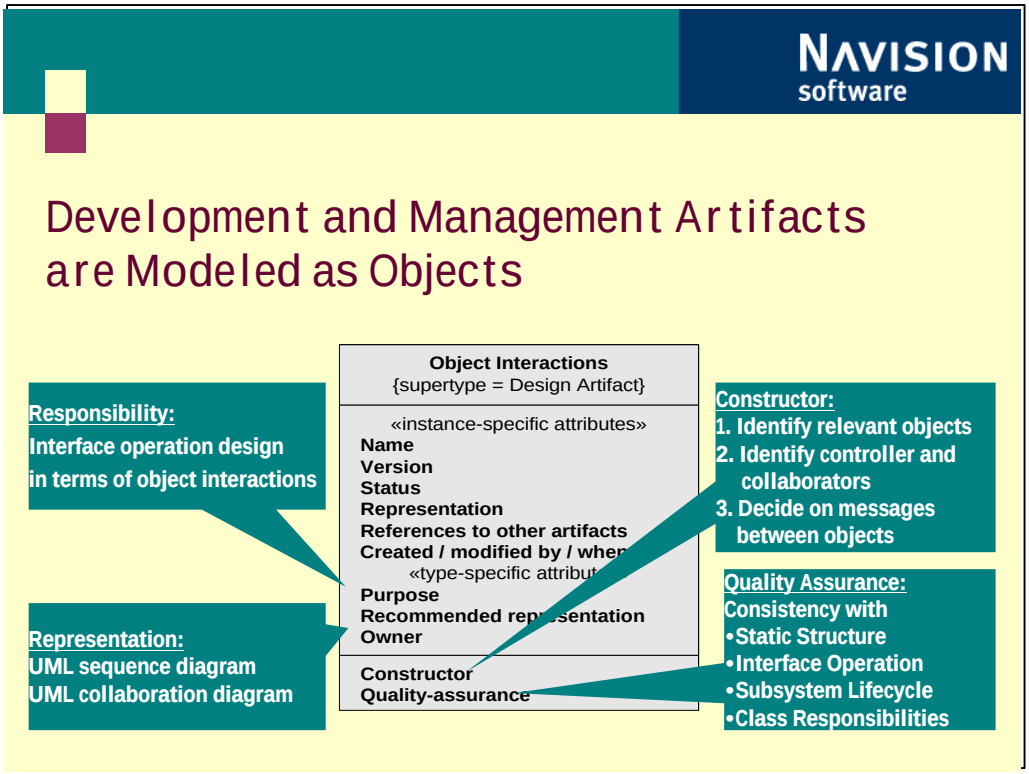
Development and Management Artifacts are Modeled as Objects



During software development, evolution is represented as collaborations between software development artifacts, management artifacts and members of a development team.

Software development and management artifacts produced during a software development process are considered objects possessing various methods and attributes..

As an example, let’s have a look at the artifact “object interactions”.



Artifact types have two kinds of methods:

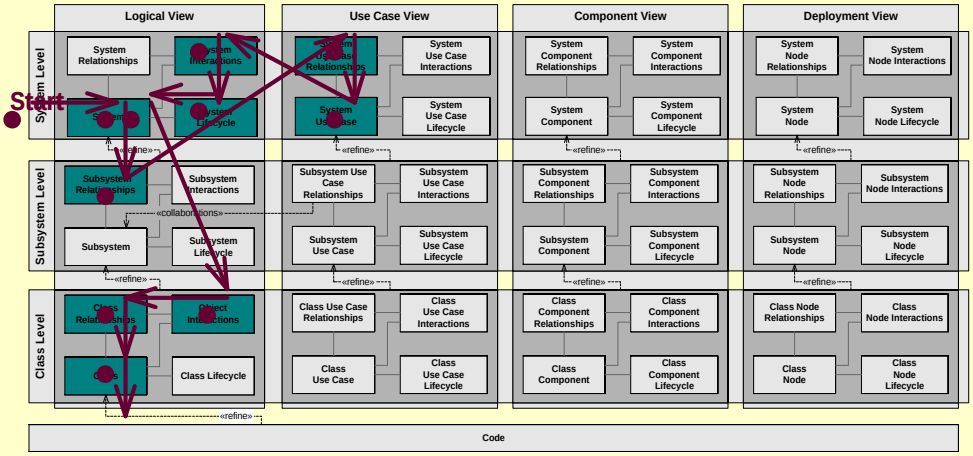
- *Constructors*, which are methods describing how to create an artifact.
- *Quality-assurance methods*, such as completeness and consistency checks.

Artifacts have instance-specific *attributes*: name; version; representation, which typically contains a diagram, a table or a text; status, such as draft, completed, tested; references to other software development and management artifacts; and attributes such as who created and modified the artifact and when.

In addition, artifact types have type-specific attributes: the purpose, the recommended representation and the owner of the artifact type. Artifacts may have other additional attributes and methods than those mentioned above.

NAVISION software

Process of Fusion Method (with Use Cases)

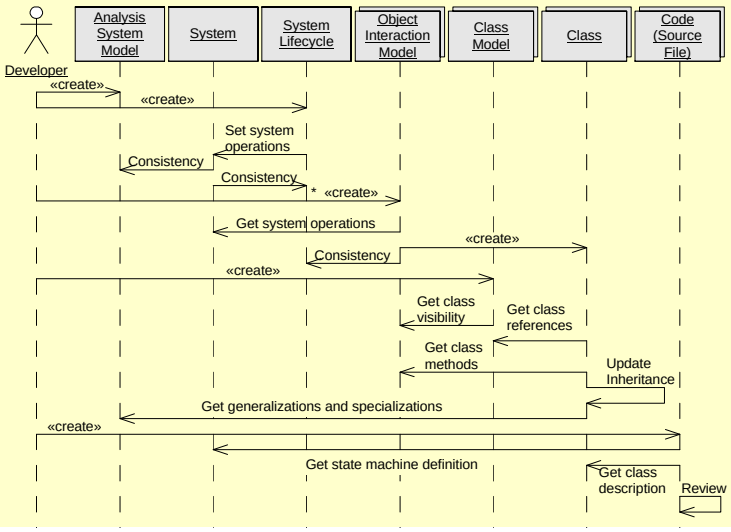


See the animation!

How would you show this scenario in UML?

NAVISION software

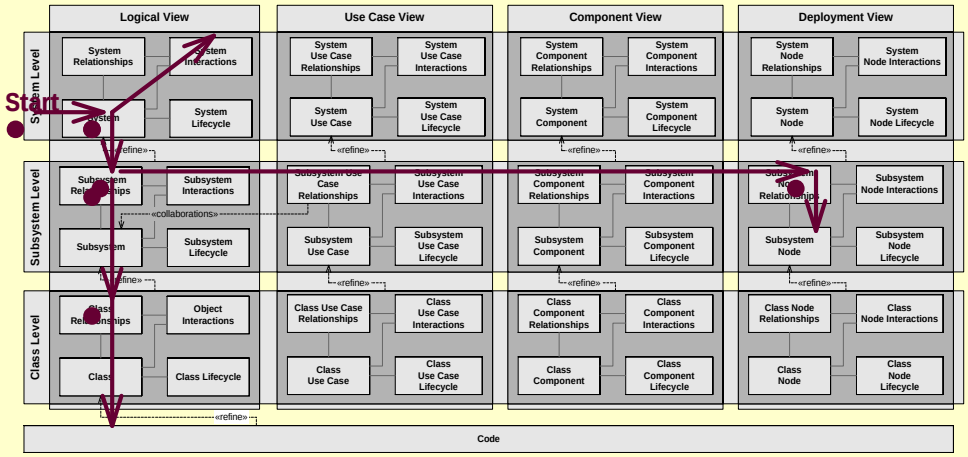
The Fusion Method Process described in UML



The Fusion method (without use cases), now as a UML sequence diagram.

NAVISION software

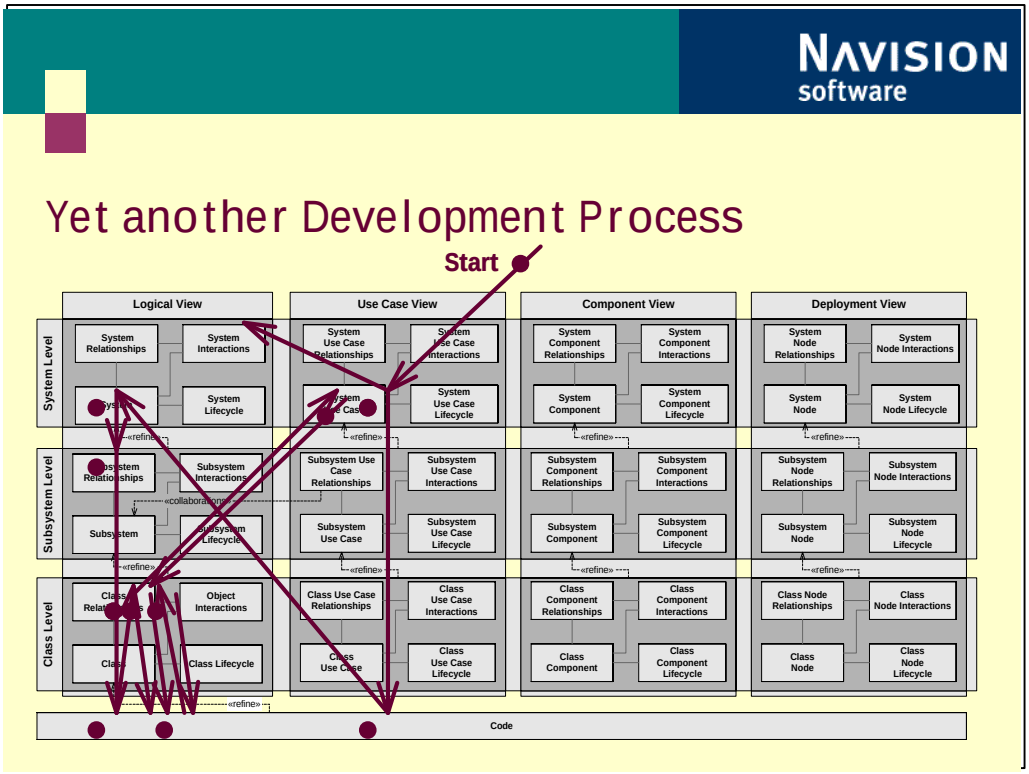
Another Development Process



We can start with the product vision and sketch the architecture. Then we determine the hardware structure (the node relationships and responsibilities).

Then we specify a couple of usage scenarios (the system interactions).

We continue with class relationships for the specified subsystems and finish with code.



But, but,….

we can also start with use cases (text) and usage scenarios (system interactions). And then make a prototype (code). Next, we make a product vision, determine the architecture and make the first iteration (code). We redesign the product (class relationships), update the code, update use cases and repeat these three steps a couple of times.

(sounds almost like eXtreme Programming, doesn't it?

NAVISION software

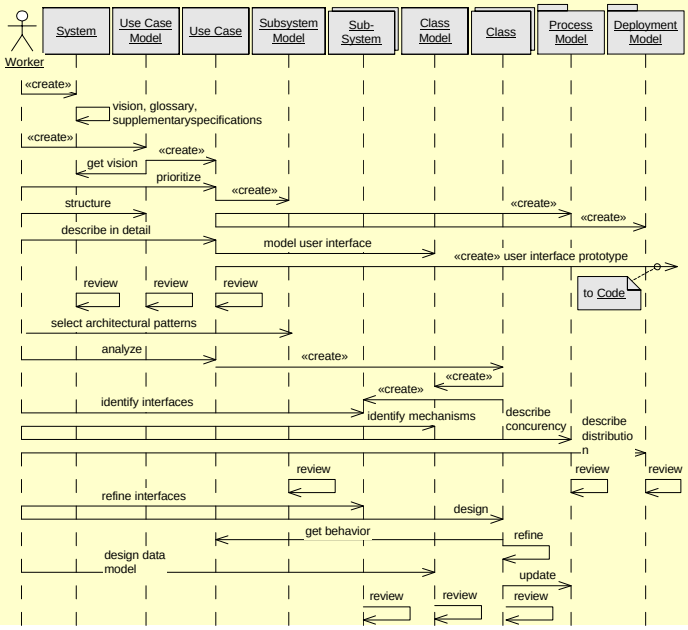
Rational Unified Process Workflows

- Business Modeling
- Requirements
- Analysis and design
- Test
- Deployment
- Project Management
- Configuration and change management
- Environment

We have many other development processes.
Rational Unified Process calls them *workflows*.

NAVISION software

Analysis and Design Workflow of RUP Described in UML



This is an example of a workflow in the Rational Unified Process, called *analysis and design workflow*.

Notice that the UML sequence diagram has the ability to specify which design artifacts are created and modified.

NAVISION software

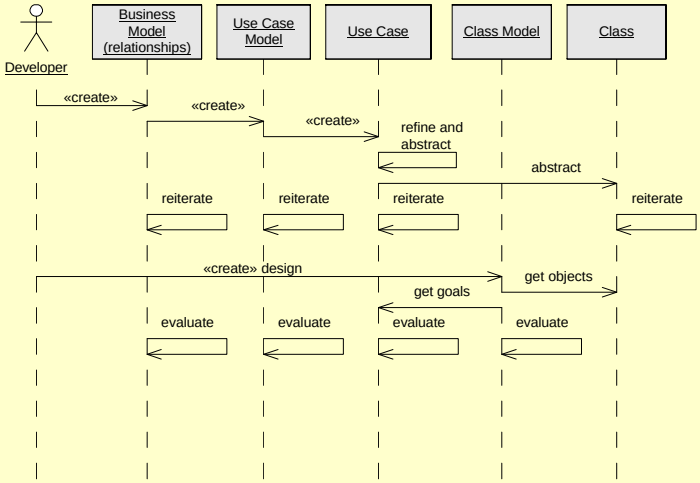
Catalysis Process Patterns

- Object Development from Scratch
- Short-Cycle Development
- Reengineering
- Business Process Improvement
- Make a Business Model
- . . .

We have many other development processes.
The Catalysis method calls them *process patterns*.

NAVISION software

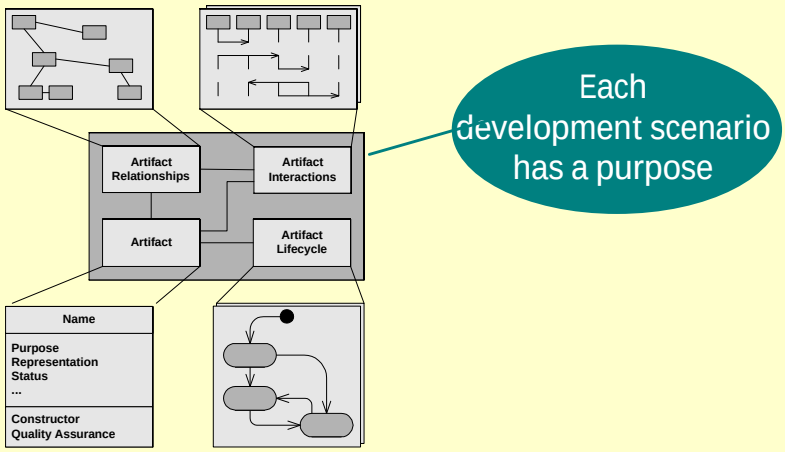
Catalysis Business Process Improvement Described in UML



This is an example of the process pattern in the Catalysis method called *business process improvement*.

NAVISION software

Specifying Software Development Processes



How do we specify our process framework?

Well, we can use the same pattern as we described earlier:

The artifact called *artifact relationships* specifies the static relationships between software development and management artifacts.

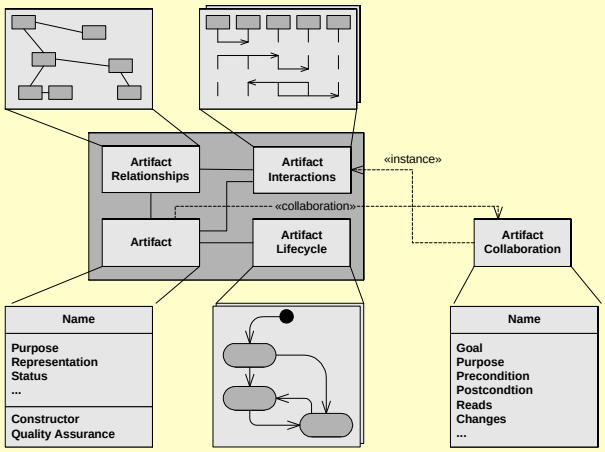
The artifact called *artifact interactions* specifies the development scenario.

The artifact called *artifact* specifies the purpose, constructor, quality assurance, and specific attributes of software development and management artifacts.

The artifact called *artifact lifecycle* specifies the artifact states and the events that change the artifact state, such as creation, completion and approval.

NAVISION software

Specifying Software Development Processes

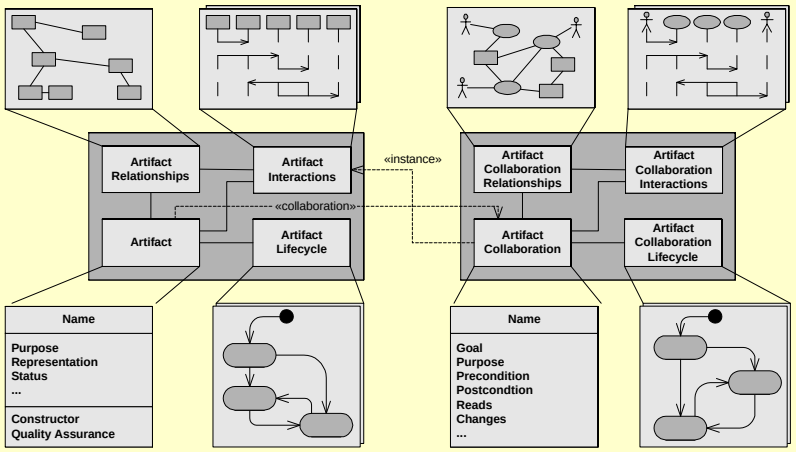


As mentioned earlier, we consider each development process as a collaboration between development and management artifacts and users of the method.

Each development scenario (the artifact collaboration) has a specific purpose. This purpose can be described as an abstract type, similar to the way we describe use cases.

NAVISION software

Specifying Software Development Processes

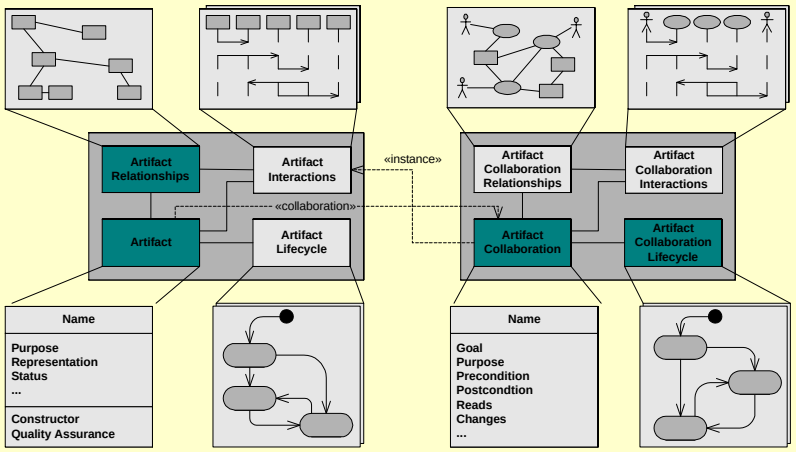


The previous slide depicted the development processes as collaborations between development and management artifacts.

There are also artifacts called artifact collaborations relationships and artifact collaboration interactions and artifact collaboration lifecycle.

NAVISION software

Specification of the Rational Unified Process



How is the Rational Unified Process specified?

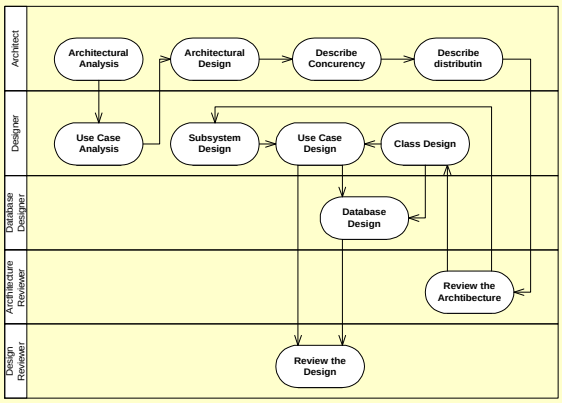
The Rational Unified Process (version 5.0) specifies development and management artifacts - and the relationships between them. It also specifies artifact collaborations (called workflow descriptions) and lifecycles of artifact collaborations (activity diagrams associated with each workflow).

The Rational Unified Process (version 5.0) does not specify the lifecycles of the development and management artifacts, interactions between artifacts, relationships between workflows (artifact collaborations), or sequences of workflows (artifact collaboration interactions).

NAVISION software

Artifact Collaboration Lifecycle

- Defined in terms of activity states
- Does not specify development and management artifacts

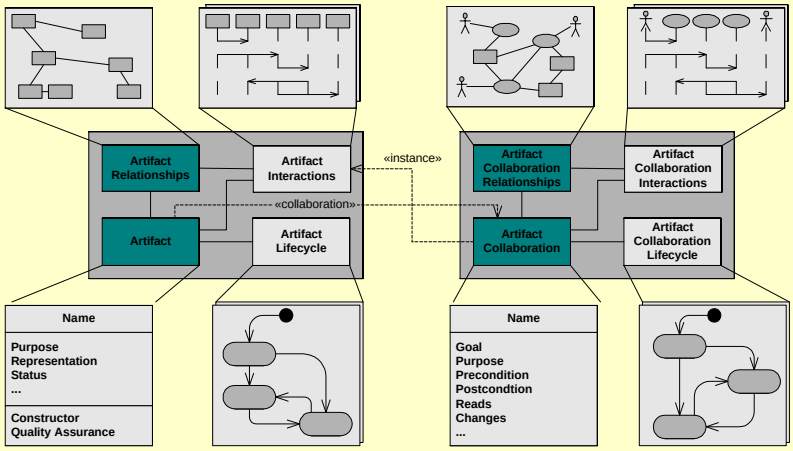


An example of the artifact *collaboration lifecycle* is the analysis and design workflow of the Rational Unified Process.

You will notice that the collaboration lifecycle does not show the design artifacts that are modified during the workflow. If the Rational Unified Process would describe artifact interactions (the sequence diagram shown earlier) instead of the collaboration lifecycle (the activity diagram shown in this slide), the description could visualize the design artifacts that are accessed and modified during the workflow.

NAVISION software

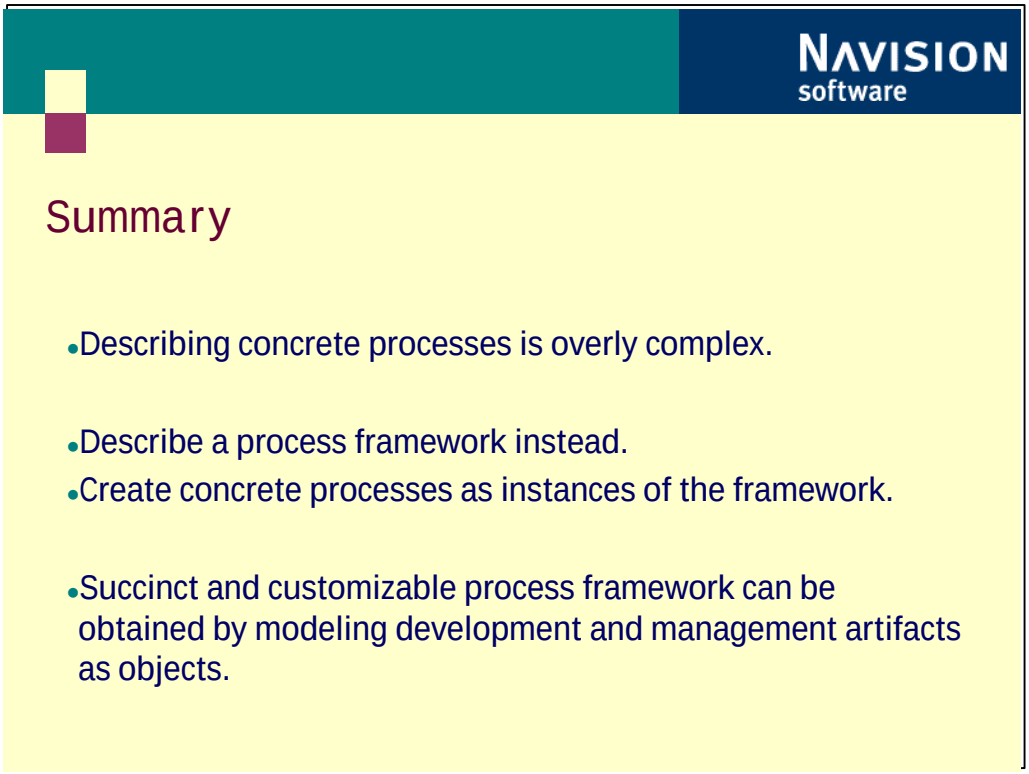
Specification of the Catalysis Method



How is the Catalysis method specified?

The Catalysis method specifies development and management artifacts, and relationships between them. It also specifies artifact collaborations (called process patterns) and informally specifies relationships between them (that is, Catalysis specifies that certain collaborations are included in others).

Catalysis does not specify artifact lifecycles, artifact interactions (instances of process patterns), collaboration lifecycles (activities within process patterns), or collaboration interactions (sequences of process pattern instances).



This paper discussed the product-focused object-oriented framework for the specification of software development processes. The software development and management artifacts are modeled as objects with constructors and quality-assurance methods, together with a number of specific attributes.

The object-oriented specification of a development process is simpler and more consistent than a traditional specification based on tasks and deliverables.

Various specific software development processes were illustrated to show instances of the process framework described in this presentation.

NAVISION software

More Information

Pavel Hruby
e-mail: ph@navision.com
Internet: www.navision.com
(Click “Corporate Info” and “Methodology”)

Please let me know if you have other solutions or comments.

Whether you have downloaded this presentation from our web site, or obtained it from some other source, I am interested in hearing your opinion or alternative view.