Specialization in DITA

Erik Hennum, Michael Priestley, Dave A. Schell
IBM

Introduction to DITA

- IBM's XML architecture for topic - oriented information
  - *Mostly* Business as Usual - some important exceptions
- Features specialization
  - From a base "topic" we provide "task, reference, and concept"
  - We also allow others to create specializations
- Features vocabulary domains
  - From a generic topic we provide "highlighting, software, programming, and GUI" vocabularies that are common
- Various reuse mechanisms: content, design, code
- Base modules, DTDs, transforms available at developerWorks

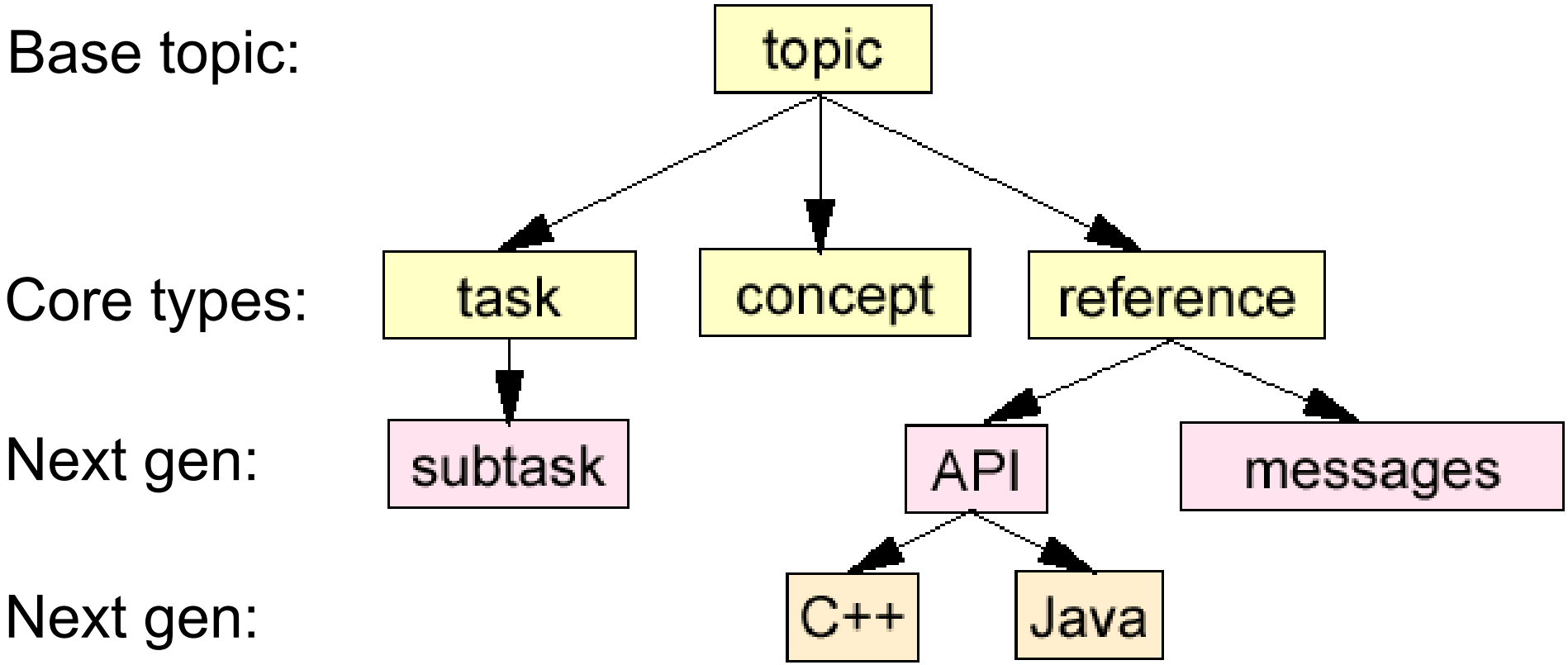
Extensibility

- Specialization
  - Information types
  - Domains
  - Code
- Customization of code
- Integration of design

Specialization

- Making semantic distinctions
- Adding to a hierarchy of distinctions
- Mapping new, more specific elements to existing, more abstract elements
- Reusing existing elements whenever possible
- Markup separated into modules based on information types and domains

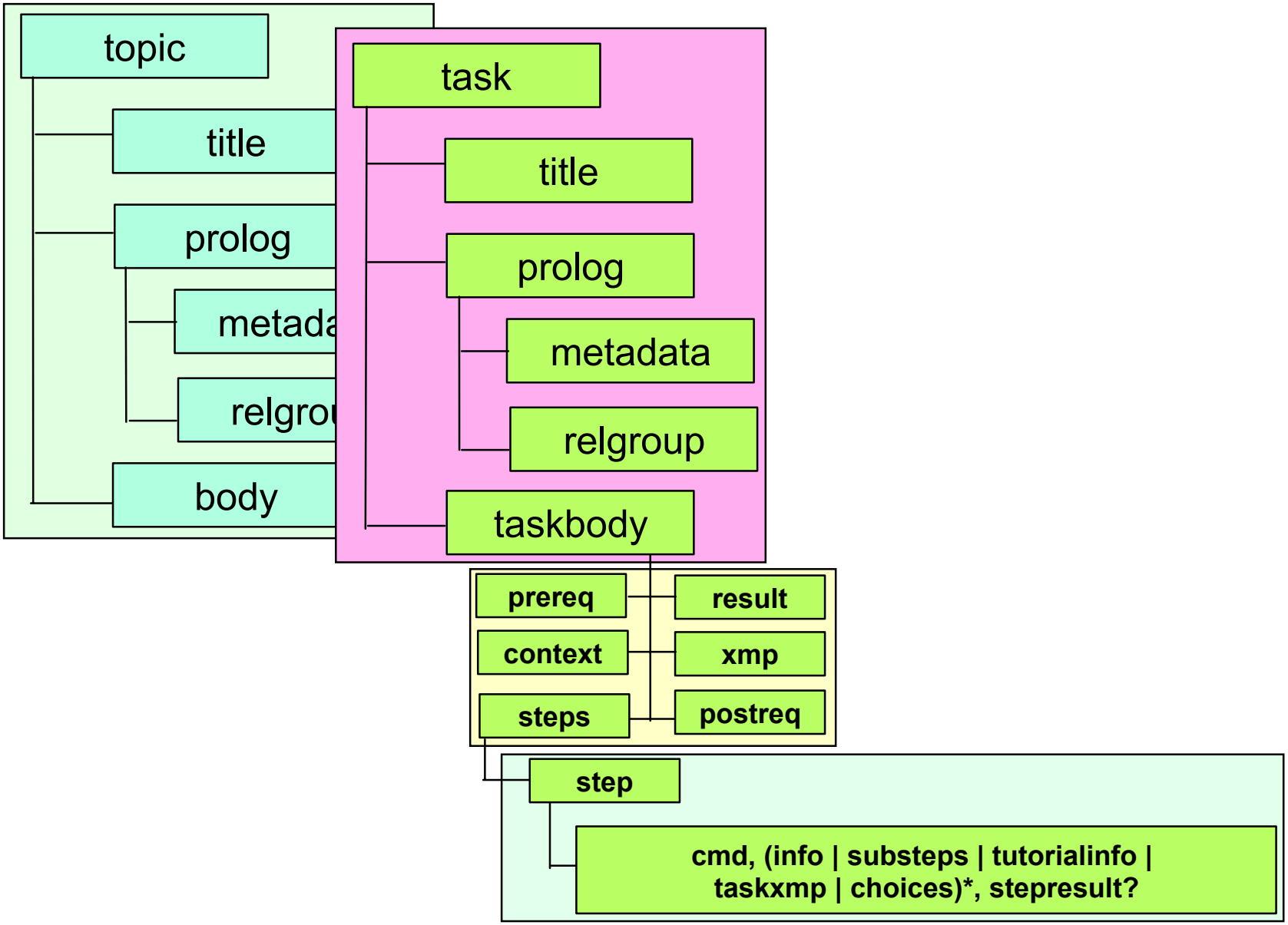
As part of a hierarchy



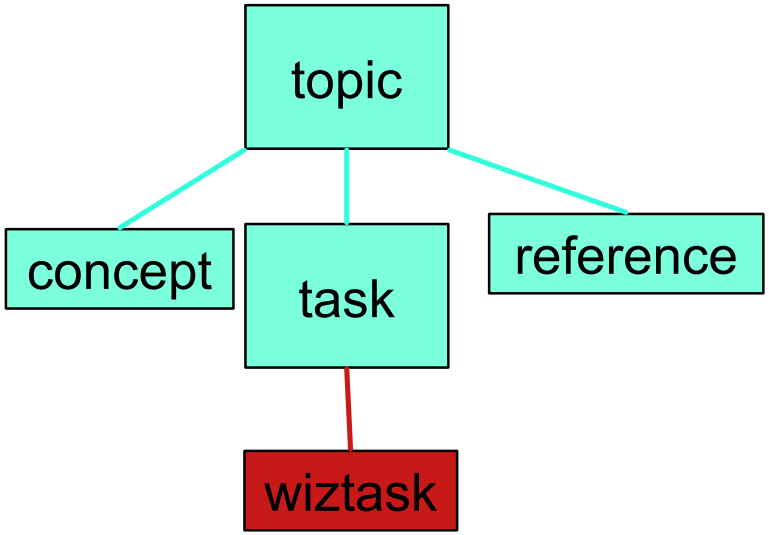
Specializing information types

- Define kinds of topics (concept, task, reference, etc.)
- Start from the top (the container for the whole topic) and work down
- Have to specialize containers to allow in new markup
- Can work all the way down to individual keywords or phrases

Specializing task from topic



As part of a hierarchy



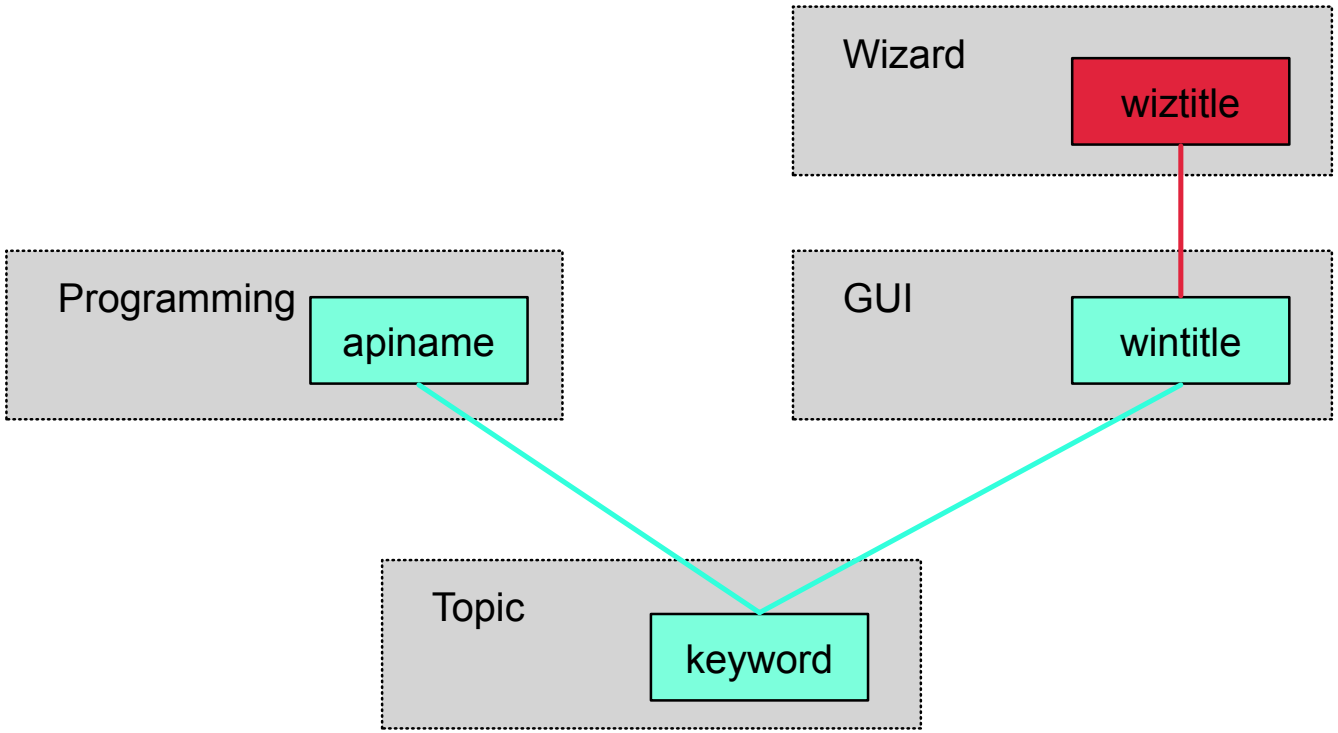
Using information types

- Create a shell DTD
- Embed the modules for each information type, and its ancestors, starting at the top of the hierarchy
- Redefine entities to determine which information type can nest with which
- Allow one topic per file for simplest reuse
- EG: for wiztasks, would embed topic.mod, task.mod, wiztask.mod

Specializing domains

- Define a category of related elements
- Part of a domain (common subject area)
- Found across information types
- For example, programming domain (APIs, syntax diagrams); UI domain (wintitle, uicontrol)
- Can still define complex structures (like syntaxdiagram)
- But can start from any level in a topic

As part of a hierarchy



Using domains with information types

- Create a shell DTD
- Include declarations for domain entities
- Override definitions for content models
- Override definitions for domains attributes
- Include modules for information types
- Include modules for domains

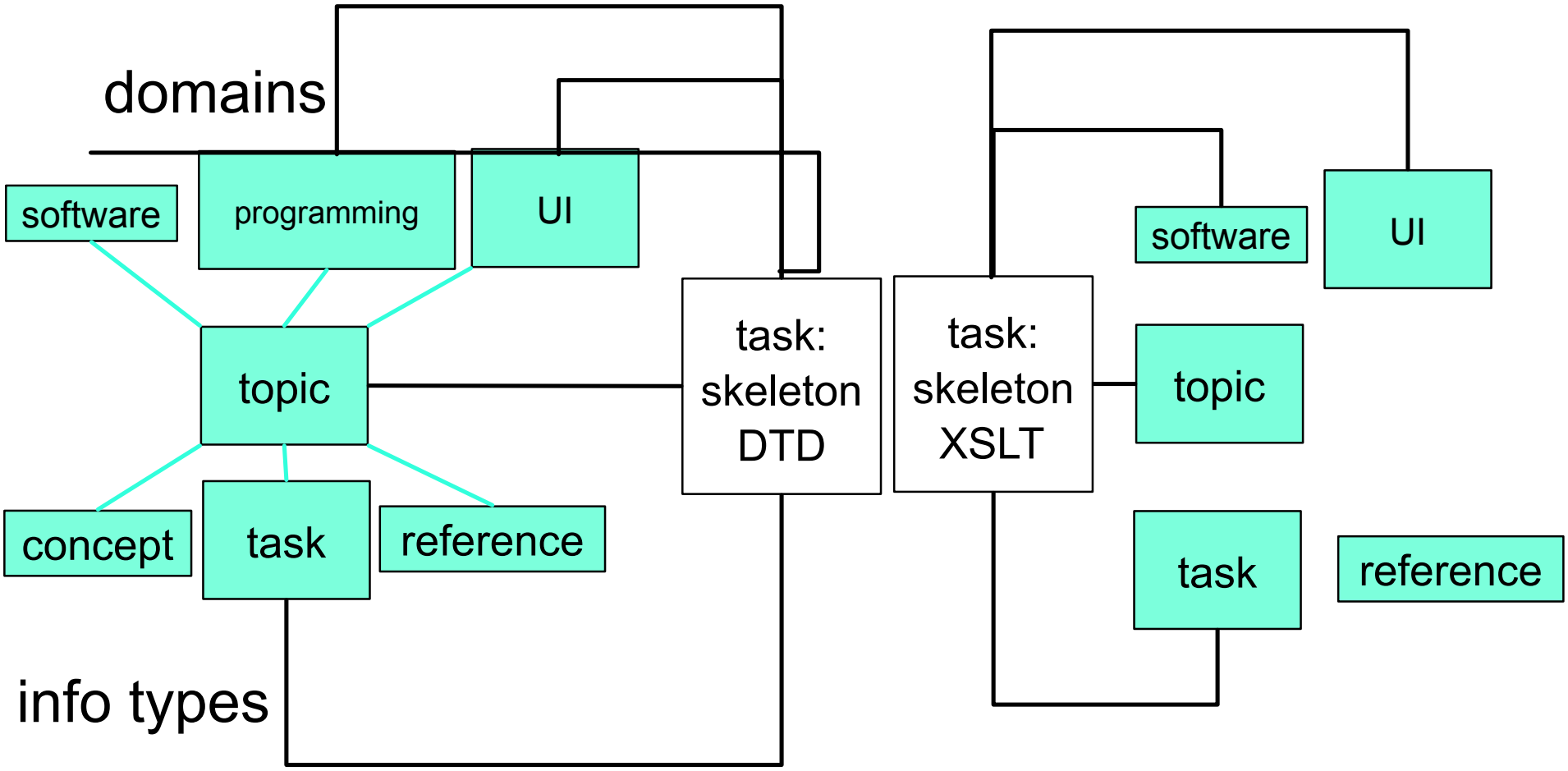
Specializing code

- Only necessary when existing general rules insufficient
- For example, task steps get formatted as an ordered list with no step-specific code
- But, for example, might want to add "fastpath" icon when outputting wiztitles, vs. wintitles
- Create as modules, parallel to information types and domains
- Pull together with a shell transform that imports modules, starting with general and proceeding to specific

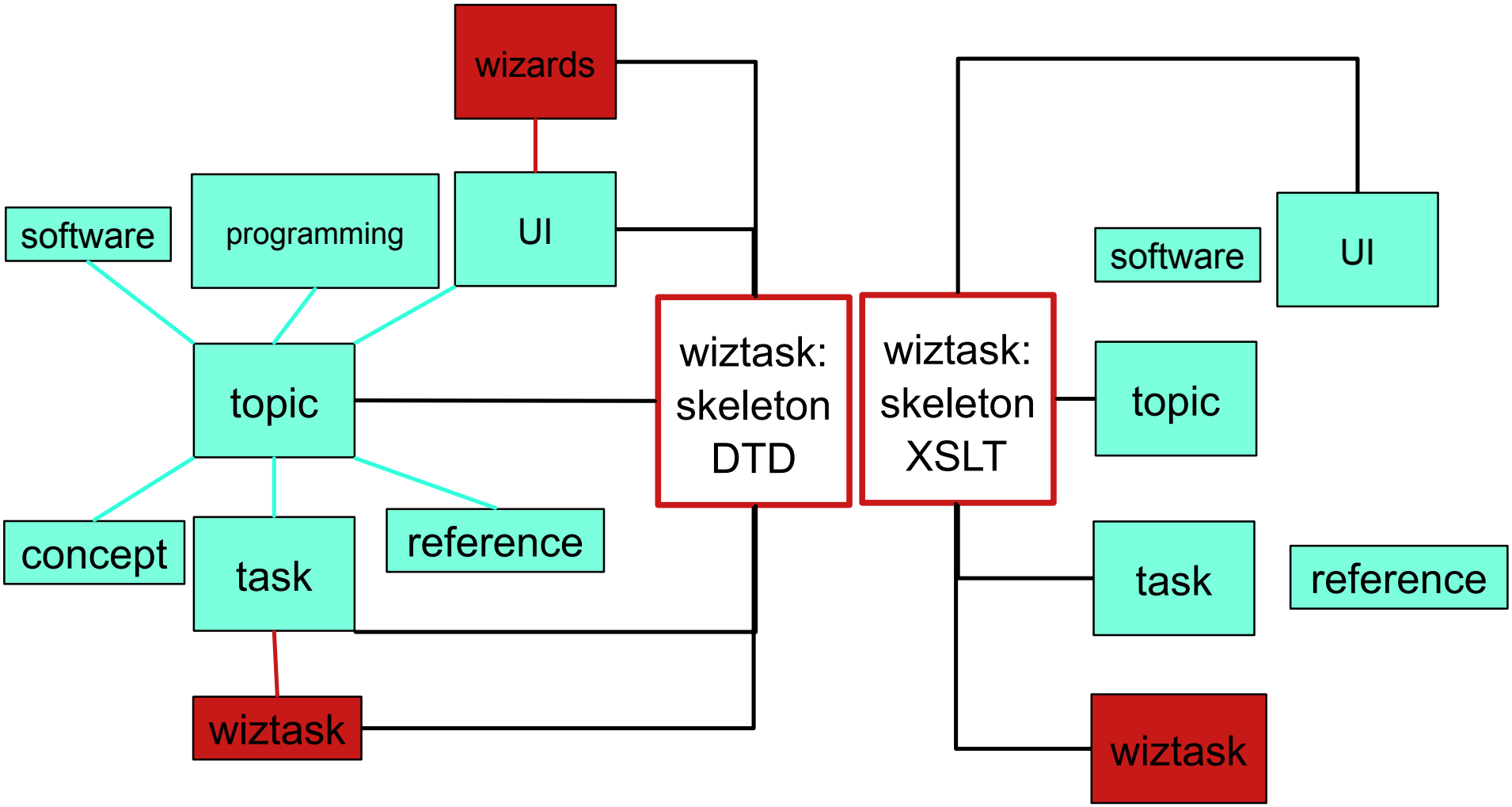
When code meets content

- Code and design are separate but parallel hierarchies
- If your processes encounter unknown information types or domains, they get processed as instances of the closest known ancestor type
- If your processes are specialized for your content but another group sends you more general content, it gets processed according to the general rules
- Sum: when the two hierarchies interact, they find their lowest common denominator

Specialization hierarchies: principled reuse



Specialization hierarchies: extensible reuse



Customization

- Adding new output rules without accompanying design distinctions (not parallel to a design hierarchy)
- Lets different groups get customized output from common content
- Insulates content from locally-driven design initiatives
- Incorporate customization modules using a shell transform that imports it after specialization code modules

Integration

- Select subset of available designs
- From full spectrum of types and domains, select the information types and domains you need
- For example, exclude highlighting if other domains are sufficient
- When others add to the hierarchies, you only get the additions you choose to integrate
- Again, using a shell DTD

Summary: Specialization

- Artifacts
  - Specialized DTD module
  - Shell DTD
  - (Specialized XSLT module)
  - (Shell XSLT)
- Costs/Benefits
  - Small cost
  - New design elements
  - (New code)
  - Migration/interchange supported by architecture (generalization transform)
  - Reuse of most existing design and all or most code

Summary: Customization

- Artifacts
  - Customized XSLT module
  - Shell XSLT
- Costs/Benefits
  - Less cost
  - No new design elements
  - Some new code
  - No migration/interchange issues
  - Reuse of all existing design and most code

Summary: Integration

- Artifacts
  - Shell DTD
- Costs/Benefits
  - Close-to-zero cost
  - No new design elements
  - No new code
  - No migration/interchange issues
  - Reuse of all existing design and code

Compare: New design from scratch

- Artifacts
  - Complete DTD
  - Complete XSLT
  - Any migration/interchange transforms when required
- Costs/Benefits
  - High cost
  - New design elements
  - New code
  - Migration/interchange supported by single-purpose transforms; no built-in mappings (transform may be complex, may require cleanup before and after)
  - No reuse of design or code

Next

- Get the DTDs and transforms:
  - www.ibm.com/developerworks/xml/library/x-dita1/xdita10.zip
- Discuss them, ask questions, make suggestions:
  - news.software.ibm.com/ibm.software.developerworks.xml.dita