General Overview

Dewayne E Perry
ENS 623A
Office Hours: T/Th 11:00-12:00
perry @ ece.utexas.edu
www.ece.utexas.edu/~perry/education/382v-s06/

Background

- Architecture and Design Intent in Component/COTS Based Systems
  - Keynote talk for ICCBSS 2006
  - Continues in the vein of my previous research at Bell Labs
    - Inscape
    - Software architecture
  - Current, important, emerging topic in software architecture
    - Eg, see Bosch 2004, Duenas 2005
  - Reflects some of the initial thinking for Matt Hawthorne's and Paul Grisham's thesis proposals

Basic Issues

- In creating systems we make choices because we have some *intent* in mind
  - Some requirements over others
  - One architecture instead of another
  - A specific algorithm or data structure over others
- When we create a product or component we have some idea of how we *intend* it to be used
  - May be specific or it may be general
- We use products or components with specific *intent* in mind
  - If a general product or component, may only use a part of it
  - If a specialized product or component may still use only a part of it
- In evolving systems
  - We often have to divine the original *intent* to understand how to make changes
  - We change things because we have some new intent in mind

Basic Facts about COTS/Components

- Integrating COTS and components often results in disasters from architectural mismatch [Garlan et al]
  - Lack of understanding of the intent (or assumptions) of components with respect to
    - Resources
      - ✓Who controls them, who uses them, how they are used, etc
    - Interactions with other components
      - ✓Who they interact with, their characteristics, their data models, etc
      - ✓How they interact, eg, synchronously or asynchronously, etc
    - Global architecture
      - ✓Topology, control flow, etc
    - System Construction
      - ✓Construction, instantiation, interactions, etc

Basic Facts

- Spend 80% of our time in (re)discovery to understand legacy systems [Bell Labs Study]
  - A large part of that is trying to determine the original intent of the architecture, design and code
- Rarely a problem in small group projects
  - But even there can be forgotten or misunderstood
- Coordination of multiple developers a major problem
  - Intent critical in choreographing multiple developers in initial development
  - Intent even more critical in evolutionary development

Basic Problems & Benefits

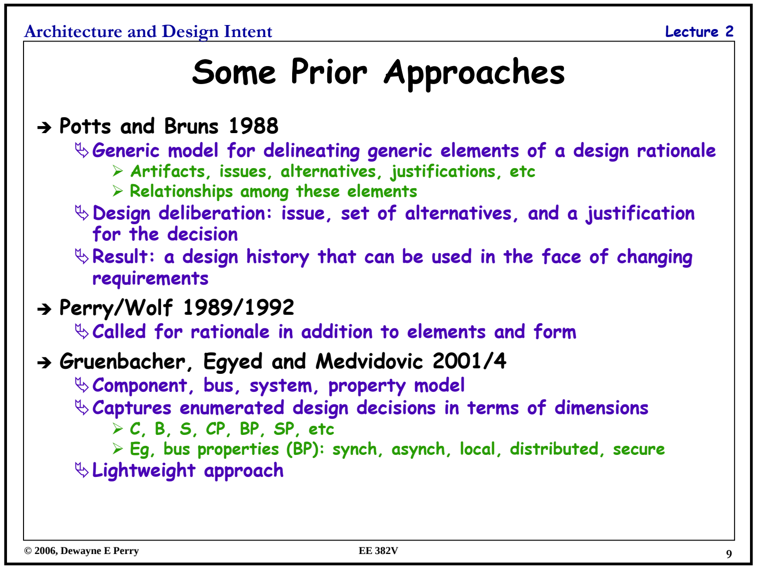
- Traditionally, intent conveyed by documentation
  - OS360 documentation
    - 6 months project workbook: 5 feet
    - Daily changes: 2 inches
- One release of Lucent's 5ESS system
  - 11.8% of design and implementation faults due to ambiguous requirements
  - 30.6% of design and implementation faults due to incomplete or omitted requirements or design
  - 42.4% - due to traditional problems of documentation
- Documentation as a shared model of intent
  - Requirements – a shared model of the problem
  - Architecture – a shared model of the basic solution structure
  - Design and code – shared model of the machine in more detail

Intent and Evolution

- Everything changes
  - World changes: uses and requirements change
  - Technology changes
  - Operating context changes
  - System itself changes: improvements, faults fixed
- What persists in the face of evolution: CODE
  - Requirements, architecture, design documents out of date
  - Code is only thing up to date
  - Code: desiccated relic of a long intellectual process
    - Difficult to reconstruct the intent and reasoning
    - Too many ways to backtrack
- Difficulties result:
  - Not clear how requirements changes impact the system
  - Not clear how structural changes impact the system
  - Not clear how code changes impact the architecture or the system

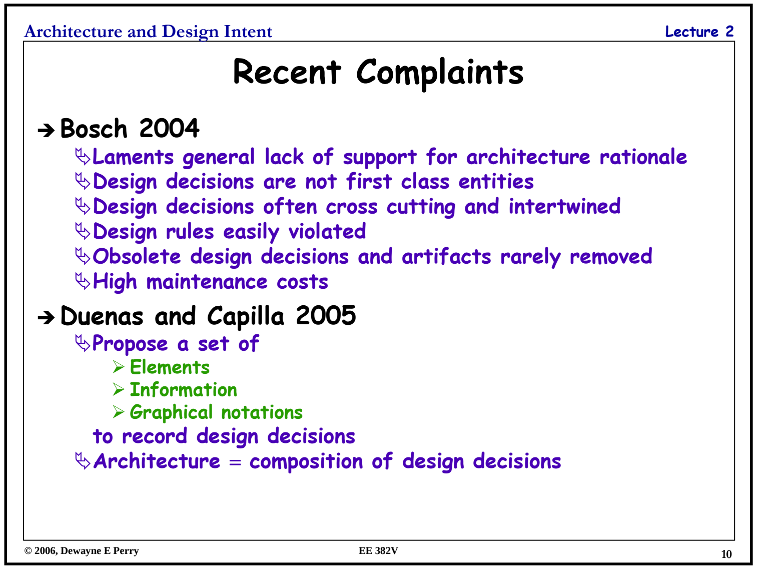
Some Basic Distinctions

- Decisions – is at best a description of what is decided
  - May indicate alternatives
  - May have some considerations about the alternatives
  - May go further and evaluate the alternatives
  - May indicate why the decision was made
- Intent – why decisions were made the way they were
  - Why alternatives were not chosen
  - What effect the evaluations had on the design choices
  - What expectations result from the choice



Some Prior Approaches

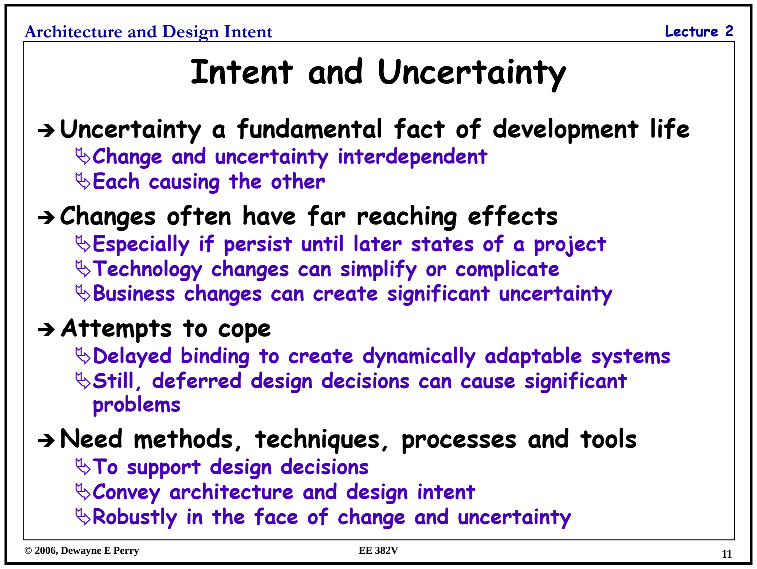
- Potts and Bruns 1988
  - Generic model for delineating generic elements of a design rationale
    - Artifacts, issues, alternatives, justifications, etc
    - Relationships among these elements
  - Design deliberation: issue, set of alternatives, and a justification for the decision
  - Result: a design history that can be used in the face of changing requirements
- Perry/Wolf 1989/1992
  - Called for rationale in addition to elements and form
- Gruenbacher, Egyed and Medvidovic 2001/4
  - Component, bus, system, property model
  - Captures enumerated design decisions in terms of dimensions
    - C, B, S, CP, BP, SP, etc
    - Eg, bus properties (BP): synch, asynch, local, distributed, secure
  - Lightweight approach



Recent Complaints

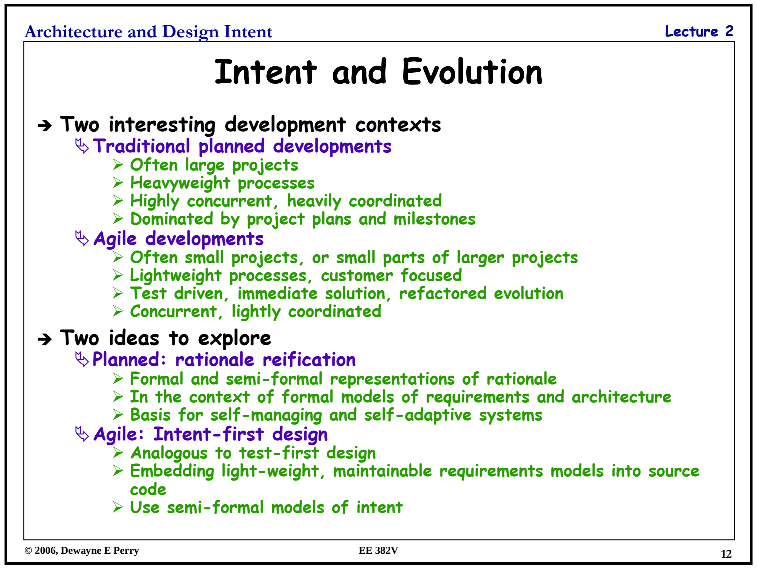
- Bosch 2004
  - Laments general lack of support for architecture rationale
  - Design decisions are not first class entities
  - Design decisions often cross cutting and intertwined
  - Design rules easily violated
  - Obsolete design decisions and artifacts rarely removed
  - High maintenance costs
- Duenas and Capilla 2005
  - Propose a set of
    - Elements
    - Information
    - Graphical notations

    to record design decisions
  - Architecture = composition of design decisions



Intent and Uncertainty

- Uncertainty a fundamental fact of development life
  - Change and uncertainty interdependent
  - Each causing the other
- Changes often have far reaching effects
  - Especially if persist until later states of a project
  - Technology changes can simplify or complicate
  - Business changes can create significant uncertainty
- Attempts to cope
  - Delayed binding to create dynamically adaptable systems
  - Still, deferred design decisions can cause significant problems
- Need methods, techniques, processes and tools
  - To support design decisions
  - Convey architecture and design intent
  - Robustly in the face of change and uncertainty



Intent and Evolution

- Two interesting development contexts
  - Traditional planned developments
    - Often large projects
    - Heavyweight processes
    - Highly concurrent, heavily coordinated
    - Dominated by project plans and milestones
  - Agile developments
    - Often small projects, or small parts of larger projects
    - Lightweight processes, customer focused
    - Test driven, immediate solution, refactored evolution
    - Concurrent, lightly coordinated
- Two ideas to explore
  - Planned: rationale reification
    - Formal and semi-formal representations of rationale
    - In the context of formal models of requirements and architecture
    - Basis for self-managing and self-adaptive systems
  - Agile: Intent-first design
    - Analogous to test-first design
    - Embedding light-weight, maintainable requirements models into source code
    - Use semi-formal models of intent

Earlier work

- **The Inscape Environment**
  - Constructive approach based on
    - Formal interface specifications
    - Semantic interconnections determined during construction
    - Set of propagation rules
  - Basic rule: all preconditions and obligations must be satisfied or propagated to the interface
  - Preconditions or obligations unpropagated and unsatisfied represent faults
    - Called precondition ceiling and obligations floors
  - Specification contributions
    - Obligations
    - Multiple results, some of which are considered as exceptions
      - ✓Set of rules for handling them
      - ✓Useful for fault tolerance and reliability
  - Predicate based retrieval of components

Earlier Work

- **Perry/Wolf Architecture model**
  - Architecture = (elements, form, rationale)
  - Components and connectors the basic elements
  - Form is properties and relationships (ie, interactions) and constraints on those properties and relationships
  - Rationale is the justification for the elements and form
    - The primary carrier of architectural intent
  - Architecture styles codify basic aspects of intent to be applied to elements and form
  - Rationale and styles are critical for managing evolution

Earlier Work

- **Architectural Prescriptions**
  - Transforming software requirements into architecture prescriptions
  - KAOS → Preskriptor
    - Goals → constraints
    - Architect has freedom to chose how goals are distributed among architectural elements as constraints
    - Goals as a means of expressing requirements intent
    - Prescriptions as a means of expressing architectural intent
  - Architectural styles important as a form of constraint codification
    - Incomplete architecture prescriptions
    - Applied to specific elements, collections of elements of the entire system
    - Also capture architecture intent

Earlier Work

- **Intent-based Architectures**
  - Introduces architecture intent as a key concept
  - Intent of an element encapsulates its functional purpose
  - Intent associated with roles in architecture
    - Elements with similar intent can be substituted for each other
    - Based on higher levels of abstraction
    - Direct link between requirements and architecture
  - Enables reification of an architecture in one or more functionally equivalent implementations
  - Basis for self-configuring adaptive systems
    - Respond to changes in environmental or operational conditions
    - By reconfiguring – subject to functional and nonfunctional constraints

Rationale Reification

- Basic idea:
  - Begin with formally specified requirements and architecture
    - Eg, KAOS requirements specifications and architecture prescriptions
  - Requirements are in problem domain terms; architecture often in solution domain terms
    - Systems drivers such as user needs, business goals, strategies are incorporated in requirements
  - Currently no connection between the two
    - No rationale, even informally
    - Mapping from problem domain to solution is problematic
  - Current focus of architecture:
    - Elements and form
    - Rationale, if treated at all, is informal and general
  - Rationale reification
    - Capture refinements and transformations used by architects in creating the architecture from the requirements

Rationale Reification

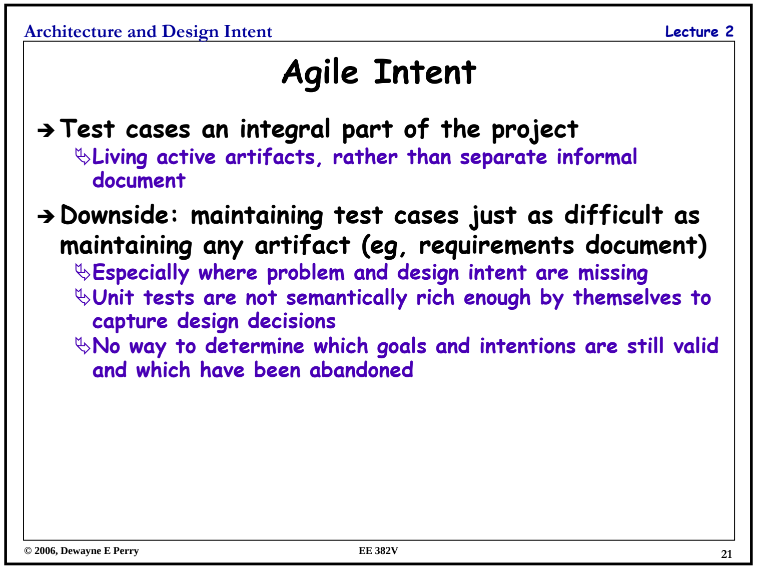
- Basis for systematic requirements and architecture based evolution
  - Changing requirements lead to changes in rationale and associated changes in the architecture
  - Requirements become an integrated part of the system structure rather than something separate and apart
- Rationale determines the mapping between the functional and non-functional requirements and the architecture
  - Abstract architecture in terms of problem domain (ala Preskriptor) and models functional intent
  - Concrete architecture then related to abstract via intent
  - *Refinement* used to decompose functinality into smaller functional elements
  - *Transformations* used functional structure into an architecture that satisfies the non-functional requirements
  - Requirements → (rationale) → architecture
    - Captures semantics and conditions for mappings
    - Enables traceability from goals to structure

Rationale Reification – Tool Support

- Requirements modeling support
  - Such as the KAOS System of Axel van Lamsweerde
- Rationale modeling support
  - Create and evolve mappings and transformations
- Architecture modeling support
  - Create, edit and view the architecture models
- Intent modeling and visualization support
  - Ties everything together
- Self-managing/adaptive support
  - Styles, components, etc

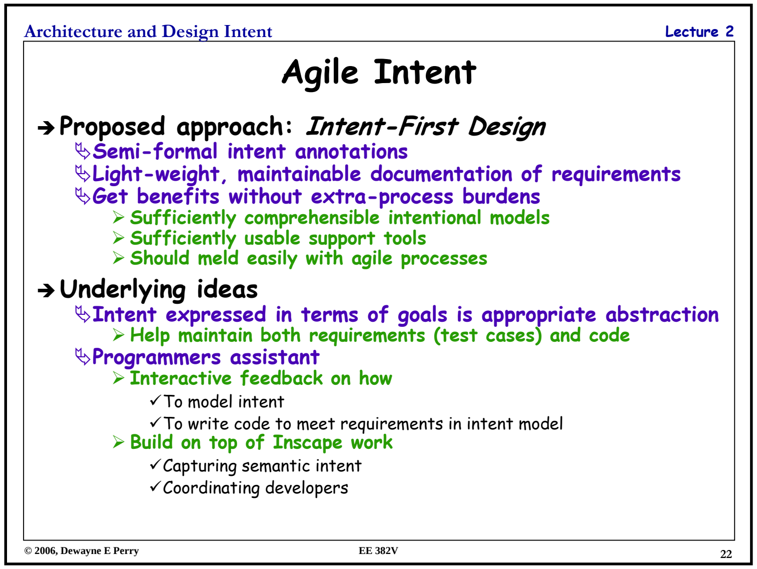
Agile Intent

- Agile context
  - Set of methods, techniques and processes to cope with changing and uncertain requirements. Eg,
    - Feature oriented milestones
    - sort iterations with frequent deliveries
    - Close interactions with customers
    - Deferred design decisions
  - Most popular: extreme programming (XP)
    - Requirements captured as acceptance and unit test cases
      - ✓Written from customer stanpoint
    - Test-first design
      - ✓Written before code
      - ✓Can determine if requirements already satisfied
    - Code written to meet minimum needs of requirements
      - ✓Just sufficient, complex as needed – simple as possible
    - Requirements change → test cases added or evolved



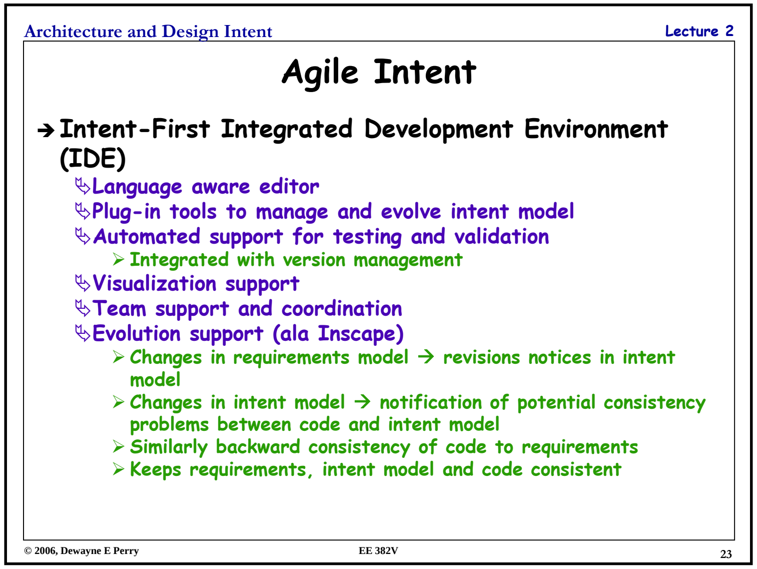
Agile Intent

- ➔ Test cases an integral part of the project
  - Living active artifacts, rather than separate informal document
- ➔ Downside: maintaining test cases just as difficult as maintaining any artifact (eg, requirements document)
  - Especially where problem and design intent are missing
  - Unit tests are not semantically rich enough by themselves to capture design decisions
  - No way to determine which goals and intentions are still valid and which have been abandoned



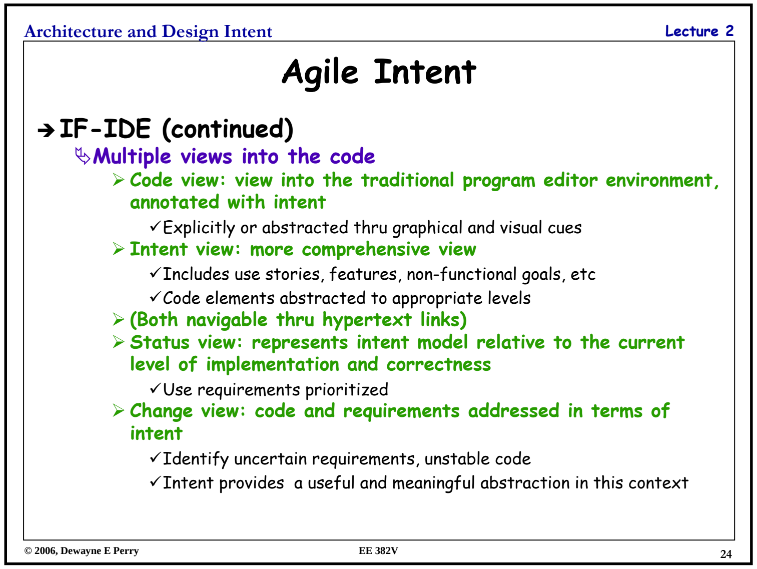
Agile Intent

- ➔ Proposed approach: *Intent-First Design*
  - Semi-formal intent annotations
  - Light-weight, maintainable documentation of requirements
  - Get benefits without extra-process burdens
    - ➢ Sufficiently comprehensible intentional models
    - ➢ Sufficiently usable support tools
    - ➢ Should meld easily with agile processes
- ➔ Underlying ideas
  - Intent expressed in terms of goals is appropriate abstraction
    - ➢ Help maintain both requirements (test cases) and code
  - Programmers assistant
    - ➢ Interactive feedback on how
      - ✓To model intent
      - ✓To write code to meet requirements in intent model
    - ➢ Build on top of Inscape work
      - ✓Capturing semantic intent
      - ✓Coordinating developers



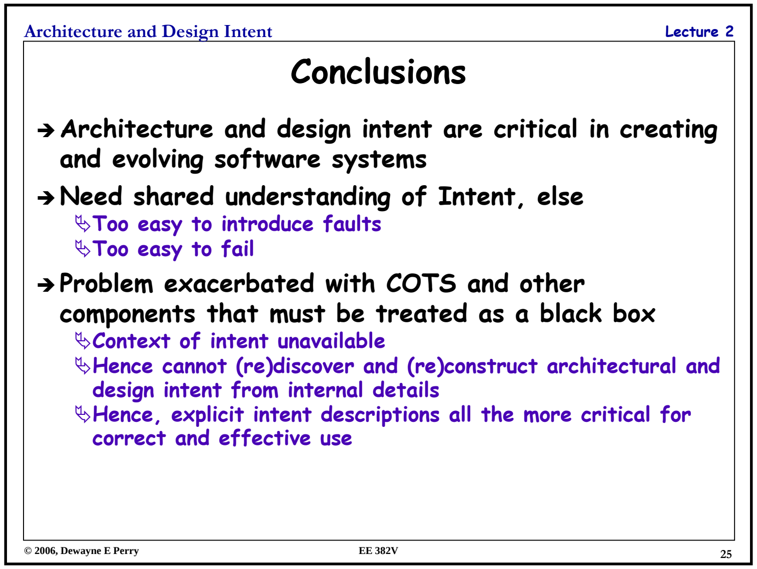
Agile Intent

- ➔ Intent-First Integrated Development Environment (IDE)
  - Language aware editor
  - Plug-in tools to manage and evolve intent model
  - Automated support for testing and validation
    - ➢ Integrated with version management
  - Visualization support
  - Team support and coordination
  - Evolution support (ala Inscape)
    - ➢ Changes in requirements model → revisions notices in intent model
    - ➢ Changes in intent model → notification of potential consistency problems between code and intent model
    - ➢ Similarly backward consistency of code to requirements
    - ➢ Keeps requirements, intent model and code consistent



Agile Intent

- ➔ IF-IDE (continued)
  - Multiple views into the code
    - ➢ Code view: view into the traditional program editor environment, annotated with intent
      - ✓Explicitly or abstracted thru graphical and visual cues
    - ➢ Intent view: more comprehensive view
      - ✓Includes use stories, features, non-functional goals, etc
      - ✓Code elements abstracted to appropriate levels
    - ➢ (Both navigable thru hypertext links)
    - ➢ Status view: represents intent model relative to the current level of implementation and correctness
      - ✓Use requirements prioritized
    - ➢ Change view: code and requirements addressed in terms of intent
      - ✓Identify uncertain requirements, unstable code
      - ✓Intent provides a useful and meaningful abstraction in this context



Conclusions

- ➔ Architecture and design intent are critical in creating and evolving software systems
- ➔ Need shared understanding of Intent, else
  - ↳ Too easy to introduce faults
  - ↳ Too easy to fail
- ➔ Problem exacerbated with COTS and other components that must be treated as a black box
  - ↳ Context of intent unavailable
  - ↳ Hence cannot (re)discover and (re)construct architectural and design intent from internal details
  - ↳ Hence, explicit intent descriptions all the more critical for correct and effective use