Architecture and Design Intent

Dewayne E. Perry
(perry@ece.utexas.edu)
Paul S Grisham
(grisham@ece.utexas.edu)
Empirical Software Engineering Laboratory (ESEL)
The University of Texas at Austin

Overview of this Research

- Explore approaches of capturing design "intent"
  - Documenting design, decisions, and decision processes
- Explore approaches of applying design "intent"
- Study the nature of architectural design
  - Relationship between high level abstractions and low level details in problem solving
  - Relationship between opportunistic and rational design
  - Relationship between initial and evolutionary design
- Describe new design methods and documentation systems
  - Goal-oriented prescriptive architectures
  - Methodical exploratory test and design
  - Intent-based refactoring and reification systems

“The issue is not documentation, the issue is understanding.”

Jim Highsmith

Agile Software Development Ecosystems (2003)

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

Definitions of Intent

- **Functional Intent**
  - Describe *WHAT* a program element is and does
    - Functional Requirements
    - Functional Specifications
- **Design Intent**
  - Describe HOW a program element interacts with other program elements
    - Scenarios / Use-cases
    - Contracts
    - Obligations
- **Design Rationale**
  - Describe *WHY* program element was designed a certain way
    - Selection Criteria
    - Plans and Methods
    - Alternatives
    - Non-functional Requirements (?)

Traditional Approaches to Intent

- 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 - more detail
- But . . . everything changes
  - World changes: uses and requirements change
  - Technology changes
  - Operating context changes
  - System itself changes: improvements, faults fixed
- 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 arch/system
  - Not clear how context changes impact the arch/system

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 ceilings 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 are 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

Design Intent Modeling

- **Cited Benefits:**
  - Design Analysis
    - Claim: Formalizing decisions facilitates identifying and avoiding early mistakes
  - Communication and Coordination
    - Claim: Formalizing decisions prevents large teams from making incompatible decisions
  - Maintenance and Evolution
    - Claim: Documenting decisions captures the designer's thoughts
      - ✓ Aids program comprehension
      - ✓ Prevents architectural mismatch in new components
      - ✓ Assists in impact of additive and corrective changes
- **When needs and contexts change over time, designers can see which design decisions can or must be changed**

How We Use Documented Intent

- Replication
  - Use existing patterns and processes to build something new
    - Strategies, Patterns and Idioms
  - Be sure we are replicating the important things
    - Cutting off the end of the ham
- Reuse
  - Include legacy modules in new systems
    - Identify opportunities for reuse
    - Make sure we use those modules correctly
    - Identify assumptions about usage
- Modification
  - Perform risk analysis
    - Explore semantic and operational dependencies
- Maintenance
  - Identify out-of-date or invalidated assumptions

Problem Structuring

- **Well-Structured Problems:**
  - Relationship between problem, solution methods, and criteria
    - Coding a well-defined algorithm
- **Ill-Structured Problems:**
  - Not well-structured (i.e., no domain guidance on solution methods or evaluation)
    - Deciding what to build (requirements selection)
- **Problem Structuring:**
  - The act of turning ISPs into WSPs
  - Software Analysis and Design:
    - Select requirements to implement
    - Given a requirement, decompose into a set of goals
    - Transform goal into a detailed design
    - Treat design as a WSP, and abstract its complexity, and use to solve another goal

Designers and Decisions

- Opportunistic Decision Making
    - Decisions made with partial knowledge influence later decisions as fact
  - Emergent knowledge and partial solutions
    - Discovery of partial Well-Structured Problems from domain knowledge
  - Emergent requirements need attention
    - Immediate Structuring ISP into WSP
  - Drifting
    - Explore dependencies and assumptions
  - Scenario exploration
    - Make ill-structured requirements concrete
    - Verify partial solutions
    - Confirm inferred requirements
  - Early design activities are opportunistic, rather than methodical or rational

Designers and Decisions

- Rational Decision-Making
  - Made based on criteria and rationale
    - Consequential choice of an alternative
    - Set of possible options are known
    - Probabilities of outcomes are known
- Natural Decision-Making
  - Situational decisions using partial knowledge + personal experience
  - Ill-defined tasks or goals
  - Situational assessment over consequential choice
  - Alternatives not considered until rejection
    - Satisficing solutions
- Software design decisions are cross-cutting
  - May operate on multiple levels of abstraction simultaneously
    - E.g., arch. style impacts implementation language and technology infrastructure selection

Software Design Decisions

- So, in the life of a piece of software
  - Some decisions were rational
    - E.g., Technology vendor selection
  - Some decisions were opportunistic
    - E.g., Spike solution, then integrate if possible
  - Some decisions were arbitrary
    - E.g., Requirement prioritized as "low-hanging fruit"
  - Some decisions were deferred
- Over time:
  - As rationale is lost, distinction between decision types is lost
    - Rational decisions relate to well-structuredness and optimality
    - Natural decisions were situationally satisficing based on partial solutions and incomplete knowledge
  - Assumptions and Dependencies are forgotten or ignored
- Problem: Many design decisions happen without designers being aware of them

Faking It

- Because there is something satisfying about rational decisions, treat all decisions as rational
  - In mature engineering professions, many tasks are WSP
  - We want to believe that Software Engineering is an engineering profession
  - Express SE problems as WSP with well-defined goals and decision processes (i.e., that it is rational)
  - Emphasis on design methods

"We will never find a process that allows us to design software in a perfectly rational way… [but] we can present our system to others as if we had been rational designers and it pays to pretend do so during development and maintenance."

D. Parnas and P. Clements. "A Rational Design Process: How and Why to Fake It"

Problems with Faking Design Rationale

- Natural decisions are situational
  - Difficult to differentiate between essential domain criteria and dynamic or volatile criteria
- Faked rationale tends to be uniform
  - What level of abstraction / granularity to use?
- Does not necessarily reflect real alternatives
  - How should alternative solutions should be faked?
  - Are these alternatives realistic or practical?
  - Are these alternatives desirable under emergent criteria?
- Bad or failed solutions are interesting
  - Faked rationale describes successful designs
  - "The best prototype is a failed project" (Curtis, et.al.)
- Faked rationale uses "timeless" inferential reasoning
  - Argumentation-based rationale studies emphasize "reconstructing" rationale
  - If you can infer rationale, why document faked rationale?

Research Issues

- **Initial Design:**
  - Use and documentation of design methods which apply prior design knowledge
  - Methods of documenting and analyzing exploratory techniques
    - Early specification
    - Test-driven design
  - Structuring requirements and methodically driving design
    - Requirements structuring and prioritization is a design activity
- **Evolutionary Design:**
  - The software understanding problem is an attempt to reconstruct:
    - The rationale for rational design decisions
    - The situational context and expert knowledge for opportunistic decisions
    - The relationship between design elements and design decisions
    - Prioritization of criteria for proscribing and prescribing change

Science of Design

- Guindon's studies of designers (late 80s)
  - Applied generally whenever we talk about software design activities
  - Are they still relevant? Do they describe arch. design?
    - Low level of abstraction
    - Developers working in isolation
    - Small projects (one-sitting projects)
    - Initial design only
- We need current studies of architectural designers
  - Teams of architects and lower-level designers
    - Study the interactions between them
  - Study the cognitive issues of architectural (high-level) design and low level design
  - Currently interviewing software architects on a number of issues
  - Differentiate between activities of initial and evolutionary design

Exploratory Design and Design Artifacts

- Inspired by agile development and designs:
  - Use executable specifications to drive design
  - Tests are internally consistent, but not complete
  - Tool support to record historical relationships between test evolution and design evolution
- Exploratory testing:
  - Use tests to describe problem to be solved piecemeal
  - Exploratory tests bind subsequent design and refactoring
  - Test suites are reduced into regression tests or specifications
- Current approaches:
  - Treat integration tests as design intent model
  - Explore relationship between operational tests and semantic interconnection models (as in Inscape)
  - Develop a design framework in eclipse to support evolutionary test design

Using Test to Drive Design

- Traditional approach
  - Design → Implement → Test
- Test-First Design or Test-Driven Development
  - Test → Implement → Iterate → Refactor
- Since we treat requirements structuring as a design activity:
  - Test selection and design is a design activity
  - Tests constrain and guide development
  - Test evolution is a record of design changes
- Tool support:
  - Integrate test management to version control
  - Use tests to describe corrective and additive changes
- Tests as Program Comprehension
  - Integration tests are scenarios that describe intent

Incremental Design and Iterative Specification

- Problem: Generalize correctness for ill-defined task
  - Requirements prioritization and structuring a design activity
  - Tests and scenarios only describe a part of the problem
  - We want some way to relate specifications to tests and vice versa
- Goal: Methodical approach to test design
  - Prescriptive guidance on test selection
  - Structured/annotated tests for program analysis and specification building
  - As tests are added, specification becomes more complete
- Test Maintenance
  - Process of minimizing test suite to reduce testing costs
  - As tests are eliminated, intent is lost

Specification → Generate Test

Tests → Specification Building → Coverage Test Suite

Prescriptive Architectures

- Knowledge about design implicit in architecture
  - Descriptive models show the results of those decisions
  - Requirement prioritization is lost
  - Relationship between decisions is lost
- Prescriptions → restricts design elements and relationships
  - Essential constraints are differentiated and specified
    - Essential design intent → architecture prescription
    - Opportunistic decision → traceability between prescription and detailed architecture or design
  - Describes *classes* of solutions
    - Including future adaptive, perfective, and corrective changes
  - Supports incremental and exploratory design activities
  - We need mechanisms and tool support for enforcing and checking constraint satisfaction

Rationale Reification

- **Basic idea:**
  - Begin with formally specified requirements and architecture
    - E.g., 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 functionality 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

Approaches to Design Intent

- Capture Intent through:
  - Maintenance and reuse of existing design artifacts
    - Incremental and evolutionary design histories
  - Methodical, prescriptive approaches that relate domain, design, and constraints, reusing design knowledge
    - Process model (a priori)
    - Input knowledge (method by-product)
    - Intermediate and final models (method by-product)
    - Justification for overriding method where appropriate
  - Apply best practices of intentional design
    - E.g., styles, patterns, and idioms
    - Intent can be identified through metonymic clues
  - Understand the difference between initial design documentation needs and evolutionary design needs
    - Prefer approaches that address both needs
    - Treat all design as incremental

New Graduate Course

- Architecture and Design Intent (Spring 2006)
  - Emphasis on representing designs along with various types of intent
  - Covered published research and state of the art in:
    - Cognitive and social interactions in software design
    - Empirical studies of design and designers
    - Design rationale modeling (representations and tools)
    - Architecture design rationale and design drivers
    - Styles, patterns, idioms
    - Using design histories and intent models in evolution
    - Design reuse and design process reuse
    - Opportunistic vs. rational decision-making
  - Students used architecture design methods to solve an evolution/maintenance problem for a well-defined architecture
    - What information was useful in understanding the design issues?
    - What information was missing?

Results from Class

- Comments from Students about the project:
  - Need multiple paths/views to information
    - Single representation, but ability to form different queries
  - Tool support is critical to maintain relationships between design elements and decisions
  - Low-level details were later discovered to be unnecessary
    - However, initial comprehension searches involved finding those details
    - It takes experience to know what's relevant
  - Difficult to differentiate between what is planned and what is actually implemented
  - Hard to differentiate between:
    - Enhancement requirements
    - Domain requirements
    - General-purpose requirements
    - Requirements driven by the existing application

Experiences with CBSP and Archium

- Problem: Evolve the design of a browser accessibility module
  - Initial undocumented attempt at design failed
  - Very general idea problem; no specific designs for a solution
- Approach: Use CBSP and Archium to structure requirements and evaluate design possibilities
- Conclusions:
  - CBSP
    - Useful in constraining requirements and providing traceability
    - No support for selecting architecture elements
    - Led to rejecting a candidate solution
  - Archium
    - Explicit capture of candidate solution evaluation
    - No methodical support for selecting candidate solutions

FSE 2006 Workshop on Architecture and Design Intent

- Portland, Oregon – November 5, 2006
- Discussion format
- Invitation on the basis of position papers (5 pages)
- Topics Include
  - Design decisions, rationale and intent in the context of initial and evolutionary design
  - Using intent and rationale to manage evolution
  - Decision support and capture tools
  - Design of empirical studies for measuring the usefulness of intent and rationale in design and maintenance activities
- Full-day workshop with presentations and discussion
  - Digital library publication of position papers

See you all there!