University of Twente
department of
computer science

A Constructive Approach to Software Evolution

Selim Ciraci

Pim van den Broek

Mehmet Aksit

Univeristy of Twente

Outline

University of Twente
department of computer science

- The limitations of current design/evaluation techniques w.r.t. evolution
- Software Design Process – Problem solving approach
  - Evolution Changes
- Approaches to software evolution
  - Destructive
  - Constructive
- Constructive approach to software evolution applied to integration problem
  - How to Apply
  - The contexts of evolution problems

Limitations of design/evaluation techniques

University of Twente
department of computer science

- Design processes:
  - Evolution is not considered, software is evolved by changing the initial components
  - No systematic way for finding mechanisms that can allow the software to evolve without changing components
- Evaluation techniques (Scenario based)
  - Scenario's find problematic components
    - E.g. what components are going to change in near future
  - How to change the identified components so they can withstand these changes?
    - Not addressed by evaluation techniques

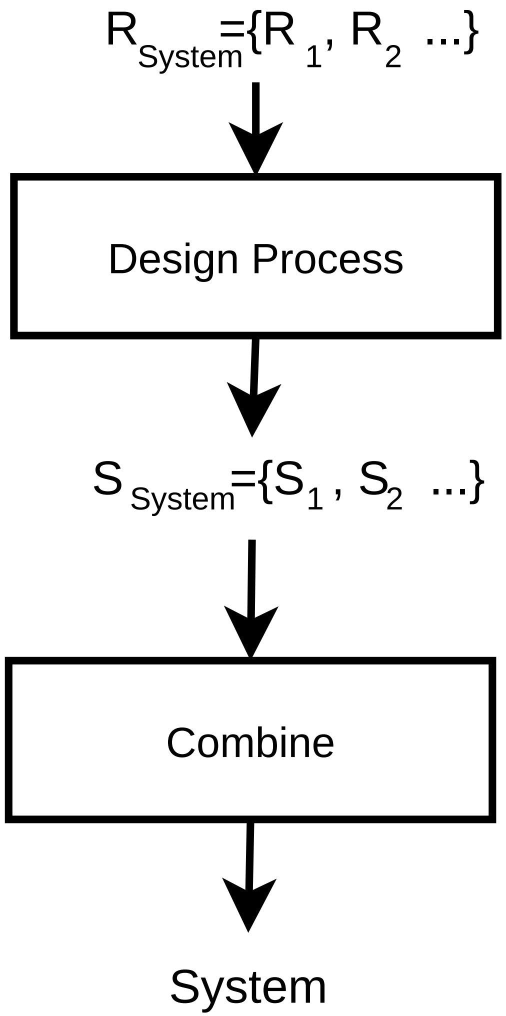
Limitations of design/evaluation techniques

- Design patterns and styles (mechanisms)
  - They provide extensible interfaces
    - Can withstand changes
  - But which design pattern can be used for which evolution problem?

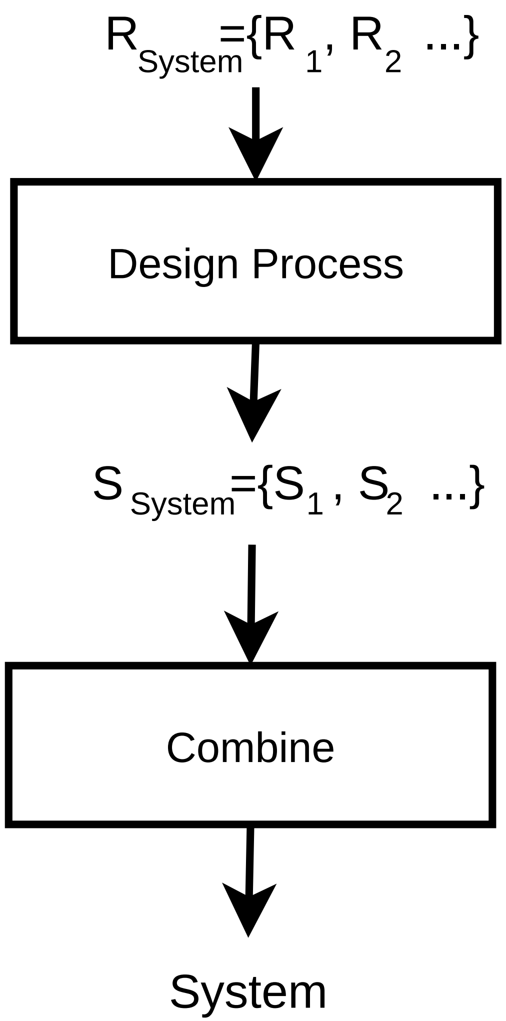
In summary: Design/Evaluation techniques do not include steps that points out which mechanisms can be used to apply the changes

Result: Changes applied by changing the initial components, design drift

- Software system starts its life-cycle with a set of Requirement specifications
- The design process converts the requirements to solutions
- A solution can be viewed as a set that contains the software components that solve the requirement(s)
  - Contents of a solution set depends on the design process used



- The set S_{System} is a set of sets that contain the solutions of the system.
- The solutions in S_{System} are then combined to form the overall software system
  - *System=Combine(*S_{System}*)*



University of Twente
department of computer science

Example PDA Input and Storage System

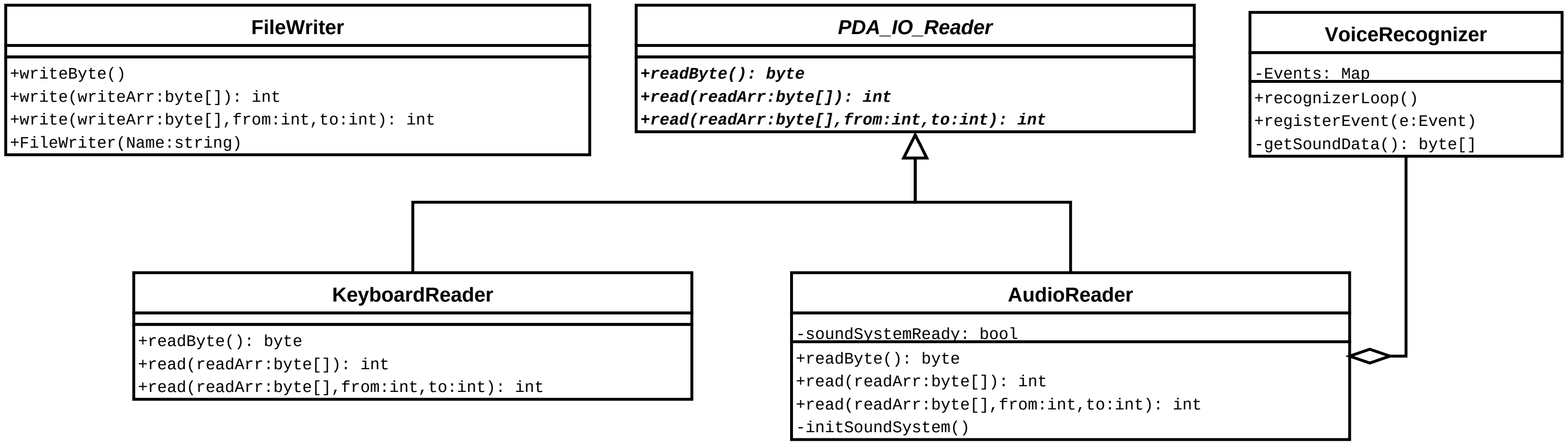
- The Requirements:
  - R_1: The system should be able to accept textual input
  - from the user.
  - R_2: The system should be able to accept spoken input
  - R_3: The system should be able to store the given input in text format on a local disk
- $R_{System} = \{R_1, R_2, R_3\}$

Example PDA Input and Storage System

- The solutions for these requirements
- $S1 = \{C1, C2, R1\}$, $S2 = \{C3, C4, R2, R3\}$, $S3 = \{C5\}$ where:
  - $C1$: Abstract I/O Reader class
  - $C2$: Keyboard Reader Class
  - $R1$: Inheritance relation between $C1$ and $C2$
  - $C3$: Audio Recorder class
  - $C4$: Voice Recognizer class.
  - $R2$: Inheritance relation between $C1$ and $C3$
  - $R3$: Aggregation relation between $C4$ and $C3$
  - $C5$: File writer class.

Software Design Process – Problem solving approach
Design of PDA Input & Storage System

University of Twente
department of computer science



Software Design Process – Problem solving approach Evolution

- Evolution causes the requirements of the system to change
- Requirement changes causes the solutions of the software system to change
- Three types of changes:
  - Integration: S_{System} -> $S_{System} \cup \{S_{New}\}$
  - Removal: S_{System} -> $S_{System} - \{S_{Old}\}$
  - Modification: S_{System} -> $(S_{System} - \{S_{Old}\}) \cup \{S_{New}\}$

Approaches to Software Evolution

- Requirement changes affect the solutions
- Destructive Approach: Due to changes at S_{system} the combine operation is restarted
  - The interactions between components are re-identified
- Constructive approach: Find the context of the evolution problem, find the mechanism(s) that allow extensions for this context and apply the changes without breaking about the *System*

Constructive Approach

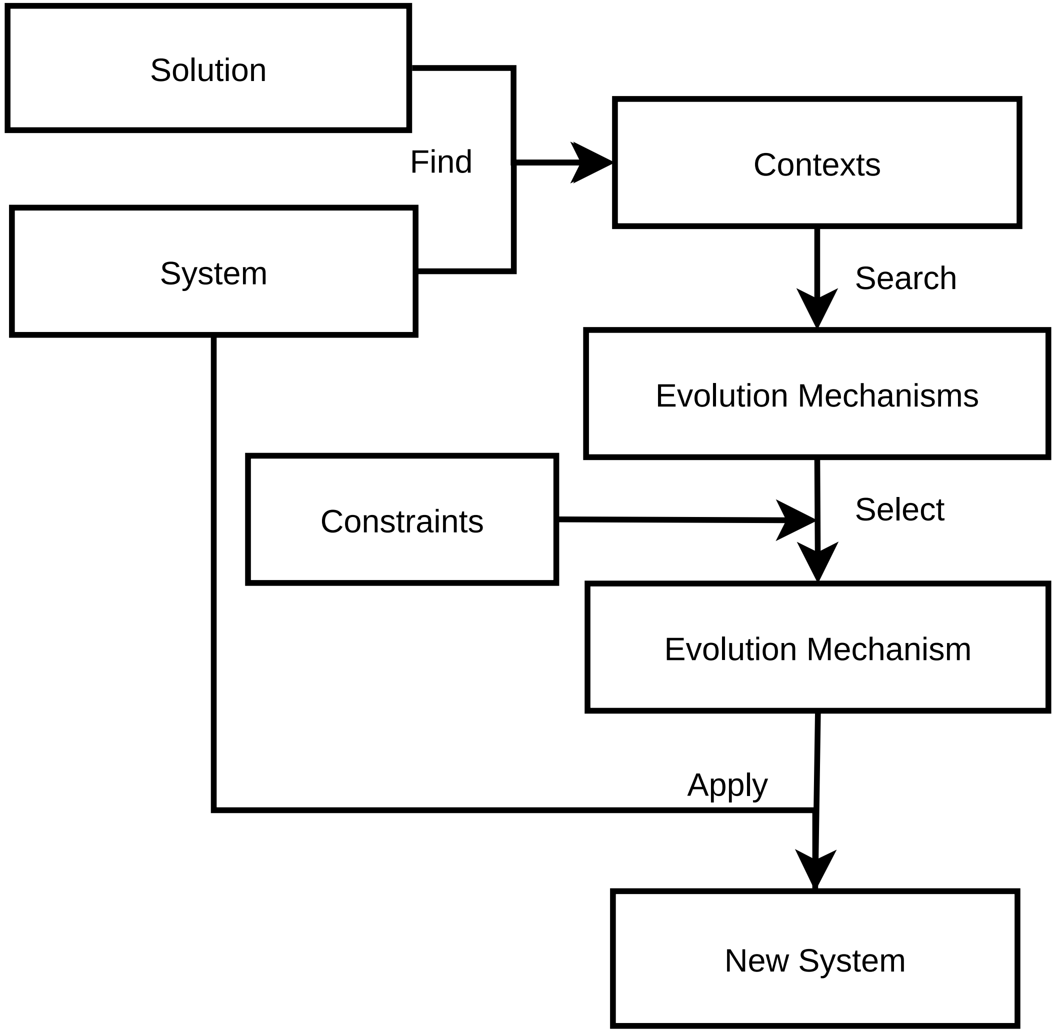
- Works without breaking up the *System* to its solutions
  - $NewSystem = (S+, S-) \oplus System$
    - S+ : Set of solutions to be added to the system
    - S- : Set of solutions to be removed from the system
- For the types of changes:
  - Integration: $NewSystem = (\{S_{New}\}, \{\}) \oplus System$
  - Removal: $NewSystem = (\{\}, \{S_{Old}\}) \oplus System$
  - Modification: $NewSystem = (\{S_{New}\}, \{S_{Old}\}) \oplus System$

Approaches to Software Evolution Example

- Example PDA Input & Storage
  - R4 – The system should support encrypted file writing
  - Solution to R4 – *EncryptedFile* class
  - Destructive approach: add a new class and force the client to use different classes (possibly also different interface) for write operations
  - Constructive Approach:
    - $NewSystem = (\{EncryptedFile\}, \{\}) \oplus System$
    - Find the properties of the evolution problem (the context), then for the context find the mechanisms

Constructive Approach – How to Apply

University of Twente
department of computer science



Constructive Approach Applied to Integration Problem

University of Twente
department of computer science

- To apply the constructive approach we need to find the context of the evolution problem
  - The context contains parameters that details the evolution problem
  - We can find the details of the problem by looking at the relation and properties of S_{New} and S_{System}
  - We identified 3 parameters that detail the evolution problem
    - Characteristic of S_{New} (Sta)
    - Relationship between S_{New} and S_{System} (Rel)
    - Enviroment (Env)

Constructive Approach Applied to Integration Problem - Contexts

1. The status of S_{New} (Sta)
   1. Composition (C) – The change has occurred
   2. Extension (Ex) – extend the system with scenarios so that it can withstand anticipated changes
   3. Exception – We cannot find a solution for the new requirement

Constructive Approach Applied to Integration Problem - Contexts

2. The relationship between S_{New} (Rel)
   1. Non-overlapping (NO) – $\forall S_j \in S_{system}$, $S_j \cap S_{New} = \varnothing$
   2. Overlapping (O) – $\exists S_j \in S_{system}$, $S_j \cap S_{New} \neq \varnothing$
   3. Specialization (S) – $\exists S_j \in S_{system}$, $S_j \subset S_{New}$
   4. Interpertation (I) – $\exists S_j \in S_{system}$, $S_j \supset S_{New}$

3. The Environmental Factors (Env)
   1. Run-time adaptation (RA)
   2. Compile-Time adaptation (CA)
   3. Installation (In)

- The context of an evolution problem is a triple {Char, Rel, Env}
  - Char ranges over C, Ex
  - Rel ranges over NO, O, S, I
  - Env ranges over RA, CA, In

Constructive Approach Applied to Integration Problem - Contexts

- There are 36 contexts for evolution problems
  - Char takes 3 values, Rel takes 4 values, Env takes 3 values
- Not all possible combinations of the parameters give a feasible context
  - When Char=Ex, SNew doesn't exist; we can't find a value for Rel parameter
  - Thus we have 24 feasible contexts
- For each context, we listed applicable mechanisms from SE literature

Constructive Approach Applied to Integration Problem -Example

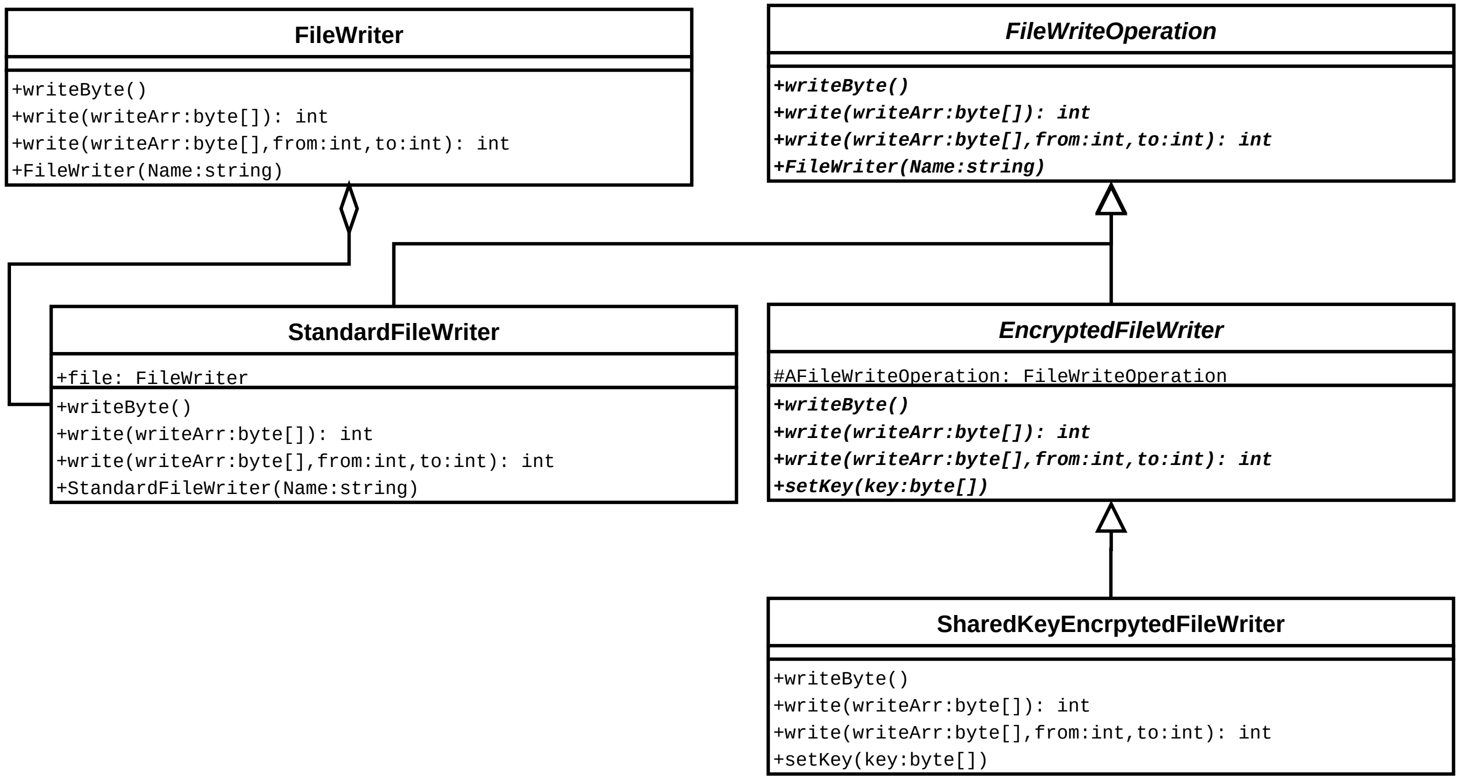
University of Twente
department of computer science

- Example PDA Input & Storage
  - R4 – The system should support encrypted file writing
  - Solution to R4 (S_{New})– *EncryptedFile* class
- The context of this evolution problem:
  - Char = C since the change has occurred
  - Rel = NO since S_{New} doesn't intersect with the any solution in S_{system}
  - Env = Assume we want to achieve this composition with compile time techniques (CA)
  - {C,NO,CA}

Constructive Approach Applied to Integration Problem -Example

University of Twente
department of computer science

- {C,NO,CA} – The mechanisms are polymorphic calls, decorator pattern,
  - The System is evolved using decorator pattern



Conclusion And Future Work

University of Twente
department of computer science

- Constructive Approach to Software Evolution allows the system to evolved without changing the initial design
  - Knowledge about the design stays the same
  - No design drift
- Future work
  - Mechanisms for removal and modification