Domain Specific Languages

Wave of the future

Jim Duey
@jimduey

From the metal (Imperative)

Object Oriented Programming (C++, Java)

Block Structured Programming (Pascal, C)

Higher Level Languages (Fortran, Algol)

Subroutines (Hardware Stack)

Assembly Language (Mnemonics)

Machine Language (Numbers)

Hardware

From the metal (Functional)

?

OOP

Structured

Languages

Haskell (Purity, Patterns)

LISP

Subroutines

Assembly

Machine Language

Hardware

From the metal (Functional)

Domain Specific Languages

OOP

Structured

Languages

Haskell (Purity, Patterns)

LISP

Subroutines

Assembly

Machine Language

Hardware

Signs of the times

- *Paul Graham*

  On Lisp, etc

  "Use Lisp to write a DSL to solve your problem"

  - paraphrase

Signs of the times

- *Alan Kay*

  View Points Research Institute

  vpri.org

  2011 Annual Report

Video: "Programming and Scaling"

Signs of the times

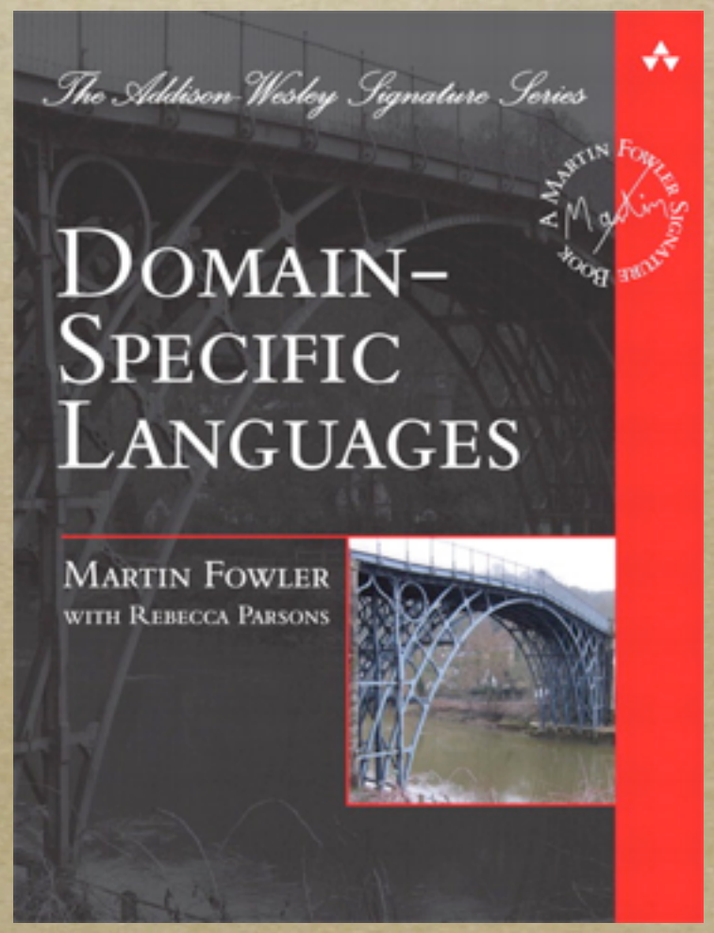
- *Simon Peyton-Jones*

  Videos. Particularly one with John Hughes in Sydney.

  “Haskell is great for implementing DSLs”

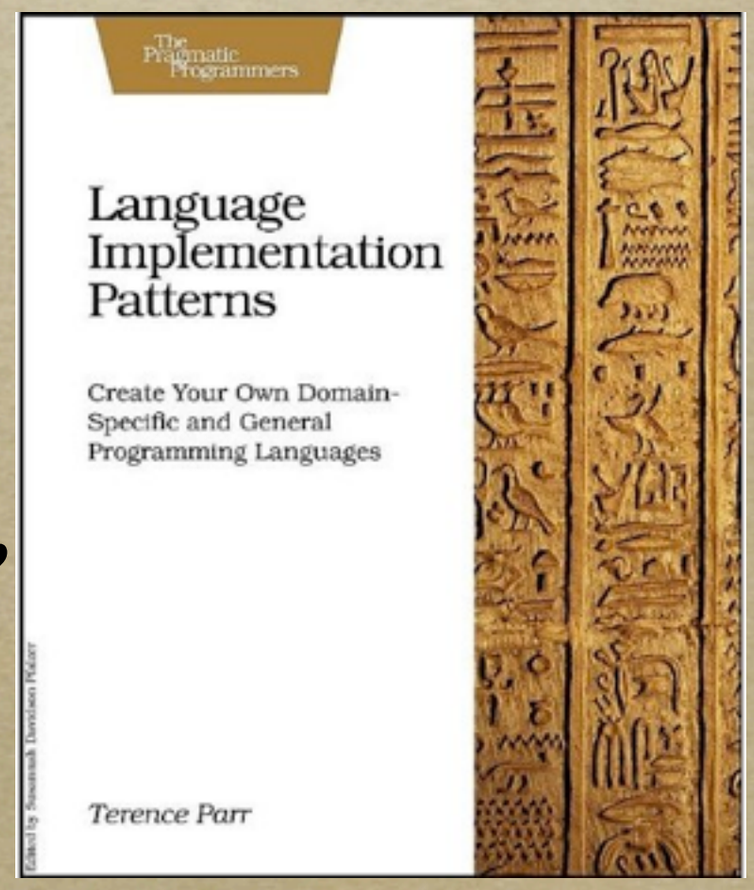
Writings

- *Martin Fowler*
  - *Domain Specific Languages*
  - *2010*
  - *Limited to OOP*
  - *Focused on providing DSLs for users.*



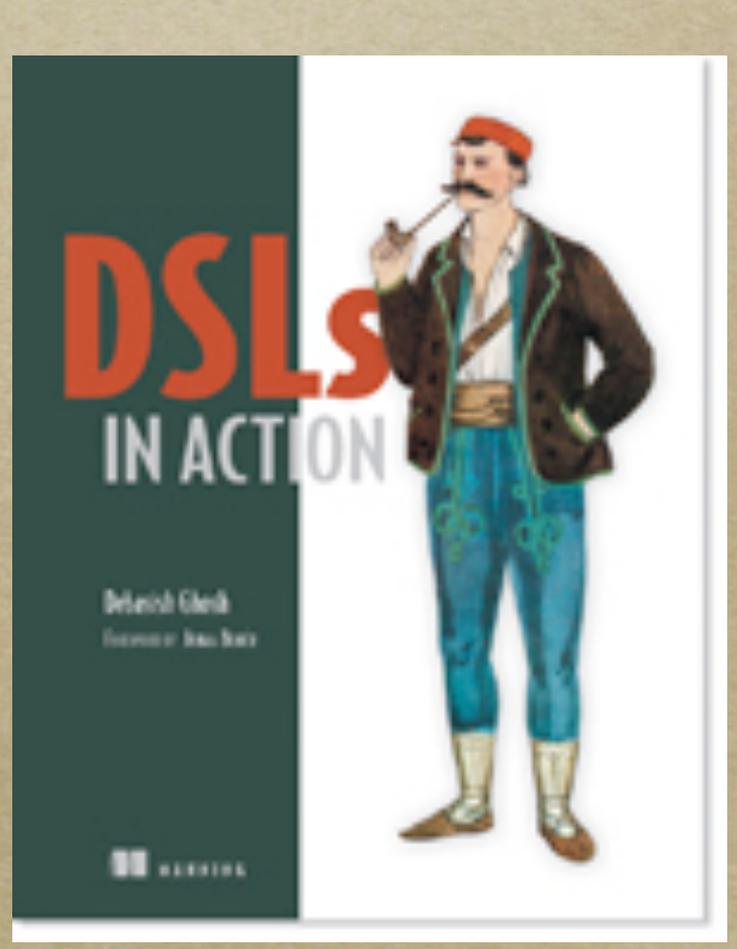
Writings

- *Terence Parr*
  - *Language Implementation Patterns*
  - *2010*
  - *Good info about implementing parsers, lexers, etc.*



Writings

- *Debasish Ghosh*
  - *DSLs in Action*
  - *2010*
  - *OOP and FP*
  - *Java, Clojure, Scala, Ruby*



How?

- *What I wanted was a process to create DSL's.*
- *A methodology.*

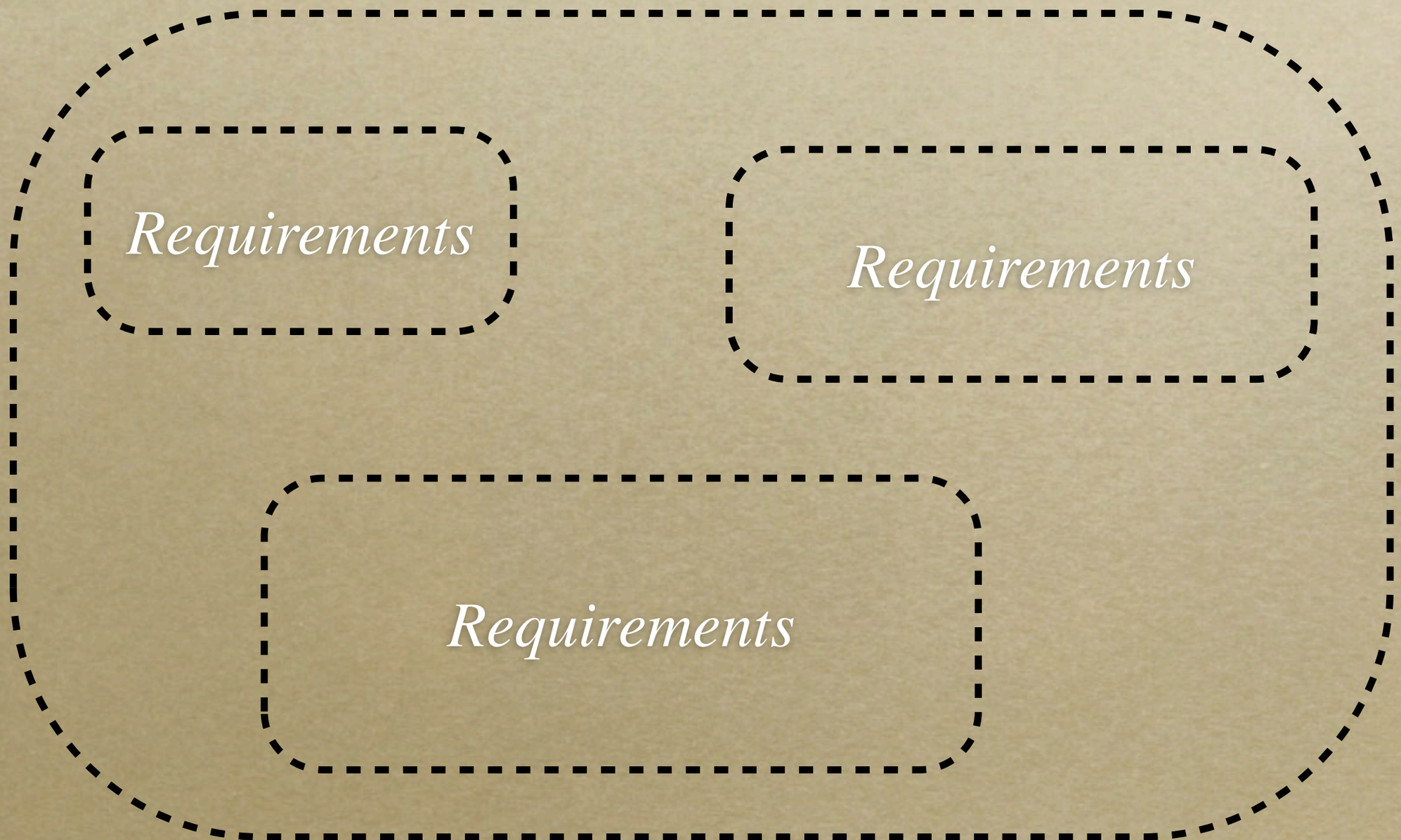
Writing Software

Solution

Requirements

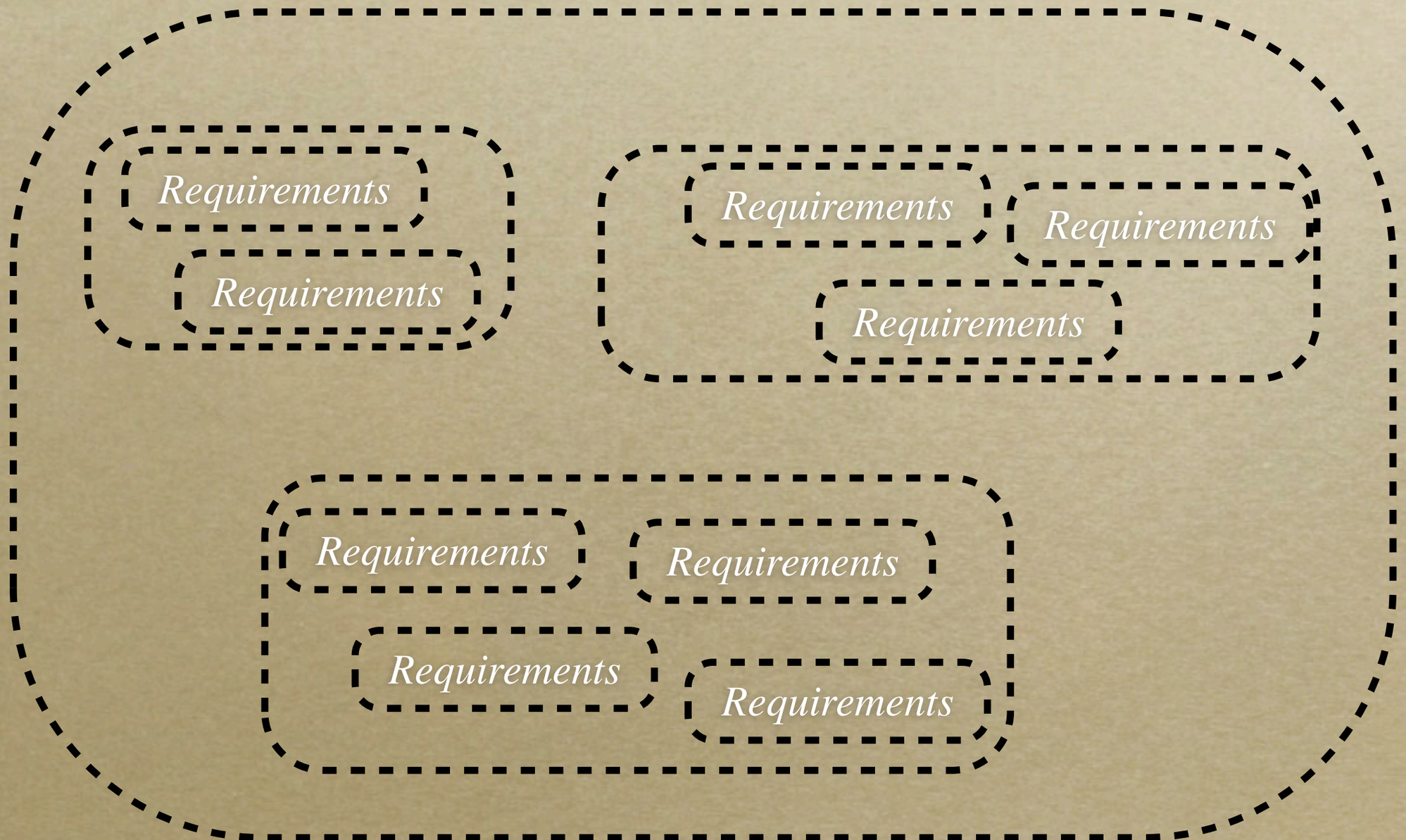
Writing Software

Solution

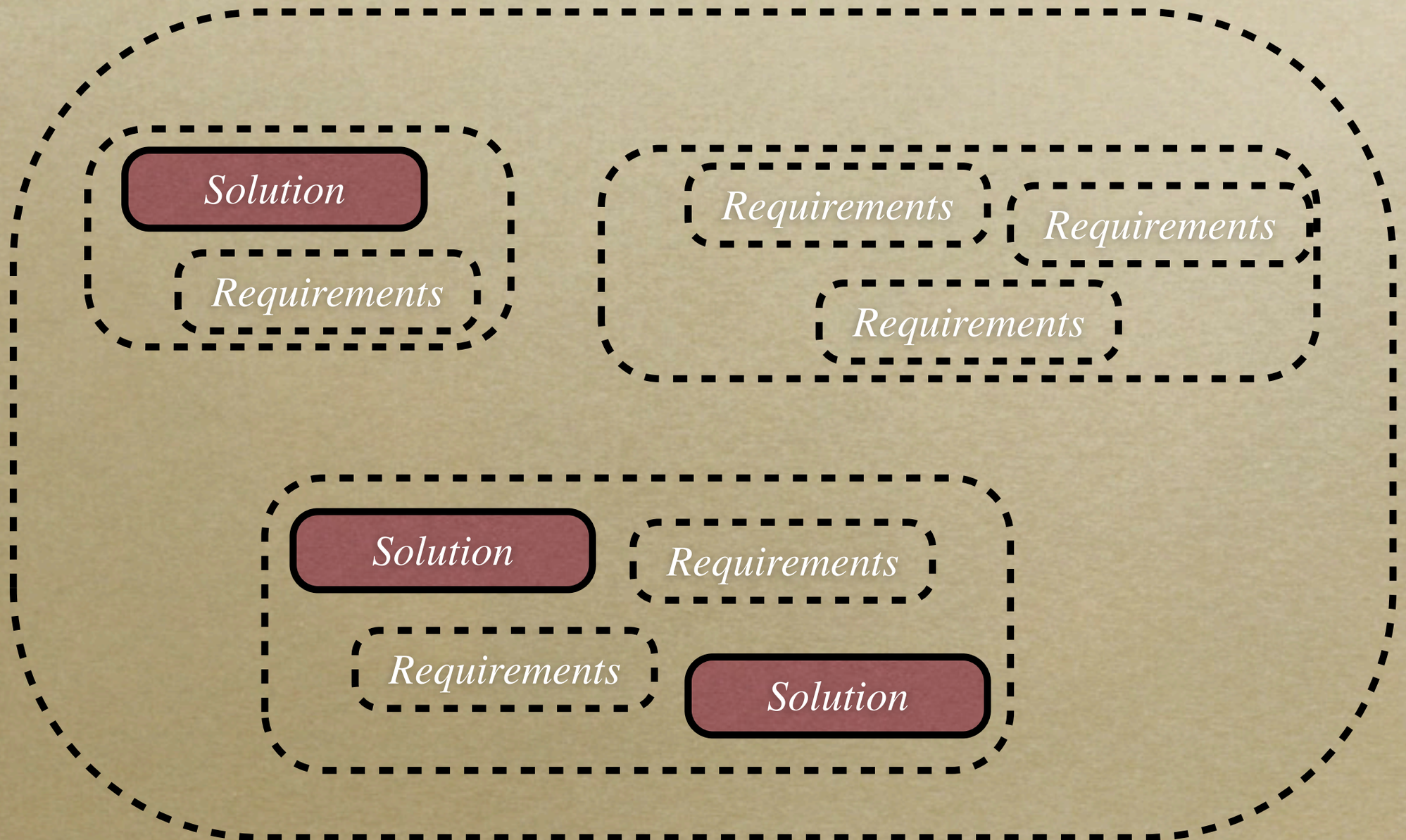


Writing Software

Solution

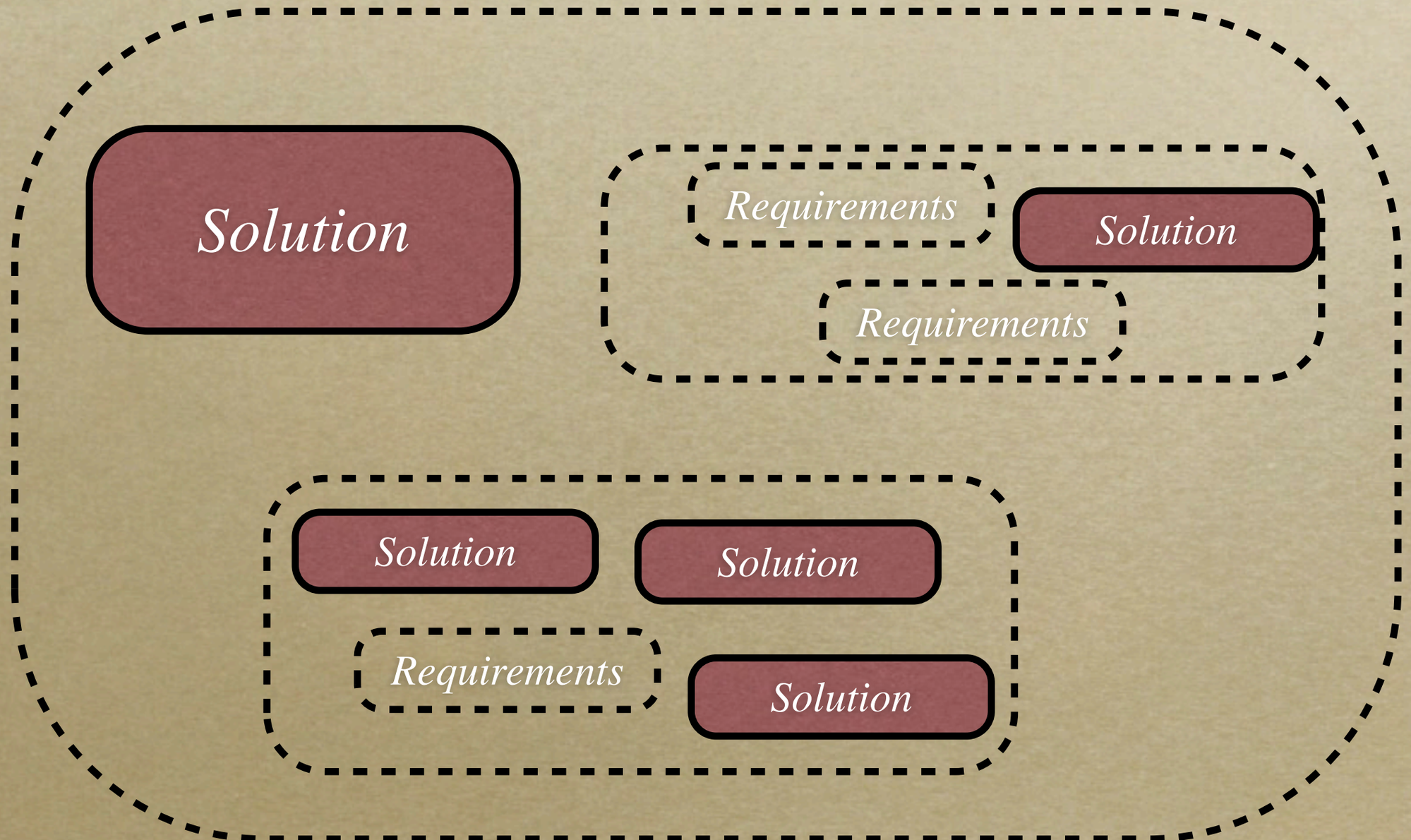


Writing Software



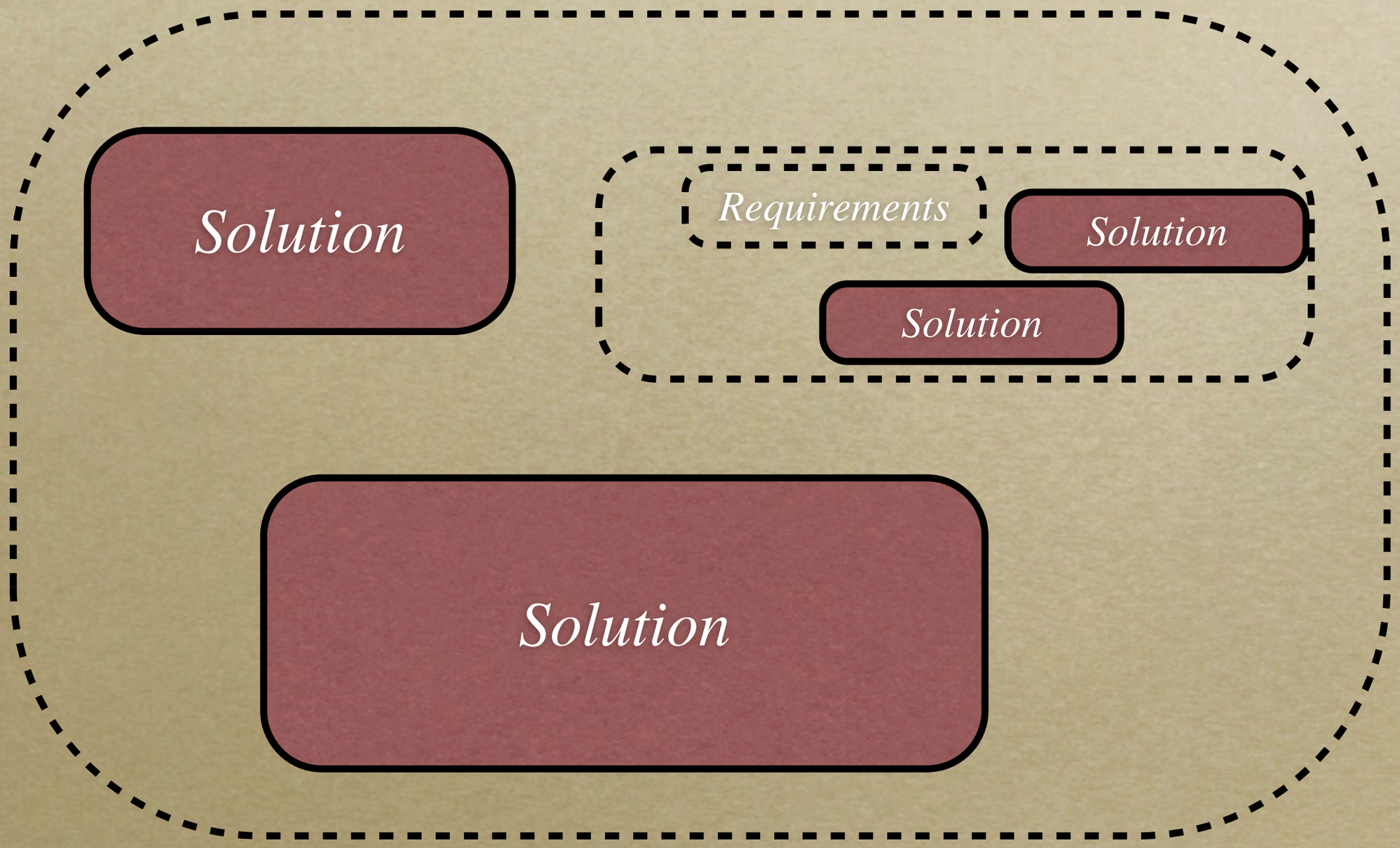
Writing Software

Solution



Writing Software

Solution



Writing Software

Solution

Solution

Solution

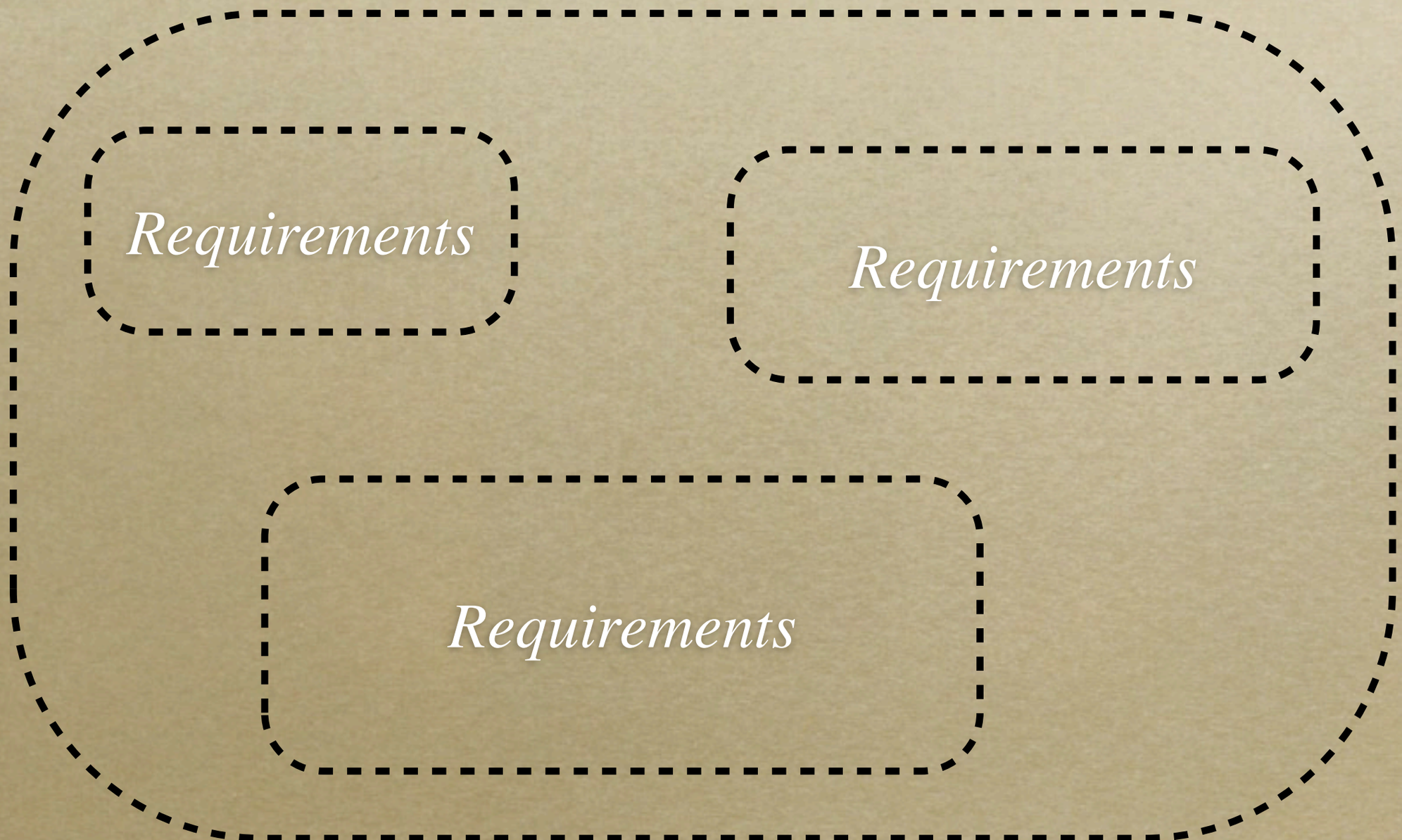
Solution

Writing Software

Solution

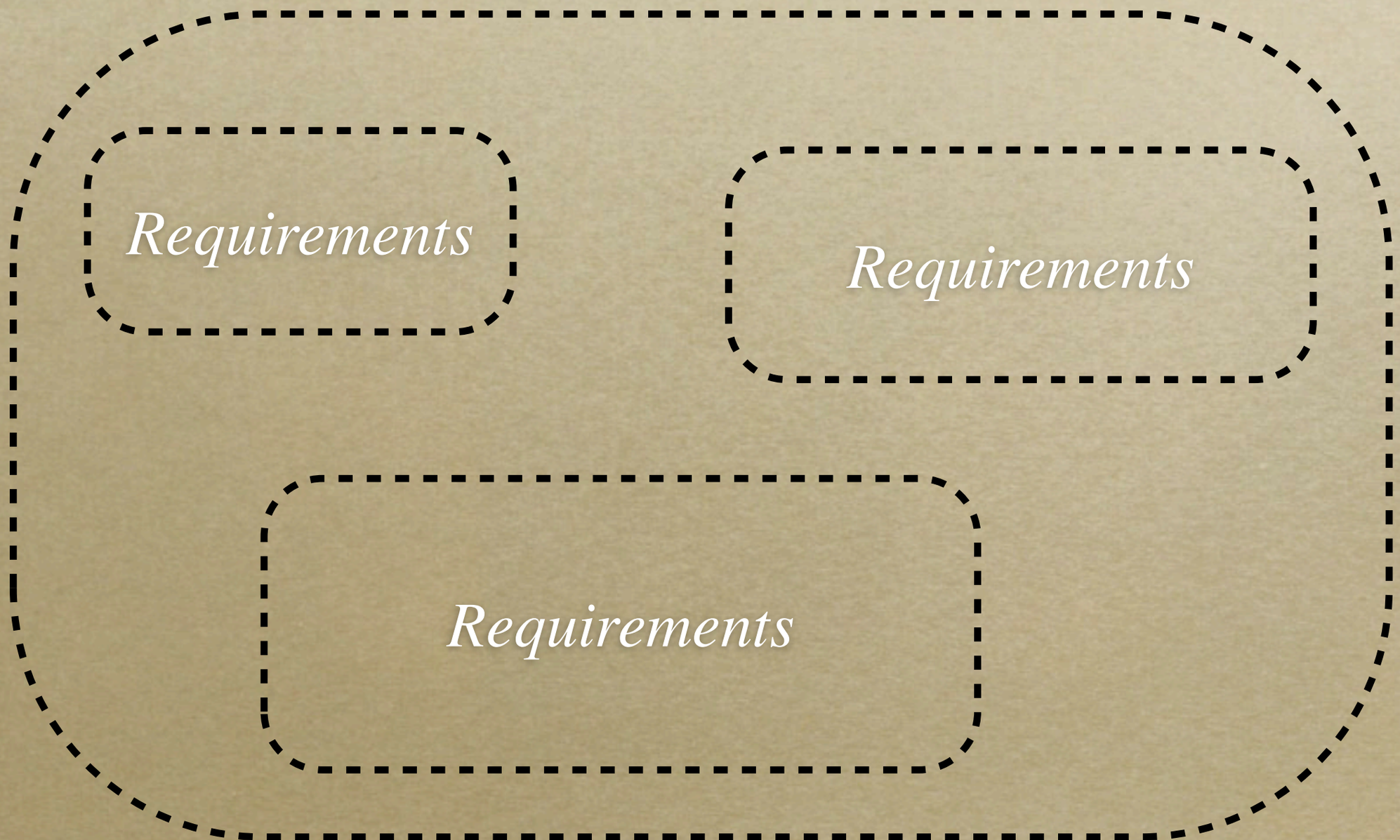
Unasked question

Solution



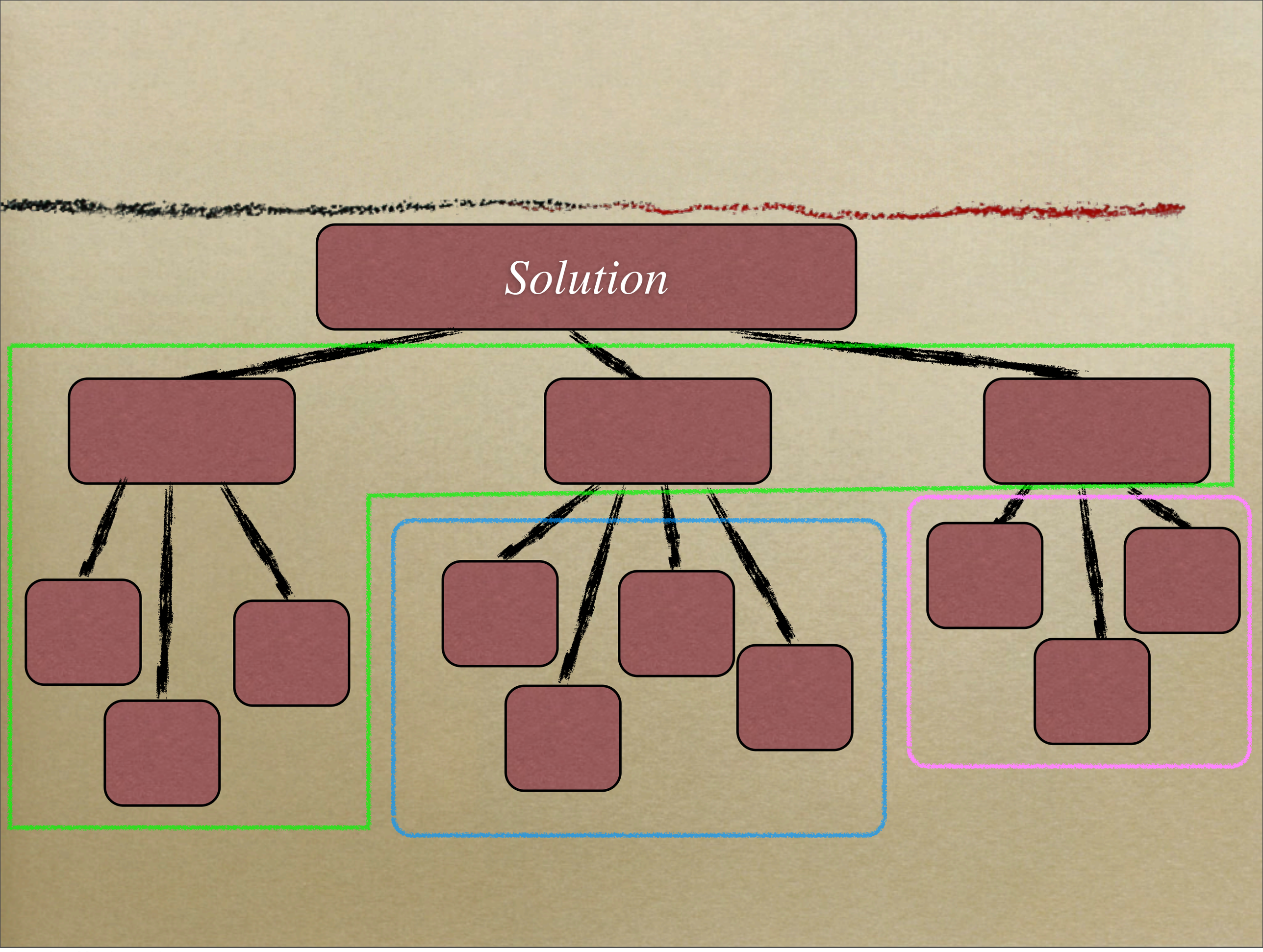
How do I put these back together?

Solution



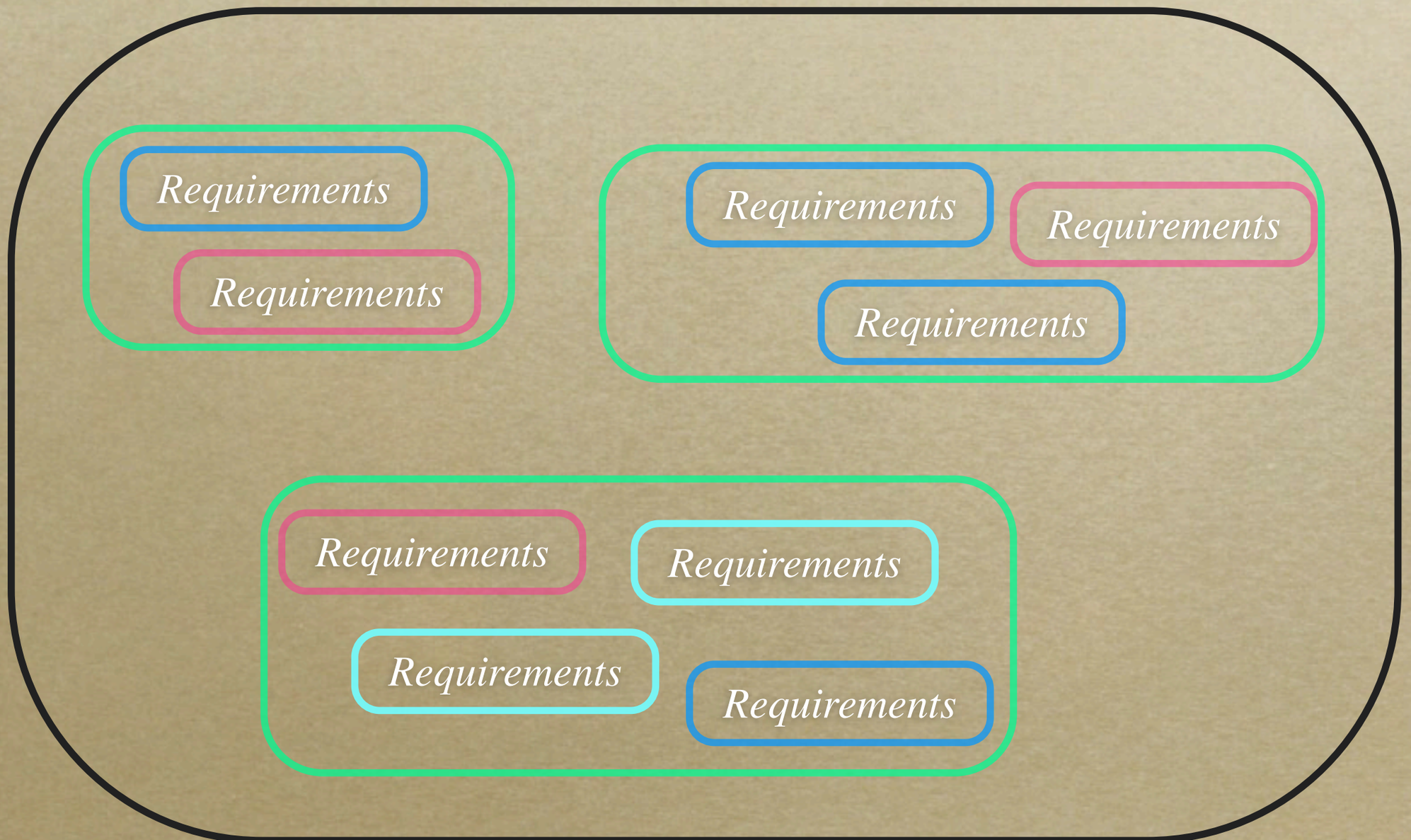
Solution

Solution



Writing Software

Solution



Some Real DSLs

Compojure

- *James Reeves (weavejester)*
- *https://github.com/weavejester/compojure*
- *Composes: HTTP routes/handlers*
- *Operator:* `routes`
- *Hides: matching request to handler*
- *run function: ring adapter*

```
(def r (GET "/" []
            "howdy, globe"))

(def s (GET "/sub" []
            "sub route"))

(def ss (GET "/sub/sub" []
             "sub, sub route"))

(def rs (routes r s))

(def app
  (handler/site (routes rs ss)))
```

Korma

- *Chris Granger (ibdknox)*
- *https://github.com/ibdknox/Korma*
- *Composes: query parts (*`where, aggregate`*)*
- *Operators: Clojure's -> macro*
- *Hides: building SQL queries*
- *run function:* `select`

```
(select users
  (aggregate (count :*) :cnt)
  (where (or (> :visits 20)
             (< :last_login
                a-year-ago))))
```

```
(def base
   (-> (select* "user")
     (fields :id :username)
     (where {:email [like "*@gmail.com"]})))

(def ordered-and-active
       (-> base
         (where {:active true})
         (order :created)))

(def constrained (-> ordered-and-active
                     (limit 20)))

(exec constrained)
```

core.logic

- *David Nolen (swannodette)*
- *https://github.com/clojure/core.logic*
- *Composes: relations (created by ==)*
- *Operators:* `fresh,all,conde,lcons`
- *Hides: backtracking, passing substitutions*
- *run function:* `run` *et.al*

What's the process?

- *Decompose your problem*
- *Decide what you want to say*
  - *What are you composing*
  - *How are you composing them*
- *What do you want to hide*
- *What are the boundaries/interfaces*

Rules of thumb

- *Don't implement a general purpose language*
- *Keep operators to a minimum*
- *Make operators return the same kind of thing as primitives*
  - *Results should serve as inputs to operators*
- *Allow assignment of results to symbols*
- *Leverage your host language*

In practice

A DSL to test Ring handlers

```
(script
    (request :get “/”)
    (return-code? 200)
    (body-contains? “some str”)

    (request :get “/illegal”)
    (return-code? 404))

(run-script app test-script)
```

Requirements

- *Requests and tests are types of actions*
- *Actions compose sequentially*
- *Need to pass a response through actions*
- *Also deal with cookies*

How to implement?

- *State monad*
- *Actions become monadic values*
  - `(fn [s] ... [v new-s])`
- `script` *is just* `m-seq`
- `script` *produces monadic values*
- *that can be assigned to symbols*

More requirements

- *Want to abort when a test fails*
- *Maybe monad*
- *Monad transformer*

Implementation

```
(def script-m
    (state-t maybe-m))

(defmacro script-let [bindings body]
  `(domonad script-m
        ~bindings
        ~body))

(defn script [& actions]
  (with-monad script-m
     (m-seq actions)))
```

Implementation

```
(defn request [method uri & [req-map]]
 (script-let
   [handler (fetch-val :handler)
    cookies (fetch-val :cookies)
    :let [request (assoc req-map
                         :request-method method
                         :uri url
                         :cookies cookies)
          response (handler request)]
   _ (set-val :response response)
   _ set-cookies]
response)
```

Implementation

```
(defn body-contains? [strn]
  (script-let
    [response (get-val :response)
     :when (.contains (:body response) strn)]
    true))

(defn response-code? [code]
  (script-let
    [response (get-val :response)
     :when (= code (:status response))]
    true))
```

Implementation

```
(def set-cookies
  (script-let
    [cookies response-cookies
     _ (update-val :cookies merge cookies)]
    cookies))

(defn run-script [handler script]
  (-> {:handler handler}
    script   ;; run the script on the handler
    first    ;; throw away the final state
    last     ;; get final result
    boolean)) ;; convert to a boolean
```

In practice

A DSL to test Ring handlers

```
(script
    (request :get “/”)
    (return-code? 200)
    (body-contains? “some str”)

    (request :get “/illegal”)
    (return-code? 404))

(run-script app test-script)
```

Usage

```
(def counter (atom 0))

(defn home []
  (swap! counter + 1)
  {:status  200
   :headers {"Content-Type" "text/html"}
   :body (str "counter: " @counter)})

(defroutes main-routes
           (GET "/" [] (home))
           (route/not-found
              "Page not found"))

(def app
  (handler/site main-routes))
```

Usage

```
(def test-script
     (script
       (request :get "/wrong")
       (response-code? 404)))

(deftest test-not-found-route
         (is (run-script app test-script)))
```

Usage

```
(def hit-counter
  (script
    (click "/")
    (response-code? 200)))

(def test-script
  (script
     hit-counter
     (body-contains? "counter: 1")

     hit-counter
     (body-contains? "counter: 2")

     hit-counter
     (body-contains? "counter: 3")))
```

From the metal (Functional)

Domain Specific Languages

OOP

Structured

Languages

Haskell (Purity, Patterns)

LISP

Subroutines

Assembly

Machine Language

Hardware

More Info

- *Blog - http://clojure.net*
- *Twitter - @jimduey*
- *Code - https://github.com/jduey/appraiser*