ESPERIMENTI DI DDD IN C++

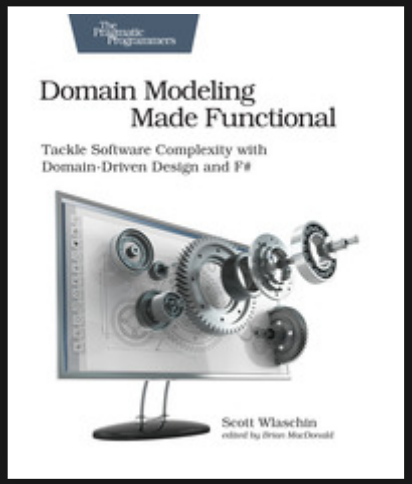
- Domain Driven Design
- Event storming
- Pianificatore meetup in C++

YURI VALENTINI

- SW Windows e Linux
- Linguaggi: C/C++, C#, Python
- Videoconferenza e VOIP
- yuroller [AT] gmail.com
- https://github.com/yuroller

DOMAIN MODELING MADE FUNCTIONAL

Scott Wlaschin



OBIETTIVI

- provare design con DDD
- sperimentare con C++17
- optional, expected, variant
- stili alternativi cpp (auto, east const)
- cmake multipiattaforma

DDD

- applicazioni business/enterprise
- workflow e gestionali
- **NON** object oriented design
- **NON** database oriented design

DDD IN BREVE 1

Porre attenzione:

- su eventi business
- **NON** su strutture dati

DDD IN BREVE 2

Partizionare il problema in sottodomini

Dominio: area di cui l'utente è esperto

DDD IN BREVE 3

Creare un modello per ciascun sottodominio

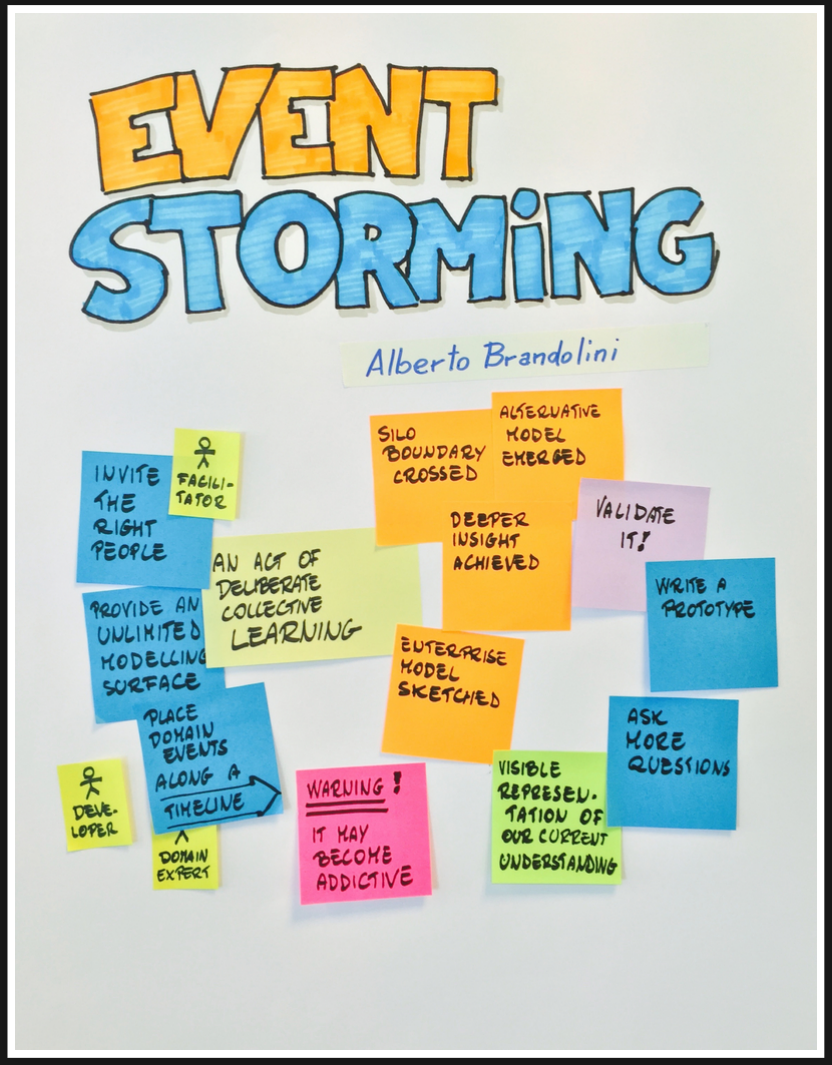
Modello: astrazione che coglie gli aspetti rileventi del problema

DDD IN BREVE 4

Sviluppare un Ubiquitous Language condiviso

NON si parla di:

- ProposalManager
- CalendarHelper
- AvailabilityFactory


EVENT STORMING
Alberto Brandolini
INVITE THE RIGHT PEOPLE
FACILI-TATOR
AN ACT OF DELIBERATE COLLECTIVE LEARNING
PROVIDE AN UNLIMITED MODELLING SURFACE
PLACE DOMAIN EVENTS ALONG A TIMELINE
DEVE-LOPER
DOMAIN EXPERT
SILO BOUNDARY CROSSED
ALTERNATIVE MODEL EMERGED
DEEPER INSIGHT ACHIEVED
VALIDATE IT!
WRITE A PROTOTYPE
ENTERPRISE MODEL SKETCHED
ASK MORE QUESTIONS
WARNING!
IT MAY BECOME ADDICTIVE
VISIBLE REPRESEN-TATION OF OUR CURRENT UNDERSTANDING

EVENTI DDD

- arancione
- verbo al passato
- qualcosa di significativo succede nel dominio

EVENTI PIANIFICATORE

- Letto/Validato file date meetup
- Ricevuto/Validato elenco relatori
- Ricevuto/Validato elenco proposte
- Proposte votate
- Calendario meetup creato
- Calendario approvato
- Calendario pubblicato
- Relatori notificati

COMANDI DDD

- azzurro
- conseguenza di azione utente
- causano eventi DDD

COMANDI PIANIFICATORE

- Avvia programma
- Vota proposte
- Approva calendario

SISTEMI ESTERNI DDD

- viola
- provengono dall'esterno
- conseguenza del tempo che passa
- causano eventi DDD

SISTEMI ESTERNI PIANIFICATORE

- valutatore proposte

AGGREGATI DDD

- giallo
- raggruppa comandi ed eventi
- riceve un comando
- decide se eseguirlo
- genera eventi DDD

AGGREGATI PIANIFICATORE

- DateMeetup
- Relatori
- Proposte
- Calendario

OBIETTIVI PIANIFICATORE IN C++

- rappresentare modello DDD
- logica workflow
- sfruttare il sistema di tipi c++
- gestire errori nel percorso del workflow
- no raw-pointers/nullptr

AVVERTENZE

- codice non adatto per la produzione
- alcune idee sono ancora da sviluppare
- ci possono essere problemi di performance
- usa classi c++ non ancora standardizzate
- verificare se "resiste" ai cambiamenti

IMPLEMENTAZIONE C++

- classi differenti per entità in diverse fasi
- es. ElencoProposteDto, ElencoProposte, ElencoProposteVotate
- dati immutabili che si trasformanto nel workflow
- no exception (interrompono il flusso)
- istanze delle classi sempre valide (no .isValid())

WORKFLOW 1

Acquisisci Input (side-effect)

- LeggiDateMeetup() → DateMeetupDto
- RiceviRelatori() → RelatoriDto
- RiceviProposte() → ProposteDto

WORKFLOW 2

Valida Dto (pure)

- DateMeetupDto → ValidaDM() → DateMeetup
- RelatoriDto → ValidaRelatori() → Relatori
- ProposteDto → ValidaProposte() → Proposte

WORKFLOW 3

Business (pure)

- (Proposte,Relatori) → Vota() → ProposteVotate
- (ProposteVotate,DateMeetup) → OrganizzaCal() → Calendario
- Calendario → SerializzaCal() → CalendarioDto

WORKFLOW 4

Emetti Output (side-effect)

- CalendarioDto → PubblicaCal()
- Calendario → NotificaRelatori()

WORKFLOW COMMENTI

- side-effect → pure → side-effect
- core è testabile facilmente
- tipi specifici
  1. impediscono stati non rappresentabili
  2. servono da documentazione
  3. contengono solo quello che serve

ELENCO PROPOSTE DTO

- Data Transfer Object
- Rappresenta il file di memorizzazione
- Nel nostro caso lo creiamo da un file json
- Lo rappresentiamo come `std::vector<PropostaDto>`

ELENCO PROPOSTE DTO JSON

```
[{
  "codice": 1000,
  "codice_relatore": "yv",
  "titolo": "Esperimenti di DDD in C++",
  "sommario": "Da Event Storming a ...",
  "disponibilita": {
    "tipo": "date_specifiche",
    "date_specifiche": [ "2019/9/12" ]
  }
}]
```

ELENCO PROPOSTE DTO C++

```
class PropostaDto {
public:
  static auto CreaElenco(string const &json)
        -> expected<vector<PropostaDto>, Errore>;
  // getters ...
private:
  int codice_;
  string codiceRelatore_;
  string titolo_;
  string sommario_;
  TipoDisponibilita tipoDisponibilita_; // enum class
  optional<vector<year_month_day>> dateDisponibilita_;
  optional<year_month_day> dataInizioDisponibilita_;
};
```

TIPO DISPONIBILITA

```
enum class TipoDisponibilita {
  SempreDisponibile,
  DateSpecifiche,
  DopoUnaData
};
```

ERRORE

```
class Errore {
public:
  explicit Errore(string testo)
    : testo_(move(testo)) {}
  auto testo() const
    -> string { return testo_; }
private:
  string testo_;
  friend auto operator<<(ostream &os, Errore const &err)
    -> ostream &;
};
```

YEAR_MONTH_DAY

- Howard Hinnant date library
- libreria header-only
- basata su `std::crono`
- attenzione a date non valide
  `date::year(2019)/2/30`

UTILIZZO YEAR_MONTH_DAY

```
using namespace date;
auto d = year(2019) / 9 / 12;
d.ok(); // true
d.year(); // year(2019)
d.month(); // month(9)
d.day(); // day(12)
```

OPTIONAL

- disponibile in c++17 ma uso versione TartanLlama
- implementa P0798R0: Monadic operations for std::optional
- libreria header-only
- tipo "somma" fra T e Unit
- concatenabile

UTILIZZO OPTIONAL C++17

```
using namespace std; // tl
auto a = optional<int>(5); // make_optional(42)
a.has_value(); // true
*a; // 5

auto b = optional<int>(nullopt);
b.has_value(); // false
*b; // Undefined Behaviour
b.value_or(11); // 11
```

OPTIONAL CONCATENAZIONE

```
using namespace std;
class Tweet;

auto leggiIndirizzo() -> optional<string>;
auto componiTweet(string const &indirizzo, string const &testo
  -> optional<Tweet>;
auto inviaTweet(const Tweet& tweet) -> bool;
```

STD::OPTIONAL COMPOSIZIONE

```cpp
auto notificaConTweetCpp17() -> bool
{
  auto indirizzo = leggiIndirizzo();
  if (!indirizzo)
    return false;

  auto tweet = componiTweet(*indirizzo, "tweet di notifica");
  if (!tweet)
    return false;

  return inviaTweet(*tweet);
}
```

TL::OPTIONAL CONCATENAZIONE

```
auto notificaConTweetTl() -> bool
{
  auto res = leggiIndirizzo()
    .and_then([](string const& indirizzo) {
      return componiTweet(indirizzo, "tweet di notifica");
    })
    .map(inviaTweet);
  return res.value_or(false);
}
```

EXPECTED

- uso versione TartanLlama
- estende P0323R3: Utility class to represent expected object
- libreria header-only
- tipo "somma" fra T e TErrore
- concatenabile

EXPECTED BASE

```
using namespace tl;
enum class Err { FileNonTrovato, Generico, Conversione };

auto a = expected<int, Err>(5);
a.has_value(); // true
*a; // 5;

// auto b = tl::expected<int, Err>(
//       tl::unexpected(Err::Conversione));
tl::expected<int, Err> b
  = tl::make_unexpected(Err::Conversione);
b.has_value(); // false;
b.error(); // Err::Conversione
```

EXPECTED CONCATENAZIONE 1

```
using namespace tl;
class Errore;
class Tweet;

auto leggiIndirizzo() -> expected<string, Errore>;
auto componiTweet(string const& indirizzo, string const& testo
  -> expected<Tweet, Errore>;
auto inviaTweet(const Tweet& tweet) -> bool;
```

EXPECTED CONCATENAZIONE 2

```
// expected<std::monostate, Errore>
auto notificaConTweetExp() -> optional<Errore> {
  auto res = leggiIndirizzo()
    .and_then([](string const& indirizzo) {
      return componiTweet(indirizzo, "tweet di notifica");
    })
    .map(inviaTweet);

  if (!res) {
    return nullopt;
  }

  return res.error();
}
```

TIPI ELEVATI

- es. `expected<T,E>`, `optional<T>`, `vector<T>`
- concatenabili con `.and_then()`, `.map()` o template appositi
- adattatore per funzione T → C applicabile a tipi elevati (es. vector<T> → vector<C>)
- vedi FoldExpected, Apply nel codice di esempio
- range-v3 rende concatenabili i container

TIPI UNIONE

DisponibilitaRelatore:

- QualunqueData
- DateSpecifiche(lista di Date)
- DopoUnaData(Data)

VARIANT

```
auto intOrStr = variant<int, string>(15);
auto n = get<int>(intOrStr); // 15
auto s = get<string>(intOrStr); // exception
```

STD::VISIT

```
template<class... Ts> struct overloaded : Ts... {
  using Ts::operator()...;
};
template<class... Ts> overloaded(Ts...)->overloaded<Ts...>;

visit(overloaded{
  [](int num) { cout << num + 1 << endl; },
  [](std::string const& str) { cout << str << endl; },
}, IntOrStr);
```

MIGLIORAMENTI

- diversi tipi di errore
- tipi per gli elenchi proposti e non `std::vector`
- finire il codice per fare un eseguibile funzionante
- funzioni asincrone

FINE

- Domande?
- Dubbi?
- Opinioni?
- Grazie