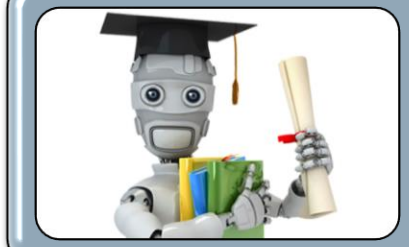
TSTL

TypeScript Standard Template Library

https://github.com/samchon/tstl

Index



Outline

- TypeScript-STL
- Standard Template Library
- In My Case

Features

- Containers & Iterators
- Algorithms
- Functors

Miscellaneous

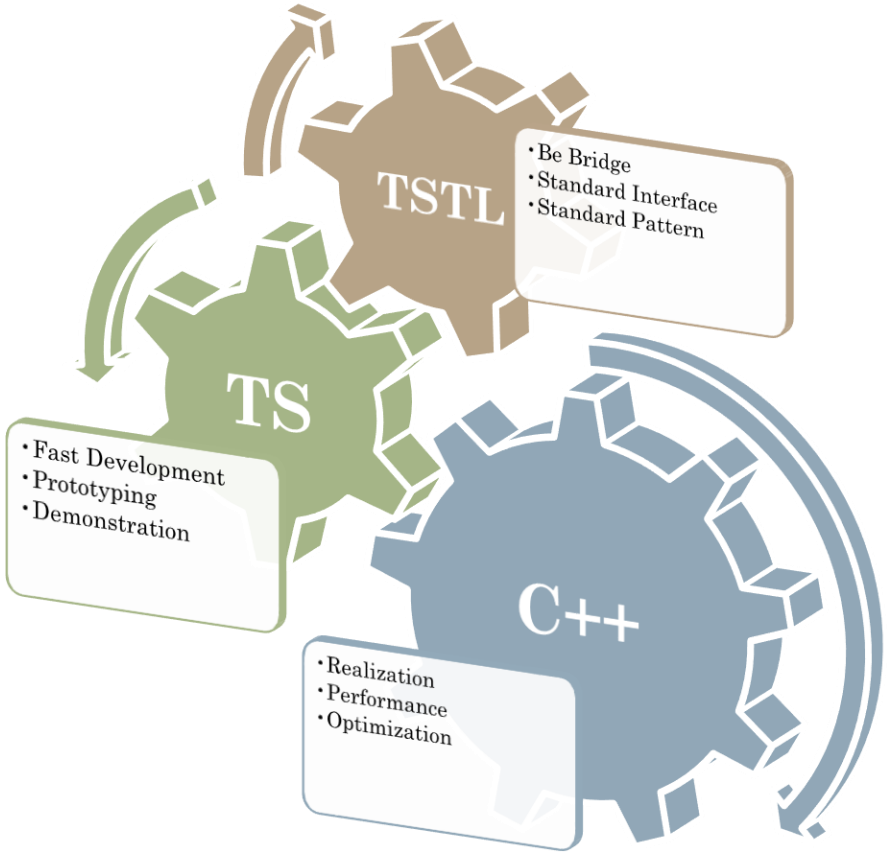
- Releases
- Utilization Cases
- Blueprint

Outline

1. TypeScript-STL
2. Standard Template Library
3. In My Case

1. TypeScript-STL

- **TypeScript** 의 **STL** 구현체
- Features
  - Containers
  - Iterators
  - Algorithms
  - Functors
- Repository
  - https://github.com/samchon/tstl
  - Made by Samchon since 2016



2. Standard Template Library

Defined by C++ Committee
(interface only)

2. Standard Template Library

Implemented by
Someone Else

2. Standard Template Library

- (C++) 표준 템플릿 라이브러리
- 다음 4개 부문으로 나뉨
  - Containers
  - Iterators
  - Algorithms
  - Functors

- **C++ Standard Committee**
  - 3년마다 표준안 개정
  - 단, 인터페이스만 정의한다.
  - 구현체는 각자 알아서 만들어야 함
- **Implemented by**
  - Microsoft STL
  - LLMV LibC++
  - LibSTD
  - EASTL
  - **TSTL**

2. Standard Template Library

- **Containers & Iterators**
  - C++ 의 *Collection Framework*
  - 다양한 컨테이너와
  - 정규화된 *Iterator Pattern* 을 제공함
- **Algorithms**
  - (함수형) 알고리즘 함수들
  - sortings, mathematics, partitions 등
- **Functors**
  - 기타 유틸리티 객체들

- **C++ Standard Committee**
  - 3년마다 표준안 개정
  - 단, 인터페이스만 정의한다.
  - 구현체는 각자 알아서 만들어야 함
- **Implemented by**
  - Microsoft STL
  - LLMV LibC++
  - LibSTD
  - EASTL
  - **TSTL**

3. In My Case

Fast Development

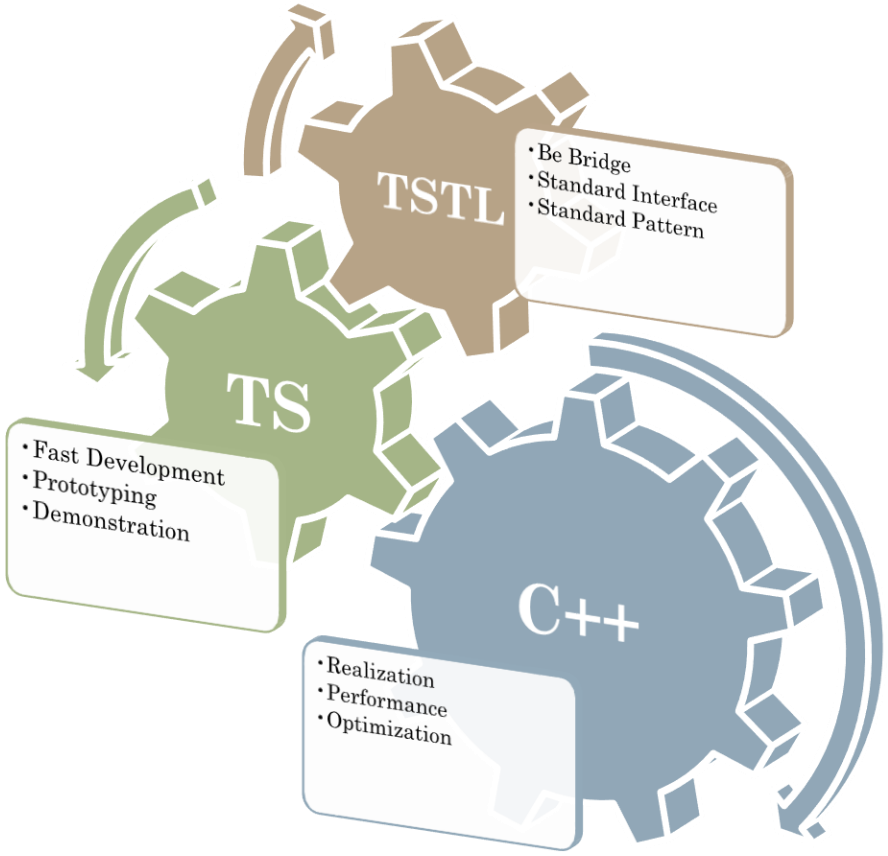
by JS + HTML

3. In My Case

Optimization
by C++ Migration

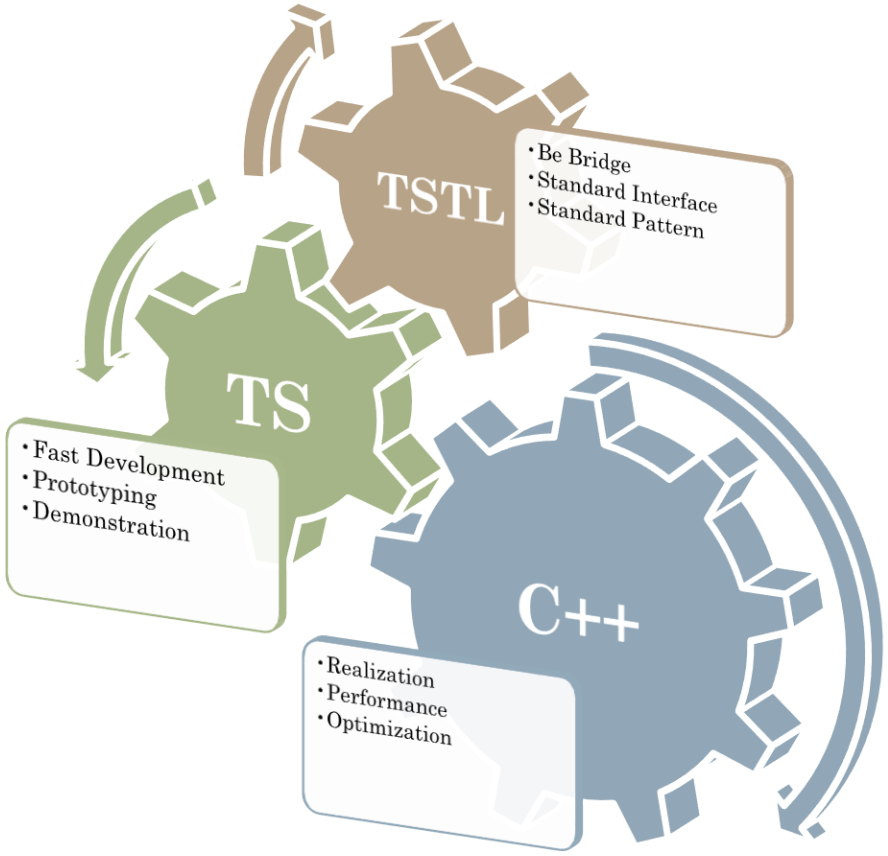
3. In My Case

- 본인은 알고리즘 개발자
- 본인의 개발 프로세스 (스타일)
  - Fast Development by JavaScript
    - Prototyping
    - Simulation in data level
  - Proof & Visualize by HTML
    - Production
    - Demonstration with visualization
    - High-level Simulation
  - Optimization by Migration to C++
    - Migration
      - Consider efficiency
      - Partial or Full migration
    - Integration with TS/HTML



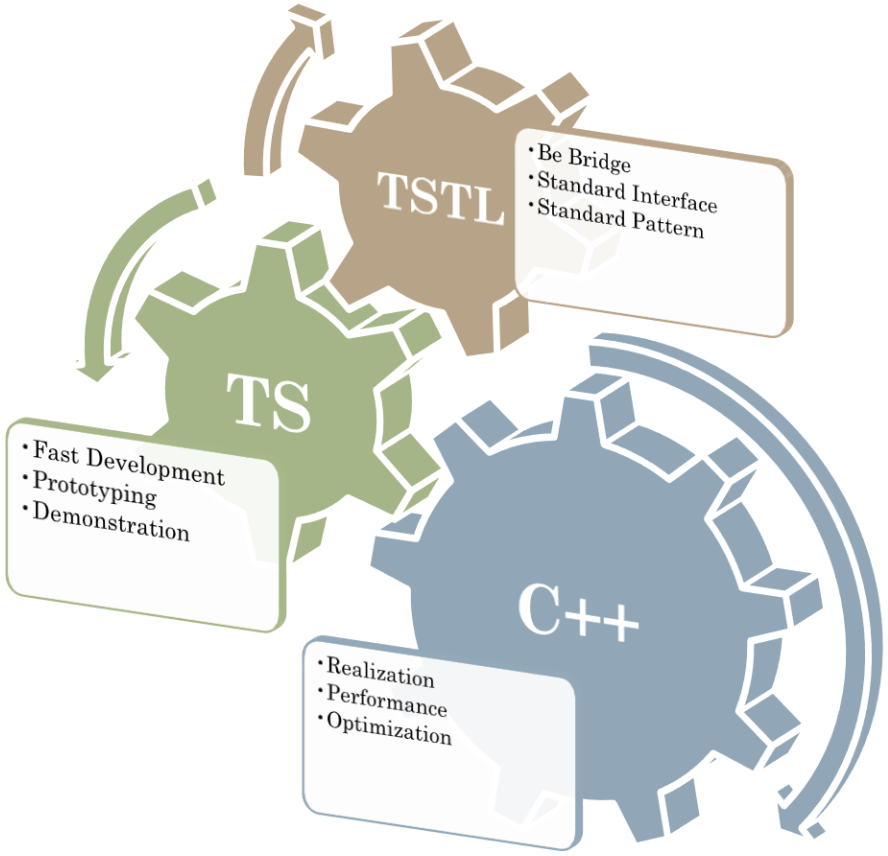
3. In My Case

- Migration 할 때마다 불편하더라
  - 기저 라이브러리의 인터페이스가 다름
    - 물론, JS 는
    - 기저 라이브러리라 할 만한 게 없다
  - 이게 같다면, 얼마나 편리할까?
    - 동일한 자료구조
    - 동일한 패턴
    - 동일한 인터페이스
- 그래서 만들었다; TSTL
  - Prototyping by TypeScript
  - Optimize by C++
  - Integrate them with TSTL



3. In My Case

- JS 는 기저 라이브러리가 너무 부실하다
  - 제공되는 Containers 는 달랑
    - Array 와
    - Dynamic Object 뿐
  - 알고리즘을 구현하기 위한
    - 기본 제공 함수들도 너무 적음
  - 유틸리티 객체들도
    - 너무 단조로우며
    - 마찬가지로 너무 적다
- 그래서 만들었다; TSTL
  - Containers & Iterators
  - Algorithms
  - Functors



3. In My Case

자기개발

Features

1. Containers & Iterators
2. Algorithms
3. Functors

1. Containers & Iterators

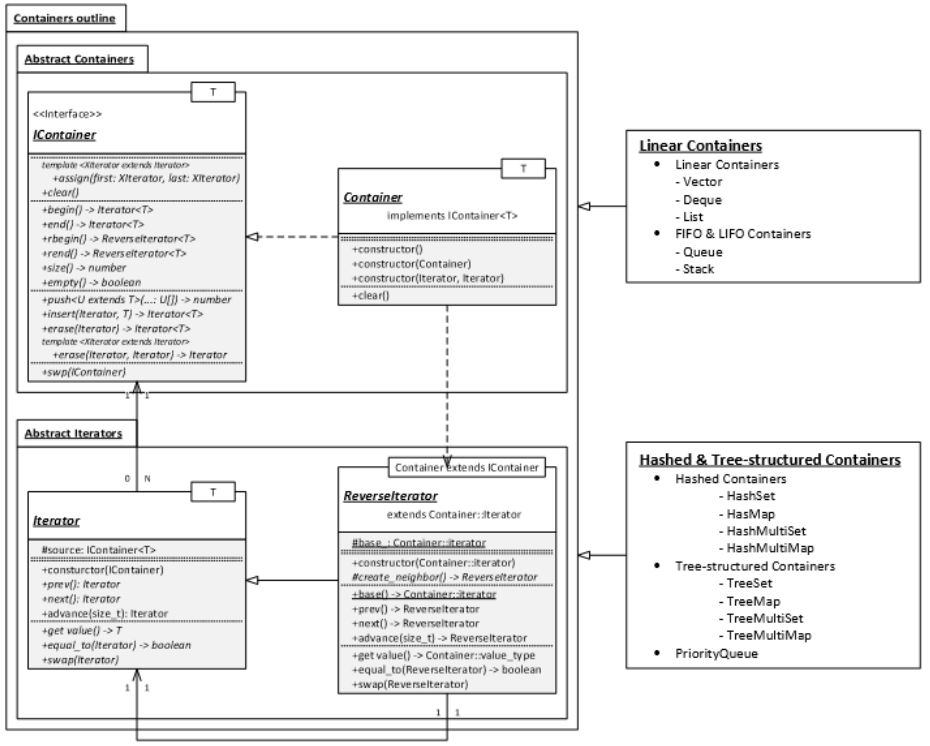
Collection Framework
with Iterator Pattern

1. Containers & Iterators

Description

- *Java* 의 *Collection Framework* 격
  - 개인적인 의견으로는
  - Collection Framework 보다 우수
    - 일관성
    - 적절성
    - 확장성
- 다채로운 Containers 와
- 정규화된 *Iterator pattern* 을 제공

Features



1. Containers & Iterators

Description

- *Java* 의 *Collection Framework* 격
  - 개인적인 의견으로는
  - Collection Framework 보다 우수
    - 일관성
    - 적절성
    - 확장성
- 다채로운 Containers 와
- 정규화된 *Iterator pattern* 을 제공

Features

- Linear Containers
  - vector
  - list
  - deque
- Adaptor Containers
  - stack
  - queue
  - priority_queue

1. Containers & Iterators

Description

- *Java* 의 *Collection Framework* 격
  - 개인적인 의견으로는
  - Collection Framework 보다 우수
    - 일관성
    - 적절성
    - 확장성
- 다채로운 Containers 와
- 정규화된 *Iterator pattern* 을 제공

Features

- Associative Containers
  - Atomic Containers
    - set — *TreeSet*
    - multiset — *TreeMultiSet*
    - unordered_set — *HashSet*
    - unordered_multiset — *HashMultiSet*
  - Dictionaries
    - map — *TreeMap*
    - multimap — *TreeMultiMap*
    - unordered_map — *HashMap*
    - unordered_multimap — *HashMultiMap*

2. Algorithms

Description

- 다양한 알고리즘 함수들의 모음
  - 일관된 패턴을 지니며,
  - 용이성과 확장성이 우수함
- 크게 두 가지 특성을 가짐
  - Iterator Pattern
  - Functional Programming

Features

- iterations
- modifiers
- sortings
- heaps
- binary_searches
- partitions
- union_sets
- mathmatics

3. Functors

Utility Objects

3. Functors

Description

- 다양한 유틸리티성 객체들의 모음
- 다른 말로, 기타 (miscellaneous)

- STL 이 개정될수록 Functors 가 커짐
- 곧 추가될 모듈 (예정)
  - File System (17)
  - Standard UI Library (20)
  - Standard Network Library (20)

Features

- <exception>
  - 예외사항 객체들
- <functional>
  - binary functions
  - hash functions
  - bitsets
- <utility>
  - swap
  - Pair, Entry 등

3. Functors

Description

- 다양한 유틸리티성 객체들의 모음
- 다른 말로, 기타 (miscellaneous)

- STL 이 개정될수록 Functors 가 커짐
- 곧 추가될 모듈 (예정)
  - File System (17)
  - Standard UI Library (20)
  - Standard Network Library (20)

Features

- <thread>
  - Asynchronous Critical Section Objects
  - 비동기 이벤트 간 임계영역 제어
  - JS 나름의 스레드 흉내

  - condition_variable
  - mutex
  - timed_mutex
  - shared_mutex
  - shared_timed_mutex
  - experiments.semaphore

Supports

1. Releases
2. Utilization Cases
3. Blueprint

1. Releases

- Github
  - Repository: https://github.com/samchon/tstl
  - Issues: https://github.com/samchon/tstl/issues
  - Version histories: https://github.com/samchon/tstl/releases
- NPM Module: https://npmjs.com/tstl
  - npm install --save tstl
  - npm test

1. Releases

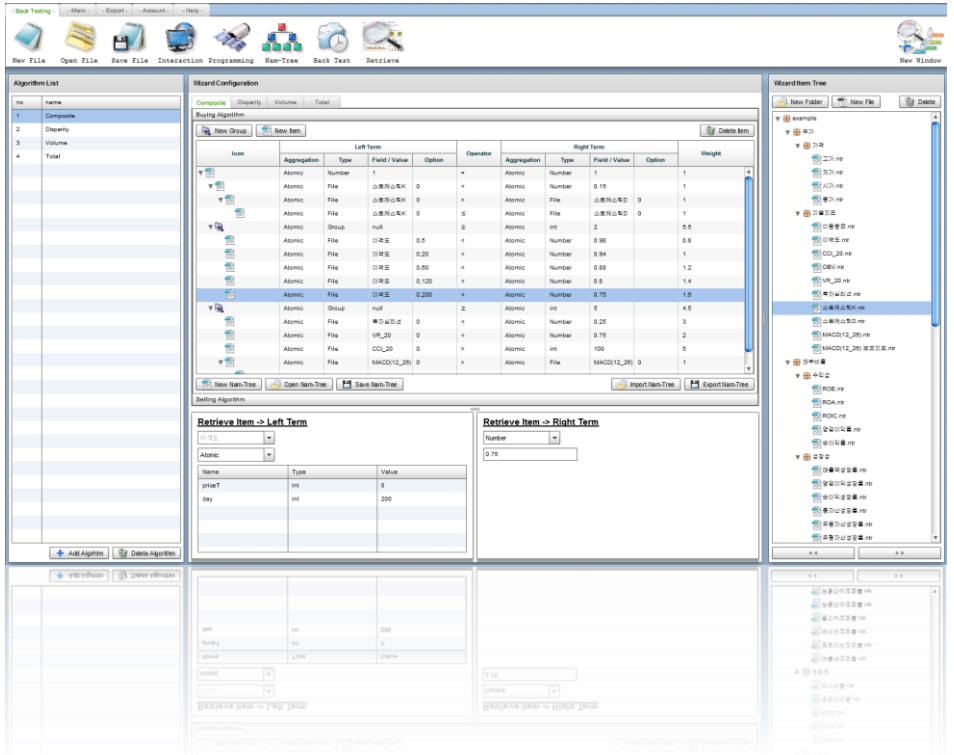
```
##########################################
# TSTL 설치, 빌드 및 테스트 수행
##########################################
# NodeJS 와 TypeScript 가 반드시 설치되어 있어야 함
brew install node
npm install -g typescript

# 깃허브로부터 TSTL 을 다운받자
git clone https://github.com/samchon/tstl.git
cd tstl

# Dependency 를 설치하고, TSTL 을 빌드한 후,
npm install --save-dev
node build/build

# 테스트 자동화를 돌려주면 끝.
npm test
```

1. Releases

- Homepage: http://samchon.org/tstl
- Guide Documents: https://github.com/samchon/tstl/wiki
- API Documents: http://samchon.github.io/tstl/api

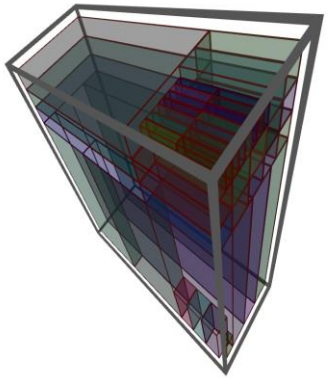
2. Utilization Cases

- **Nam-Tree**
  - 간이 A.I. 모형
  - 고차원적인 라이브러리의 필요
  - Web 시각화 & C++ Migration
- **Samchon Simulation**
  - 주식 시뮬레이션 & 시스템 트레이딩
  - 고차원적인 라이브러리의 필요
  - Web 시각화 & C++ Migration
- **Samchon Framework**
  - OON 프레임워크
    - Object Oriented Network
  - Integration with C++



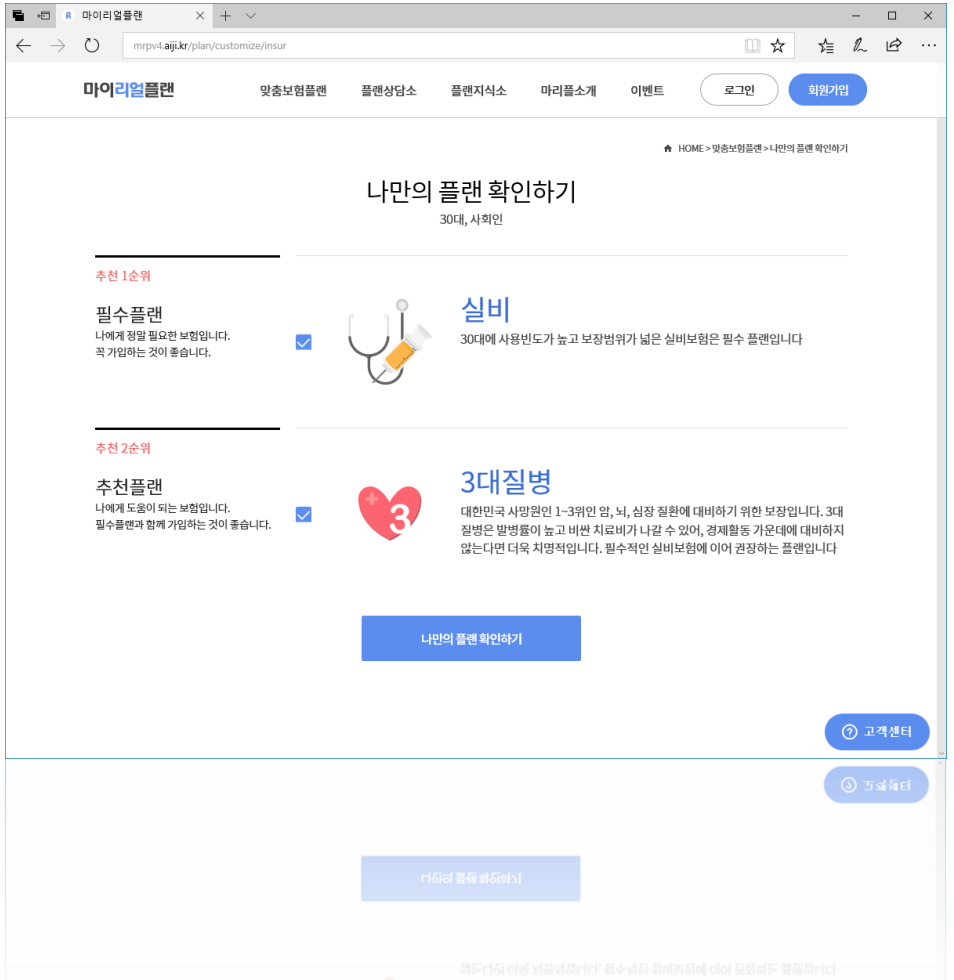
2. Utilization Cases

- **3D-Bin-Packing**
  - 공간활용 최적화
  - Web 을 통한 시각화 (3D 모델링)
  - C++ Migration
- **Folding**
  - 접지 단가 최적화
  - Containers & Algorithms 의 필요
  - Web 을 통한 시각화
- **Button-Finder**
  - 이미지 검색기
  - Interaction with C++
    - 분산처리: NodeJS
    - 이미지 프로세싱: C++



2. Utilization Cases

- **Kiosk-Editor**
  - Cloud Project
  - 고차원의 Container 와
  - Asynchronous Critical Section 필요
- **Auto-HL**
  - Cloud 장비관리
  - Kiosk-Editor 와 동일한 이유
- **Package-Retriever**
  - 보험 검색엔진
  - Collection Framework 와
  - Algorithms 및
  - Asynchronous Critical Section 의 필요



3. Blueprint

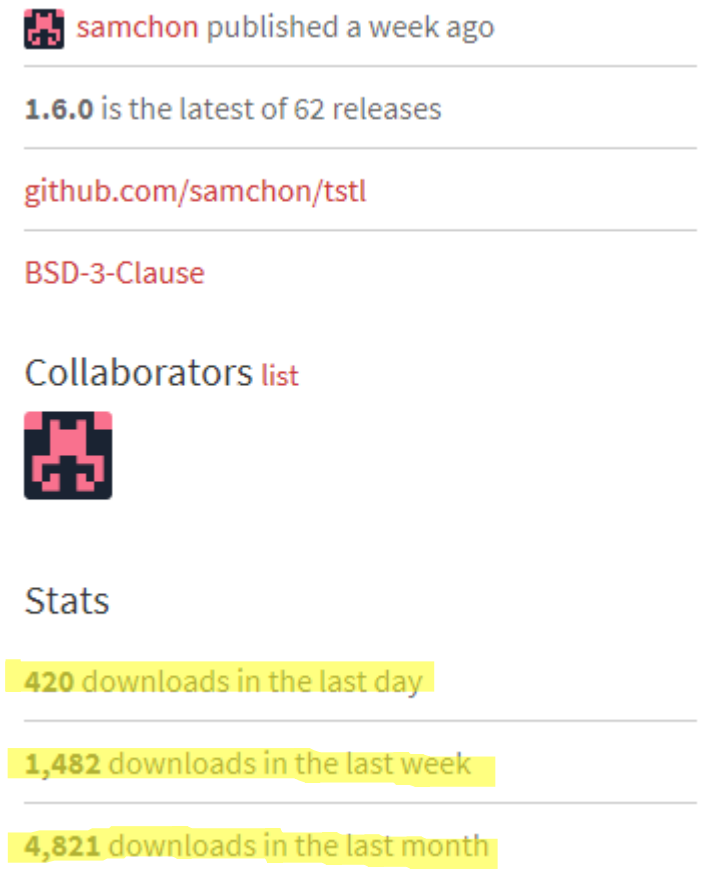
Be Famous

3. Blueprint

- TSTL 은 아직 마이너한 라이브러리
- 일일 다운로드 수가 적음
  - 수 십 ~ 수백 건 수준

- 유명한 라이브러리들은
- 하루에 수만건의 다운로드가 있음
  - Collections.JS
  - 일일 약 8,000 ~ 12,000 건

3. Blueprint



- **TSTL** 의 현재 최대 단점은
- **C++ STL** 을 아는 사람만 쓸 수 있다.

- **일반 JS 사용자**도 친숙하게 쓸 수 있게
- 내년엔 튜토리얼을 만들고,
- 여러 PR 자료들도 만들어 볼 생각

Q & A

TSTL

2017.11.10

Samchon