LEIDEN UNIVERSITY MEDICAL CENTER

Project planning

Jeroen F. J. Laros

Leiden Genome Technology Center

Department of Human Genetics

Center for Human and Clinical Genetics

Introduction

Common approaches:

- Just start.
- Make sure the simple things work, the rest comes later.

Introduction

Common approaches:

- Just start.
- Make sure the simple things work, the rest comes later.

However, it is likely that:

- Specific cases are solved, not parts of the problem.
- Unscalable solution are made.
- Unmaintainable solution are made.
- Lots of time is spent on rewriting.
  - Or worse, a stack of exceptions is made.
- More time than needed is wasted.

Introduction

Common approaches:

- Just start.
- Make sure the simple things work, the rest comes later.

However, it is likely that:

- Specific cases are solved, not parts of the problem.
- Unscalable solution are made.
- Unmaintainable solution are made.
- Lots of time is spent on rewriting.
    - Or worse, a stack of exceptions is made.
- More time than needed is wasted.

Software needs to be *designed*.

Introduction

Mutalyzer: a curational tool for *Locus Specific Mutation Databases* (LSDBs).

Introduction

Mutalyzer: a curational tool for *Locus Specific Mutation Databases* (LSDBs).

- Variant nomenclature checker applying *Human Genome Variation Society* (HGVS) guidelines.
  - Is the syntax of the variant description valid?
  - Does the reference sequence exist?
  - Is the variant possible on this reference sequence?
  - Is this variant description the recommended one?

Introduction

Mutalyzer: a curational tool for *Locus Specific Mutation Databases* (LSDBs).

- Variant nomenclature checker applying *Human Genome Variation Society* (HGVS) guidelines.
  - Is the syntax of the variant description valid?
  - Does the reference sequence exist?
  - Is the variant possible on this reference sequence?
  - Is this variant description the recommended one?
- Basic effect prediction.
  - Is the description of the transcript product as expected?
  - Is the predicted protein as expected?

HGVS nomenclature

Genomic orientated positions:

AL449423.14:g.[65449_65463del;65564T>C]

HGVS nomenclature

Genomic orientated positions:

AL449423.14:g.[65449_65463del;65564T>C]

Coding sequence orientated positions:

AL449423.14(CDKN2A_v001):c.[5A>G;106_120del]

HGVS nomenclature

Genomic orientated positions:

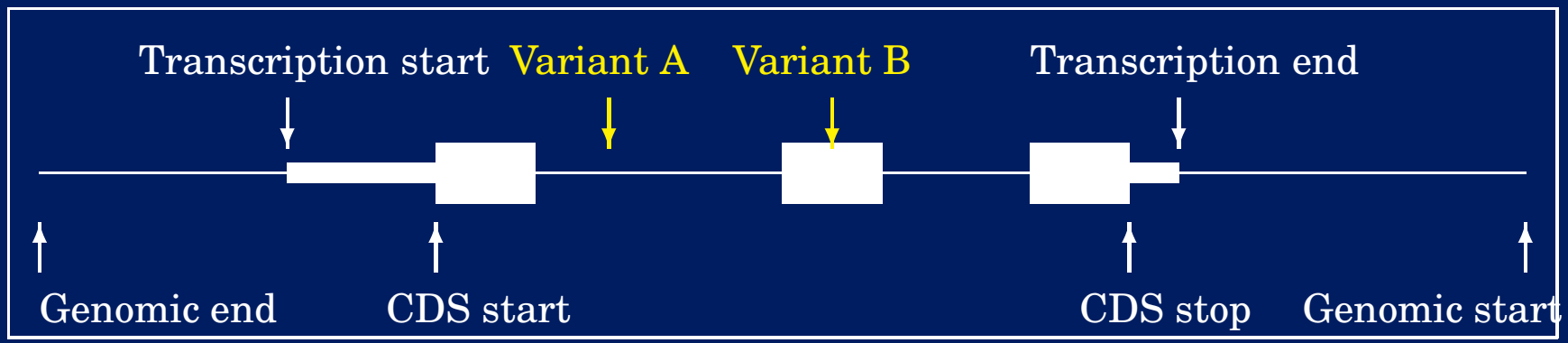
AL449423.14:g.[65449_65463del;65564T>C]

Coding sequence orientated positions:

AL449423.14(CDKN2A_v001):c.[5A>G;106_120del]

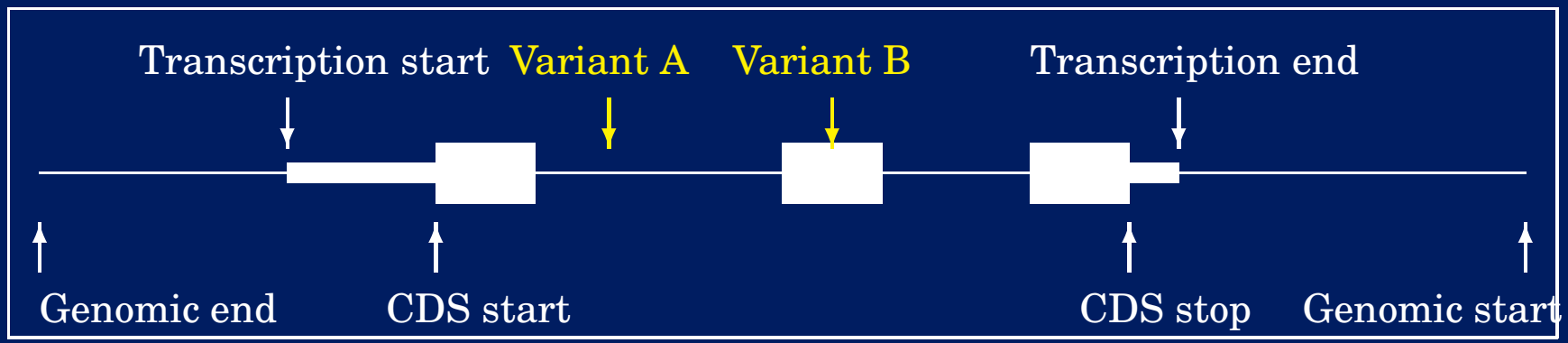
- **AL449423.14** – reference sequence.
- **CDKN2A_v001** – transcript variant **1** of gene CDKN2A.
- **c.[5A>G;106_120del]**
  - A *substitution* at position **5** counting from the start codon.
  - A *deletion* from position **106** to position **120**.

HGVS nomenclature



Name	g.	n.	c.
Genomic start	1	100+d70	*10+d70
Genomic end	300	1-u50	-30-u50
Transcription start	250	1	-30
Transcription end	70	100	*10
CDS start	220	30	1
CDS stop	80	90	60

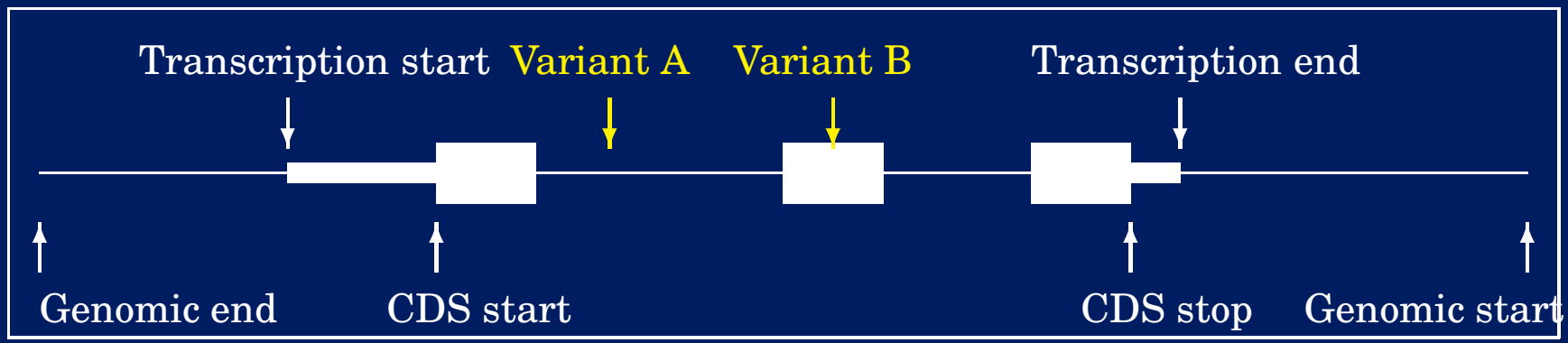
HGVS nomenclature



c. positions:

- Positions in introns are relative to the nearest exonic position.
- Positions before the CDS are indicated with a **-** sign.
- Positions after the CDS are indicated with a ***** sign.

HGVS nomenclature



c. positions:

- Positions in introns are relative to the nearest exonic position.
- Positions before the CDS are indicated with a **-** sign.
- Positions after the CDS are indicated with a ***** sign.

Position **-1** and **1** are adjacent.
If **60** is the last position of the CDS, then **60** and ***1** are adjacent.

Mutalyzer 1.0.4

Organic growth:

- Developed in over four years by multiple people.
- Originally a command line program.
- Web interface added later.
    - PHP scripts that call (exec) a python program.

Mutalyzer 1.0.4

Organic growth:

- Developed in over four years by multiple people.
- Originally a command line program.
- Web interface added later.
    - PHP scripts that call (exec) a python program.

Design flaws:

- Nomenclature rules interwoven with the code.
- HTML output interwoven with the code.
- No modularity (reuse of code is very hard).
- Reference sequence parsing not abstracted
    - Support for other reference sequences nearly impossible.
- Nomenclature was not analysed / formalised.
    - Regular expressions directly in code.
    - Virtually impossible to change (or read).

Mutalyzer 1.0.4

Implementation flaws:

- Inheritance of types (delins on DNA $\Rightarrow$ delins on protein).
- Disambiguation not general.
- No support of up- / downstream exons.
- Nothing was ever redesigned, only wrapped in loops.
  - Debugging, altering code made impossible.
  - Speed drastically diminished.

Mutalyzer 1.0.4

Implementation flaws:

- Inheritance of types (delins on DNA $\Rightarrow$ delins on protein).
- Disambiguation not general.
- No support of up- / downstream exons.
- Nothing was ever redesigned, only wrapped in loops.
  - Debugging, altering code made impossible.
  - Speed drastically diminished.

Programming flaws:

- Excessive usage of exceptions.
- Incomprehensible error messages.
- Virtually no inline comment.
- Non existent documentation (apart from the user manual).

Mutalyzer 1.0.4

Feature requests:

- Solving all mentioned problems.
- Support for other reference files (LRG).
- Programmatic access to internal functions.
- Extension of HGVS nomenclature rules.

Mutalyzer 1.0.4

Feature requests:

- Solving all mentioned problems.
- Support for other reference files (LRG).
- Programmatic access to internal functions.
- Extension of HGVS nomenclature rules.

Especially since:

- Interwoven nomenclature.
- Interwoven interface.
- Lack of documentation.

Mutalyzer 1.0.4

Feature requests:

- Solving all mentioned problems.
- Support for other reference files (LRG).
- Programmatic access to internal functions.
- Extension of HGVS nomenclature rules.

Especially since:

- Interwoven nomenclature.
- Interwoven interface.
- Lack of documentation.

Throw away four years of work and start again.

Preparing for version 2.0

Preparations for a new version:

- Setting up two version control repositories.
- Gathering all old versions and putting then under version control.
  - Critical bugfixes until there is a new version.
  - Easy to search and track changes.
  - Point of reference for the new version.
- Set up bugtracking systems.

Preparing for version 2.0

Preparations for a new version:

- Setting up two version control repositories.
- Gathering all old versions and putting then under version control.
  - Critical bugfixes until there is a new version.
  - Easy to search and track changes.
  - Point of reference for the new version.
- Set up bugtracking systems.
- Talking for months.
  - Figuring out what the HGVS language is.
  - Formalising that language (BNF).
  - Semantic rules.
- Identify and implement functional modules.
- Designing interfaces (web (TAL), webservice, command line, . . .).
- Documentation (API (epydoc), TRM, User manual, . . .).

Preparing for version 2.0

Although the complexity might seem low at first glance. . .

Non-trivial problems can be found:

- Position systems.
  - Reference sequence related peculiarities.
- The HGVS nomenclature.
- Parsing reference sequence files and their annotation.
  - Keeping the option open for other formats (LRG).
- Keeping track of positions after mutation.
- . . .

Preparing for version 2.0

Although the complexity might seem low at first glance. . .

Non-trivial problems can be found:

- Position systems.
  - Reference sequence related peculiarities.
- The HGVS nomenclature.
- Parsing reference sequence files and their annotation.
  - Keeping the option open for other formats (LRG).
- Keeping track of positions after mutation.
- . . .

HGVS parser

The previous versions used *regular expressions* for input parsing.

- The HGVS nomenclature turns out to be a *context free* language.
  - Strictly stronger than regular language.
  - Can not be parsed with a finite automaton.
  - No regular expressions.
- The semantics of the HGVS is even more complex than the syntax.

HGVS parser

The previous versions used *regular expressions* for input parsing.

- The HGVS nomenclature turns out to be a *context free* language.
  - Strictly stronger than regular language.
  - Can not be parsed with a finite automaton.
  - No regular expressions.
- The semantics of the HGVS is even more complex than the syntax.

These observations remind us of *compiler construction*.

- We need a context free parser.
- A parse tree must be generated.
- And then there is the semantic checking.
  - This should be the real focus of the project, the rest can be solved with existing techniques.

HGVS parser

Definition of a gene symbol.

```
1   TransVar   -> '_v' Number
2   ProtIso    -> '_i' Number
3   GeneSymbol -> '(' Name (TransVar | ProtIso)? ')'
```

Listing 1: Abstract HGVS nomenclature

Gene name and optionally a transcript or isoform number.

HGVS parser

Definition of a gene symbol.

```
1   TransVar   -> '_v' Number
2   ProtIso    -> '_i' Number
3   GeneSymbol -> '(' Name (TransVar | ProtIso)? ')'
```

Listing 1: Abstract HGVS nomenclature

Gene name and optionally a transcript or isoform number.

```
1       TransVar = Suppress("_v") + Number("TransVar")
2       ProtIso = Suppress("_i") + Number("ProtIso")
3       GeneSymbol = Suppress('(') + \
4           Group(Name("GeneSymbol") + \
5           Optional(TransVar ^ ProtIso))("Gene") + \
6           Suppress(')')
```

Listing 2: HGVS nomenclature in Python

HGVS parser

After parsing, a nested python object (parse tree) is generated:

HGVS parser

After parsing, a nested python object (parse tree) is generated:

The generated object of **(CDKN2A_v001)**:

```
1       Gene.GeneSymbol = CDKN2A
2       Gene.TransVar = 001
```

Listing 3: Python object

HGVS parser

After parsing, a nested python object (parse tree) is generated:

The generated object of **(CDKN2A_v001)**:

```
1       Gene.GeneSymbol = CDKN2A
2       Gene.TransVar = 001
```

Listing 3: Python object

Complete BNF consists of 63 production rules.

HGVS parser

After parsing, a nested python object (parse tree) is generated:

The generated object of **(CDKN2A_v001)**:

```
1       Gene.GeneSymbol = CDKN2A
2       Gene.TransVar = 001
```

Listing 3: Python object

Complete BNF consists of 63 production rules.

A formalized description of the standard human variant nomenclature will improve sequence variant recognition in databases and literature.

Jeroen F. J. Laros[1], André Blavier[2], Johan T. den Dunnen[1], Peter E. M. Taschner[1]

[1] Department of Human Genetics, Center for Human and Clinical Genetics, Leiden University Medical Center, Leiden, Nederland

[2] Interactive Biosoftware, Rouen, France

Interfaces

Name checker	-	Full nomenclature / semantic check.
Syntax checker	-	Only nomenclature check.
Position converter	-	Mapping, lifting over (build / transcripts).
SNP converter	-	DbSNP rsId to HGVS.
Name generator	-	Point and click to make a description.
GenBank Uploader	-	Custom reference sequences.

Interfaces

Name checker	-	Full nomenclature / semantic check.
Syntax checker	-	Only nomenclature check.
Position converter	-	Mapping, lifting over (build / transcripts).
SNP converter	-	DbSNP rsId to HGVS.
Name generator	-	Point and click to make a description.
GenBank Uploader	-	Custom reference sequences.

Bulk / RPC interfaces:

- Upload a table (CSV, Excel, Open Office Spreadsheet):
  - Name checker.
  - Name checker.
  - Syntax checker.
  - Position converter.
  - SNP converter.
- Webservices (SOAP).
  - 18 functions available.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Name checker.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Syntax checker.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Position converter.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

SNP converter.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Name generator.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

GenBank Uploader.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Added when using a batch interface.

Components

Config	-	Parsing config file.
Crossmap	-	Position conversions.
Db	-	Mapping, linking, queues, caching info.
File	-	CSV, Excel, OpenOffice tables.
GenRecord	-	Abstraction of annotated reference sequences.
GBparser	-	Instance of GenRecord (GenBank files).
LRGparser	-	Instance of GenRecord (LRG files).
Misc	-	
Mutator	-	Modify the reference sequence and annotation.
Output	-	Communication with the interfaces.
Parser	-	HGVS nomenclature parser.
Retriever	-	Retrieve / cache reference sequences.
Scheduler	-	Batch jobs scheduler.
Serializers	-	SOAP definitions of complex objects.

Added when using webservices.

Comparison to the old version

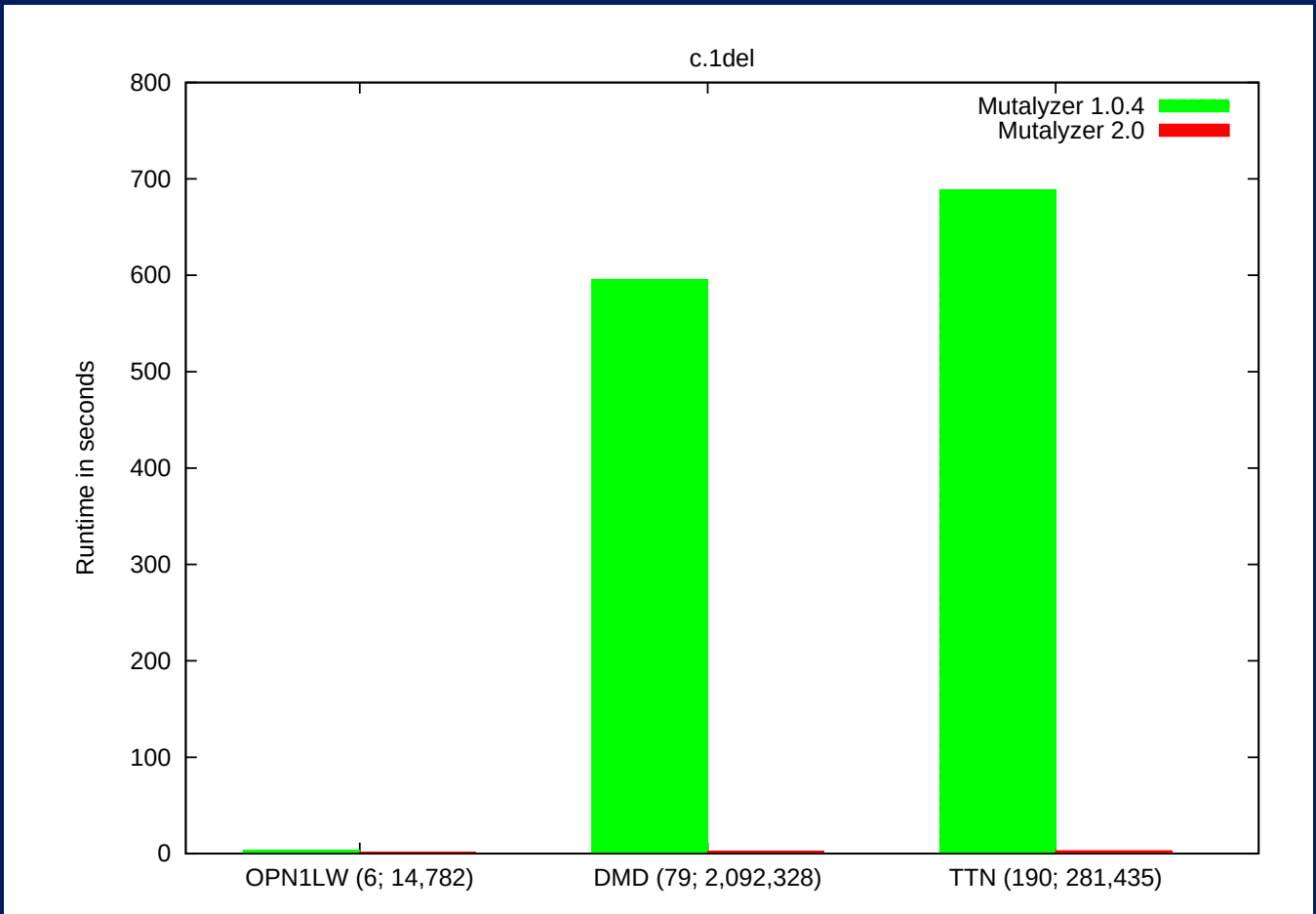


Figure 1: Runtime comparison

A 229× speedup was measured (12*min* to 3*s*).

Comparison to the old version

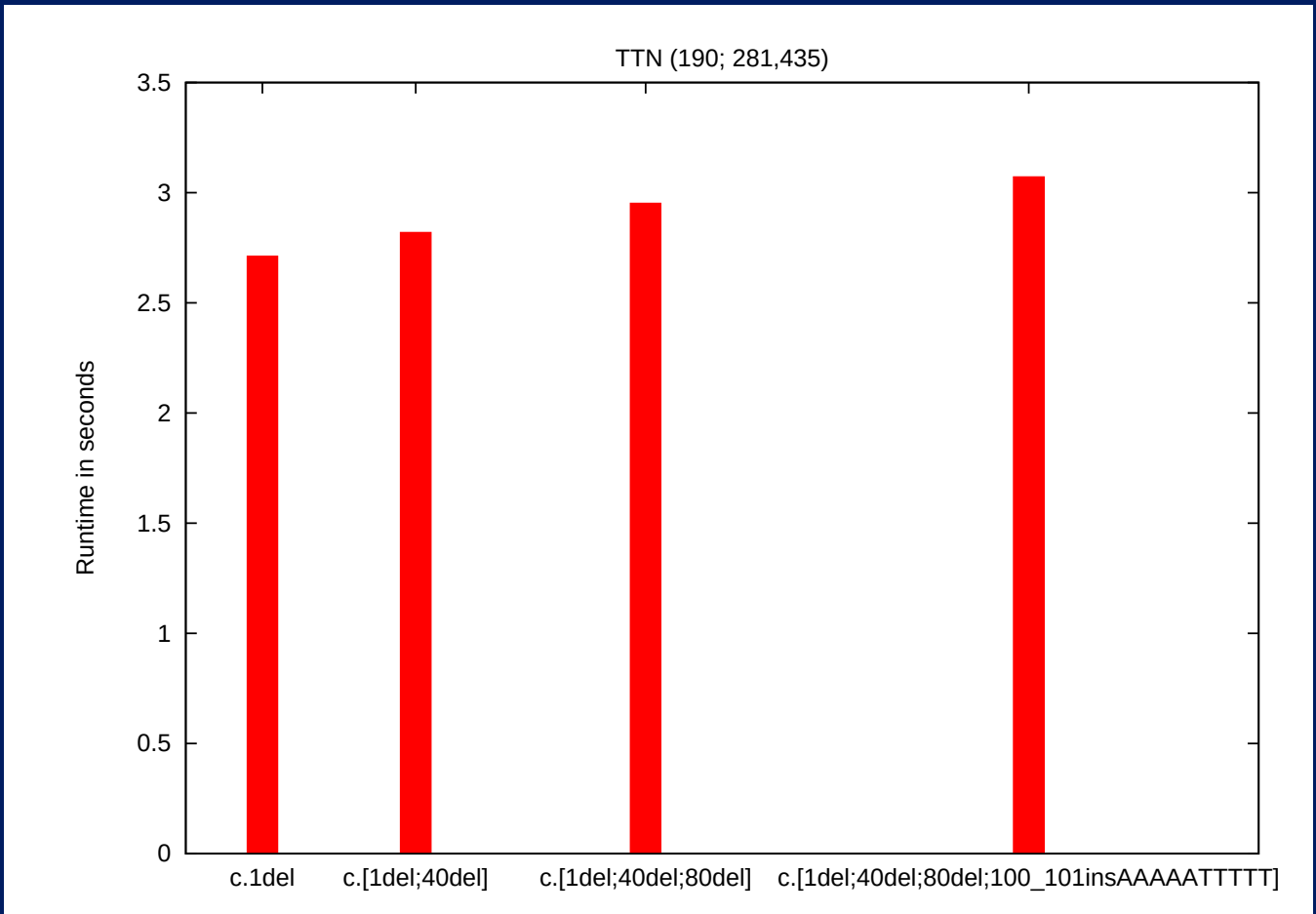


Figure 2: Scalability
Overhead ($\pm 2.5s$): loading the reference sequence.

Comparison to the old version

Special focus on documentation.

	2.0	1.0.4
User manual	12 pages	12 pages
TRM	31 pages	–
API documentation	215 pages	–
Total size	493,816 bytes	367,836 bytes
Code	198,512 bytes	245,677 bytes
Comment	60%	33%*
Development time	< 1 year	> 4 years

Table 1: Other characteristics.

* Mostly discarded code.

Conclusions

Things to keep in mind:

Conclusions

Things to keep in mind:

- Do not get tempted to implement small exceptions.

Conclusions

Things to keep in mind:

- Do not get tempted to implement small exceptions.
- Make a design.
  - Requirements will change in the future.
  - Make your design capable to deal with these changes.
- If requirements change, go back to your design.

Conclusions

Things to keep in mind:

- Do not get tempted to implement small exceptions.
- Make a design.
  - Requirements will change in the future.
  - Make your design capable to deal with these changes.
- If requirements change, go back to your design.
- Document everything.
  - Especially your code.

Conclusions

Things to keep in mind:

- Do not get tempted to implement small exceptions.
- Make a design.
  - Requirements will change in the future.
  - Make your design capable to deal with these changes.
- If requirements change, go back to your design.
- Document everything.
  - Especially your code.
- Use a version control system.

Conclusions

Things to keep in mind:

- Do not get tempted to implement small exceptions.
- Make a design.
  - Requirements will change in the future.
  - Make your design capable to deal with these changes.
- If requirements change, go back to your design.
- Document everything.
  - Especially your code.
- Use a version control system.

Think before you start.

Questions?

Acknowledgements:

Martijn Vermaat
Gerben Stouten
André Blavier
Gerard Schaafsma
Ivo Fokkema
Jacopo Celli
Peter Taschner
Johan den Dunnen

http://www.mutalyzer.nl/