Requirements Elicitation from Social Media with AI.

Requirements Specification and Validation with Formal Methods.

João Pascoal Faria (jpf@fe.up.pt)
MESW, ERMS, 15/11/2024

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

REQUIREMENTS ELICITATION FROM SOCIAL MEDIA WITH AI

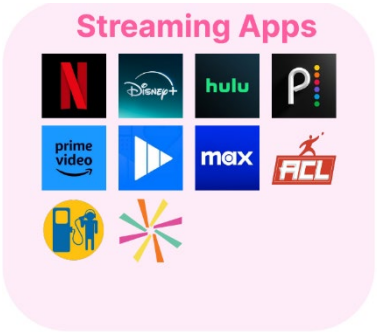
Automated Social Media Feedback Analysis for Software Requirements Elicitation: A Case Study in the Streaming Industry , Melissa Silva, J. P. Faria, ACM SAC 2025 – Requirements Engineering Track (*submitted*)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

rrog
DIGITAL MEDIA

Context

- Requirements Engineering (RE) is a critical activity for ensuring product success.
- RE is particularly challenging for software products with a wide user base, as is the case with **content delivery platforms** in the **streaming industry**.
- To guide product evolution in the face of evolving user needs, developers must continuously collect and analyze user feedback, from diverse sources
- However, manual collection and analysis are impractical due to the sheer volume of feedback and the diversity of its sources.



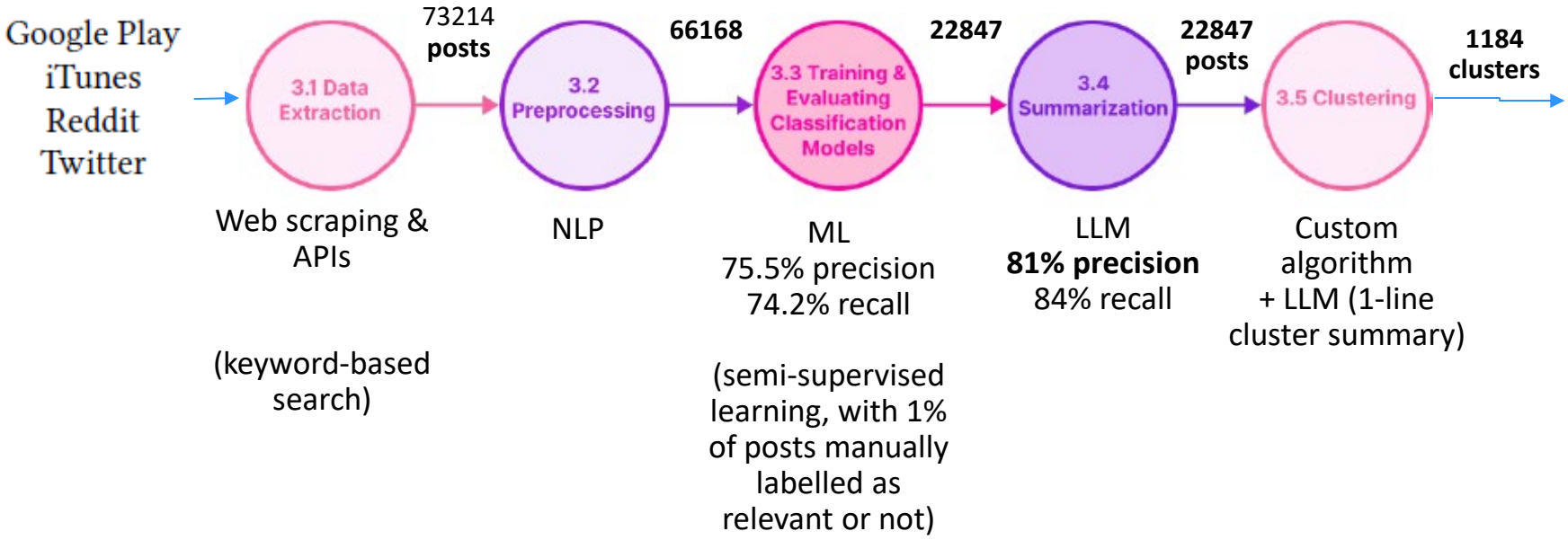
Goals

- We address these challenges by **automatically collecting, filtering, summarizing and clustering user feedback (posts) from social media**, and by **suggesting corresponding feature requests and bug-fixing requests (issues)** through an intuitive visualization **platform (Watch the Watchers).**
- Data is gathered from diverse social media platforms (Reddit, Twitter, iTunes, and Google Play) using web crawlers and APIs
- Data is processed using a novel combination of:
  - natural language processing (NLP)
  - machine learning (ML)
  - large language models (LLMs)

Results

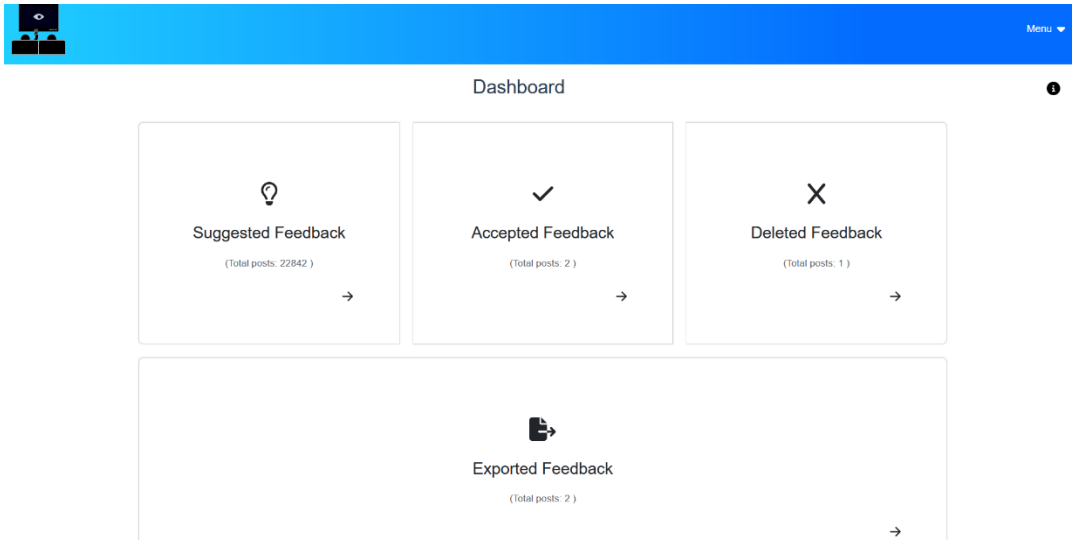
- We applied and evaluated our approach with a partner company in the streaming industry (MOG Technologies).
- We extracted **66,168 posts** related to 10 streaming services of interest, and selected **22,847 as potentially relevant** using an ML-based classifier (with 75.5% precision and 74.2% recall).
- From the top 100 posts presented to a user by our platform, they found 89 relevant and **generated 47 related issues in about 80 minutes** (exported to GitLab).
- A usability study with 6 company specialists yielded a **SUS score of 78.7 ("Good")** and very positive feedback.
- These results suggest that **our approach can effectively help software companies in requirements elicitation and product evolution**.

Social Media Feedback Processing Steps

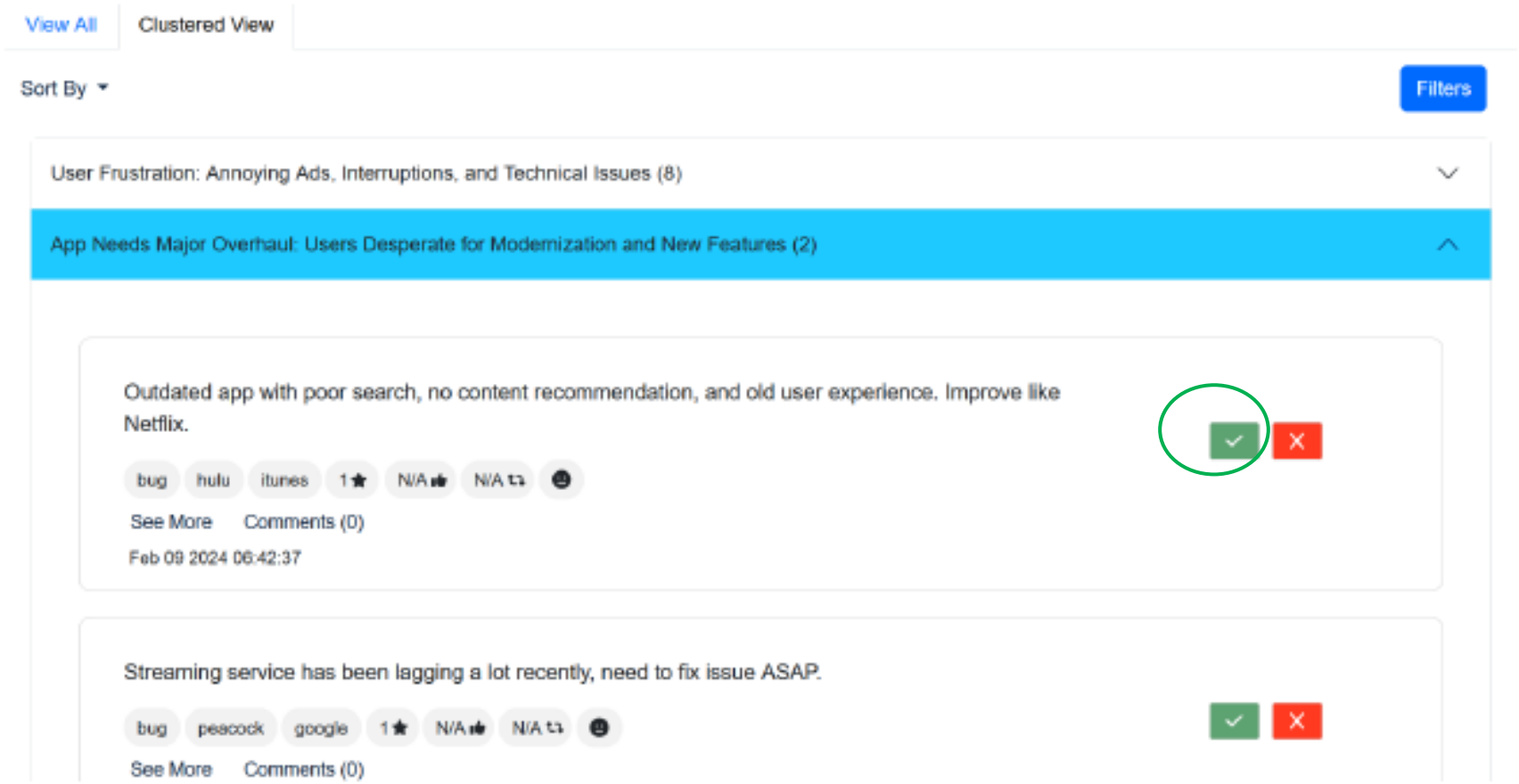


Watch the Watchers: Dashboard Page

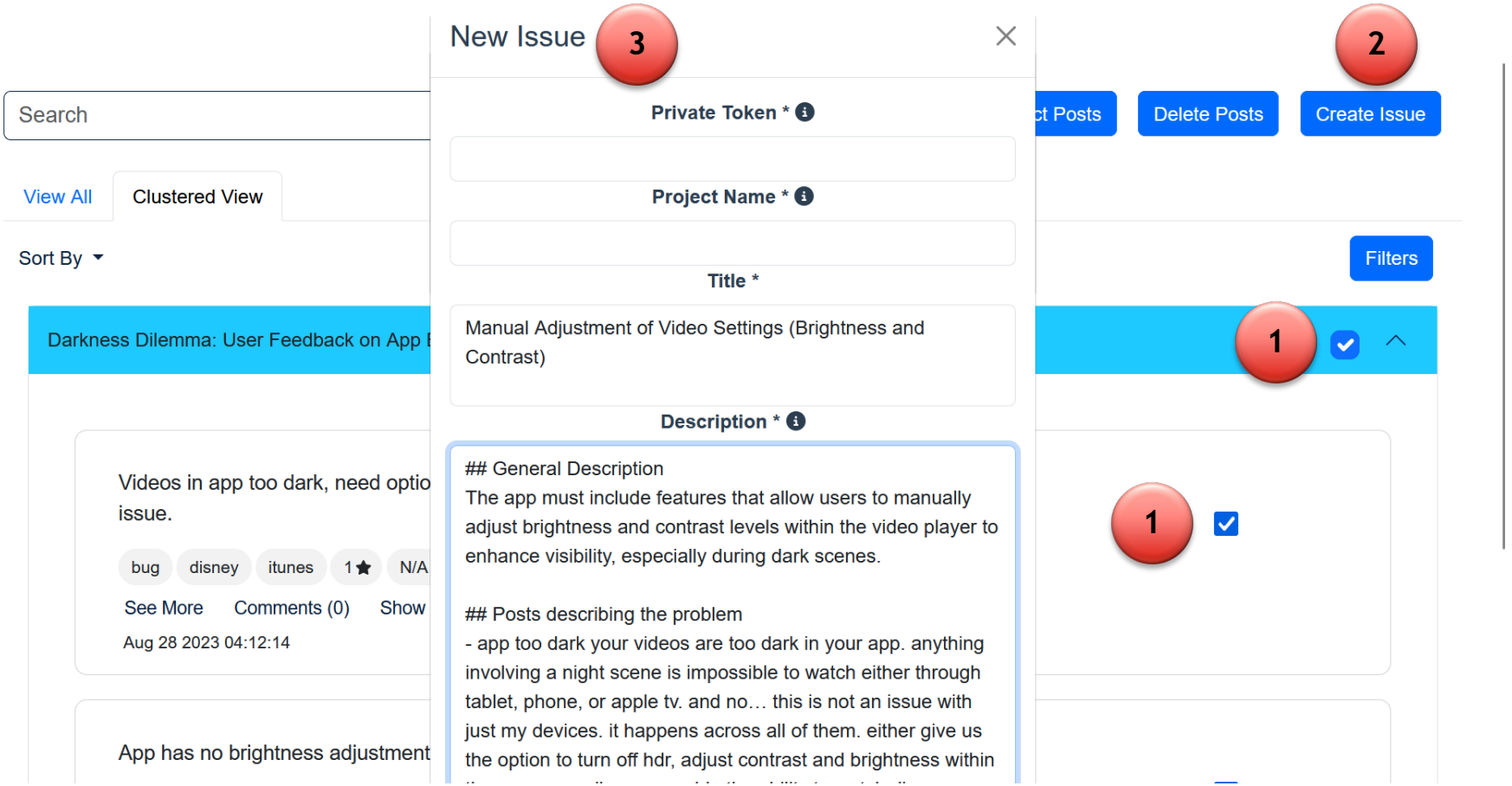
- **Suggested** Feedback: Contains all new and unseen posts.
  - Users can review these posts and move them to Accepted or Deleted;
- **Accepted** Feedback: Posts reviewed and confirmed as relevant.
  - Users can create issues from these posts or reclassify them as irrelevant;
- **Exported** Feedback: Posts processed and converted into issues.
- **Deleted** Feedback: Posts reviewed and deemed irrelevant.



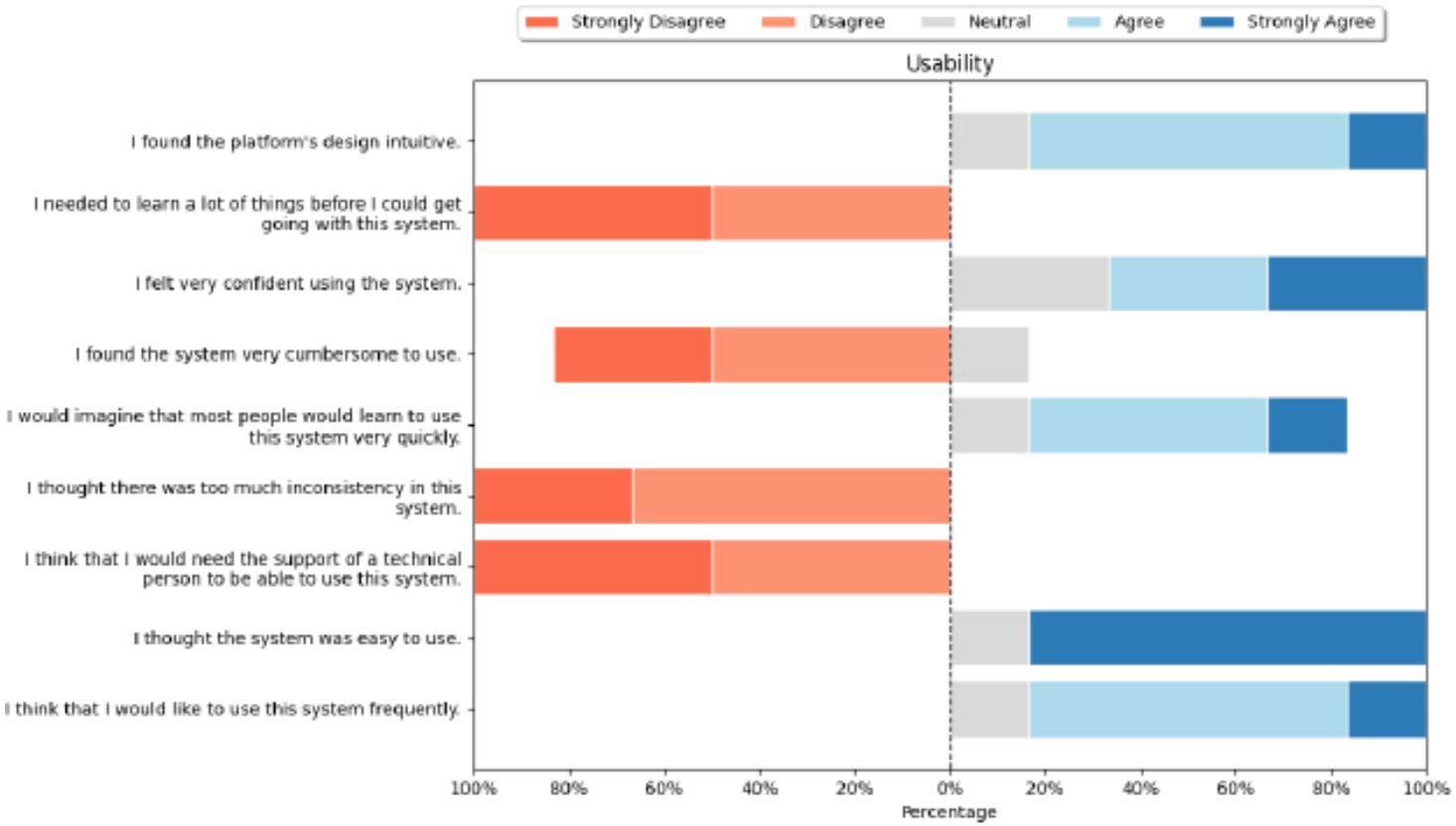
Watch the Watchers: View New Posts in Clustered View and Accept/Reject



Watch the Watchers: Create Issues from Accepted Posts



Validation Results



From the top 100 posts presented to a user @MOG by our platform, they found 89 relevant and generated 47 related issues (GitLab) in about 80 minutes.

REQUIREMENTS SPECIFICATION AND VALIDATION WITH FORMAL METHODS

What are formal methods?

- Formal methods are a set of **mathematical** techniques used for the **(formal) specification**, synthesis (creation), and **(formal) verification** of software & hardware systems.
- The goal is to provide a high level of **assurance** that the behaviour of a system is as intended, and to ensure that the system is free of certain types of defects.
- They are especially useful in **critical systems**, such as those in the aerospace, automotive, and medical industries, where the consequences of failure can be catastrophic.
- Even for other types of systems, formal specification helps **remove the ambiguity (*) and limited analyzability** inherent to informal NL specifications
  - (*) And also identify missing or inconsistent information

When do we need formal methods?

- For the development & certification of critical components, and, in general, systems whose failures are “unacceptable”.

- The IEC 61508 Standard on Functional Safety defines 4 Safety Integrity Levels (SIL), from the least (1) to the most (4) dependable, based on acceptable probability of failure (from 10^{-1} to 10^{-4}), and indicates when to apply formal methods in the lifecycle activities (R - recommended, HR - highly-recommended).

Lifecycle activity	SIL1	SIL2	SIL3	SIL4
Software Safety **Requirements Specification**	--	R	R	HR
Software **Design** and Development	--	R	R	HR
Software **Verification**	--	R	R	HR

Formal methods in the media (PT)

BASE DE DADOS FOI A MESMA

Nova empresa resolveu colocação de professores em seis dias

SOFIA RODRIGUES · 30 de Setembro de 2004, 9:35

0 PARTILHAS

O problema da colocação de professores foi resolvido através de uma nova solução informática pensada em seis dias e executada em 30 minutos, com o apoio de um novo servidor vindo de Espanha. A revelação foi feita por um dos cinco elementos da equipa da ATX Software, empresa externa contratada pelo ministério da Educação para "desbloquear" o programa informático de colocação de professores criado pela Compta.

A partir da mesma base de dados do ministério que contém todos os docentes por colocar, a ATX Software criou um novo algoritmo, ou seja, uma solução informática, "pensado na íntegra durante seis dias e baseado em princípios matemáticos muito sólidos", afirmou ontem o engenheiro informático e autor da solução, Luís Andrade, durante uma conferência de imprensa, em Lisboa.

https://www.publico.pt/2004/09/30/portugal/noticia/nova-empresa-resolveu-colocacao-de-professores-em-seis-dias-1204777

Formal methods in the media (EN)

Kevin Hartnett

Senior Writer

September 20, 2016

COMPUTER SECURITY

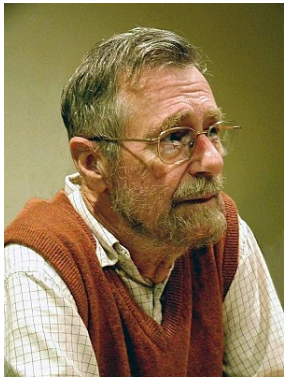
Hacker-Proof Code Confirmed

Computer scientists can prove certain programs to be error-free with the same certainty that mathematicians prove theorems. The advances are being used to secure everything from unmanned drones to the internet.

https://www.quantamagazine.org/formal-verification-creates-hacker-proof-code-20160920

Why do we need formal verification?

- **Testing is not enough** (at least for critical systems)!



E. W. Dijkstra
(ACM Turing Award)

"Program testing can be used to show the presence of bugs, but never to show their absence."

The FAA Rigorously Tested the Boeing 737's Software

Testing versus formal verification

- Testing
  - Requires test cases
  - Finds some bugs
    - **If *a bug* is found, the system under test is proved to be incorrect**
  - Applicable to all sorts of software

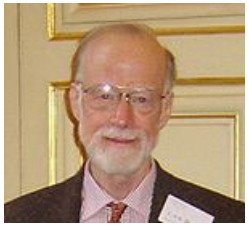
- Formal verification
  - Requires a formal specification
  - Finds ALL bugs
    - **If *no bug* is found, the system under test is proved to be correct**
  - Applicable to critical software modules

Specification with contracts

- Common style used in formal specifications
- Design by Contract (DbC) and the Eiffel language (first to implement DbC) were developed by Bertrand Meyer in the 1980's at ETH Zürich

- He received the 2006 ACM Software System Award
  - *"For designing and developing the Eiffel programming language, method and environment, embodying the Design by Contract approach to software development and other features that facilitate the construction of reliable, extendible and efficient software."*

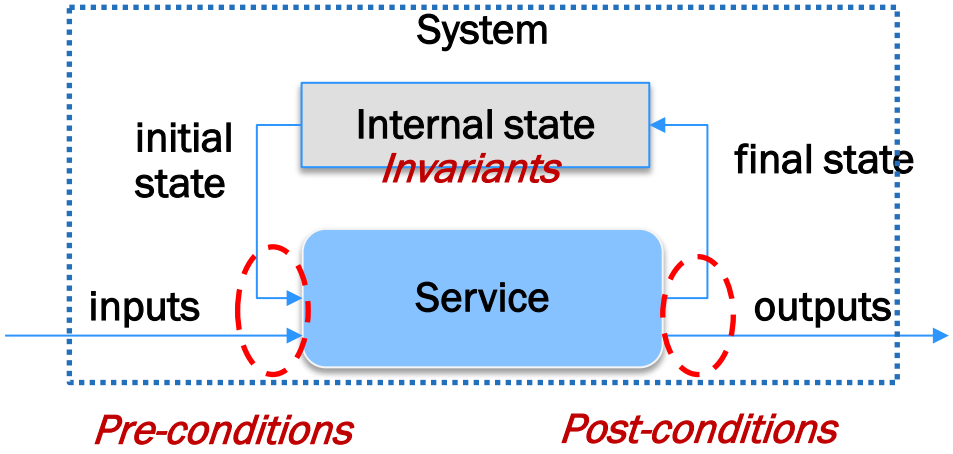
- DbC has its roots in work on program verification concepts (pre and post-conditions, etc.) developed by C.A.R. Hoare in the 1960's (author of quicksort)



What is a contract in DbC?

A **contract** is an agreement between the client and the supplier of a service, that sets the obligations and benefits of each party, comprising sets of:

- **Pre-conditions** – obligations that the client must respect when requesting the service, usually expressed in terms of constraints on input parameters and initial system state
- **Post-conditions** – benefits that the client can expect from a valid service request (obligations for the supplier), expressed in terms of constraints on output parameters and final system state, possibly in relation to the values of the input parameters and initial system state



Example 1: Pre/Post specifications

- double **sqrt**(double x)

- What are the pre-conditions?
  - x is not negative
- What are the post-conditions?
  - the result squared equals x (up to a rounding error!)
  - the result is not negative

Example 2: Pre/Post specifications

- void **sort**(T[] a) – sort array passed by reference

- What are the pre-conditions?
  - Not null
  - No null elements

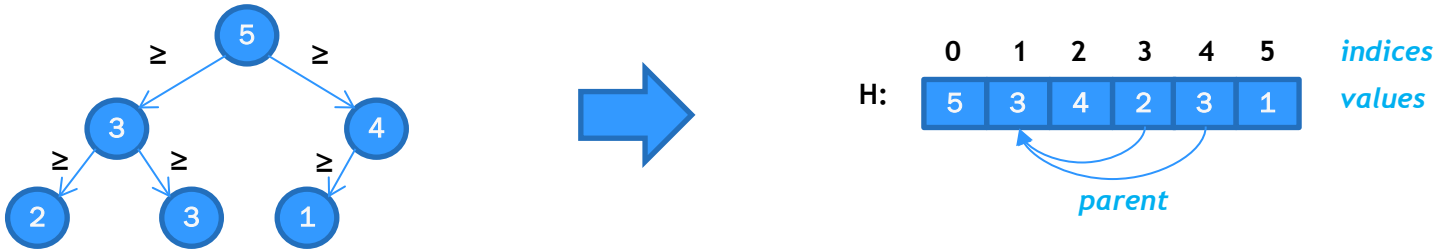
- What are the post-conditions?
  - Sorted by ascending order
  - Same elements as in the initial state (permutation)

State invariants

- Third concept in DbC
- A state invariant is an (integrity) constraint that defines the valid states (attribute values) of an object or system.
- Besides pre/post-conditions, operations must respect state invariants:
  - Each (public) constructor must ensure that all the invariants hold at the end of its execution
  - Each (public) operation must ensure that all the invariants hold at the end of its execution, assuming they hold at the beginning.

Example 3: Binary Heap

- A binary heap is a partially ordered binary tree represented in an array by levels, supporting operations *insert* and *deleteMax* in time O(log n).



- What is the heap invariant (with max at top)?

$$\forall i \in \{1, \ldots, |H| - 1\} \bullet H[(i - 1) \text{ div } 2] \geq H[i]$$

Example 4: Formal specification in Dafny

https://github.com/dafny-lang/dafny

Informal specification

Formal specification with pre- (requires) post-conditions (ensures)

Implementation is verified automatically against the specification!

Tests the specification (for consistency and completeness), not the implementation!

```
// Computes the quotient 'q' and remainder 'r' of
// the integer division of 'n' (>=0) by 'd' (>0).
method Div(n: nat, d: nat) returns (q: nat, r: nat)
  requires d > 0
  ensures q * d + r == n && r < d
{
  q := 0;
  r := n;
  while r >= d
    invariant q * d + r == n
  {
    q := q + 1;
    r := r - d;
  }
}

// A simple test case checked statically by Dafny!
method TestDiv() {
    var q, r := Div(15, 6);
    assert q == 2 && r == 3;
}
```

verification helper

Demo: Div.dfy

theorem proved with Z3:

```
n, d, q, r: nat && d > 0 && q * d + r == n && r < d
&& n == 15 && d == 6 ⇒ q == 2 && r ==3
```

Example 5: Stable Matching

- Stable matching (or stable marriage) problem
  - There is a set of *N* men and *M* women
  - Each has a list of preferences for the opposite sex
  - We want to find a stable matching of men and women
  - A matching is not stable if there is a pair *(m, w)* that prefer each other as compared to their current partners
  - Can be solved in time O(NM) by the Gale-Shapley algorithm
- The problem can be formalized in Dafny with appropriate data structures (collections), pre-conditions and post-conditions
- The algorithm can be encoded in Dafny and its correctness verified semi-automatically.

Stable Matching Specification in Dafny

```
// Stable matching with incomplete lists and no ties. Receives the lists of
// preferences of men and women and returns the couples created.
method stableMatching<Man, Woman>(menPrefs: map<Man, useq<Woman>>,
                                  womenPrefs: map <Woman, useq<Man>>)
                            returns(couples: inmap <Man, Woman>)
  // P1: women referenced in men preferences must exist
  requires forall m <- menPrefs, w <- menPrefs[m] :: w in womenPrefs
  // P2: man referenced in women preferences must exist
  requires forall w <- womenPrefs, m <- womenPrefs[w] :: m in menPrefs
  // Q1: men and women can be engaged only if mentioned in their preferences
  ensures isValid(couples, menPrefs, womenPrefs)
  // Q2: stable marriage (and maximality)
  ensures ! exists m <- menPrefs, w <- womenPrefs ::
              unstable(m, w, couples, menPrefs, womenPrefs)
```

Demo: StableMatching.dfy

```
w in menPrefs[m] && m in womenPrefs[w] &&
// m prefers w to his current wife
(m in couples ==> precedes(w, couples[m], menPrefs[m])) &&
// w prefers m to her current husband
(forall m' <- couples :: couples[m'] == w ==> precedes(m, m', womenPrefs[w]))
```

Example 5: Teachers' Placement

- Teachers' application to vacancies in schools
  - Each teacher has a list of preferences for vacancies
  - Some teachers already have a current position but want to change if possible
  - Teachers are sorted by their classification
- Can be reduced to stable matching and solved in polynomial time:
  - Men: teachers; the list of preferences is their initial list, with the current position appended to the end of the list.
  - Woman: vacancies; the list of preferences is the list of teachers sorted by their classification, with the current teacher occupying the position moved to the top
- The problem and reduction can be formalized and verified in Dafny!

Demo: StableMatching.dfy

Example 6: Students Application for Singular Curricular Units at FEUP

- Problem:
  - Each student has a list of preferences for curricular units
  - Each student has a maximum number of curricular units he/she wants to be placed in
  - Each curricular unit has a capacity (maximum number of students it can accept)
  - Each curricular unit 'sorts' the students that are applying to it, according to a grade assigned to their original courses, and their current marks in those courses
- Can be reduced to a variant of stable matching
  - Similar to polygamic & polyandric marriages with many-to-many matchings and limited capacities
  - Can be solved by a generalization of the Gale-Shapley algorithm
  - Formalized and proved in Dafny!

Demo: ManyToManyStableMatching.dfy