LOG8371 : Ingénierie de la qualité en logiciel

Software Quality Assurance

Hiver 2017

Fabio Petrillo
Chargé de Cours

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Why study **Software Quality**?

Because software is **critical** in our life!

VT100

Therac-25 Accidents

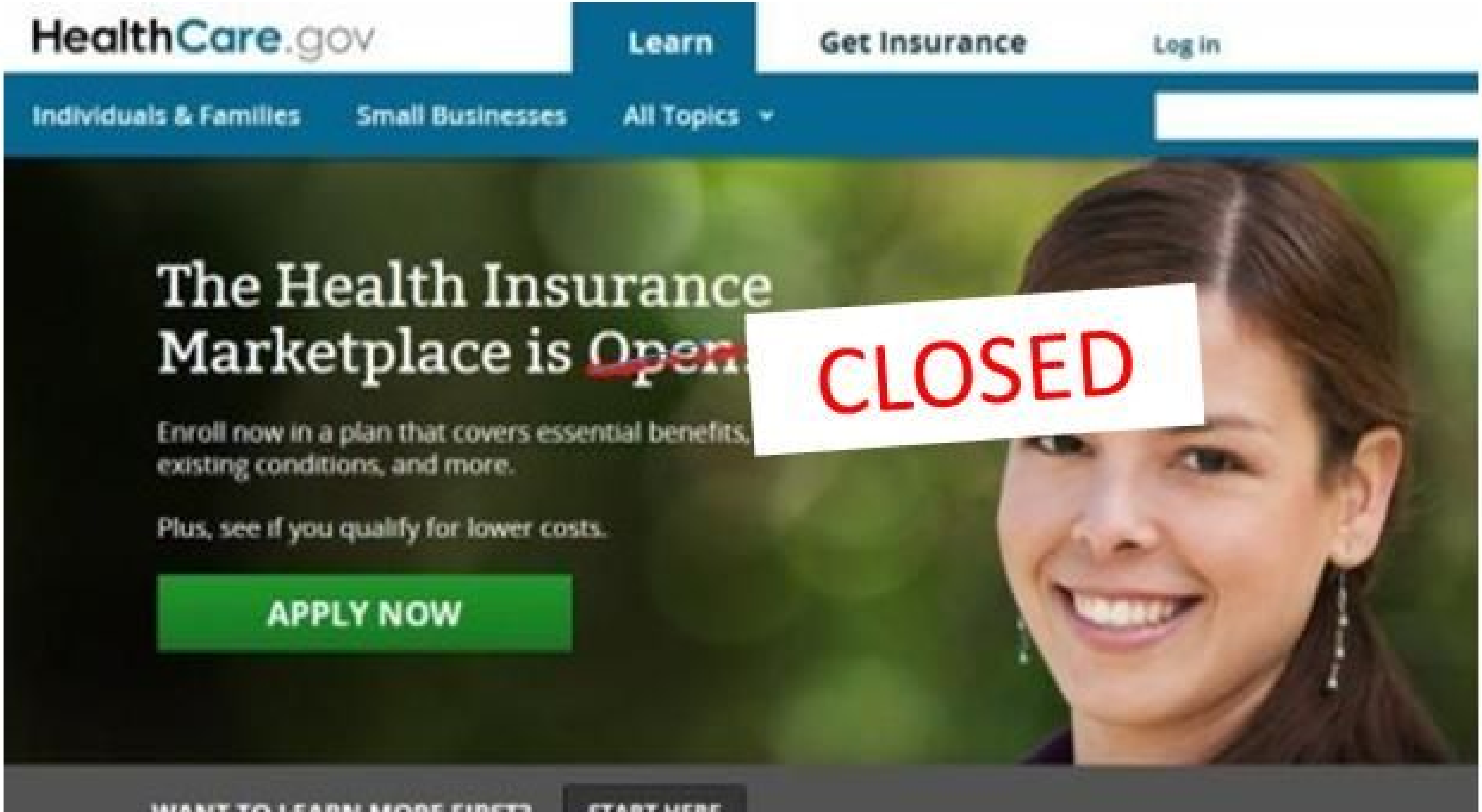
- Radiation therapy machine
- Companies
  - Atomic Energy of **Canada** Limited
  - CGR (France)
- Failure
  - Beta radiation overdose - 100 times the normal dose
  - 1985 - 1887
- Impact
  - **the death of at least three patients**
  - several injuries

So, what is the main issue on **“traditional”** Software Quality approach?

Therac-25 Accidents - Causes

- **Software** complex race condition
- AECL did not have the software code independently **reviewed**
- AECL had never tested the **combination** of software and hardware until it was assembled at the hospital
- Developed by a **single** person using PDP11 assembly
- Several years of development!
- Lack of documentation and software test plan
- The investigators could not reproduce the fault condition!

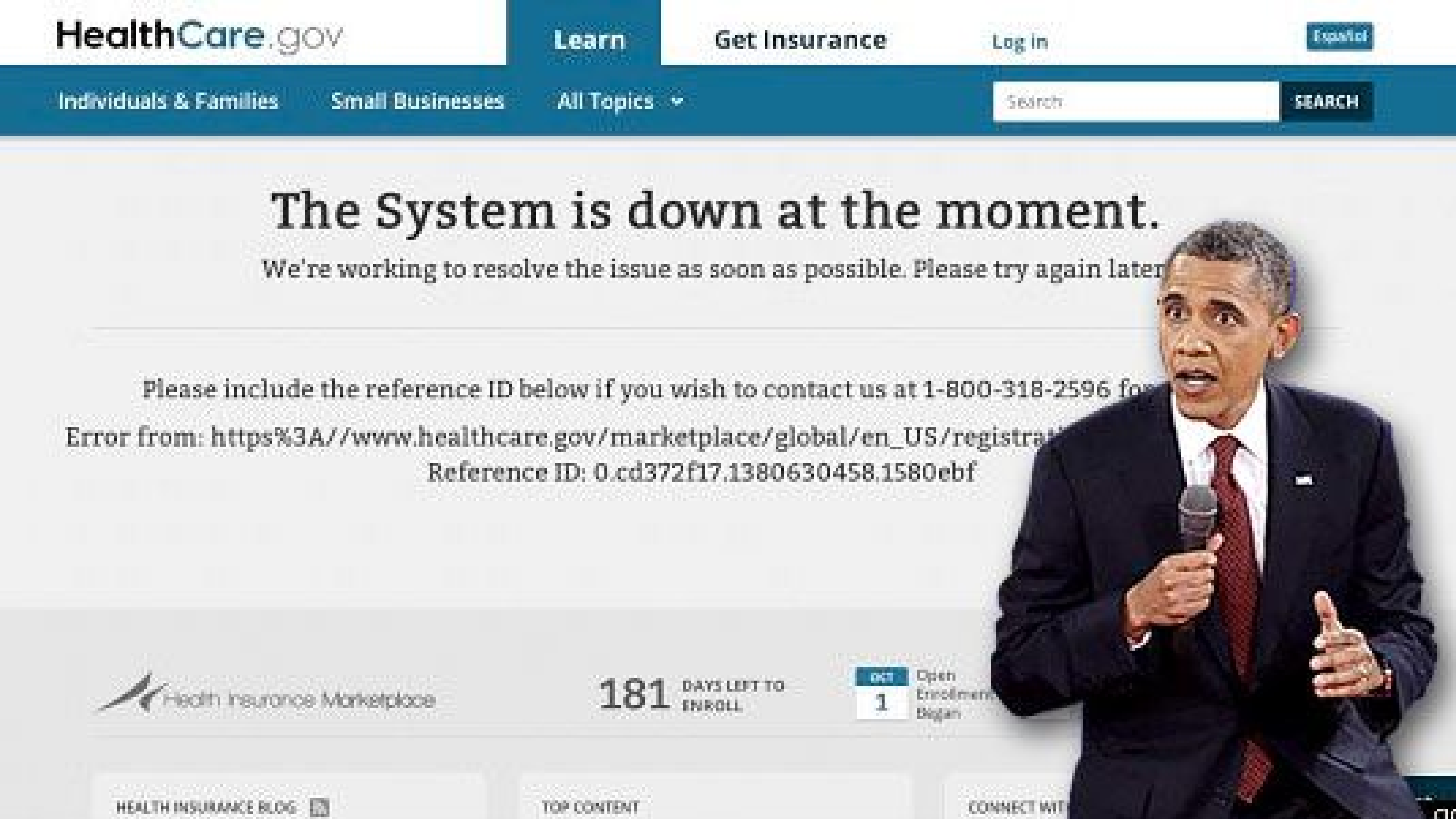
Leveson, Nancy G.; Turner, Clark S. (July 1993). "An Investigation of the Therac-25 Accidents". IEEE Computer. 26 (7): 18–41.



The Health Insurance Marketplace is ~~Open!~~ CLOSED

Enroll now in a plan that covers essential benefits, existing conditions, and more.

Plus, see if you qualify for lower costs.



The System is down at the moment.

We're working to resolve the issue as soon as possible. Please try again later

Please include the reference ID below if you wish to contact us at 1-800-318-2596 fo

Error from: https%3A//www.healthcare.gov/marketplace/global/en_US/registra

Reference ID: 0.cd372f17.1380630458.1580ebf

HealthCare.gov Application Crash (“ObamaCare”)

- USA Federal Health Insurance
- Company
  - CGI Group Inc. (Montreal/Canada)
  - CGI was a key contractor
- Failure
  - problematic launch - application poor performance
  - 2013
- Impact
  - The program does not achieve its objective of facilitating health insurance services for **millions of Americans**
  - American **election** results???

Venkatesh, V., Hoehle, H., Aljafari, R. (2014). "A usability evaluation of the Obamacare website". Government Information Quarterly. Volume 31, Issue 4, 2014, Pages 669-680

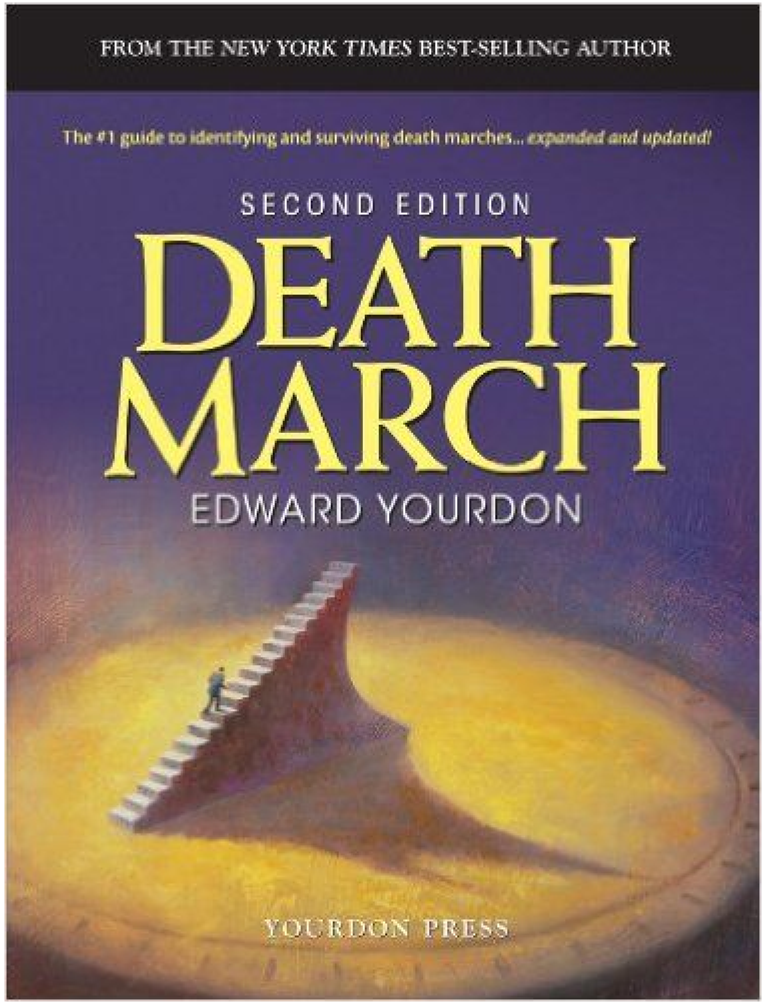
“ObamaCare” Application Crash - Causes

- integration/infrastructure issues
- Lots of subcontractors (47!!!)
    - who was ultimately responsible???
- ~60k user sim -> 250k (naïve)
- Large requirements
- “Big Bang” approach
- Huge pression
- Lack of integration testing

“We **didn’t** see full **end-to-end testing** until a **couple of days** leading up to the launch”

http://adage.com/article/digitalnext/lessons-user-experience-healthcare-gov/244933

http://www.nytimes.com/2013/10/25/us/politics/bipartisan-dismay-over-health-plan-woes-at-house-hearing.html



FROM THE NEW YORK *TIMES* BEST-SELLING AUTHOR

The #1 guide to identifying and surviving death marches... *expanded and updated!*

SECOND EDITION

DEATH MARCH

EDWARD YOURDON

YOURDON PRESS

What is a Death March project?

Death March Defined

A Death March is a project is destined to fail

In most software death march projects, this usually means one or more of the following constraints has been imposed:

- The **schedule** has been compressed to **less than half the amount of time** estimated by a rational estimating process
- The budget and associated resources have been cut in half
- The staff has been reduced to less than half the number of people that would normally be assigned to a project of this size and scope
- The **functionality**, features, performance requirements, or other technical aspects of the project **are twice** what they would be under normal circumstances and/or **discussed previously (change of requirements)**

Why it happens?

Politics,
politics,
politics.

Death March - Why it happens (more)? (1997)

- **Naive promises** made by marketing, senior executives, naive project managers, etc.
- **Naive optimism** of youth: "We can do it over the weekend!"
- The **"start-up"** mentality of fledgling, entrepreneurial companies.
- The "Marine Corps" mentality: **Real programmers don't need sleep**
- Intense competition caused by **globalization** of markets.
- Intense competition caused by the appearance of **new technologies**.
- Intense pressure caused by **unexpected government regulations**.
- **Unexpected** and/or unplanned **crises** — e.g., your hardware/software vendor just went bankrupt, or your three best programmers just died of Bubonic Plague.

So, how to address that problems? What could we do?

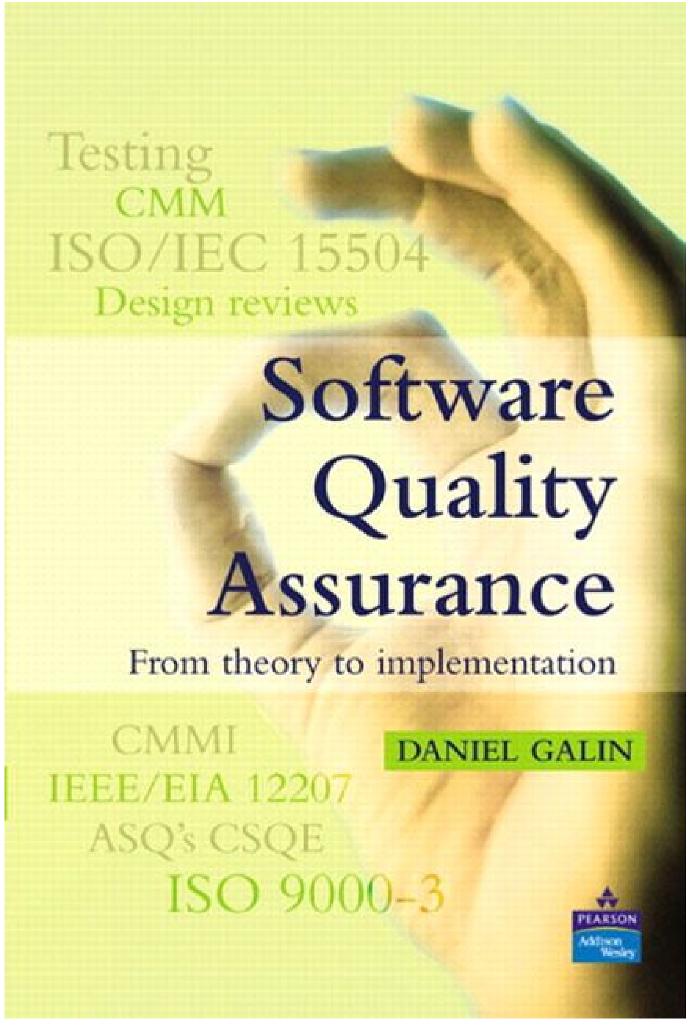
So, how to address that problems? What could we do?

In groups, discuss and propose solutions to
Death March Projects.

There is **no way to rescue** Death March Projects!

Quit as quick as possible, if it is possible.

Classic Software Quality Assurance



Testing

CMM

ISO/IEC 15504

Design reviews

Software Quality Assurance

From theory to implementation

CMMI

DANIEL GALIN

IEEE/EIA 12207

ASQ's CSQE

ISO 9000-3

PEARSON

Addison Wesley

The software quality challenge

- Product complexity
- The uniqueness of software
- Product visibility
- Traditional industry: one design phase and several manufacturing cycles
  - ISO 9000
- Software industry: only **development phase (code is a design artifact too)**
- Manufacturing software -> compiler!
- A software project never finish -> continuous **evolution** or perish

Software Definition: IEEE and ISO/IEC 9000-3

Software is:

Computer **programs**, **procedures**, and possibly associated **documentation** and **data** pertaining to the operation of a computer **system**.

Software quality (adapted from Pressman)

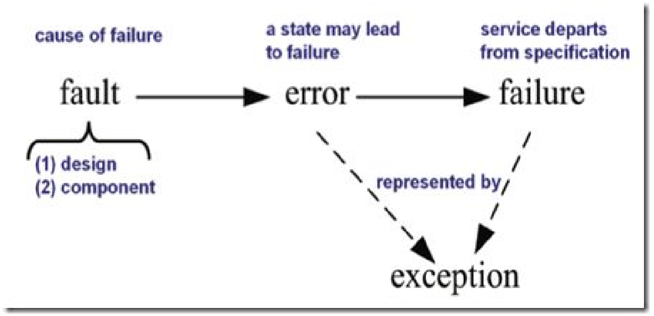
The degree of **conformance** to specific **functional** and **nonfunctional** requirements, specified software quality standards, and Good Software Engineering Practices (GSEP).

Software quality (adapted from Pressman)

The degree of **conformance** to specific **functional** and **nonfunctional** requirements, specified software quality standards (ISO 25000?), and Good Software Engineering Practices (GSEP) (???).

Fault, Error, Failure (fault-tolerant computing)

- **Fault**: A fault is the adjudged cause of an error.
- **Error**: An error is a state of the system. In the absence of any corrective action by the system, an error state could lead to a failure which would not be attributed to any event subsequent to the error.
- **Failure**: A failure is said to occur whenever the external behavior of a system does not conform to that prescribed in the system specification.



Software quality assurance objectives

1. Assuring (promising???), with **acceptable** levels of confidence (???), conformance to **functional** technical requirements.
2. Assuring, with **acceptable** levels of confidence, conformance to managerial requirements of **scheduling and budgets**.
3. **Initiating and managing activities for the improvement and greater efficiency of software development and SQA activities.**

Assurance, Quality Assurance and Quality Control

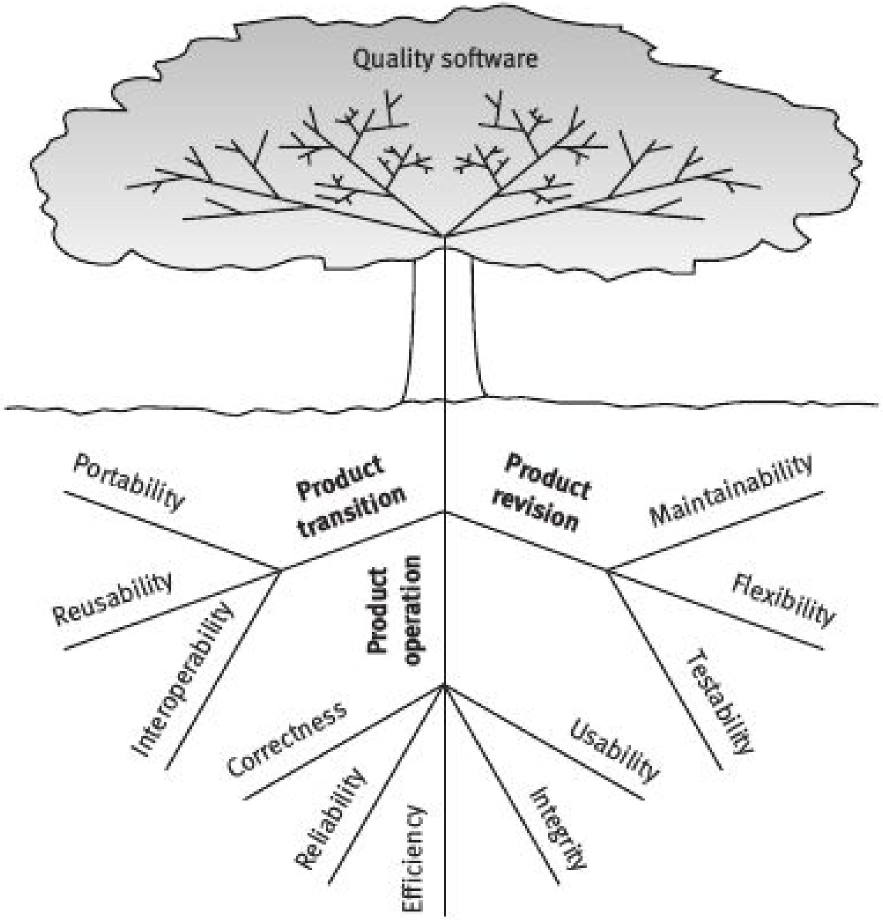
- **Assurance**: The act of giving confidence, the state of being certain or the act of making certain. **Assurance ~= Promise**
- **Quality Assurance**: The planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled.
- **Control**: An evaluation to indicate needed corrective responses; the act of guiding a process in which variability is attributable to a constant system of chance causes.
- **Quality Control**: The observation techniques and activities used to fulfill requirements for quality.

Main “classic” causes of software errors

- faulty requirements definition - natural language
- client-developer communication failures - natural language
- deliberate deviations from software requirements - political
- logical design errors - complexity
- coding errors - complexity
- non-compliance (???) with documentation and coding instructions - political
- shortcomings of the testing process - complexity
- procedure errors - complexity
- documentation errors - complexity/natural language

Software quality factors

- Comprehensive software quality requirements
- Classic **model** of software quality - **McCall**, 1977
- McCall’s factor model = 11 factors - **three** categories
- **Product operation factors**: Correctness, Reliability, Efficiency, Integrity, Usability.
- **Product revision factors**: Maintainability, Flexibility, Testability
- **Product transition factors**: Portability, Reusability, Interoperability
- Alternative models: add 5 new factors
  - Verifiability, Expandability, Safety, Manageability, Survivability


Quality software
Product transition
Product revision
Product operation
Portability
Reusability
Interoperability
Maintainability
Flexibility
Testability
Correctness
Reliability
Efficiency
Integrity
Usability

Product operation - **Correctness**

- list of the software system’s required **outputs**
- Dimensions
  - Mission (e.g. alarms temperature > 250 F)
  - Accuracy
  - Completeness
  - Up-to-dateness
  - Availability
  - Standards for coding and documenting the software system

Product operation - **Reliability**

- Reliability requirements deal with **failures** to provide service.
- Maximum allowed software system failure rate
- Example
  - Heart-monitoring reliability: less than one fail in 20 years.

Product operation - **Efficiency**

- Efficiency requirements deal with the hardware **resources** needed to perform all the functions of the software system in conformance to all other requirements.
- Example
    - Energy efficiency: consume less than 1% of a mobile battery per day (mobile app)

Product operation - **Efficiency**

- Efficiency requirements deal with the hardware **resources** needed to perform all the functions of the software system in conformance to all other requirements.
- Example
  - Energy efficiency: consume less than 1% of a mobile battery per day (mobile app)

Product operation - **Efficiency, Integrity, Usability**

- **Efficiency** requirements deal with the hardware **resources** needed to perform all the functions of the software system in conformance to all other requirements.
    - Energy efficiency: consume less than 1% of a mobile battery per day (mobile app)
- **Integrity** requirements deal with the software system security (authentication and authorisation), and to maintain correctness and harmony among all related pieces of data.
- **Usability** is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use. **Easy to use and learn**.

Product operation

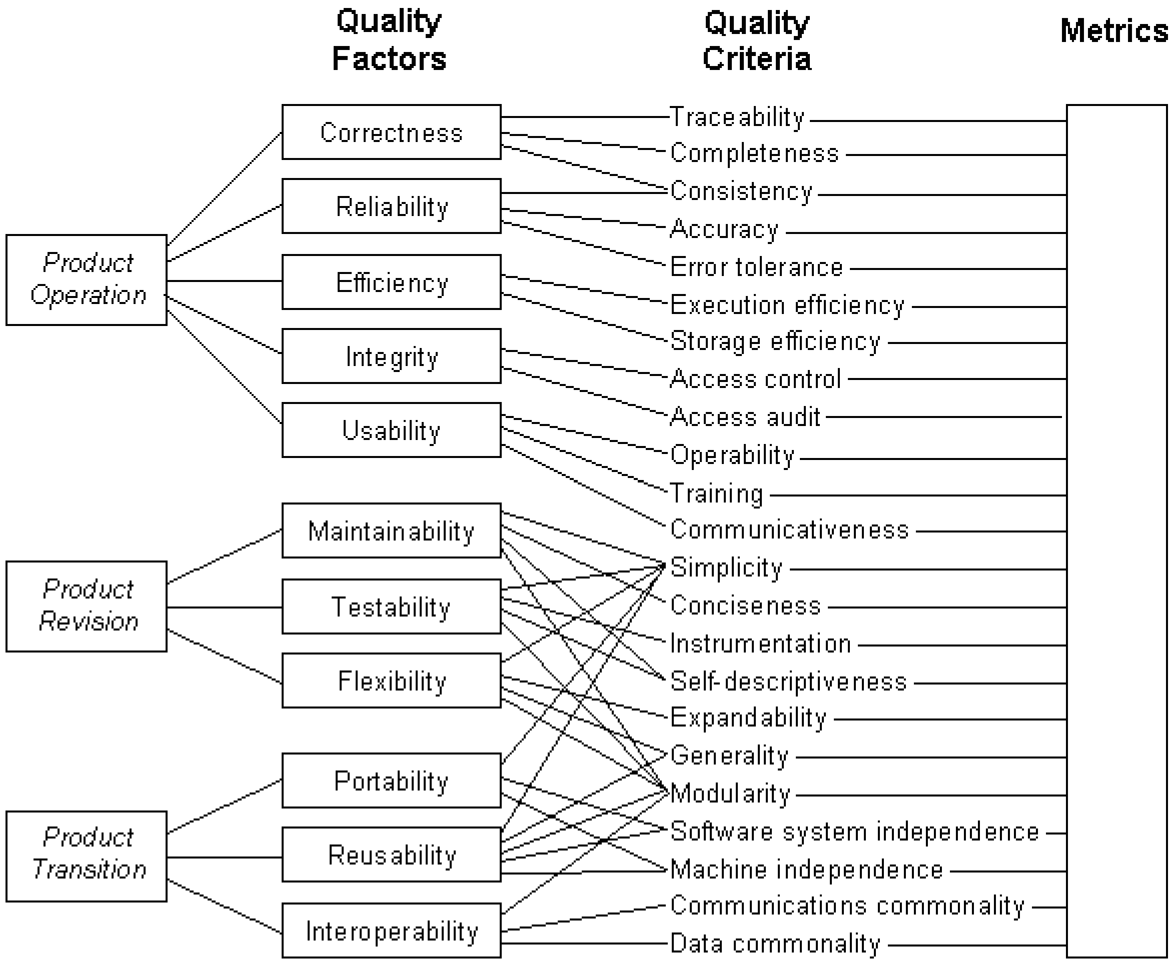
- Efficiency requirements deal with the hardware **resources** needed to perform all the functions of the software system in conformance to all other requirements.
- Example
    - Energy efficiency: consume less than 1% of a mobile battery per day (mobile app)

Product revision - **Maintainability, Flexibility, Testability**

- **Maintainability** requirements determine the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections. **Coding practices and engineering**!
- **Flexibility** is capabilities and efforts required to support adaptive maintenance activities are covered by the flexibility requirements. **Software architecture!**
- **Testability** requirements for the ease of testing are related to special features in the programs that help the tester, providing predefined results and **log** files. Testability requirements related to software operation include **automatic diagnostics** and to obtain a **report** about the detected faults.

Product Transition - **Portability, Reusability, Interoperability**

- **Portability** requirements tend to the **adaptation** of a software system to other environments consisting of different hardware, different operating systems, and so forth.
- **Reusability** requirements deal with the use of software **modules** originally designed for one project in a new software project currently being developed.
  - Old video game industry problem
- **Interoperability** requirements focus on creating **interfaces** with other software systems or with other equipment firmware.


Quality Factors
Quality Criteria
Metrics
Product Operation
Product Revision
Product Transition
Correctness
Reliability
Efficiency
Integrity
Usability
Maintainability
Testability
Flexibility
Portability
Reusability
Interoperability
Traceability
Completeness
Consistency
Accuracy
Error tolerance
Execution efficiency
Storage efficiency
Access control
Access audit
Operability
Training
Communicativeness
Simplicity
Conciseness
Instrumentation
Self-descriptiveness
Expandability
Generality
Modularity
Software system independence
Machine independence
Communications commonality
Data commonality

SQA System Components

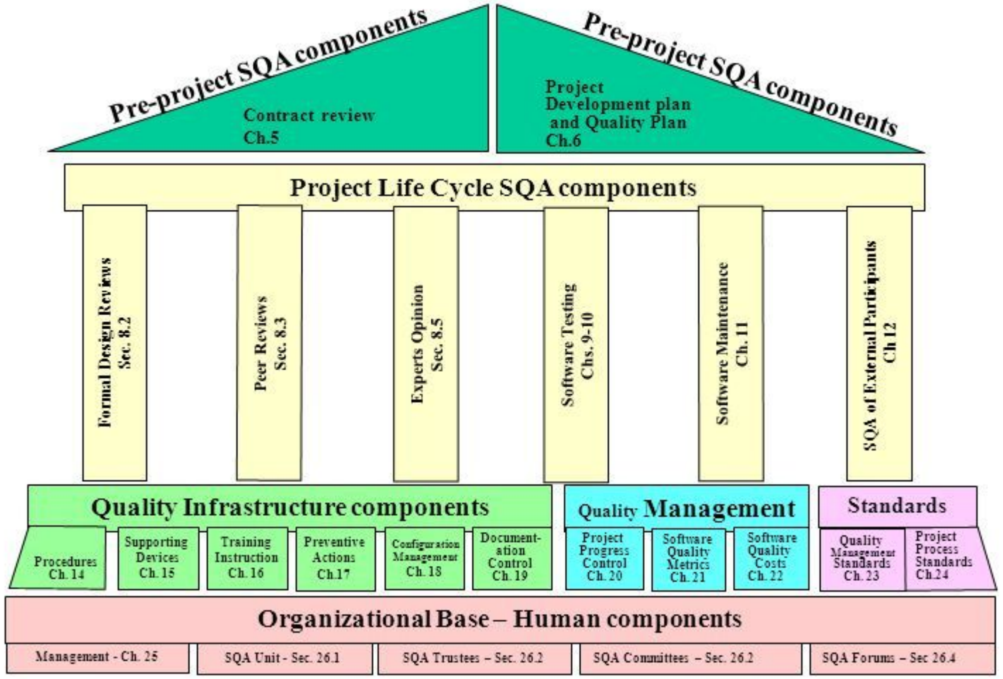
- Pre-project
- Project life cycle activities assessment
- Infrastructure error prevention and improvement
- Software quality management
- Standardization, certification, and SQA system assessment
- Organizing for SQA – the human components

DANIEL GALIN'S QUALITY SHRINE

Pre project	*Software Project Life Cycle*	Quality Infrastructure	Quality Management	Standards	Organizational Based – Human
Contract Review	Review	Procedures	Project progress Control	Quality Management Standards	SQA Unit, *SQA Trustees*, *SQA Committees* dan *SQA Forums*
Development and Quality Plans	Expert Opinions	Supporting Devices	Software Quality Metrics	Project Process Standards	
	Software Testing	Preventive & Corrective Actions	Software Quality Cost		
	Software Maintenance				
	3rd Party Quality				

OHT 4.18

The Software Quality Shrine

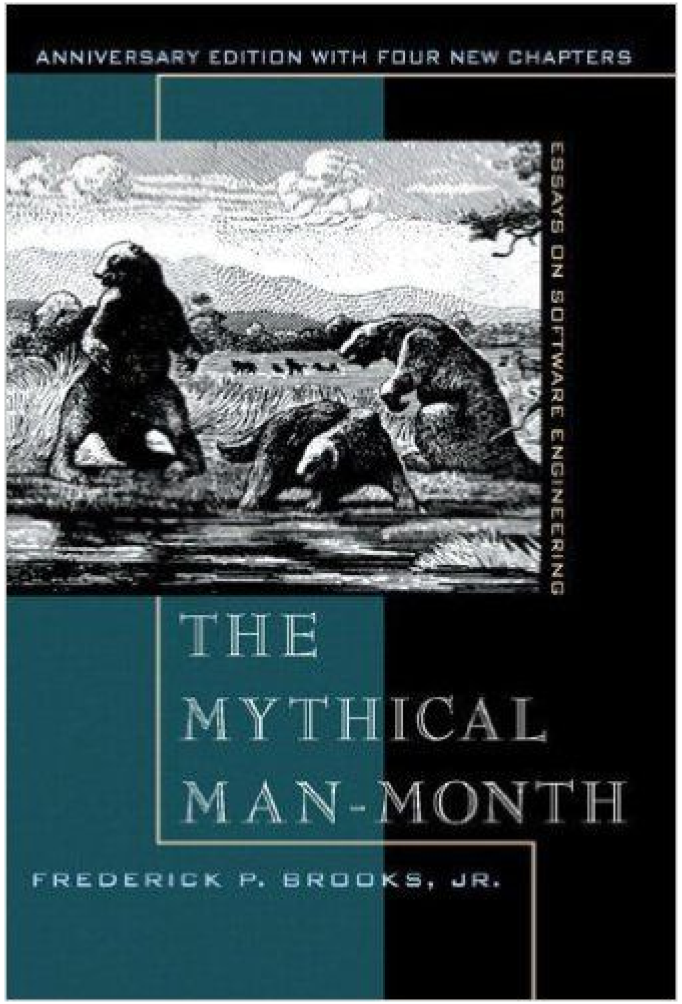


Thus, what is the
main issue
on the **“classic”**
Software Quality Assurance
approach?

Is Therac-25 or HealthCare.gov a **same** kind of project that a **Indie game**?

What is the main issue on the **“classic”** Software Quality Assurance approach?

To use the **same approach** for any kind of software project!!!


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

No Silver Bullet!

(Brooks, 1975)

Software is a wicked problem

*“A **wicked problem** is a problem that is **difficult** or **impossible** to solve because of **incomplete, contradictory, and changing requirements** that are often difficult to recognize.” [Wikipedia, 2016]*

Complex interdependencies!

Agile Quality Assurance Perspective

McBreen (2003) defines agile quality assurance as the development of software that can **respond to change**, as the customer requires it to change. This implies that the **frequent delivery** of tested, working, and **customer-approved software** at the end of each iteration.

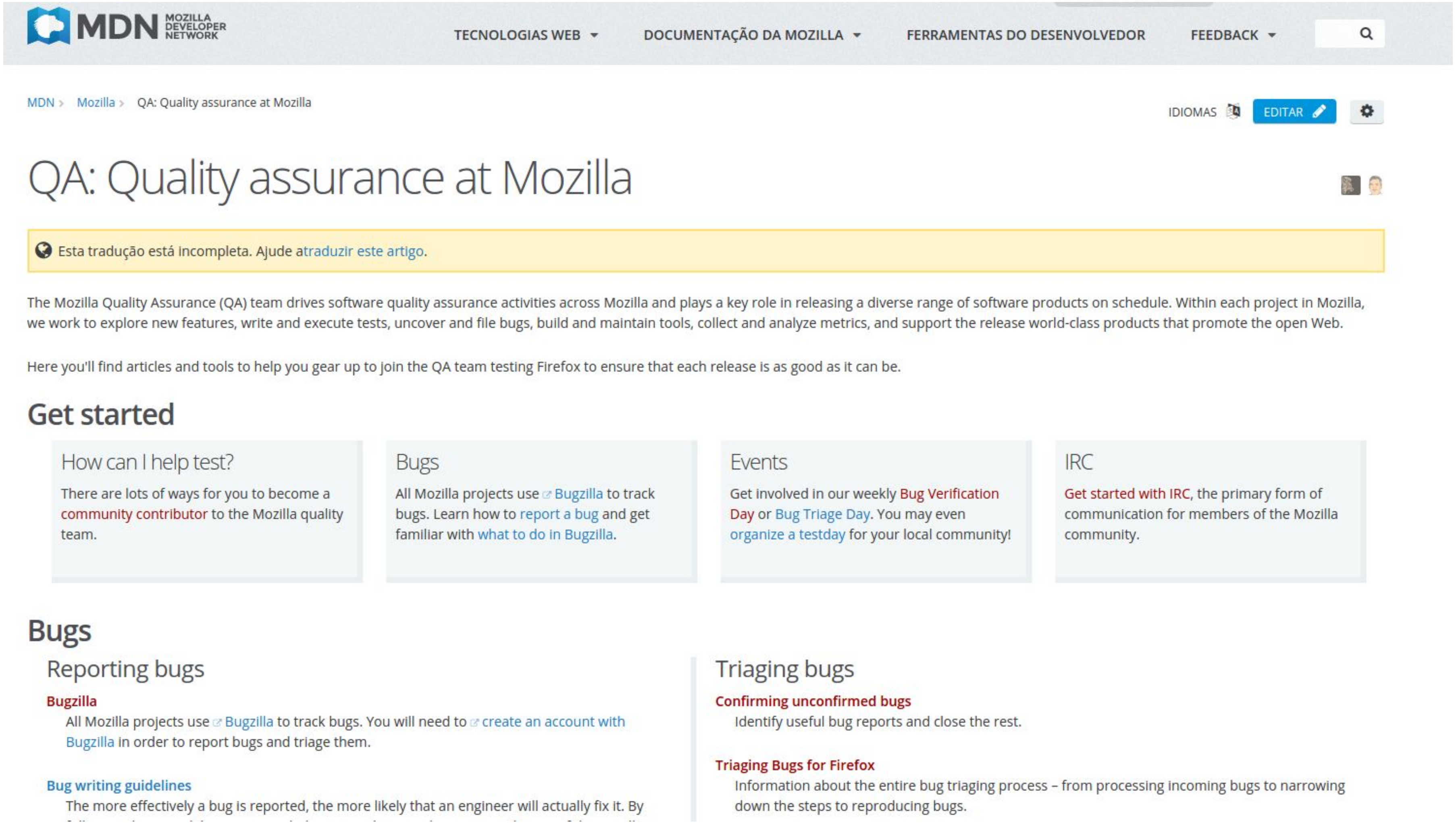
Ambler (2005) considers agile quality to be a result of practices such as effective **collaborative work**, **incremental** and **iterative** development as implemented through techniques such as **refactoring**, t**est-driven development**, **modelling**, and effective **communication** techniques.

Next course

Agile Software Quality Assurance

TP 1 - Analyse et Diagnostic

- **Objective**: analyse and report a diagnostic from a software quality assurance process using this 4 dimensions/perspectives:
    - 1) McCall's Software Quality factors model
    - 2) Galin's SQA System Components
    - 3) Agile SQA
    - 4) ISO 25000
    - Or follow the ISO 15504-5:2006 (Process Evaluation Norm) - Optional
- Find and choose an open source project (or a company that support you)
- Analyse, evaluate and report the project Software Quality Assurance on that 4 dimensions
- Grade is proportional to complexity, completeness, conciseness
- Groups of 3 participants (ideal, max. 4). **One report per group on Moodle!**
- Deadline: **February, 12 to the BOTH GROUPS (B1 AND B2)!**



MDN > Mozilla > QA: Quality assurance at Mozilla

QA: Quality assurance at Mozilla

Esta tradução está incompleta. Ajude atraduzir este artigo.

The Mozilla Quality Assurance (QA) team drives software quality assurance activities across Mozilla and plays a key role in releasing a diverse range of software products on schedule. Within each project in Mozilla, we work to explore new features, write and execute tests, uncover and file bugs, build and maintain tools, collect and analyze metrics, and support the release world-class products that promote the open Web.

Here you'll find articles and tools to help you gear up to join the QA team testing Firefox to ensure that each release is as good as it can be.

Get started

How can I help test?

There are lots of ways for you to become a community contributor to the Mozilla quality team.

Bugs

All Mozilla projects use Bugzilla to track bugs. Learn how to report a bug and get familiar with what to do in Bugzilla.

Events

Get involved in our weekly Bug Verification Day or Bug Triage Day. You may even organize a testday for your local community!

IRC

Get started with IRC, the primary form of communication for members of the Mozilla community.

Bugs

Reporting bugs

Bugzilla
All Mozilla projects use Bugzilla to track bugs. You will need to create an account with Bugzilla in order to report bugs and triage them.

Bug writing guidelines
The more effectively a bug is reported, the more likely that an engineer will actually fix it. By

Triaging bugs

Confirming unconfirmed bugs
Identify useful bug reports and close the rest.

Triaging Bugs for Firefox
Information about the entire bug triaging process – from processing incoming bugs to narrowing down the steps to reproducing bugs.

QA

In other languages: **English** (en) | Español (es) | Italiano (it) | Português do Brasil (pt-br) | 中文(中国大陆) (zh-cn) | Русский (ru) [edit]

Fedora Quality Assurance

Welcome to the Fedora QA project page. Fedora QA is the project which covers all testing of the software that makes up Fedora. It's our goal to continually improve the quality of Fedora releases and updates.

Activities

The Quality Assurance project is engaged in the following activities:

- Testing of software as it is released into Rawhide, Branched pre-releases, updates-testing, or as it appears in a supported public release
- Testing all updates to critical path packages before they are accepted
- Acting as a bridge between users and developers that aids in fixing and closing bugs
- Developing and executing test plans and test cases to test important functionality in a systematic way, usually with multiple cooperating testers
- Developing and running tools which use automation to find potential bugs
- Running test days to co-ordinate focused testing on a specific feature or component
- Working with developers and release engineers to maintain the release criteria, which are used to determine what bugs should be fixed before a pre-release or final release of Fedora is made
- Managing the release process along with the ReleaseEngineering team, including requesting candidate composes, performing release validation, and managing blocker and freeze exception bugs
- Working with the Working Groups to plan testing for the Fedora Products

Pending and completed tasks are listed in Trac and, for tooling, Phabricator.

Get Involved

We're always eager to have new contributors to the QA project, no matter your experience level. If you'd like to get involved with helping to make Fedora better, read the What can I do? page to find out how to join in with QA activities. There are tasks available for every level of expertise and available spare time.

If you'd like to get involved with Fedora but QA doesn't sound like the group for you, think about joining another Fedora project instead.

Communicate

QA project meetings are held **Mondays** at **16.00 UTC** (or **15.00 UTC** during daylight savings time) in the fedora meeting channel on IRC. Everyone is welcome to come along, especially if you're thinking of getting involved with QA and would like to ask some questions first.

General info on QA meetings such as topics for future meetings and records from previous meetings can be found at QA/Meetings.

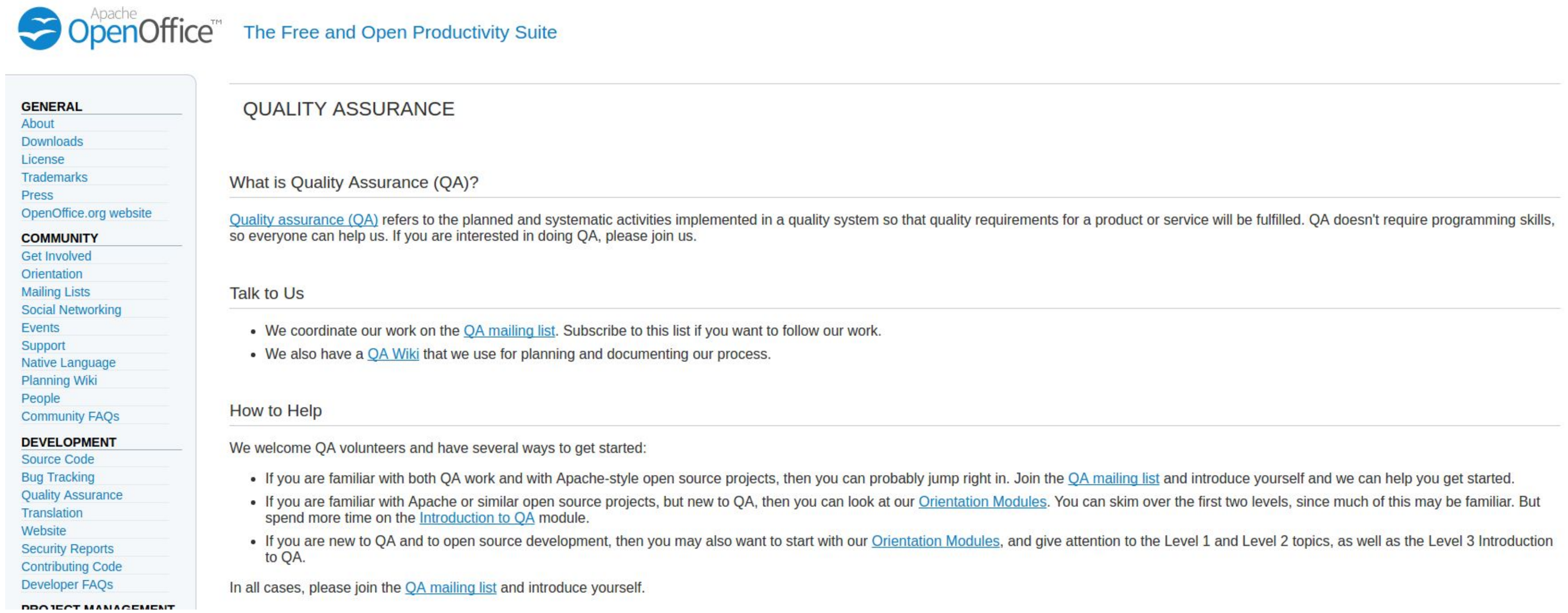
WIKI
- Fedora Project Wiki
- News
- Events
- Features
- Recent changes
- Random page
- Help

NAVIGATION
- Home
- Get Fedora
- Join Fedora

SUB-PROJECTS
- Ambassadors
- Community Operations
- Design
- Documentation
- EPEL
- Infrastructure
- Internationalization
- Localization
- Marketing
- Magazine
- Package Maintainers
- Quality Assurance
- Websites
- All projects

SEARCH
Go Search

TOOLS
- What links here



Apache OpenOffice™ The Free and Open Productivity Suite

GENERAL
- About
- Downloads
- License
- Trademarks
- Press
- OpenOffice.org website

COMMUNITY
- Get Involved
- Orientation
- Mailing Lists
- Social Networking
- Events
- Support
- Native Language
- Planning Wiki
- People
- Community FAQs

DEVELOPMENT
- Source Code
- Bug Tracking
- Quality Assurance
- Translation
- Website
- Security Reports
- Contributing Code
- Developer FAQs

PROJECT MANAGEMENT

QUALITY ASSURANCE

What is Quality Assurance (QA)?

Quality assurance (QA) refers to the planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled. QA doesn't require programming skills, so everyone can help us. If you are interested in doing QA, please join us.

Talk to Us

- We coordinate our work on the QA mailing list. Subscribe to this list if you want to follow our work.
- We also have a QA Wiki that we use for planning and documenting our process.

How to Help

We welcome QA volunteers and have several ways to get started:

- If you are familiar with both QA work and with Apache-style open source projects, then you can probably jump right in. Join the QA mailing list and introduce yourself and we can help you get started.
- If you are familiar with Apache or similar open source projects, but new to QA, then you can look at our Orientation Modules. You can skim over the first two levels, since much of this may be familiar. But spend more time on the Introduction to QA module.
- If you are new to QA and to open source development, then you may also want to start with our Orientation Modules, and give attention to the Level 1 and Level 2 topics, as well as the Level 3 Introduction to QA.

In all cases, please join the QA mailing list and introduce yourself.

TP on Savoir Faire Linux (Ring)

- **SFL** is available and open to receive **2 groups** to do the TPs
- On the Ring Team
- True “in company” work
- True consulting work
- Strongly possible to be hired after the project
- More challenging (and fun) that just visit a web site or send emails
- Bonus in the grade
- Flexible deadlines (external justification)
- Interested? Send me an email
  - Subject: [LOG8371] - TP on Savoir Faire Linux
  - Team members name and email
  - The 2 first groups to send email will be contacted to a SFL meeting!

Project d’investigation - Rémise R1

- **Définition du sujet**
- **Objectives: Vision large**
- **La question de recherche principale**
- structure du papier
- révision des travaux connexes (simplifier)
  - Principaux papier
  - Principaux auteurs
- Pondération: 10%
- **Individuel et personnelle**
  - Si plusieur personnes vont travailler avec comme co-auteur, liste les collaborateur et votre **contribution personnelle** dans le projet.
- **Remise: 27 janvier 2017 sur Moodle**

Next course

Agile Software Quality Assurance