Évaluation des Logiciels

Anne Canteaut, Miguel Angel Fernández, Luc Maranget, Sophie Perin, Mario Ricchiuto, Manuel Serrano, Emmanuel Thomé

Commission d'Evaluation

Évaluation des Logiciels

Préparé par le groupe de travail « évaluation logiciel »
composé des membres de la Commission d'Evaluation d'Inria suivants :
Anne Canteaut, Miguel Fernandez, Luc Maranget, Sophie Perin,
Mario Ricchiuto, Manuel Serrano (rédacteur), Emmanuel Thomé

Document validé par la Commission d'Évaluation Inria le 14 janvier 2021

Janvier 2021

Cette note est destinée aux membres des jurys Inria afin de les aider dans leur évaluation des logiciels développés par des chercheurs. Elle est le fruit du travail du groupe de travail « évaluation logiciel » mené par la Commission d'Évaluation de mars à décembre 2020.

1 Préambule

Les candidats chercheurs aux divers concours de recrutement et de promotions Inria sont invités à qualifier leurs éventuels développements logiciels en utilisant la terminologie décrite dans le document « Critères d'auto-appréciation pour les logiciels V2 » datant de juin 2011. Nos expériences de participations à ces jurys nous ont enseigné que, si ce document est d'une aide précieuse pour les candidats, il n'est souvent pas exploité de manière appropriée par les jurys. La présente note a pour but de corriger ce point. Elle est un complément du document remis aux candidats et aux équipes. Elle est à destination des membres des jurys mais elle peut aussi servir aux panels d'experts en charge de l'évaluation des équipes. Afin d'avoir deux documents complémentaires et cohérents il a été nécessaire de procéder à quelques révisions du document d'auto-appréciation. La nouvelle version remplaçant la précédente est nommée « Criteria for Software Self-Assessment V3 ».

Le premier hiatus qu'il nous semble important de relever est que le document d'auto-appréciation ne constitue pas, en soi, une évaluation des logiciels et ce d'autant moins qu'il est rédigé par les candidats eux-mêmes. En revanche, les candidats doivent bien saisir que, par ce document, ils indiquent les aspects de leurs développements logiciels qu'ils veulent mettre en avant, et sur quels critères ils souhaitent qu'ils soient évalués. Aux jurys alors de procéder à cette évaluation.

Le second hiatus est que l'évaluation des logiciels, comme celle de tout autre activité scientifique et de recherche, ne peut être régie par des simples critères syntaxiques et arithmétiques. Il est donc demandé aux rapporteurs d'un candidat un *avis*, qui est personnel et qui bien entendu comportera une part de subjectivité, comme cela est également le cas pour l'évaluation des articles, ni plus ni moins. Cet avis doit pouvoir être argumenté et ne peut pas se cantonner à une répétition d'une « auto-évaluation » déclarative par le candidat. L'évaluation doit donc se fonder sur ce qu'*est* le logiciel et donc en premier lieu sur l'examen du code source en s'aidant éventuellement d'évaluations par les pairs qui sont de plus en plus

fréquentes dans de nombreux domaines, ou à défaut, lorsque le logiciel ne peut être diffusé publiquement par des avis circonstanciés d'utilisateurs ou de commanditaires ou des résultats issus du logiciel. L'évaluation doit aussi porter sur la diffusion du logiciel, sa documentation, sa stabilité, l'investissement dans la durée qu'a nécessité son développement, sur des éléments attestant de son utilisation, etc. Comme pour les nombreux autres aspects développés dans un dossier de candidature ou de promotion, il appartient également aux rapporteurs d'apprécier si le développement logiciel est, ou n'est pas, un aspect *important* dans le dossier.

Cette note a pour but de donner des indications sur la façon de réaliser l'évaluation. Elle le fait de façon relativement informelle en listant les questions ou les points qui pourront être soulevés par le jury au moment de son évaluation. L'élément central que cette évaluation devra caractériser, indépendamment du type de développement, est l'importance du développement logiciel dans le dossier d'un candidat. Pour certains candidats le développement est anecdotique et ne mérite pas d'attention particulière du jury. Pour d'autres en revanche, l'activité de recherche est organisée autour de la réalisation et la diffusion d'un ou plusieurs logiciels. Pour ces candidats, le jury sera particulièrement attentif à la qualité et au rôle joué par le développement dans la recherche du candidat.

2 Évaluation du rôle du candidat

Quel que soit le type de logiciel, le premier objectif de l'évaluation sera de caractériser le rôle du candidat dans le développement selon deux axes: la nature des contributions et la durée de l'implication. Pour cela on distinguera :

- une participation anecdotique (simple réalisation d'exemples, fourniture de quelques correctifs, rédaction de tutoriels ou de quelques éléments de la documentation) ;
- une participation au cœur du logiciel (écriture de composants essentiels) ;
- une participation à la diffusion (écriture de scripts d'installation, de déploiement, etc.).
- ...

Enfin, dans le cas d'un logiciel réalisé par plusieurs auteurs, on demandera aux évaluateurs de justifier de la nature de la contribution par des arguments factuels.

Dans ce document, nous préconisons de ne considérer le développement logiciel d'un candidat que si ce dernier est directement impliqué dans sa réalisation informatique, c'est-à-dire dans l'écriture de son « code source ». A contrario, nous considérons que le terme d'« architecte » si fréquemment employé pour les chercheurs, désigne en fait le plus souvent un rôle d'« encadrement », bien souvent d'étudiants, ou une supervision d'ingénieur. Cette fonction est importante mais doit être mentionnée dans la rubrique idoine consacrée à l'encadrement et non dans celle destinée au développement. De façon analogue, les activités de transfert, comme par exemple la mise en œuvre ou l'animation d'un consortium lié à un logiciel ne devra pas être assimilée à une activité de développement mais à une activité de transfert.

Tout comme on ne considère que quelqu'un est l'auteur d'une publication que si son nom apparaît clairement en tête du texte, on ne considérera qu'un chercheur est l'auteur d'un logiciel que s'il existe des traces tangibles de son implication personnelle dans le développement

(son nom est présent dans les sources ou dans l'historique des systèmes de gestion de versions, etc.).

Le rôle d'« architecte logiciel » dans son sens plus industriel [1], se situe à mi-chemin entre la supervision et le développement. Il est donc plus difficile à caractériser. En général, l'architecte logiciel doit organiser le développement d'un logiciel conséquent devant respecter un cahier des charges destiné à satisfaire les exigences d'utilisateurs. Le développement s'étale généralement sur une longue période et les composants qui le forment sont nombreux et fréquemment développés par des équipes différentes. Certains logiciels issus de la recherche académiques rentrent effectivement dans cette catégorie. Le rôle de l'évaluation sera donc de déterminer si la complexité du logiciel développé peut justifier qu'un chercheur revendique d'en être l'architecte et dans le cas positif d'évaluer l'investissement du candidat dans cette activité.

Enfin, un autre aspect essentiel de l'évaluation sera de considérer la pertinence du développement dans le dossier d'un candidat. Voici quelques exemples de questions pour lesquelles on pourra s'attendre à trouver des réponses dans les rapports d'évaluation. *Est-ce que le développement a été conduit de façon cohérente vis-à-vis des recherches académiques du candidat ? Est-ce que les objectifs du développement logiciel sont clairs et pertinents ? Est-ce que la recherche justifiait un développement conséquent* ou au contraire *est-ce que le développement n'a pas été sur-dimensionné au regard des objectifs scientifiques recherchés* ? Ou enfin, *Est-ce que le développement logiciel a atteint ses objectifs ?*

3 Caractériser le type de logiciel

Les logiciels forment une population hétérogène. Il y a en des petits et il y en a des gros. Il y a en a des complexes dont la réalisation prend des années et d'autres qui sont presque réalisés sur un coin de table pour rendre un service particulier à un moment particulier. Il y en a dont la réalisation informatique est particulièrement ardue et nécessite un développement très minutieux et d'autres dont la difficulté est plutôt à chercher dans les développements scientifiques préalables. Il y en a qui sont des outils permettant d'obtenir des résultats scientifiques autres. Il y en a qui sont des résultats de recherche, par exemple comme preuve d'existence d'une solution algorithmique à un problème donné. La seconde tâche de l'évaluation est donc de *caractériser* un logiciel. Pour cela nous proposons de les classer en quatre grandes familles qui permettent d'affiner l'évaluation. Bien entendu, ces catégories peuvent avoir des recouvrements et il peut se présenter des logiciels qui appartiennent à plus d'une famille. Il peut aussi exister des logiciels dont le statut évolue au cours du temps.

La caractérisation du logiciel devra être mise en regard des contributions du candidat car certains logiciels bénéficient de contributions de natures très différentes. Cela sera probable-

[1] Selon Wikipedia (`https://fr.wikipedia.org/wiki/Architecte_logiciel`), un architecte logiciel est présenté comme *« un expert en informatique qui est responsable de la création et du respect du modèle d'architecture logicielle. Il se distingue de l'architecte informatique qui travaille sur le matériel. C'est un informaticien professionnel agréé ou un ingénieur logiciel membre d'un ordre professionnel dans plusieurs pays. Le rôle d'architecte logiciel peut, dans de petits projets, être tenu par l'analyste, le chef de projet ou le développeur responsable du projet. Dans les grandes entreprises, il est possible de trouver un architecte logiciel en chef qui est responsable de l'application des normes d'architecture à l'ensemble des projets et de la gestion et de la réutilisation des composants logiciels de l'entreprise. Dans les projets d'envergure, il est possible de retrouver un architecte logiciel et plusieurs sous-architectes logiciel, responsables de parties disjointes du logiciel à construire. L'analyste fonctionnel, l'architecte logiciel et l'ingénieur logiciel ont des responsabilités distinctes : le premier pilote les cas d'utilisation, le second l'architecture et le troisième le développement. »*

ment le cas des gros logiciels ayant été développés par des équipes conséquentes. Pour ces derniers, il est envisageable que certains des contributeurs se soient concentrés sur les aspects *scientifiques* du développement alors que d'autres se seront plus focalisés sur les aspects de *mise en oeuvre pratique*. Les contributions individuelles devront donc être évaluées pour ce qu'elles sont et pour la place qu'elles occupent dans le parcours et le dossier d'un candidat.

3.1 Les logiciels « vecteur de savoir »

Il s'agit de logiciels dont la finalité est double. D'une part ils réalisent une tâche précise correspondant à un objectif scientifique clair et qui donne sa valeur au logiciel. D'autre part, ils communiquent un savoir en tant qu'objet scientifique. Soit parce que l'examen de leur construction est instructive, soit parce qu'ils permettent à d'autres chercheurs de bâtir des déclinaisons ou des variantes qui permettront d'explorer d'autres pistes de recherche. Le premier aspect est important car c'est lui qui justifie l'intérêt de l'existence du logiciel. Le deuxième aspect est crucial car c'est lui qui donne au logiciel son statut d'objet scientifique.

La conséquence est que pour qu'un logiciel puisse rentrer dans cette catégorie son **code source doit obligatoirement être disponible publiquement** car c'est là, en grande partie, que se trouve le savoir diffusé.

Par ailleurs, un logiciel « vecteur de savoir » doit également satisfaire un ou plusieurs des critères suivants (liste *non exhaustive*).

- La réalisation s'appuie sur des résultats de recherche qui sont décrits dans des productions classiques telles que des articles de conférences ou de journaux, des livres, etc.
- Le développement de ce logiciel nourrit une activité de recherche décrite classiquement dans des articles scientifiques et donne lieu à d'autres développements logiciels.
- La réalisation du logiciel a suscité de nouvelles avancées scientifiques.
- Il est principalement développé par un ou plusieurs chercheurs (éventuellement, bien entendu, avec l'aide d'ingénieurs).
- Il est réalisé dans le but d'être diffusé. Ainsi, il bénéficie d'un site web qui le présente, le documente et indique comment l'installer.
- Le code source est disponible publiquement (gitlab, github, ou autre). L'historique des développements, fourni par les systèmes de contrôle de version, est d'une grande valeur pour comprendre la dynamique des évolutions du logiciel.
- ...

3.1.1 Critères d'évaluation

Voici quelques critères à retenir pour évaluer et comparer les logiciels de cette catégorie.

- Complexité scientifique du problème traité.
- Complexité de la réalisation (taille du code, spécificités matérielles, portage complexe sur machines ésotériques, difficulté informatique intrinsèque, taille de l'équipe de développement, ...).

- Liens entre le logiciel et les publications scientifiques (comment le développement logiciel et les résultats académiques se nourrissent les uns et les autres, même démarche pour les thèses peut-être en mode plus mineur, évaluation du logiciel dans les conférences qui pratiquent l'évaluation d'artefacts, ...).
- Qualité du développement (usage des bonnes pratiques, méthodes de développement « contemporaines », qualité du code, ...).
- Qualité de la diffusion (site web, documentation, facilité d'installation, packaging, etc).
- Impact (existence d'une communauté d'utilisateurs attestée par des contributions extérieures, citations dans d'autres travaux, inclusion dans d'autres systèmes ouverts, etc).
- ...

3.1.2 Exemples

- Coq, `https://coq.inria.fr/` : assistant de preuve. Son développement a nécessité de nombreuses avancées dans le domaine des langages de programmation. Son évolution s'appuie sur une réflexion autour de la « preuve » mathématique. Son impact scientifique est visible en informatique et en mathématique.
- FreeFem, `https://freefem.org/` : logiciel de calcul scientifique Open Source permettant de résoudre numériquement des équations différentielles par la méthode des éléments finis. Il possède son propre langage de script, inspiré du `C++`, pour faciliter l'écriture de la formulation variationnelle de problèmes aux limites avec des équations aux dérivées partielles. La plate-forme a été développée pour faciliter l'enseignement et la recherche fondamentale grâce au prototypage de scripts.
- SimGrid, `https://simgrid.org/` : Une plateforme logiciel pour le développement de simulateurs d'application distribuées. Cette plateforme a permis de nombreuses expériences sur de longues périodes et a permis la publication d'un nombre incalculable d'articles.
- FEniCS, `https://fenicsproject.org` : logiciel de calcul scientifique Open Source permettant de résoudre numériquement des équations différentielles par la méthode des éléments finis. Il possède une interface Python qui facilite l'écriture de la formulation variationnelle de problèmes aux limites avec des équations aux dérivées partielles et leur résolution numérique avec du parallélisme via l'interfaçage avec des bibliothèques externes (e.g., PETSc).
- GNU MPFR, `https://www.mpfr.org/` : bibliothèque open source de calcul flottant multiprécision, avec arrondi correct. GNU MPFR a eu un impact important sur les évolutions du standard IEEE-754 ainsi que via de nombreux projets logiciels qui ont bâti autour de MPFR, qu'il s'agisse de l'utiliser comme une brique de base, ou bien de l'étendre en s'inspirant de la structure des algorithmes implantés dans GNU MPFR (cas des bibliothèques proposant des extensions à $\mathbb{C}$ ou aux polynômes).
- Si les cinq exemples ci-dessus peuvent être qualifiés de *gros* projets logiciels, cette catégorie peut également concerner des projets de plus petite taille comme par exemple

les projets qDSA (`https://joostrenes.nl/software.html`) ou `relic-toolkit` (`https://github.com/relic-toolkit`). Dans ces cas, l'intérêt ne réside pas tant dans ce que le logiciel réalise, que dans la façon de le faire: le code source du logiciel, qui répond à des objectifs de petite taille, ou de performance d'exécution, est un objet d'étude essentiel.

- Unison, `https://www.cis.upenn.edu/~bcpierce/unison/` : Unison est un outil de synchronisation de fichiers. Il constitue un exemple intéressant car au début de son existence il a été un logiciel de recherche servant de terrain d'expérimentation pour des avancées en programmation distribuée. Dans un second temps, il est devenu un logiciel utilitaire. Il n'est de nos jours plus utilisé pour la recherche en programmation distribuée mais il reste largement utilisé pour la synchronisation entre machines.

3.2 Les logiciels « support de recherche »

Dans cette rubrique se trouvent les logiciels qui sont en relation étroite avec la recherche mais qui ne sont pas directement en eux-mêmes l'expression d'un résultat scientifique. Souvent le but premier des logiciels « support de recherche » est de valider des travaux scientifiques.

Ces logiciels sont principalement organisés en deux catégories. La première est constituée des logiciels utilisés pour valider des travaux ou pour les rendre reproductibles. Il s'agit par exemple de logiciels, généralement petits, qui ont été écrits pour valider une expérience ou un algorithme décrit dans un article. Parfois le soin apporté à leur développement est sommaire car leur durée de vie et leur finalité sont limitées. Parfois, un plus grand soin est apporté car leur intérêt excède la simple reproduction d'une expérience. C'est par exemple le cas lorsque leur usage peut se généraliser et qu'ils peuvent servir à la réalisation de toute une famille d'expérimentations.

La seconde catégorie est constituée des plates-formes logicielles, c'est-à-dire des ensembles de briques logicielles (routines, structure de données, interfaces d'entrée/sortie, etc.), qui servent à fédérer les développements de plusieurs chercheurs, voire d'une ou plusieurs équipes de recherche. Ces plates-formes permettent de factoriser les développements et donnent un cadre qui permet à de nouveaux chercheurs, par exemple des doctorants, de contribuer à des projets dont l'ampleur dépasse ce qu'ils pourraient développer de façon isolée.

3.2.1 Critères d'évaluation

Voici quelques critères à retenir pour évaluer et comparer les logiciels de cette catégorie.

- Si un logiciel a pour seule ambition de valider une expérience alors l'évaluation sera sommaire et ne portera que sur la connexion avec le résultat académique obtenu et sur l'éventuelle difficulté de la réalisation. Dans un tel cas, la validation et la reproductibilité des résultats sont importantes. Le logiciel doit donc être disponible (ainsi que les données de l'expérience) et documenté (installation et exécution) au minimum pour qu'une telle reproduction soit possible.

- Si un logiciel est destiné à être utilisé par d'autres chercheurs alors l'évaluation devra être plus précise. Elle devra donner des indications quant à son impact (nombre d'utilisateurs, collaboration rendue possible grâce au développement) et à sa facilité d'utilisation (existence d'une documentation, facilité d'installation, etc.).

- Dans le cadre des plates-formes la difficulté de la chose réalisée devra être évaluée ainsi que la taille du problème traité. L'évaluation devra donner des indications quant à la taille et la durée du développement et elle devra indiquer si la plate-forme ne permet que de traiter un problème particulier avec des solutions très spécialisées ou si au contraire, elle permet de traiter une vaste famille de problèmes.
- D'autres critères d'évaluation peuvent bien entendu être pertinents. Par exemple certains logiciels ont des dépendances fortes et complexes avec d'autres logiciels et qui les rendent difficile à réaliser. Ceci pourra être remarqué par l'évaluation.
- Le rôle pédagogique, et notamment les travaux de thèse intégrés au logiciel, pourront également être pris en compte par l'évaluation.

3.3 Les logiciels « support de transfert »

Il s'agit principalement de logiciels destinés à des partenaires industriels. Leur code n'est généralement pas public. Ils sont souvent associés à des licences d'exploitation idoines.

3.3.1 Critères d'évaluation

- L'utilisation du logiciel doit être attestée par un ou plusieurs industriels (logiciel mentionné dans un site web, lettre d'attestation ou de recommandation des industriels utilisateurs, mentions de verrous technologiques importants que ce logiciel permet de lever, licences significatives payées par des industriels, ...).
- Complexité du problème traité.
- Importance des résultats scientifiques qui sont transférés par l'intermédiaire du logiciel.
- Complexité du développement (indication sur la taille du code, sur le portage, sur le langage d'implémentation, durée du développement, ...).
- Usage des bonnes pratiques (méthodes de développement « contemporaines » et bien adaptées), existence d'un site web, documentation et facilité d'installation (au moins documentée).
- Il convient aussi de préciser les rapports entre la version effectivement utilisée par les partenaires industriels et les développements logiciels réalisés par les partenaires académiques.
- ...

3.4 Les logiciels utilitaires

Les chercheurs sont parfois amenés à développer des logiciels qui ne sont ni liés à leurs recherches ni développés dans le cadre d'activités de transfert. Parfois ils contribuent à des développements de logiciels libres utilitaires, à des développements pour l'enseignement, ou simplement pour satisfaire un besoin de leur environnement professionnel. Ces logiciels peuvent s'inscrire dans la durée et ils peuvent être complexes mais leurs liens plus ténus avec les activités de recherche les rendent néanmoins moins importants dans l'évaluation des candidats chercheurs que les logiciels des autres catégories.

3.4.1 Exemples

Citons quelques exemples, commis par les membres de notre institut, de logiciels appartenant à cette catégorie.

- Flyspell.el, `https://www.gnu.org/software/emacs/` : le correcteur orthographique à la volée d'Emacs.
- Gnubiff, `http://gnubiff.sourceforge.net/` : outil de détection de nouveaux mails ;
- Hevea, `http://hevea.inria.fr/` : un traducteur de LaTeX vers HTML.

3.4.2 Critères d'évaluation

- Taille de la communauté d'utilisateurs.
- Durée du développement et de l'utilisation du logiciel.
- Qualité du service rendu.
- ...

4 Software Self-Assessment, revised version

Cette annexe contient une mise à jour du document « INRIA Evaluation Committee Proposal of Criteria for Software Self-Assessment ».

The EC considers software a major research outcome of the Institute. However, experience has shown that software descriptions in the team descriptions and the candidate application files are very hard to use, because team members and applicants are much less used to self-assessment of software than to self-assessment of theoretical contributions and publications.

The goal of this document is to provide Inria teams and candidates with a self-assessment mechanism for their software developments, to be used by the Inria Evaluation Committee (EC) for evaluation seminars and internal or external recruitments/promotions. It is **a request for evaluation** that the juries and evaluation panels will use to select the appropriate criteria for evaluating software applications developed by teams and candidates. What the EC wants is a fair self-assessment of the real state, evolution, and impact of the software, which in principle would prove exact in a subsequent in-depth evaluation. Examples of some Inria Software Artifacts.

- OCaml: `Family=research; Audience=community; evolution=lts; Duration>=20; contribution=devel,softcont; Url=https://caml.inria.fr/ocaml/`
- Flyspell.el: `Family=utility; Audience=universe; evolution=basic; Duration>=20; contribution=leader; Url=https://www.emacswiki.org/emacs/FlySpell`

The software self-assessment should be followed (Section *Free Description*) by a brief description of the software you have contributed to and the nature of your contributions.

4.1 Software Family (Family)

The software families are described in the EC document "Software Artifact Evaluation", referred to as SAE.

1. `research`: Software as a Vector for Knowledge (see SAE, Section 3.1).
2. `vehicle`: Software as a Vehicle for Research (see SAE, Section 3.2).
3. `transfer`: Transfer software, (see SAE, Section 3.3).
4. `utility`: Utility, (see SAE, Section 3.4).

4.2 Audience (Audience)

1. `personal`: personal or internal team prototype (to experiment an idea);
2. `team`: to be used by people in the team or close to the team (including contractual partners);
3. `partners`: to be used by people inside and outside the team but without a clear and strong dissemination and support action plan;
4. `community`: large audience software, usable by people inside and outside the field with a clear and strong dissemination, validation, and support action plan;
5. `universe`: wide-audience software (aims to be usable by a wide public, to become the reference software in its area, etc.).

4.3 Evolution and Maintenance

- `nofuture`: no real future plans;
- `basic`: basic maintenance to keep the software alive;
- `lts`: long term support.

4.4 Duration of the Development (Duration)

Indicate here the number of years of your contribution to the software development.

4.5 Contribution Characterization

Characterize your contributions to the development. More than one contribution are possible, in particular if the nature of your contribution has evolved over the years.

- `leader`: you have been at the initiative of the development and you have been involved in the coding, debugging, and maintenance.
- `instigator`: you have been at the initiative of the development but you have not been involved in the coding.
- `devel`: you have been deeply involved in the coding.

- `softcont`: you have contributed to the documentation, maintenance, installation scripts, ...

Supervising activities must be reported in dedicated section (Form 1) in the application form.

4.6 Web Page (Url)

Indicate the software artifact URL.

Free Description

In this free section, present in **at most 10 lines**, all the pieces of information you will find useful to let the examiner better understand the software application you have developed and the nature of your contribution. This section can contain facts such as the programming language(s) used, the code size, the complexity and the nature of your contribution, etc. It can also describe the relationships between this software and your other research activities.