# GDR GPL Prospective 2024-2030

Les recherches en science du logiciel couvrent les langages, la compilation, la vérification, le test, le déploiement, le traitement de l'évolution et de la migration (rétro-ingénierie, réingénierie). L'évolution du domaine montre une progression de l'automatisation et de la montée en niveaux d'abstraction, en conceptualisation, qui touche, au-delà des langages de modélisation et de programmation, tout le cycle de vie. Nos recherches intégrent et trouvent des solutions à la complexité toujours croissante des éco-systèmes logiciels, suivent les évolutions des supports d'exécution et le développement de types particuliers de logiciels, comme les systèmes cyber physiques ou les pipelines en science des données et en intelligence artificielle, pour prévenir et résoudre les difficultés qui leur sont propres. Elles prennent en considération les dimensions environnementales et éthiques.

## I. Evolutions significatives

### I.1. De l'IDM au no-code

L'ingénierie dirigée par les modèles (IDM) a permis de mûrir une vision du logiciel intégrant la capitalisation de modèles et leur exploitation par des transformations. L'IDM sert en particulier de terreau aux techniques de génération d'artefacts comme pour le code ou le test. Elle sous-tend aussi le développement de langages spécifiques de domaine (DSLs), eux-même sources des approches *low-code* et *no-code*.

### I.2. Des méthodes de développement aux aides aux développements

L'omniprésence du logiciel a conduit au renforcement et à la dissémination de concepts et de cadres méthodologiques, à l'exemple des lignes de produits logicielles ou du mouvement DevOps. Ces méthodes sont favorisées par la production d'assistants automatiques ou semi-automatiques à tous les niveaux du cycle de vie : depuis la synthèse de programme jusqu'au déploiement automatique en passant par le *refactoring*, la réparation, la génération de tests ou de documentation, la composition, l'aide à la migration, la vérification déductive de programmes ou d'outils de développement, ou encore la maîtrise de la variabilité et des versions. Les résultats s'intensifient sur tous ces aspects.

### I.3. Des logiciels auto-adaptables aux logiciels autonomiques

Tandis que les systèmes auto-adaptatifs répartis et collaboratifs se diffusent dans notre quotidien, de nombreux progrès ont été réalisés en vérification formelle de ces systèmes, dans l'efficacité des prises de décision et de l'application d'une adaptation. Pour ce dernier point, les chercheurs s'intéressent en particulier à réduire la consommation en ressources et en énergie des adaptations. Une perspective pour ces systèmes est qu'ils deviennent "autonomiques", c'est-à-dire autonomes dans toutes leurs dimensions : fonctionnement, optimisation, sécurité, configuration, évolution.

### I.4. Des supports d'exécution de plus en plus complexes

Avec la fin de la loi de Moore, la période a connu l'avènement de plateformes hybrides complexes pour le calcul haute performance. Cela s'est accompagné du développement de systèmes distribués très dynamiques et hétérogènes (cloud, edge, microservices). L'European Processor Initiative (https://en.wikipedia.org/wiki/European_Processor_Initiative) met en

évidence l'importance de proposer une méthodologie et un produit final aboutis souverains à l’échelle européenne avec le développement de langages, de méthodes de compilation et d’exécution efficaces énergétiquement pour les machines parallèles. Le défi réside en particulier dans la prise en compte de types de parallélisme à différents grains, de problématiques systèmes de gestion énergétique, tout en proposant des méthodes de développement efficaces et sûres.

# II. Thématiques émergentes

## II.1. Science du logiciel pour et avec l'IA

Les relations entre ces deux disciplines, en particulier avec l'IA symbolique, ont toujours été très fortes, avec une conjonction des systèmes de représentation *(ex. graphes de connaissances, ontologies et objets)*, de raisonnement *(ex. systèmes de règles et preuves)* ou sur l'opérationnalisation *(ex. systèmes autonomiques et systèmes multi-agents)*.

Une première nouveauté est venue du fort développement de systèmes logiciels intégrant des méthodes d'apprentissage statistique (IA sous-symbolique) avec des objectifs de sûreté *(ex. sélection de tests pour les très grands systèmes ; vérification des systèmes intégrant des modèles statistiques*), de résilience (*ex. maintenance de ces systèmes*), mais aussi de qualité, de sobriété dans l'usage des ressources et de capitalisation des connaissances (*ex. usines logicielles*).

Une seconde nouveauté est que ces techiques d'IA sous-symbolique sont venues compléter celles venant de l'IA symbolique pour résoudre des questions de génie logiciel, telles que la production d'assistants qui demandent de capturer, de modéliser et de reproduire l’expertise humaine, ou encore la compréhension, la maîtrise et l’exploitation des masses de code produit et utilisé sur des durées très longues. Tout ceci pose de nombreux défis de recherche ayant des impacts économiques, énergétiques, ou encore de sûreté.

## II.2. Reproductibilité des systèmes logiciels et préservation du patrimoine logiciel

L'initiative Software heritage (https://www.softwareheritage.org/?lang=fr) permet la capitalisation et le traitement des grandes masses de code existantes. D'une part, la préservation des codes joue un rôle dans la reproductibilité des logiciels. D'autre part, elle offre une opportunité unique d’extraire des connaissances enfouies concernant, par exemple, les bonnes pratiques de stylistique de codage, d'architecture, de migration entre bibliothèques, ou encore la détection et la réparation de bugs.

## II.3. Un logiciel omniprésent mais pas au détriment de notre planète

Le coût environnemental exorbitant du numérique, porté à la fois par la fabrication des équipements, la consommation énergétique et les effets d'accélération induits sur tous les secteurs d'activité, force à réfléchir à de nouvelles approches pour un cycle de vie raisonné du logiciel. Cela peut prendre la forme, par exemple, de la prédiction de la consommation énergétique logicielle dès les phases de conception ou d'architectures et de méthodologies axées sur la sobriété numérique, voire une réflexion profonde sur la dé-numérisation. Il faut noter que si la conscience environnementale ne suffisait pas à envisager ces approches, l’évolution des normes et règles métier *(ex. calcul de l’impact CO2, nouvelles exigences de non-obsolescence)* forcera à aborder l'étude du logiciel sous cet angle.

# III. Enjeux sociétaux

- Les politiques de Responsabilité Sociale des Entreprises (RSE) doivent intégrer la prise en compte des logiciels.
- Les logiciels doivent être développés en respectant des principes de sûreté et de sécurité, d'éthique et de respect des données utilisateurs (privacy).
- Nous pouvons contribuer à réduire l'illectronisme : en nous posant la question des besoins réels des utilisateurs et en évitant la prolifération de logiciels inutiles ou inadaptés ; en diffusant la culture logicielle, d'une part par des opérations de médiation scientifique et d'autre part en donnant les moyens à chacun, si ce n'est de développer son propre système, au moins de l'adapter précisément à son métier par des approches low-code/no-code.
- Le manque de diversité en informatique s'observe également dans la communauté du génie logiciel et nous devrons nous impliquer dans des actions positives de remédiation.
- Nous devons poursuivre notre engagement dans la préservation du patrimoine en soutenant et en disséminant l'initiative Software Heritage.
- La prise en compte de l'accessibilité logicielle à tous doit devenir une problématique de premier plan. Cela exige d'intégrer cette problématique dans ses déclinaisons (ex. handicaps) dans les outils et les processus de développement.

# IV. Relations avec les autres disciplines

## Au Sein des thématiques de l'INS2I

1. Le lien avec le GDR SOC2 se fait par la prise en compte des modifications architecturales des machines, permises par l'essor des jeux d'instructions libres, au plus tôt dans le cycle de développement logiciel/matériel.
2. Les GDR IA et GPL partagent les problématiques de conception d'approches pour les logiciels intégrant des composants d'IA ou pour résoudre les questions propres au génie logiciel.
3. Avec le GDR RSD, notre GDR participe à la réflexion sur les systèmes autonomiques, y compris répartis et critiques.
4. Le lien avec le GDR IM se fait via les problématiques liées à l'étude des systèmes de type, de la logique et des calculs mais concerne également l'intégration des méthodes de test, de vérification déductive et de vérification par *model-checking*.
5. Les GDR GPL et Sécurité intéragissent en particulier sur le thème de l'utilisation des méthodes formelles pour la sécurité des systèmes.
6. Les problématiques environnementales rejoignent celles du GDR labos 1point5.

## Hors de l'informatique

De nombreux résultats obtenus dans des disciplines les plus diverses, allant de la biologie aux sciences humaines et sociales, reposent sur des logiciels. Cela apporte des problématiques dont le statut de validité des résultats obtenus, ou encore la reproductibilité et la diffusion. Dans ce contexte, le logiciel doit être considéré selon les principes FAIR (Findable, Interoperable, Accessible, Reusable) pour la science ouverte. En sens inverse, certaines approches du génie logiciel empruntent des concepts ou des cadres méthodologiques à d'autres disciplines, telles que les sciences de gestion (ex. lignes de produits) l'architecture (ex.

patrons et plans) ou encore la biologie (ex. éco-systèmes). De plus, les sciences humaines et sociales apportent un éclairage sur le rôle de l'humain dans le logiciel, notamment en ce qui concerne la construction et les usages.