**Objectif**

Former des experts et des cadres en informatique dans le domaine du développement logiciel

* Maîtriser les procédures (systématiques) qui permettent d’arriver à ce que des logiciels de grande taille correspondent aux attentes du client, soient fiables, aient un coût d'entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction.
* Toutes les étapes du cycle de vie du logiciel sont concernées.
* Prise en compte des interactions du génie logiciel avec un certain nombre d’autres domaines connexes (IA, robotique, santé …)
* Participation d’intervenants industriels garantit l’adéquation de la formation avec les exigences du milieu industriel

Moyenne de vie d’un logiciel : 20 ans

**Insertion professionnelle**

Cadres en informatique dans le domaine du développement logiciel

| Métiers | Secteurs d'activités |
| --- | --- |
| * Ingénieur en informatique / Dev. logiciel * Chef de projet logiciel * Architecte logiciel, urbaniste * Consultant dans le domaine du logiciel * Expert en sûreté de fonctionnement et méthodes formelles * Chercheur ou enseignant-chercheur en informatique * … | * Edition de logiciels * Service en ingénierie informatique * Services informatiques : industrie, banques, assurances, … * Recherche en développement en informatique * Enseignement supérieur * … |

Au bout de 6 mois : 84% en emploi, 13% en thèse, 3% inactivité

Presque 100% d’activité avec un master GL, 90% grâce à un stage. Plus d'offres que de demande, progression de carrière rapide.

Ce master donne les bases pour tout faire

**Définitions**

* Le génie logiciel, l'ingénierie logicielle ou l'ingénierie du logiciel ( en anglais : software engineering) est une science du génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels.
* Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d’arriver à ce que les logiciels de grande taille correspondent aux attentes du client, soient fiables, aient un coût d’entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction.

Développer de manière professionnelles (en suivant des procédures). Il existe des logiciels avec 8000 classes. Il faut que le logiciel soit sûr et sécurisé.

* L’ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi.
* Procédures, méthodes, langages, ateliers, imposés ou préconisés par les normes adaptées à l’environnement d’utilisation afin de favoriser la production et la maintenance de composantes logiciels de qualité
* Génie logiciel concerne l'ingénierie appliquée au logiciel informatique

Pour construire un logiciel, il y a un schéma à suivre avec des méthodes, des normes.

* Branche de l’informatique s’interesse plus particulierement à la manière dont le code source d’u logiciel est spécifié puis produit
* Le génie logiciel touche au cycle de vie des logiciels : (toutes les phases de la création y sont enseignés )
  + L’analyse du besoin
  + L’élaboration des spécifications
  + La conceptualisation du mécanisme interne au logiciel ainsi que des techniques de programmation
  + le développement
  + la phase de test
  + maintenance

**Une science**

Sciences (du logiciel) :

* Comme de connaissances qu’un individu possède ou peut acquérir par l'étude, la réflexion ou l’expérience (ici, dans le domaine du logiciel / développement du logiciel)
* Elle est ce que l’on sait pour l’avoir appris, ce que l’on tient pour vrai au sens large, l’ensemble de connaissances, d’études d’une valeur universelle, caractérisées par un objet (domaine) (Ici le logiciel) et une méthode déterminés, et fondés sur des relations objectives vérifiables [sens restreint]

Sciences : une expériences que l'on peut reproduire

**C’est du génie**

Selon l’American Institute of Industrial Engineers, le génie industriel englobe la conception, l’amélioration et l’installation de systèmes intégrés

Il utilise les connaissances provenant des sciences mathématiques, physiques et sociales, ainsi que les principes et méthodes propres au “génie”, ou à l’art de l’ingénieur, dans le but de spécifier, prédire et évaluer les résultats découlant de ces systèmes

Aspect de mécanisation

**Normes internationales en génie logiciel**

Le génie logiciel repose sur un ensemble de normes de niveau international :

* IEEE Standard for software Quality Assurance Processes - IEEE std. 730
* IEEE Standard for Configuration Management in Systems and Softwars Engineering - IEEE Std.828
* IEEE Standard for Software Test Documentation - IEEE Std. 829
* IEEE Recommended Practice for Software Requirements Specifications - IEEE Std.830
* IEEE Recommended Practice for Software Design Descriptions - IEEE Std.1016
* EEE Standard for Software User Documentation - IEEE Std. 1063
* Normes et guides d’ingénierie du logiciel pour les très petits organismes - ISO/CEI 29110
* Software life cycle processes - ISO/CEI/IEEE 12207
* ISO 15504 fourni un ensemble structuré de bonnes pratiques destinées à appréhender, mesurer et améliorer la qualité des produits d’une entreprise d’ingénierie informatique

Les normes permettent de cadrer le développement d’un logiciel : la conception, documentation, test, qualité…

**Domaines de connaissance**

SWEBOK : Guide de corpus de connaissances en génie logiciel (“Software Engineering Body of Knowledge”) - Edité par l’IEEE et également disponible en tant que rapport technique ISO/IEC TR 19759:2015

SWEBOK : Software Engineering Body of Knowledge est un référentiel établi par l’IEEE Computer Society pour structurer les connaissances bien identifiés, et à faire une synthèse des pratiques et techniques généralement acceptées, tout en visant à asseoir la reconnaissance du génie logiciel comme discipline de l’ingénierie. La version électronique est mise à disposition gratuitement par l’IEEE

Le premier programme est une carte perforée (date de plus de 70 ans). Le 1er ordi date des années 40

Le SWEBOK structure le génie logiciel en 15 domaines de connaissances :

* Exigences du logiciel (“Software requirements”)
* Conception du logiciel (“Software Design”)
* Construction du logiciel (“Software Construction”)
* Test du logiciel (“Software testing”)
* Maintenance du logiciel (“Software maintenance”)
* Gestion de la configuration du logiciel (“Software Engineering Models and Methods”)
* Management du génie logiciel (“Software Engineering Management”)
* Processus du génie logiciel (“Software Engineering Process”)
* Modèles et méthodes du génie logiciel (“Software Engineering Models and methods”)
* Qualité du logiciel (“Software quality”)
* Pratique professionnelles du génie logiciel (“Software Engineering Professional Practice”)
* Principes économiques du génie logiciel (“Software Engineering Economics”)
* Bases informatiques (“Computer Foundations”)
* Bases Mathématiques (“Mathematical Foundations”)
* Bases d’ingénierie (“Engineering Foundations”)

**Complexité des logiciels et Ingénierie logicielle**

1950 à 1960 : Les logiciels étaient développées par des membres des institutions pour leurs propres besoins, la distribution de logiciels était très limitée, et ceux-ci servaient essentiellement à effectuer des traitements par lots (anglais batch)

Jusqu’en 1985 : Les ordinateurs appartenaient à des sociétés ou des institutions

en 1970 : Nouvelles notions, telles que le multi-utilisateur, les interfaces graphiques, la programmation concurrente, les bases de données et le temps-réel. C’est à la même époque que sont apparus les premiers éditeurs de logiciels et que le logiciel est devenu un bien du marché.

Depuis 1973 : et a fortiori depuis l’arrivée des ordinateurs personnels en 1980 : Le logiciel devient un bien de grande distribution, orienté vers le consommateur, par l’arrivée des progiciels - des logiciels prêt-à-porter. Le prix du matériel informatique à également beaucoup diminué, ce qui augmente la proportion du coût du logiciel sur le coût total de l’ordinateur.

Entre 1985 et le début des années 2000 :

* Le logiciel passe du statut de produit stand-alone indépendant à celui d’élément d’un ensemble, dans lequel plusieurs ordinateurs et plusieurs logiciels travaillent en collectivité.
* Avènement des systèmes distribués, de l’Internet, de l’architecture client-serveur et du cloud computing.
* L’arrivée de la programmation orienté objet et la conception orientée objet transforment le travail des ingénieurs (1er langage orienté objet : Small Talk)
* Les logiciels incluent des formes d’IA telle que la reconnaissance de forme, la déduction automatique, la traduction automatique et l’exploration de données.

**Cycle de vie du logiciel**

Le développement de logiciels consiste à étudier, concevoir, construire, transformer, mettre au point, maintenir et améliorer des logiciels.

Le “cycle de vie d’un logiciel” (en alglais software lifecycle), désigne toutes les étapes du développement d’un logiciel de sa conception à sa disparition.

On est recruté en tant que développeur. Avec l’expérience, on devient architecte logiciel, capable de construire l’architecture du logiciel.

18

Analyse des besoins et faisabilité, c’est à dire l’expression, le recueil et la spécification des besoins du demandeur (le client) et de l’ensemble des contraintes.

Conception générale, il s’agit de l’élaboration des spécifications de l’architecture générale du logiciel.

Conception détaillé, consistant à définir précisément chaque sous-ensemble du logiciel

Codage (Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités conformément aux spécifications.

Tests unitaires, permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

Intégration, dont l’objectif est de s’assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l’objet des tests d’intégration consignés dans un document.

Qualification (ou recette), c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.

Déploiement et Mise en production

Maintenance, comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

conserve l’ordre dans lequel sont effectués les différents travaux de développement d’un logiciel, en cascade (détection des problèmes trop tard), en V (on fait des tests à chaque étape), itératif, en sprints ou parallèlement. Agile : boucle, analyse des petits morceaux. C’est le plus utilisé car on voit les erreurs tôt.

**Les schémas de réutilisation et productivité**

Concepts et schémas pour la réutilisation du logiciel :

* Le polymorphisme d'inclusion et paramétrique
* L’héritage, spécialisation de méthodes
* L’envoi de message et liaison dynamique en présence de typage dynamique ou statique
* Le paramétrage par fonctions d’ordre supérieur, par spécialisation et par composition
* Framework
* L’affectation polymorphique
* L’injection de dépendance et inversion de contrôle

Application concrètes des concepts précédents pour la réalisation d’architectures extensibles et adaptables (éventuellement à l'exécution), de type :

* Bibliothèques de classes
* Frameworks
* Ligne de produits logiciels

On ne crée par un logiciel de rien. On réutilise (patrons).

**Comment construire des logiciels Modulaires : les Architectures logicielles**

Concepts avancés liés au découpage modulaire et à la réutilisation des programmes et architectures logicielles

Différents types d’entités modulaires et extensibles :

* Les classes et leurs méthodes
* Les composants logiciels
* Les aspects

Schémas d’utilisation de ces entités permettant de bâtir des architectures réutilisables (frameworks, ligne de produits)

Etude de frameworks :

* Framework JEE pour le développement d’applications à modules distribués, l’injection de dépendances
* la modularité au niveau des classes avec le framework Spring
* la modularité auniveau des clusters de classes / packages avec le framework OSGi, le système de plugins Eclipse, et le système de modules de Java 9+
* Architectures à services

**Comment définir des architectures avancées du web**

Présentation des principaux frameworks permettant la création industrielle d’applications web

Les plus importantes avancées logicielles contribuant à ces frameworks :

* La réutilisabilité du code via différents niveaux de modularité (composants web, modules,...)
* L’extension du langage HTML
* L’utilisation de la programmation réactive (abonnement à des flux de données via des observables)
* La mise en oeuvre de services web permettant la création d’applications monopages
* L’utilisation d’ORM (Mapping Object-Relationnel)
* La spécification intégrée de tests

**Comment définir des architectures logicielles Distribuées**

Les systèmes logiciels deviennent de plus en plus complexes et intègrent souvent plusieurs parties communicatives

Les architectures n-tiers (distribuées) ont pour but de gérer les problèmes spécifiques à ce type de systèmes tels que :

* La communication et échange de données entre les différentes composantes du système,
* La persistance et le partage des données
* etc

Différentes technologies OO sous-jacentes, en Java et en .NET :

* Objets distribués et intergiciels à objets (RMI)
* Service web SOAP
* Service web REST
* Micro-service et architecture du cloud
* etc…

**Programmation répartie / concurrente**

Technique de base de programmation concurrente (sockets, multi-threading et/ou multi-processus), RPC (Remote Procedure Call), et MPI (Message Passing Interface).

Ensemble de principes sur lesqules s’appuient la conception d’applications distribuées destinées à être exécutées sur tout système distribué :

* Grappes de calcul (Clusters)
* Grilles
* Cloud
* Systèmes embarqués communiquant
* IoT, réseaux sans fil, réseaux de capteurs
* etc…

Algorithmes fondamentaux en algorithmes distribués, en mettant en avant les caractéristiques algorithmiques de ce nouveau paradigme par rapport au monde client-serveur (Exemple : Étude des problèmes de l’élection et de la terminaison d’un calcul)

**Développement et programmation pour supports mobiles**

Comprendre et de maîtriser le développement logiciels sur des plateformes mobiles (téléphone, tablette, montre, etc..) de manière générale

Développement sur Android en particulier

Aspect à savoir :

* Les propriétés et catégories des plateformes mobiles
* Principes et architectures des OS mobiles
* Les principes généraux des langages et framework de programmation pour mobile
* L’architecture et les principaux composants des applications mobiles
* Les modèles de structuration des interfaces utilisateurs sur mobiles
* La gestion des événements
* La gestion des ressources sur mobiles
* La gestion des processus et du multi-tâches
* La gestion de la connectivité
* La gestion des capteurs
* La gestion de la persistance des données mobile
* Ect..

**Développement mobile avancé, IoT et embarqué**

Deux parties

Programmation mobile multi plateformes : Il s’agit de passer d’un développement pour un seul OS à un développement où un même code est adapté/transformé pour être déployé/exécuté sur plusieurs OS :

* Flutter
* Xamarin
* Cordova
* Etc…

Développement de logiciels pour l’IoT (internet des objets)

* Il s’agit essentiellement d’aborder les problèmes liés au développement logiciel où l’interaction avec le matériel est prédominante

**Ingénierie des modèles**

L’ingénierie dirigée par les modèles (IDM) est une approche de développement logiciel qui consiste à mettre les modèles et leurs méta modèles au centre du processus.

Les enjeux liés à l'ingénierie des modèles (capitalisation du métier, indépendance vis-à-vis des technologies, manipulations de modèles, méta modélisation) : Permet de mettre en pratique certains aspects :

* Etude du méta modèle d’UML et méta modélisation
* Utilisation d’OCL pour la conception de méta modèles et de transformations de modèles
* Transformation de modèles
* DSL et DSML (création de syntaxe textuelle et de syntaxe graphique).

**Systèmes réflexifs, models@runtime**

Les systèmes informatiques réflexifs sont auto-adaptables, ils peuvent assurer leur propre re-conception et re-programmation sans interrompre leur fonctionnement.

Méta-programmation, qui partage avec l'ingénierie dirigée par le modèles le concept de méta-modélisation, mais y ajoute l'idée de réflexivité et d'exécution des méta-modèles (ou models@runtime).

Les langages réflexifs les plus élaborés et puissants : Smalltalk(Pharo), CLOS, et les environnements de méta-modélisation comme MOF-EMF (Eclipe Modeling Framework)

**Comprendre le Processus de transformation d'un langage de haut-**

**niveau en un langage de bas niveau : La compilation**

Processus de transformation d'un langage de haut niveau en un langage de bas niveau, qui permet de rendre un programme informatique exécutable sur une machine.

Deux types de compilation :

* La compilation native, qui transforme le code source en instructions machines pour une architecture cible donnée ;
* La compilation bytecode, qui transforme le code source en code intermédiaire exécutable par une machine virtuelle.

Technique de compilation à la volée, aussi connue sous le nom de compilation JIT (Just-In-Time), qui vise à améliorer la performance des systèmes bytecode- compilés par la traduction de bytecode en code machine natif au moment de l'exécution.

**Conduire et gérer un projet Informatique**

Les enjeux et techniques de gestion et conduite de projet.

• CMMI est donnée.

• Les méthodes agiles (Scrum)

**Assurer la sécurité des logiciels**

Développement de systèmes critiques, c'est-à-dire des systèmes dont la panne peut avoir des conséquences dramatiques (morts, dégâts matériels importants, conséquences graves pour l’environnement).

Les domaines d’applications critiques sont nombreux. On peut citer en particulier les transports, la production d’énergie, la santé, le système financier ou le domaine militaire.

Pour développer de tels systèmes, l'idée est de produire des logiciels sûrs (sans bugs, zéro défaut) en utilisant des méthodes rigoureuses de développement, à savoir les méthodes formelles.

Une des méthodes de développement sont les méthodes déductives, qui permettent de démontrer (avec une preuve) qu'un programme est conforme à sa spécification, c'est-à-dire qu'il fait bien ce qu'on attend de lui.

Outil d'aide à la preuve tel que Coq.

**Évolution et restructuration des logiciels**

Principes fondamentaux et les bonnes pratiques de la maintenance et de l’évolution des logiciels :

* Maintenance et migration des systèmes patrimoniaux.
* La rétro-ingénierie et la compréhension des programmes.
* La réingénierie des logiciels.
* Les bad smells dans le code, son refactoring et sa restructuration.
* Etc.

Techniques de maintenance/Evolution :

* Analyse statique.
* Analyse dynamique.
* Techniques issues du domaine de l'intelligence artificielle symbolique ou sous-symbolique.
* Etc.

**Gestion des données au delà de SQL (NoSQL)**

Le NOSQL (Not Only SQL) regroupe un ensemble de systèmes de gestion de données spécialisées qui sont des alternatives aux SGBD relationnels. Ils sont notamment utilisés par les plateformes du BigData.

Les grands axes sont :

* Différences entre NOSQL et NewSQL, le Théorème CAP.
* Les systèmes orientés clé-valeur et document (techniques de stockage et indexation, langages, évaluation de requêtes).
* Les systèmes orientés colonnes (techniques de stockage et indexation, langages, évaluation de requêtes).
* Les systèmes orientés graphe (techniques de stockage et indexation, langages, évaluation de requêtes).
* Les principaux outils utilisés sur le marché (Mongodb, CouchDB, Cassandra, Hbase, Neo4j, RDF

**Conférences Génie logiciel**

Apporter des connaissances supplémentaires en génie logiciel qui ne sont pas abordées

dans les autres UE du parcours GL. Voir les interactions actuelles du génie logiciel avec un certain nombre d'autres domaines connexes (intelligence artificielle, robotique, santé, environnement, etc.). Ces conférences seront données majoritairement par des intervenants

extérieurs, mais également par des membres de l'équipe pédagogique du parcours GL.

En 2021 :

• 14 septembre : Jules coulon, Sopra Steria

• 21 septembre : Simon Robillard, Lirmm, UM

• 28 septembre : Julien Morgan, Berger Levrault

• 05 octobre : Olivier LeGoaer, Université de Pau

• 12 octobre : Bachar RIMA, Ing. R&D-LIRMM

• 19 octobre : Gilles Fabre, CapGemini

• 26 octobre : Contrôle

• 9 novembre : Robin Passama, Dept. Robotique Lirmm

• 16 novembre : Tewfik Ziadi, Mobioos/Univ Paris Sorbonne

• 23 novembre : Stéphane Ducasse/Nicolas Anquetil, INRIA

• 30 Novembre : Naoel Moha ou Hafedh Mili, UQAM

• 07 décembre : Contrôle

**IA pour le génie logiciel**

Principes de l’Intelligence Artificielle (AI) dont des applications en génie logiciel sont établies.

* Notion d'apprentissage en s'appuyant sur la Fouille de Données (DM).
* Raisonnement avec la Programmation Par Contraintes (CP).
* Vérification et validation des logiciels basées sur des techniques AI.
* Etc.

**Aide à la décision**

La prise de décision repose souvent sur des informations imparfaites. L'imperfection peut être due à la présence d'incohérences (informations provenant de sources multiples) et/ou d'incertitude/priorité associée aux informations.

Le problème de décision se décline en trois parties :

* Représentation des informations imparfaites.
* Raisonnement.

Plan succinct :

* Représentation : logiques pondérées, logiques conditionnelles, graphes ;
* Raisonnement : gestion des incohérences, fusion d'information ;
* Décision : décision inhérente à la gestion des incohérences et la fusion d'information, vote, algorithmes d'affectation, décision multicritères ;
* Implémentation des problèmes de décision (méthodes et outils) : SMA, jeux.

Domaines scientifiques connexes : Big-data, robotique, fouille de données, web sémantique, bases de données. Applications potentielles : commerce électronique, systèmes de recommandation, domaine juridique, sécurité informatique, développement durable, santé.

**Machine learning**

Classification supervisée.

Méthodes de classification non supervisées : K-means, Hiearchical Clustering, Density-based clustering, pattern mining, ensemble learning, etc.

Méthodes de classification semi-supervisées : Active learning,... Evaluation de modèles : validation croisées, précision, rappel, F-mesure, courbes de Roc.

Techniques les plus récentes en apprentissage automatique et notamment sur les approches de type deep learning.

* Un focus particulier sera fait sur la classification supervisée d’images.
* Deep learning : descente de gradient, réseaux de neurones profonds, optimisation, CNN, AutoEncoder, RNN, LSTM.
* Complément sur la classification non supervisée : motifs complexes (trajectoires, séquences, graphes).
* Apprentissage de représentation : plongement (embeddings) de mots et de graphes.
* Méthodes avancées de traitement statistique de langage naturel : topic models, LDA.
* Méthodes hybrides basées sur les graphes de connaissance.
* Extraction de connaissance à partir du contenu web : scrapping, enrichissement, web sémantique.

**Entrepôts de données et Big-Data**

Optimisation de requêtes (choix du meilleur plan d'exécution, estimation de sélectivité).

Entrepôts de données : principes et meilleures pratiques de la modélisation multidimensionnelle.

Optimisation pour les entrepôts de données (index, vues matérialisées).

Procédures d’intégration des données (ETL).

Plateformes de Big-Data.

Principes d’un système de fichiers distribués : réplication, distribution, tolérance aux pannes, la négociation des ressources.

Principes de la programmation dans map / reduce, les problèmes parallélisables (et non) en lien avec leur complexité, l’équilibrage des charges (nombre de mappers et reducers).

Outils et solutions utilisés dans le marché.

**Fondements de l'IA symbolique**

Fondements algorithmiques et logiques de l'IA symbolique.

Trois grandes thématiques :

* Espace d'états, modélisation de problèmes sous cette forme, algorithmes classiques d'exploration (non-informés versus informés, profondeur versus largeur, algorithme A\* et variantes), problème de planification.
* Problème de satisfaction de contraintes (CSP) :
  + modélisation d'un problème sous forme de CSP
  + algorithme de backtrack et raffinements (arc-consistance, forward checking)
  + liens avec le problème SAT.
* Approche logique de la représentation de connaissances et des raisonnements :
  + notions de base de connaissances et de raisonnement
  + systèmes à base de règles et mécanismes de chaînage avant et chaînage arrière
  + autres méthodes de raisonnement
  + hypothèses fondamentales du monde ouvert et du monde clos.

**POA/SMA**

Notions de base autour des SMA et de permettre la programmation de tels systèmes.

Présente et discute les différences pratiques entre la POO et le paradigme de programmation multi agent (POA).

Extensions du modèle de base pour la POA :

* Les agents qui font du raisonnement pratique (notions de raisonnement pour atteindre des notions de "means-ends reasoning" et de délibération). Pour illustrer cette notion les étudiants programmeront en JASON des agents illustratifs (tels que la domotique, les tuteurs virtuels etc).
* Cette notion de POA basique sera étendue à des notions d’interaction entre agents (le dilemme du prisonnier et des notions d’équilibre Nash) ainsi que des langages de communication dans le SMA (FIPA).
* Ces deux dernières notions seront mises en œuvre par de la programmation NetLOGO afin de simuler des larges SMA. Les étudiants imagineront une extension du projet de la première partie dans laquelle tous les agents développés dans la classe devront interagir ensemble et définir des scénarios de simulation.

**Programmation efficace d'algorithmes**

Définition :

* La programmation efficace d'algorithmes s'intéresse à l'adéquation entre un algorithme et son implantation sur une architecture de calcul particulière.
* De manière générale, l'utilisation fine des caractéristiques techniques de l'architecture cible pour adapter l'algorithme et ses structures de données est un enjeu majeur pour atteindre les meilleures performances possibles.

Objectif :

* Connaissances de base leur permettant de commencer à optimiser des codes dans les environnements de calcul actuels (processeur superscalaire, multi-coeur, GPU).
* Techniques classiques d'optimisation en calcul haute performance qui s'appuient sur : l'optimisation des accès mémoires, la vectorisation des instructions, la parallélisme multithread avec OpenMP et l'utilisation d'accélérateur de calcul (carte graphique GPU).
* Questions algorithmiques posées par l’utilisation des technologies sous-jacentes et de leurs caractéristiques.

**Recherche opérationnelle**

Principes de base en recherche opérationnelle.

Les aspects géométriques seront introduits et plusieurs algorithmes de programmation linéaire seront développés (simplexe, algorithme dual, ...) avec les complexités associées (NP et CO-NP).

Mise en avant du Lemme de Farkas et les aspects modélisation.

Présentation de la notion de dualité ainsi que les implications au niveau des méthodes de résolutions.

Présentation des méthodes classiques de résolution branch and bound, branch and cut (la programmation linéaire en nombres entiers).

Prérequis nécessaires : Connaissance en mathématiques de licence et Classes de complexité.

**Traitement sémantique des données**

Concepts et techniques basés sur des connaissances pour mieux tirer parti des données :

* Langages de représentation de connaissances pour l'accès aux données : règles en logique du premier ordre, logiques de description, langages du web sémantique (RDFS, OWL) ;
* Techniques de raisonnement associées ;
* Application à l'interrogation de données médiatisées par une ontologie et l'intégration de données multi-sources.

**Programmeur (ou développeur) informatique**

Maillon essentiel dans la création d'un logiciel, le programmeur (ou développeur) informatique est la personne qui va concevoir la logique applicative.

A l'aide d'outils de programmations (les langages), il va définir les actions réalisables par l'utilisateur du logiciel et créer les processus qui mènent au résultat.

Les programmeurs travaillent généralement en équipe sur des gros projets mais peuvent aussi travailler seul sur des logiciels de plus petites envergure.

De part la variété des travaux qu'ils ont à faire, ils doivent aussi maîtriser différents logiciels et langages de programmation. Dans la carrière d'un informaticien, programmeur est parfois une étape transitoire qui mène à un métier avec plus de responsabilités comme responsable des tests ou chef de projet.

**Ingénieur Génie Logiciel**

Poste de haut niveau Technique

L'ingénieur Génie logiciel étudie et met en place les méthodes d'aide à la réalisation de logiciels et les outils associés.

Il est ainsi l'un des éléments de base pour un travail efficace des ingénieurs développeurs et pour tous ceux qui sont associés à ces équipes.

Son travail comprend le développement, le paramétrage en maîtrisant les outils de développement les plus couramment employés sur le marché

Enfin, un bon profil d'Ingénieur Génie logiciel comprend les compétences d'administration de systèmes UNIX et ou NT, celles de bases de données relationnelles (Oracle ou SQL serveur ODBC, OLEDB..)

**Intégrateur web ou développeur web**

Le développeur web ou intégrateur web a pour principale mission de créer un site internet (ou d'assurer une refonte d'un site existant) à partir d'un cahier des charges retraçant les différents besoins du client. Un développeur full-stack sera apte à travailler à la fois sur la partie visible d'un site (le rôle du développeur front-end) et sa partie cachée (la mission du développeur back-end).

**Responsable de tests et d'intégration / qualité**

Un bon développeur doit assurer le bon fonctionnement du logiciel qu'il a produit. Pourtant, la personne qui a fabriqué le logiciel n'est généralement pas la meilleure personne pour effectuer cette tâche. C'est la raison pour laquelle il arrive que des équipes soient spécialement créées pour assurer l'étape cruciale des tests logiciels. Un responsable de tests et d'intégration (il s'agit de l'étape qui permet de s'assurer que le nouveau logiciel - ou fonctionnalité - s'intègre bien avec les autres) va donc définir les tests à réaliser et s'assurer de la qualité du produit fini. Généralement, ce responsable connaît très bien les technologies employées pour créer le logiciel ou la nouvelle fonctionnalité. Il connaît aussi très bien l'environnement existant (tout ce que doit faire le logiciel et avec quels autres logiciels il communique). C'est donc une personne d'expérience qui pourra être amenée à travailler en collaboration avec les programmeur et qui pourra aussi déléguer certaines tâches à ses collaborateurs.

**Chef de projet**

Le chef de projet est en quelque sorte le chef d'orchestre d'un projet de développement informatique (comme la création d'un logiciel ou d'un site Internet). C'est lui qui gère les ressources nécessaires au projet, qu'elles soient humaines ou financières (car il y a souvent un budget à respecter ou un nombre de jours de travail à ne pas dépasser). Pour exercer pleinement son métier, le chef de projet informatique doit à la fois connaître les technologies employées par son équipe (comme le langage de programmation ou le type de base de données) mais aussi avoir des compétences en planification et en management. Comme son but est de faire travailler tout le monde ensemble, le chef de projet doit savoir comprendre et analyser les problèmes rencontrés par chaque personne travaillant sur son projet.