Chapter 1
Introduction

1.1 Problem statement

The advent of various biomolecular technologies has boosted biological and medical research. Figure 1 illustrates the heterogeneous types and large amounts of biomolecular data and the complex protocols needed to produce and analyze these new data (see section 1.3 for a glossary of terms):

Multiple plants (Fig 1a) or mice (Fig 1b) are crossed with smart breeding strategies to produce hundreds of genetically different offspring. Each individual of these offspring is fingerprinted with molecular marker technologies (AFLP in Fig 1a, SNPs in Fig 1b) to generate 10,000-100,000 pieces of information about their genetic make-up (genotypes). Each individual is also profiled using gene expression technologies (Qiagen-Operon microarrays in Fig 1a) or mass spectrometry technologies (LC-MS in Fig 1b) to get 100,000 pieces of information about which of the 20,000-30,000 genes are ‘switched on’ (gene expression), or which genes result in protein and metabolite molecules (visible as mass peaks). All technical steps in the complex data production process have to be well documented, e.g. and what batch of DNA microarray chips was used, what temperature for hybridization, in order to be able to track and trace data and, if needed, re-do and re-interpret certain analyses. The (pre-)processing of gene expression (Fig 1a) or mass spectrometry (Fig 1b) data will require several days at a computing cluster of 100s state-of-the-art PCs, and will generate output that exceeds input in size and complexity. Although different in origin, both types of data may end up in a similar type of analysis to uncover the genetic determinants of plant/animal features and to reconstruct biological networks (QTL, correlation and network analysis in Fig 1).

Storing, managing, processing, cross-linking and interpreting these data won’t work anymore using only a paper lab journal, some rewritable CD’s, a stack of Excel files and a copy-and-paste into analysis software, and there is little debate about the need for software infrastructures to archive, analyze and integration of all these data. Timely production of such infrastructures has become a major bottleneck.
Figure 1a | A high-throughput molecular biology protocol. 10+ lines of the *A. thaliana* plant are inbred and genotypically profiled using biomolecular markers. The expression of thousands of genes is measured for 100+ individuals of these inbred lines, using two-color microarrays that may use 10,000+ of probes (that hybridize with the gene products). The expression data is preprocessed on the computer to reduce systematic errors. Finally, the QTL likelihood along the genome is calculated and correlated to reconstruct regulatory network.
Figure 1b | Another variant of a molecular biology protocol. The process shown has much in common with the process described in Figure 1a. However, important differences have to be accounted for, e.g. difference in species (strains of mouse instead of lines of A. thaliana), genotyping device (SNP arrays instead of biochemical markers), and omic level measured (metabolite abundance instead of gene expression). This requires another variant of software infrastructure in order to accommodate those differences.
Figure 1 also illustrates why development of such software infrastructures is non-trivial and time-intensive:

Software developers have to make a large effort to master the complex biological terminology and biotechnological processes before they can start helping the biologists. Once they understand the terminology, they can start to tailor and assemble many software components for storing, finding, updating, calculating and user interfacing each data type. They have to face the large scale of data: standard components may not work and all steps need to be automated. For example, copy-and-pasting thousands of ‘probe annotations’ one-by-one into the tool used for network analysis is not an option. A new or improved biomolecular method (SNP instead of AFLP in Fig 1) has different input/output signatures that require changes in the software infrastructure parts that support other steps. New and invaluable software components may be developed elsewhere outside the control of the infrastructure owners (e.g. a revolutionary tool for QTL calculation or network reconstruction). These heterogeneous software components are often implemented on a different technical platform (Oracle, Windows, Linux, using Java, Perl, C++, etc) making it hard to make them fit in the infrastructure software. Although experiments may follow roughly a common protocol – in Fig 1: generate offspring, fingerprint and profile offspring, pre-process data, map QTLs and reconstruct networks – there are also important differences requiring many variants of software infrastructure. Moreover, it is unpredictable today what new (combinations of) molecular and statistical technology will be around tomorrow. Still, researchers want to have the new or modified infrastructure variant with a short time-to-market to keep ahead in the competitive world of science.

This thesis aims to introduce and validate innovative informatics solutions to address these challenges. The result, I hope, is that bioinformaticists will have a new strategy to timely produce, maintain and evolve software infrastructures that suit biologists’ needs.

1.2 Thesis road map

Figure 2 provides a road map to the four parts of this thesis. The current chapter provides an introduction into the research problem (previous section), a short outline of each chapter (this section) and a glossary of terms (next section). Chapters 2-3 describe methods to tackle the research problem and review related work. Chapters 4-7 describe detailed cases covering different (parts of) software infrastructures for (systems) biology. Chapter 8
discusses the contributions of this thesis and outlines future work. Below a short outline of each chapter is given.

**Methods**

**Chapters 2-3** describe methods to tackle the research problem and review related work. We conceptualize and implement an informatic strategy to alleviate the problems described above and describe how to implement this strategy in practice.

**Problem analysis and approach** | In Chapter 2 we further analyze the problems described above. Biologists need software infrastructures that easily connect to work that is done in other laboratories, for which (numerous) standardization initiatives have been helpful. However, the infrastructure must also accommodate the specifics of their biological system, but appropriate mechanisms to support variation have been lacking.

<table>
<thead>
<tr>
<th>Chapter 1: Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem statement; Thesis road map; Glossary of terms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter 2</strong></td>
</tr>
<tr>
<td>Problem analysis and approach</td>
</tr>
<tr>
<td>Beyond standardization: dynamic software infrastructures for systems biology.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter 4</strong></td>
</tr>
<tr>
<td>Infrastructure for the wet-lab</td>
</tr>
<tr>
<td>Molecular Genetics Information System (MOLGENIS): alternatives in developing local experimental genomics databases.</td>
</tr>
</tbody>
</table>

| **Chapter 5** |
| Infrastructure for dry-lab |
| MGG: A customizable software infrastructure for genetical genomics |

| **Chapter 6** |
| Infrastructure for clinical trials |
| Towards a uniform treatment of clinical data. |

| **Chapter 7** |
| Reusable assets for processing |

| **Chapter 8: Discussion and Future work** |
| Evolution of a biological software family, dealing with commonalities and variation, how to bring many more benefits to biology. |

Figure 2 | Road map of this thesis.
In this thesis I argue that a strategy that goes ‘beyond standardization’ using a minimal computer language (to describe what biologists need), and a software tool called a generator (to automate software implementation), can be used to quickly produce customized software infrastructures that ‘systems biologists really want to have’. We review six recent example initiatives, one of which is presented in this thesis, that give a glance into a future with many more benefits.

Generative development in action | In Chapter 3 we report on the use, working and implementation of a generator to quickly produce ‘Molecular Genetics Information Systems’. The MOLGENIS generator is an example of a ‘domain specific’ tool to generate ready to use software infrastructures for biology, including user interfaces for biologists and programmatic interface for bioinformaticians. We show a concrete ‘domain specific language’ to describe how for example a microarray experiment is to be organized in terms of data and their relationships, and how simple “text file” generators automagically generate all parts of database, application logic and user interfaces. Only parts specific to research supported by a software variant need to be generated, on top of reusable assets that have mechanisms to easy variation. We found a 48 fold reduction of effort in comparison with hand-writing software.

Cases

Chapters 4-7 describe case studies covering various (parts) of working software infrastructures for systems biology. For each case we describe what functionality is needed by biologists/bioinformaticists and how we implemented a suitable software infrastructure to accommodate these needs.

Infrastructure for the wet lab | In Chapter 4 we report on the case of a molecular genetics group establishing a microarray laboratory. They required infrastructure to support the work in their ‘wet’ lab, i.e. preparation of measurement materials, treatment of biomaterials, and production of raw experimental data. We evaluated twenty existing microarray databases (Table 4.1) and then decided to build a new system. Five typical requirements were identified and alternative solutions evaluated for: (i) evolution ability to keep up with the fast developing genomics field; (ii) a data model to deal with local diversity; (iii) storage of data files in the system; (iv) easy exchange with other software; and (v) low maintenance costs. This resulted in the first generated MOLGENIS-variant and since then supported over 90 projects.

Infrastructure for the dry lab | In Chapter 5 we report on the case of a bioinformatics group analyzing many genetical genomics experiments for their collaborators. They required infrastructure to support the work in their ‘dry’ lab, i.e. import of heterogeneous data from their collaborators, management of in-process data, and sharing of processed
results. A central infrastructure would ease sharing of resources (data, processing algorithms, visualization tools) within and across organisms and collaborators. The diversity of the data made this challenging: transcript, protein, metabolite, and genotype data were provided from hundreds of genetically different individuals from human, mouse, *A. thaliana*, and *C. elegans* (remember Figure 1). The MOLGENIS generator tool was evolved to provide variation mechanisms for data (such as ‘inheritance’) and processing centered interfaces (such as data access from the R statistical language). This resulted in a second MOLGENIS variant ‘for genetical genomics’ (MGG) that can be customized by individual research groups to suit their needs.

**Infrastructure for clinical trials** | In Chapter 6 we report on the case of a clinical trial coordination center involved in the analysis of many clinical studies. They required infrastructure to preserve and (re)distribute databases that were produced in different clinical studies and that allow biomedical researchers to extract (cross-study) data. We explain our design decisions, describe a generic and flexible data model, and give mechanisms to preserve and extract data in a custom, reproducible and labor extensive fashion. Such a uniform system eases the reuse of methods by data administrators and provides clinical researchers with a uniform (web-based) user interface to quickly extract custom datasets that suit their analysis needs. The detailed descriptions also constitute a foundation for local system developers to base their own projects upon.

**Reusable assets for processing** | In Chapter 7 we report on the case of a bioinformatics group that needed to process (metabolite) genetical genomics data. They required infrastructure to (i) map metabolite quantitative trait loci (mQTLs, regions on the genome containing genes) that supposedly underlie variation in metabolite abundance in genetically different individuals, (ii) predict potential associations between metabolites using correlations of mQTL profiles, rather than of abundance profiles, (iii) assess statistical significance using simulation and permutation procedures, and (iv) provide visual and textual report on predicted metabolic networks. We developed a package of reusable processing modules that can be used in alternative combinations because of the use of standardized (intermediate) data structures that allows the modules to ‘talk to each other’. In addition, this package can also talk to MOLGENIS for genetical genomics, described in Chapter 5. We describe in detail how these programmatic modules work and how they can be used.

**Discussion and future work**

In Chapter 8 we evaluate and discuss our findings on use of novel ‘generative’ strategies for the production, maintenance, and evolution of biological software infrastructures. We first discuss the benefits for bioinformaticists involved in the development of local software infrastructures for particular experiments, based on the experiences with the MOLGENIS
generator tool developed in this thesis. Next, we discuss the benefits of generative methods for the public software infrastructures, for example to ease the integration of biological data and processing tools that are provided by numerous distributed and heterogeneous web resources. Finally, we discuss how even more can be gained if emerging ‘generative bioinformatics’ tools would join hands and define future informatics projects to support these initiatives, to allow future bioinformaticists to generate even more comprehensive software infrastructures for the management, processing and integration of systems biology data.

1.3 Glossary of terms

The research presented in this thesis integrates knowledge and methods from the fields of biology, medical sciences, bioinformatics, (information) management and software engineering. Below, we provide glossaries of terms. We thank many anonymous sources for this information. We apologize if we use these terms in a too loosely manner; we did it to improve readability for a broad public. References to the glossary are indicated in the main text with **BOLD SMALL CAPS**. Interested readers can find also a list of hyperlinks to software infrastructures for biology (or parts thereof) and software engineering tools in the appendix. References to this online information are indicated by *THIN ITALIC SMALL CAPS*.

**Molecular biology terms.**

Below a short glossary is provided to introduce the reader into molecular biology:

**CENTRAL DOGMA OF MOLECULAR BIOLOGY**

DNA is the blueprint of life. DNA can be ‘read’ using a process called transcription resulting in the production of messenger RNA (mRNA). In most species mRNA is then processed to splice out non-relevant parts and then moved from the nucleus (where DNA lives) to the cytoplasm (outer part of the cell). mRNA is then translated into PROTEINS molecules by cell components called ribosomes.

**DNA (DEOXYRIBONUCLEIC ACID), POLYMORPHISM, SNP**

Deoxyribonucleic acid (DNA) is a molecule encodes hereditary information and that is mostly found inside the nucleus (core) of cells. DNA consists of very long strands of chemical bases (also called nucleic acids). Typically $10^5$-$10^8$ adenine, guanine, cytosine or thymine (AGCT) molecules are chained together by sugars and phosphate groups to form a very large molecule. Two DNA strands are paired together into a double helix such that each base (nucleotide) on one strain is paired to a specific partner on the other strand (basepairs A-T or G-C). The specific sequence of these nucleotides determines GENES.
or regulatory elements influencing **GENE EXPRESSION**. By convention, one end of the DNA strand is called 5’ (five prime) and the other end 3’ (three prime). The specific nucleotide sequence varies between individuals of one population (**polymorphism**), for example, there are positions where a single nucleotide is different (single nucleotide polymorphism, **SNP**). In cells, continuous strings of DNA are organized in structures called **chromosomes**. Sexually reproducing species have pairs of chromosomes (i.e. are **diploid**), e.g. Human has 23 pairs of chromosomes, and *Arabidopsis thaliana* has 5 pairs of chromosomes. Asexually reproducing species have one set of chromosomes instead of pairs.

### Genes, Alleles, Gene Expression and Genomics
Genes are discrete units on the **DNA** that code for proteins. This involves specific sequences to switch on a process called **gene expression** which transcribes the DNA into **mRNA** and then translates it into **proteins** (see below) The complete set of **genes** as scattered over all chromosomes is called the **genome**. Due to **polymorphisms**, genes may have different variants (**alleles**) over individuals, which may lead to variation in function and regulation of genes, and as a consequence to different **traits** (or disease). The study of an organism’s entire genome is called **genomics**, with **functional genomics** mainly concerned with the dynamics of gene expression, the function of the proteins produced and protein-protein interaction while **structural genomics** studies the structure of these proteins (typically before function is known).

### mRNA, Transcript and Transcriptomics
**Messenger RNA (mRNA, Messenger Ribonucleic Acid or transcripts)** molecules are copies of a part of DNA. The enzyme RNA polymerase uses the **non-coding strand** of the DNA sequence as **template** for the RNA copy and using base pairing **complementary** to ensure the correct copying, i.e. G-C, A-U (n mRNA T is replaced by U). An mRNA may be **protein-encoding** which means it can be **translated** into a protein. The transcription of mRNA is known as **gene expression** and the study of gene expression is also called **transcriptomics**.

### Proteins, Proteomics and Protein-Protein Interaction
**Proteins** are long chains of molecules consisting of up to 20 different amino acids and include **enzymes** to catalyze chemical processes; **structural components** that shape cells; **hormones** for signaling; **antibodies** to bind (intruding) molecules; and **transport molecules**. Proteins fold into unique 3-dimensional structures and typically have **active sites** that participate in chemical reactions and **protein-protein interactions**. The total set of proteins in a cell is known as the **proteome** and the study of proteins is also called **proteomics**.

### Metabolites and Metabolomics
**Metabolites** are small molecules that are starting, intermediate or end products of **metabolism** (often catalyzed by enzymatic **proteins**). So-called **primary metabolites** are involved in growth, development and reproduction and are necessary for survival. So-called **secondary metabolites** are involved in less important functions such as
pigmentation or antibiotics. Metabolism is the sum of all chemical reactions that take place in every cell of living organisms such as synthesizing cellular material or breaking down complex molecules releasing energy. Metabolism is a complex web of interconnected reactions and its study is called metabolomics.

**Genotype**
Genotype is the result of measurement (Genetic profiling, see below) on how an individual differs from other individuals regarding its DNA sequence and variant of allele.

**Phenotype, Trait**
A phenotype is any observable trait of an organism such as flowering time (known as classical phenotype), but also gene expression or metabolite abundance (known as molecular phenotypes).

**Model Organisms**
Some species are extensively studied to understand particular phenomena. The expectation is that discoveries made in an organism model will give insight in the workings of other organisms. For example, mouse is a model for humans. Other well studied model organisms are budding yeast (Saccharomyces cerevisiae), roundworm (Caenorhabditis elegans), fruit fly (Drosophila melanogaster) and the plant mouse ear cress (Arabidopsis thaliana).

**Systems Biology**
Many components of biological systems, and their interactions, have been successfully identified by studying parts of systems in isolation, such as genes, transcripts, and proteins. However, static diagrams of genes/proteins and their connections are not sufficient to fully understand a biological system. Comprehensive and quantitative data on concentrations and dynamics are needed to make clear how and why cells function the way they do. Increasingly integrative experiments are being designed that combine evidence from multiple ‘ome’-levels (genome, proteome, metabolome), as well as phenotypic data on normal and disease processes, for many model organisms or even humans (clinical data). These systems biology studies promise the comprehensive information needed to understand how whole systems function See for example (Joyce and Palsson, 2006; Kitano, 2002; Sauer, et al., 2007).
Biotechnology terms

Below a short glossary is provided to introduce the reader into the many technologies that have been developed to measure the biomolecular entities as described above.

**Microarrays, Probes**
A microarray is a device that has thousands measurement probes (spots), each with a DNA sequence complementary to the mRNA sequences that are expressed. When the mRNA is available it binds to the probes, thus allowing measurement of gene expression, i.e., how much is a gene ‘switched on’. There are many microarray platforms available that measure one sample (single channel), to estimate the absolute value of gene expression, or two samples (dual-channel) to compare the ratio of gene expression (using two color labels: Cy5/red and Cy3/green). Common platforms are Affymetrix™, Agilent™, Qiagen-Operon™, and Illumina™.

**Mass Spectrometry (MS), Peaks**
A mass spectrometer measures the abundance of molecules (or compounds), i.e., how much of a protein or metabolite molecule is available. A sample is prepared by removing unwanted compounds for example using a separation method like liquid chromatography (then it is then called LC/MS). Next, a mass spectrometry machine ionizes the compounds and measures traveling time in an electro/magnetic field. The traveling time of molecules in the machine is a function of both mass $m$ (resisting acceleration) and charge $z$ (speeding up acceleration). The result is a spectrum of peaks, where each peak position represents a certain mass-over-charge ratio (mass of the molecule divided by its charge, m/z) and each peak height represents the total number of ions counted (intensity).

**Liquid Chromatography (LC)**
A liquid chromatography device separates chemicals in a mixture of chemical compounds (analyte). This mixture is dissolved using a special solvent (mobile phase) and is then passed through solid matter with (solid phase) that has different affinity for different compounds. For example, one phase has affinity for organic materials. Depending on this affinity each compounds takes longer or shorter to pass through the solid phase, which results in separation. For example, first less and later more organic molecules. These separated compounds can later be identified using another method such as mass spectrometry, mass spectrometry (in combination known as LC/MS).

**Polymerase Chain Reaction (PCR), Primers**
A polymerase chain reaction (PCR) can be used to multiply a fragment (partial sequence) of DNA using the enzymatic replication mechanism that normally occurs inside organisms. It can be used to generate millions of copies of DNA which can then easily be measured. A pair of primers is used to select which part of the DNA should be amplified: one primer acts as duplication starting point from one end (5’) and another as duplication starting point from the other end (3’). At each duplication cycle the parts between 5’ and 3’
primer are duplicated the most, while the other parts of DNA are less duplicated, resulting in an exponential surplus of the desired fragment of DNA.

**GENETIC FINGERPRINTING, AFLP, SNPs**

**Genetic fingerprinting** is used to find out what gene variants **alleles** are present in an individual, i.e. what is the allelic combination that defines **genetic make-up** of each individual in a screen. An technology to do genetic fingerprinting uses amplified fragment length polymorphism (**AFLP™**) that first cuts genomic DNA and then uses **PCR** on **primers** that amplify specific **polymorphic** regions of DNA (called **markers**). The result of PCR is then applied onto a ‘gel’ which results in lanes of visual “bands” for each individual, with the availability of the band indicating what variants are available for each marker. Another technology for genetic fingerprinting is using **SNP** microarrays to create a profile of which SNP variant is available in each individual.

**GENETICAL GENOMICS, QTLs**

Genetical genomics is a strategy to map genetic determinants that underlie variations in transcript, protein or metabolite abundance that are observed in genetically different individuals. For transcriptome data, this strategy works as follows: **genetically profile** individuals and measure gene expression (preferably genome-wide) in genetically different individuals, treat the transcript abundances of each gene over all individuals as a quantitative trait, use molecular markers to fingerprint the individuals, use **quantitative trait locus** (QTL) mapping to identify regulators. QTLs are region(s) of DNA that are most likely to regulate a particular trait. In case of gene expression (**eQTL**) a QTL is called **cis-acting** if a QTL is very close to the gene that is expressed and may be that the gene regulates itself. Otherwise it is called **trans-acting**.

**BIOINFORMATICS**

Analysis of the results of molecular experiments requires methods from applied mathematics, informatics, statistics, chemistry, artificial intelligence (etc). This thesis mainly deals with the informatics, and does not touch the numerous other aspects of bioinformatics. In several chapters we shortly mention some statistics therefore we provide some minimal terminology here. Typically an hypothesis is tested, e.g. the **null hypothesis** is that gene X is not differentially expressed between sick and healthy tissues, and the **alternative hypothesis** is that there is a difference. A **statistic test** such as ‘Students t-test’ estimates the probability that the null hypothesis is not true, e.g. with a significance level of 5%. Type 1 error is a **false positive**, meaning the null hypothesis is rejected while actually true. Type 2 error is a **false negative**, meaning the null hypothesis was not rejected while the alternative hypothesis was actually true. Problem with modern high throughput experiments is that there are thousands of hypotheses tested (e.g. for each gene) and, consequently, also hundreds of false positives (e.g. 5%*20.000) and negatives. In **Chapter 7** we mention the need for a mechanism to control this **false discovery rate**.
**Software engineering and information management terms**

Below a short glossary is provided to introduce the reader into the basics of software engineering and information management.

**SOFTWARE INFRASTRUCTURE**

Software infrastructure is the combination of data structures, user interfaces, processing tools and supporting elements that together provide functionality to support certain (biological research) processes. A large part of the work is software ‘plumbing’ to make sure that all components are connected together and data and events flow between them. Software infrastructures tend to become large and complicated programs with many software components written in a programming language. Therefore, appropriate software architecture is a key success factor.

**PROGRAMMING LANGUAGE, COMPIATION, SOFTWARE**

A programming language is an artificial language that can be used to write programs that control the behavior of a computer. A collection of computer programs that performs some task is called software. A computer is a platform that consists of hardware (e.g. Intel, PowerPC) and operating system (e.g. Windows, Linux, Apple). Files written in programming language have to be translated (compiled) into language of a platform by a compiler before they can be executed. Some programs however can run on multiple platforms and are therefore called multi-platform or platform independent. Examples of general programming languages are C++, Java, Python, and Perl.

**SOFTWARE ARCHITECTURE**

Software architecture is described as the software components, the external properties of those components and the relationships among them. A software infrastructure is deemed suitable if it enables all the necessary features and qualities, e.g. if two functions are not always necessary in combination then the components providing this function should be uncoupled to allow to independent use. A common architecture for data centric applications is a three layer architecture: a front-end layer shows a graphical user interface, a back-end layer stores data using a database management system and an intermediate ‘business’ layer that contains all application logic.

**DATABASE MANAGEMENT SYSTEM**

A database management system (DBMS) is computer software that helps to manage databases, i.e. a structured collection of files containing records of data. Typically, a DBMS comes with a specific programming language to query and maintain the data (e.g. SQL), i.e. to retrieve selections of records and use transactions to add, update and delete records. Many DBMS are ‘relational’ (RDBMS) which means they arrange the data into tables with columns as attributes and rows as records. There are also ‘hierarchical’ and ‘object-oriented’ dbms’s.

**SOFTWARE FAMILY / SOFTWARE PRODUCT LINES**
Instead of developing software applications one by one, multiple applications having many ‘phenotypic’ similarities can be approached as one **software family**. Thus, the **software family-members** can exploit their common ‘genetic’ make-up while dealing with variations in a systematic manner. To achieve this, **domain engineers** create **REUSABLE ASSETS** with adequate **variation points** that allow **application engineers** to create a **product line** of software systems more efficiently than when developing each product variant in isolation. (Note: although the terms software family and software product line are often used interchangeably there is an important difference: software families focus on commonalities in building blocks while product lines focuses on commonalities in features from an application domain/market perspective. Much insight on product lines, domain engineering and product engineering comes from non-software product industry).

**MODULE / REUSABLE ASSETS**

To ease the assembly of software, units of functionality are often designed as **modules**. Related modules have a **uniform interface** so they can be **easily assembled** and (re)used interchangeably. Good modules **hide implementation details** so that a change inside one module does not require changes in other modules (i.e. there is only ‘loose’ coupling). Often modules are designed to be reused following systematic procedures, either as as-is, or with systematic modification/configuration. When modules are tailored to a specific domain they are often called **reusable assets**, e.g. a module to visualize regulatory gene networks. When they have a more general purpose they are often called **components**, e.g., a module that shows a calendar dialog on the screen.

**GENERATIVE SOFTWARE DEVELOPMENT**

Generative software development is a **SOFTWARE FAMILY** approach that focuses on automating the mapping from user requirements into working software variants for a specific domain. A desired system can be automatically produced using **CODE GENERATION** from a higher level specification in a **DOMAIN SPECIFIC LANGUAGE**, on top of **REUSABLE ASSETS**. In **GENERATIVE SOFTWARE DEVELOPMENT**, a model of a complete application is described using such a language, and therefore this approach is also known as **model driven** software development.

**DOMAIN-SPECIFIC LANGUAGE (DSL)**

A **domain specific language** (DSL) is a minimal programming language to describe features for a **certain domain** in a compact and easy way. In contrast to ‘general programming languages’ such as Java en Perl, a DSL does not try to cover all behavior of a machine but instead tries to make a **subset of functionality particularly easy to use** for development of certain applications. Examples of domain specific languages are R (statistics), HTML (layout), and MOLGENIS (biological databases).

**CODE GENERATOR / INTERPRETER**

A **code generator** translates a **DOMAIN SPECIFIC LANGUAGE** into a general programming language (such as Java), which is then again **compiled** (by Java) into an executable **PROGRAM**. Alternatively, an **interpreter** can be created that reads the DSL and directly executes the desired behavior.