МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт машиностроения
(наименование института полностью)

Кафедра «Сварка, обработка материалов давлением и родственные процессы»
(наименование кафедры)

15.04.01 Машиностроение
(код и наименование направления подготовки)

Системы автоматизированного проектирования в машиностроении
(направленность (профиль)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему Разработка приложения автоматизированного проектирования деталей и элементов многопозиционных штампов в CAD

Студент	Д.В. Пельникевич		
	(И.О. Фамилия)		(личная подпись)
Научный руководитель	А.В. Скрипачев		
	(И.О. Фамилия)		(личная подпись)
Консультанты	П.Н. Шенбергер		
	(И.О. Фамилия)		(личная подпись)
	П.А. Путеев		
	(И.О. Фамилия)		(личная подпись)

Руководитель программы
канд. техн. наук., доцент Е.Н. Почекуев _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____» _____20_____г.

Допустить к защите

Заведующий кафедрой
д-р. техн. наук., профессор В.В. Ельцов _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____» _____20_____г.

Тольятти 2019

СОДЕРЖАНИЕ

Введение

Актуальность работы. В настоящее время на разработку вытяжных штампов для многопозиционной штамп затрачивается немало времени. В связи с этим, действия, приводящие к снижению времени проектирования, позволят удешевить проект штампа. Среди возможных вариантов снижения трудоемкости существуют дополнительные модули в САПР. Например, Progressive Die Wizard для NX или аналогичные продукты для Creo или программ группы Autodeck. Однако в подобных приложениях типовые элементы конструкций в штампах такие как транспортные приливы, поверхности крепления, ребра жёсткости, карманы для снижения веса плит, пользователю приходится выбирать и вставлять вручную, в свою очередь в вытяжных штампах многопозиционной штамповки для конкретного оборудования параметры указанных элементов постоянны или меняются в небольшом интервале, следовательно целесообразно разработать программный продукт, который учтет автоматизированное размещение и параметры типовых элементов при выборе конкретного оборудования для создания штампа.

Цель и задачи исследования. Целью работы является разработка методики проектирования элементов штампов многопозиционной штамповки за счет разработки приложения проектирования вытяжных деталей и элементов штампов многопозиционной штамповки, что повлечет за собой снижение трудоемкости.

Для решения обозначенных в работе проблем были сформулированы основные **задачи исследования:**

1 объектно-ориентированно описать конструкции штампа на C#;

2 написать программу для автоматической загрузки типовых конструкций штампа, их позиционирования относительно вытяжного перехода детали,

объединение и вычитание элементов штампов для создания конечных ассоциативных копий готовых электронных моделей;

3 апробировать метод для вытяжного штампа абстрактной листовой детали лонжерона.

Научная новизна работы. Разработанное приложение для проектирования элементов штампов многопозиционной штамповки позволяет на базе типовых решений, существующего программного обеспечения создать конструкции штамповой оснастки для существующих процессов.

Методы исследования. Для решения поставленных в работе задач были использованы:

- методы автоматизированного проектирования конструкций штампа;
- научные основы
- технологии ОМД;
- основы программирования на языке C#.

Предмет исследования: приложение для проектирования вытяжных деталей и элементов штампов многопозиционной штамповки.

Объект исследования: многопозиционная штамповка.

Практическая ценность. Объединение традиционных решений и средств автоматизации проектирования на основе функционального анализа позволит повысить эффективность производственных процессов многопозиционной штамповки.

Объем и содержание работы. Диссертация состоит из введения, четырех глав, заключения, списка использованных источников и одного приложения. Общий объем диссертации – 107 страниц. Список использованных источников состоит из 30 наименований. Диссертация содержит 123 рисунка.

1. Изучение и редактирование параметрических схем и типовых конструкций штампов

1.1 Типовой состав штампа для вытяжки многопозиционной штамповки

Вытяжные штампы подразделяются на две категории, обусловленные операциями, для которых их проектируют: штампы для первой операции такой как вытяжка и штампы для второй и последующих операций вытяжки. Штампы, не оснащенные прижимным устройством, используют для вытяжки неглубоких полых деталей и при вытяжке из толстого листового материала, когда нет опасения в образовании складок на штампуемой детали. Для неглубокой вытяжки используют прижимные устройства, действующие от пружин или резиновых амортизаторов, размещаемых в верхней части штампа, когда необходимо создавать большое давление, буферное устройство помещают под нижней плитой штампа и плитой пресса. Пневматические буферные устройства используют при вытяжке глубоки и крупноразмерных. В основе вытяжного штампа лежат плиты для быстрого закрепления пуансона, пуансон, верней плиты, прижима, прижимных секций, адаптера, секций матрицы, матрицы, правой/левой направляющий, нижней плиты, упора, планки, ограничителя высоты цилиндрический формы, шпонки, ловителей.

1.2 Обзор классификаций деталей штампов

В литературе и известных проектных решениях применяются различные классификации деталей штампов. Необходимо рассмотреть одну из них, представленную следующим образом классификацию:

- деформирующие детали, такие как пуансоны, матрицы, пуансоны-матрицы, ножи, зубила, скдадкодержатели, предающие конечную форму изделию;
- устанавливающие детали, такие как упоры временные, упоры постоянные, ловители, направляющие ленты, фиксаторы, прижимы боковые, задающие и ориентирующие заготовки относительно штампа;
- управляющие детали, такие как съемники, выталкиватели, сбрасыватели, пружины, облегчают съем и удаление детали;
- сборочные детали, такие как монтажные, плиты, пуансонодержатели, матрицедержатели, корпуса, крепёжные, направляющие, хвостовики, болты, винты, гайки, штифты, прихваты, колонки, втулки, закрепляют детали штампа относительно друг друга.

1.3 Моделирование рабочих частей штампа

С помощью NX 9.0 были смоделированы такие тела как заготовка, матрица, пуансон и прижим. У рабочих поверхностей инструмента геометрическая форма получена с помощью WAVE-технологий, которые представляют собой нисходящий метод проектирования сборки вместе с сохранением её ассоциативных связей между данными объектами. Далее задаться ориентация заготовки и рабочих инструментов относительно друг друга. При разработке штамповой оснастки используется технология WAVE-моделирования для сборок. Ранее была разработана геометрия вытяжного перехода на основании геометрии детали, она использовалась для моделирования рабочего инструмента пуансона, матрицы, прижима. Поверхности вытяжного перехода копируются со ссылкой в модель рабочего инструмента. Этими поверхностями производиться резание заготовки для матрицы прижима и пуансона которые были получены ранее с помощью стандартных средств моделирования таких как: вытяжка, вычитание, фаски, скругление и прочие. WAVE-технология позволяет в случае изменения

геометрии вытяжного перехода автоматически изменять геометрию рабочего инструмента. Остальные детали штампа получают стандартными средствами моделирования на основании ГОСТ, мат моделей и отраслевых стандартов.

1.4 Моделирование стандартных деталей штампа

Пуансон, прижим и плиты располагаются на нижней плите. Отлитый пуансон оснащен приливами для отвода воздуха и служит для закрепления плиток, а также и направляющих планок, закрепленных винтами и так же штифтами. Пуансон крепится к плите при помощи Т–образных пазов винтами, а также штифтами. Отлитая также нижняя плита имеет приливы для осуществления транспортировки и приливы служащие местом закрепления направляющих планок и плиток, ограничителей высоты цилиндрической формы, а также шпонок. Это все крепится за счет винтов и штифтов так же, как и нижняя плита, к адаптеру. Прижим крепится на нижней плите на которой находятся секционные части прижима, которые ограничивают течение металла при выполнении операции. Прижим оснащен приливами для направляющих, находящихся в нижней части данного штампа.

Нижняя плита и система выталкивания находятся на нижней части штампа. Винтами и штифтами нижняя плита закрепляется к адаптеру. В нижней части данной нижней плиты находиться оригинально разработанная система выталкивания, оснащенная толкателями, которые совпадают по форме с геометрией готового полуфабриката. К магистрали цеха подключена пневмосистема.

Необходимо рассмотреть более подробно: вытяжной пуансон крепится адаптеру, для его крепления используются шесть винтов и два штифта. В пуансоне выполняются два отверстия для транспортировки и переналадки, сам же пуансон закреплен в маркетной плите на шести толкателях, которые проходят сквозь монтажную плиту и адаптер к маркетным шпилькам. Шпильки закрепляются на пневматическом цилиндре, который приводит в

движение маркетную плиту. Секции прижима крепятся к маркетной плите, для ограничения хода которой применяют ограничители. Для крепления адаптера в верхней плите штампа используют резьбовые отверстия, а на самом адаптере отверстия для закрепления пуансона и матрицы. Втулки применяют для точного направления нижней половины штампа относительно верхней. В плитах применяют пазы, обычно в количестве четырех, для того что бы зашел болт, чтобы прикрепиться к прессу. На каждой плите имеются грузовые приливы, в количестве четырех, для транспортировки плит штампа.

1.5 Инструменты сборки

Не мало важной функцией NX 9 является сборка. Тела изменяются напрямую на уровне детали, при необходимости, а на уровне сборки к компонентам тел прямого доступа нет, их предварительно необходимо для моделирования сделать доступными.

Тело компонента на уровне сборки в NX возможно изменить следующими способами: переносом тела, вырезом в сборке и WAVE-копией тела.

Перенос тела – операция, позволяющая сделать на уровне сборки твердое тело компонента доступным для других операций. На уровне сборки все те изменения, что произошли с телом, содержатся только в ней, в то время как в детали исходное тело не претерпевает изменений. Данная функция позволяет работать на уровне детали, и на уровне сборки с одним и тем же телом. Автоматически нельзя поменять источник операции переноса тела.

При помощи операции выреза в сборке создается вычитание твердого тела из других твердых тел компонентов этой сборки на уровне сборки.

Редактор геометрических связей WAVE, содержащийся в NX, позволяет создавать на уровне сборки копии тела компонента, и применять к

копии все операции дерева построения, как и к любому другому телу. Все действия производятся не над самим телом компонента, а над копией тела. Соответствует по свойствам телу, непосредственно созданному на этом уровне, являясь при этом телом на уровне сборки.

1.6 Процесс проектирования

Для автоматизации процесса проектирования штамповой оснастки разрабатывается программа для NX Siemens PLM Software с описанием методов и алгоритмов, с формированием электронной модели штампа и типовых его агрегатах. Объектно-ориентированное программирование дает возможность создать математическую модель штампа.

Проектирование вытяжных штампов предназначенных для вытяжки листовых заготовок требует значительных затрат ресурсов. Для целей упрощения создаются прикладные приложения, в основе которых положен системный подход, базирующийся на иерархической структуре конструкции, в состав которой входят состоящей из параметризованный механизм и его узлы. В дальнейшем производится параметризация узлов модели штампа, которая выигрывает в рациональности модели с обособленными параметризованными деталями.

Системный процесс написания программного обеспечения реализуется системным подходом что подразумевает по собой представление модели штампа иерархической структурой взаимно связанных деталей и узлов. Это фиксирует целостные параметры у объекта такие как динамику и организацию. Системный подхода в данной работе представлен объектно-ориентированным программированием.

Вытяжной штамп — это подмножества деталей и узлов такие как группы штампа рабочего инструмента, штамповые плиты, узлы задающие движения плит, устройства фиксации и направления материала, прижимные и удаляющие механизмы материал, агрегаты, ограничивающие ход

подвижных частей штампа, узлы транспортные, агрегаты крепления штампа. Структуры разняться, но по признакам идентичности обосабливаются конструкции, являющимися типовыми.

Модель структуры иерархии, выявленные взаимосвязи узлов и деталей штампа реализованы в данных которые созданы объектно-ориентированным программным приложением в среде NX OPEN на языке C#. Основываясь на установленных потоков данных и сформированной диаграмме классов реализовалось программирование с использованием данных из библиотек NX OPEN, которое упаковало классы в dll-компонуемую библиотеку. Пользователю предлагается интерфейс с последовательным прохождением процедуры проектирования, которое в итоге сформирует файлы, которые соответствуют по содержанию определенному коду из динамической библиотеки программы.

Создание типовых конструкций штампов при использование разработанного приложения приводит сокращает цикл проектирования в среднем в 1,5 - 2 раза.

2. Составление исходного кода приложения автоматизированной загрузки штампа

2.1 Сравнение C# и C++

Известны возможности обоих объектно-ориентированных языков: C# сохранил основы возможности C++, которые нбе реализовались в Java, к примеру, перечисления. Для C# перечисления обеспечивают безопасность для типов. Так же C# перенял из C++ основные понятия такие как операторы и ключевые слова. Реализовано обеспечение перегрузки операторов. C# может добавлять блоки ненадежного кода.

Так же существуют и различия: C# реализован комплексным методом программирования. C++ разработчики объявляют классы в заглавном файле и исполняют классы в виде отдельного файла. Interface Definition Language - язык описывающий интерфейс и сложно исполненные интерфейсы не обязательны. C++ имеет инициализированные в конструкторе переменные для экземпляров объектов, в C# это задается заранее. В C# для массива, размером [n-1], недопустимо установка ссылки на это значение n которое нереальное, выходить за границы массива, как это реализовано в C++.

C++ совместим с множественным наследованием классов, что усложняет код. В C# нет поддержки множественного наследования только путем представления посредством интерфейса. C# классы, причем все, включают в себя базовый единый класс. Класс автоматически наследует члены и свойства из System.Object, только если он не производный от другого класса.

Код C/C++ компилируется в формат языка, ориентированного на исполнение на базе определенного процессора и операционной системы.

В C# предустановлена компиляция программных продуктов на промежуточный язык IL (Intermediate Language), код которого может

инициируется любой платформой, использующей среду .NET. JIT-компилятором происходит преобразование в зависимый код для платформы .NET. Компилируются программы напрямую в стандартную двоичную форму выполнения.

С точки зрения скорость разработки C# позволяет прототип решения получить быстрее, но, когда инфраструктура создана под проект, настроен билд, выбраны библиотеки, скорость разработки с C++ уравнивается.

Рассмотрение производительность кода и требовательность к ресурсам. Благодаря возможностям оптимизации почти любая задача на C++ решается более ресурсоемко. В задачах с обработкой большого количества данных C++ перед C# имеет сильные преимущества. Преимущества C++ в возможности писать код, выполняемый непосредственно процессором и напрямую с памятью.

Библиотеки C++ оптимизированы, часто кроссплатформенны, с открытым кодом, имеют разную, архаичную архитектуру, часто не объектный, а структурно-процедурный интерфейс. Неприятная особенность C++ библиотек — это создание и переопределение своих базовых типов. Многие C++ библиотеки переопределяют некоторые базовые типы заводя свои типы строк, контейнеров. C# имеет в базе огромное количество библиотек с .net и свободно доступные библиотеки.

Код C++ и C# имеют внешнее сходство. Но C++ является одновременно и C и C++ и C++0x и все это можно использовать одновременно. У C# его синтаксис постоянно расширяется. Код C# проще и лаконичнее. В C++ нет в базе reflection сборки мусора и позднего связывания. В C# нет полноценных деструкторов, макросов, грубая настройка наследования, отсутствие константных членов и методов, отсутствие глобальным методов. Синтаксис C# это примерно упрощённая версия C++.

В C# больше шансов написать чистый код, меньше шансов допустить ошибку сложном коде при равных ресурсах. C++ имеет проблемы с бинарной

совместимостью библиотек, если библиотеки не в исходниках, то важна их совместимость. Структуры C# не поддерживают наследование, но способны к реализации интерфейсов.

Полной самодостаточности у C++ и C# приложений нет. Для C++ необходим runtime, а для C# .net framework. Но runtime C++ может быть статический линкован в исполняемый модуль и исполняемый модуль станет самодостаточным так как будет содержать все необходимое для работы, в случае C# такое не реализуемо стандартными средствами. Для условной компиляции существуют директивы препроцессора, предупреждений, ошибок и контроля, но существуют операторы способные к перезагрузке и которые нет.

Сборка C# проектов заметно легче сборки проектов C++. Из-за быстрого прототипирования под Windows C# являться предпочтительным решением относительно C++.

2.2 Сравнение C# и Java

Среди известных схдств двух языков C# включает языковые свойства, которые реализованы в Java. Это интерфейсы, единичное наследование, компиляция в промежуточный формат и близкий синтаксис. C# все же имеет отличия от Java, такие как использование свойств, позаимствованные из Delphi, реализованы в виде непосредственной интеграцией с COM моделью (Component Object Model). Компиляция выполняется программным кодом, не зависящим он оборудования и языка программирования, инициализация происходит в рамках управляемой среды выполнения. Так же были позаимствованы и булевы операции. Между любым типом данных и булевым типом прямого преобразования нет. Так же как в Java, управление обработкой ошибок происходит за счет захватывания объектов исключения.

Концепция JVM (Java Virtual Machine) схожа с подсистемой CLR (Common Language Runtime) языка C#. Реализация функции сборщика

мусора. Предварительная проверка кода перед тем как его выполнении. Отсутствие констант или глобальных функций, определенная принадлежность к классу. C# и Java не используют заголовочные файлы, но осуществляет поддержку внутренних классов.

Так же существуют и различия: в основном это полная интеграция C# со стандартом COM. Наследование компонентов, созданных любым языком программирования, поддерживающего .NET, которые могут стать базовыми для других языков в дальнейшем. Код на промежуточном языке MSIL не интерпретируется, а компилируется в машинный код с помощью JIT-компилятора, из этого следует что созданные приложения на языке C# быстрее в исполнение чем разработанные на языке Java.

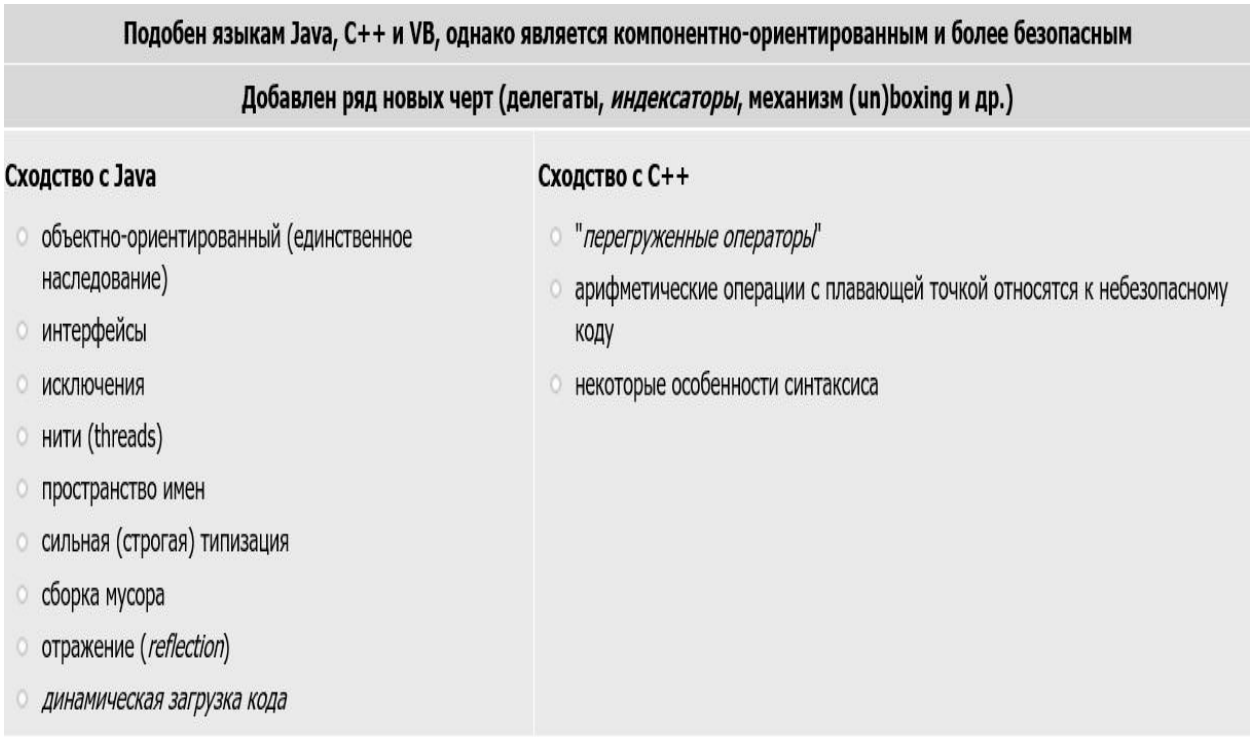


Рисунок 2.1 – Основные возможности C#

2.3 Уникальные особенности C#

C# учитывает все его возможности FCL и CLR. Типы, встроенные в язык представлены классами. В C# допускаются, но не поощряются такие опасные свойства C++ как адресная арифметика, указатели, разыменование,

адресация. Исполнительная среда CLR предоставляет собой компилятор промежуточного языка.

Создавая программу в namespace разработчик получает один или более классов. Вне класса реализована возможность объявление интерфейсов, structs и enums. Адресовать содержимое иного namespace возможно используя ключевые слова.

В C# существует Фундаментальные типы данных, более разнообразные чем в C, C ++ или Java. Такие как - ushort, int, uint, long, bool, byte, ubyte, short, ulong, float, decimal, double. Все типы имеют установленный размер, могут быть знаковыми и без знаковыми. В C# тип decimal имеет ёмкость до 28 десятичных цифр, char содержит 16-ти битный unicode символ.

В языке реализовано два фундаментальных класса таких как object - базовый класс всех классов и string - базовый класс. Являясь частью языка он, используя компилятор, заключает создаваемую строку в программе в кавычки. Так же существует ассемблирование это объединены в отдельном файле коллекция компилируемых классов и выполнение других элементов языка. Так же рассматривается два элемента дизассемблера, C:\Program Files\Microsoft Visual Studio .Net\FrameworkSDK\Bin\ildasm.exe и JetBrains dotPeek. Они имеют признаки каждый член класса: public, private, protected internal, protected, internal.

Прохождение аргумента, для принятия некоторого числа аргументов объявляться методы. Для передачи значения по определенной ссылке используется ключевое слово ref, возвращающее значение. Переход по ссылке без передачи значения реализуется ключевым словом out.

В языке известны виртуальные методы. Перед переписыванием метода в базовом классе он объявляется как virtual. Метод, который будет переписан в подклассе, объявляется ключевым словом override, что предотвращает случайную перезапись метода.

C# внутри любого класса позволяет определять свойства. Каждому свойству внутри C# класса дается тип данных и имя. Для объявления

выполняемого кода при обновлении свойства или чтении существуют ключевые слова get accessor и set accessor. Delegate и callback объект delegate содержит информацию, необходимую для вызова определенного метода. Для безопасного запроса к представленному методу обращаются посредством объекта delegate. Метод callback - пример delegate. В определении методов используется ключевое слово event, которые при возникновении события вызываются.

C# позволяет разработчикам поддерживать множество версий классов в двоичной форме, помещая их в различных namespace. Без вероятности потери данных происходит преобразование типа byte в int, но вероятность потери данных есть если при преобразовании int в byte выполняется явное преобразование. C# для других элементов программы расширяет эту способность.

2.4 Классы в C#

Класс – ключевое понятие для объектно-ориентированного программирования. Если проанализировать типы данных можно увидеть, что структурами задаются встроенные типы-значения, а классами задаются ссылочные типы такие как строки и объекты.

Кроме того, класс – это некий шаблон, определяющий форму объекта. В нем содержится код и данные, для последующего оперирования этими данными. C# использует спецификацию класса для того что бы построить объекты, являющееся экземплярами класса. Класс представлен по сути в виде ряда схематических описаний способа построения объекта. Так же класс представляет собой логическую абстракцию. Только после создания объекта класса в оперативной памяти компьютера появится физическое представление этого класса. Структуры отличаются от классов простотой и заложенной в них функциональностью.

Классы и структуры — это шаблоны, необходимые для создания объектов. Данные и методы содержатся в каждом объекте, необходимые для манипуляции этими данными. При определении класса необходимо объявить данные, содержащиеся в нем и код, оперирующий этими данными. Если наиболее простые классы содержат в себе не более одного кода или только данные, то в большинстве существующих классов содержится и код и данные. В данных-членах содержатся данные, определяемые классом, а код находится в функциях-членах. C# предусматривает для данных-членов и функций-членов несколько разновидностей.

Далее рассматривается следующая схема, которая визуализирует и структурирует наше понимания того что содержит по своей структуре в себе класс, информация представлена на рисунке 2.2:

Рисунок 2.2 – Структура классов

Данные-члены — это члены, содержащие данные класса такие как константы, поля и события. Они бывают статическими static. Член класса (объект) если только он не объявлен явно как static, является членом экземпляра. Статические данные-члены могут быть использованы как общие данные для нескольких объектов.

Виды данных:

1. Поле (field) — это все переменные, которые ассоциированы с классом.
2. Константа (const) — ассоциируется с классом аналогично переменным, объявляется ключевым словом const. Если константа или поле объявляются как public, то они становятся доступными извне класса.
3. Событие (event) — это члены класса, разрешающие объекту сообщать вызывающим кодом о том, что произошло что-то необходимое для упоминания, к примеру, изменение свойства класса или взаимодействие с пользователем. Клиент может включать в себя код, обработчик событий, откликающийся на них. Возможно создание собственных событий-делегатов.
4. Функции-члены — это члены, обеспечивающие функциональность для манипулирования данными класса. Такие как методы, конструкторы, финализаторы, свойства, операции и индексаторы:
5. Метод (method) — это функции, ассоциированные с определенным классом. Они являются членами экземпляра по умолчанию, как и данные-члены. Метод становится доступным без указания-объявления объекта, если они были объявлены статическими с помощью модификатора static.
6. Свойство (property) — это наборы функций, способ доступа к которым осуществляется аналогично, как и к общедоступным полям класса. В C# ревизован специальный синтаксис для записи свойств и чтения для классов, поэтому писать методы начинающимися на Set и Get, нет необходимости. Не существует специального синтаксиса для свойств, отличающий их от обычных функций, поэтому создается видимость

объектов как реальных сущностей, предоставляемых пользовательскому коду.

7. Конструктор (constructor) — это специальные функции, инициализирующиеся автоматически при создании объекта. Их имена идентичны с классом, к которым они принадлежат, не имеющим типа возврата. Конструкторы применяются для создания полей класса.
8. Финализатор (finalizer) — это функция, вызывающияся при определении средой CLR ненужности объекта. Их имена идентичны с классом, но с предшествующим символом тильды. Предсказать появление финализатора (деструктора) невозможно.
9. Операция (operator) — это простейшие действия, такие как + или -. При сложении применять операцию + к целым числам. C# позволяет так же указать каким образом существующие операции будут взаимодействовать с пользовательскими классами (перегрузка операции).
10. Индексатор (indexer) — это функция, позволяющая индексировать объекты аналогично массивам или коллекциям.

2.5 Платформа Microsoft .NET

.NET – это платформа, содержащая в себе средства разработки для современных приложений и вместе с этим средства обеспечения их функционирования. Платформа включает в себя несколько основных составляющих таких как: .Net Framework (фундамент платформы, отвечающий за создание программы, её запуск и последующее управление за выполнением как программы так и её библиотечных компонентов), .Net Enterprise Servers (семейство серверов корпоративных), Visual Studio .Net (средство для разработки приложений), клиенты типа Windows операционные системы, XML Web-службы в виде расположенных по всей сети Internet блоков, приложения с которыми, основываясь на

спецификациях языка XML, могут взаимодействовать, протоколах HTTP или SOAP.

Framework .Net — единый каркас среды разработки, состоящий из двух основных компонентов: FCL (Framework Class Library) статический, содержащий библиотеку классов, а именно в формате DLL набор сборок, которые содержат определённые типы в количестве несколько тысяч, причем каждый из них это некоторая функциональность динамический. CLR (Common Language Runtime) — это общеязыковая исполнительная среда для программ, на .NET-совместимых языках программирования.

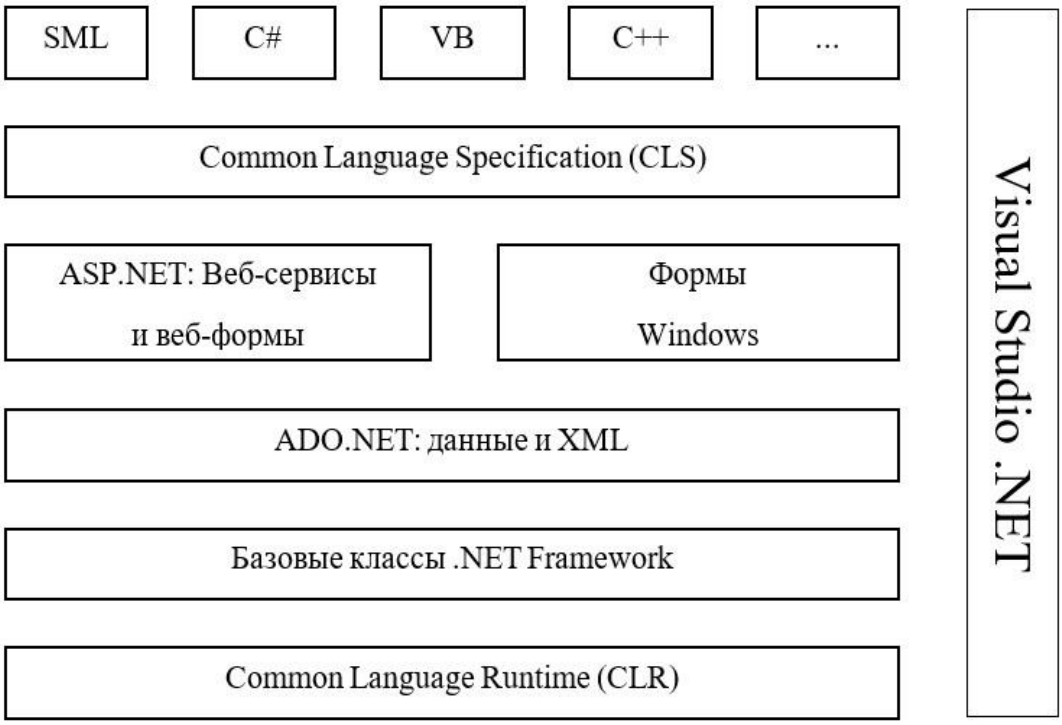


Рисунок 2.3 – Архитектурная схема .NET Framework и Visual Studio.NET

Рассмотрение примера трансляции многокомпонентного программного проекта под управлением Microsoft .NET. Компонент проекта написан на языке C#. Исходные тексты нашего компонента у проекта компиляторами транслируются с языка C# в унифицированный MSIL-код с дальнейшим сохранением в виде сборок в файлах. В процессе компоновки и выполнения

программного проекта Just-In-Time (JIT) CLR компилятор среды выполняет проект с означиванием промежуточного оттранслированного кода сборок, отвечает за безопасность, а также работоспособность кода и программы программист-разработчик, а не среда проектирования и разработки ПО .NET.

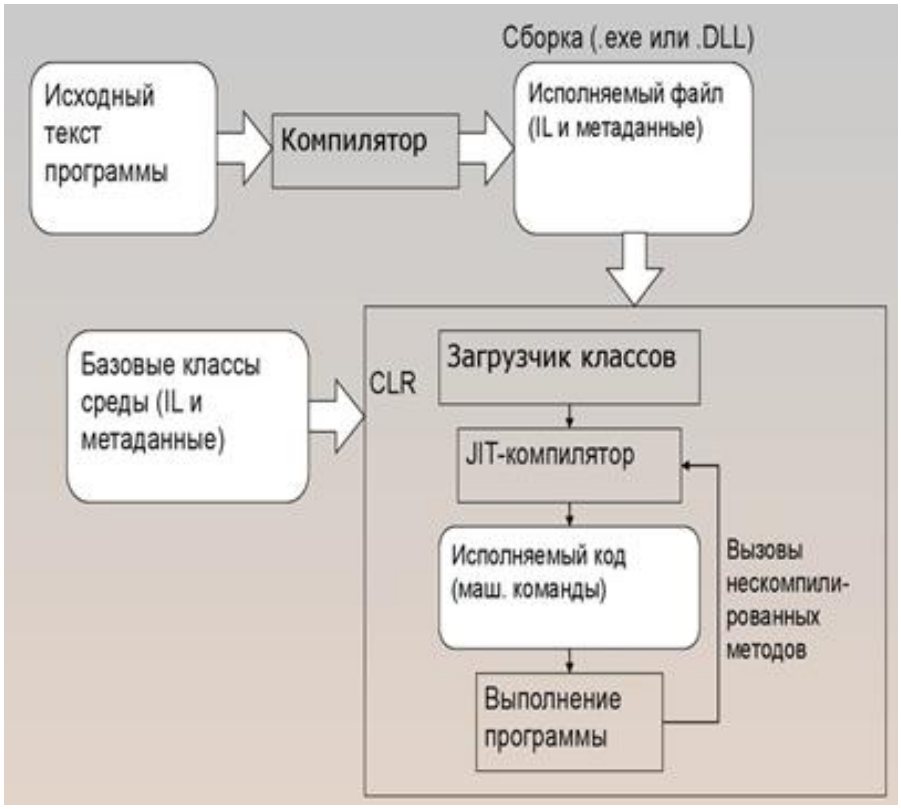


Рисунок 2.4 – Схема выполнения программы в .NET

Рисунок 2.5 – Схема компиляция исходного кода в управляемые модули

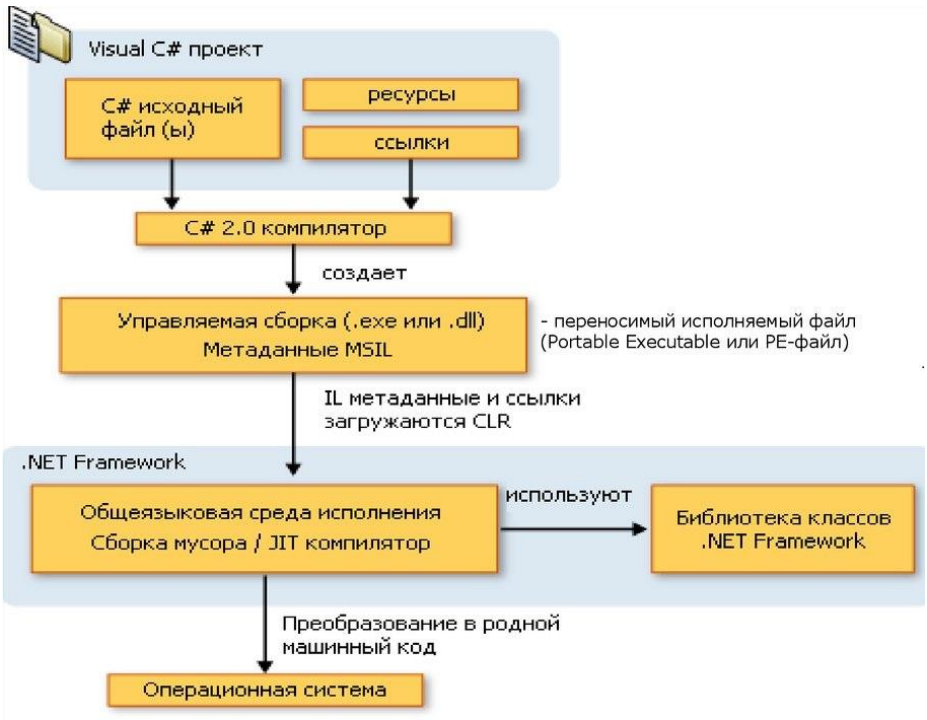


Рисунок 2.6 – Схема C# проекта

CLR — это виртуальная машина, необходимая для функционирования приложений .NET. CLR является совокупностью служб и утилит, включающие все необходимые компоненты для создания, последующего запуска и выполнения разработанного приложения.

CLR включает Class Loader (компонент, отвечающий за управление загрузкой классов MSIL (Microsoft Intermediate Language), Code Manager (компонент, отвечающий за управление запуском кода), Carbage Collector (сборщик мусора, предназначен для освобождения ресурсов памяти, которые не используются приложением), Security Engine (механизм защиты приложений), Debugger (компонент, отвечающий за отладку приложения и контроля за выполнением, приложения которое запущенно), Type checker (компонент отвечающий за гарантируемое корректное приведение типов данных), Exception manager (механизм передачи исключения). Thread support (это компонент, отвечающий за поддержку классов для создания многопоточных приложений), COM marshaler (компонент отвечающий за обеспечение взаимодействия с COM компонентами).

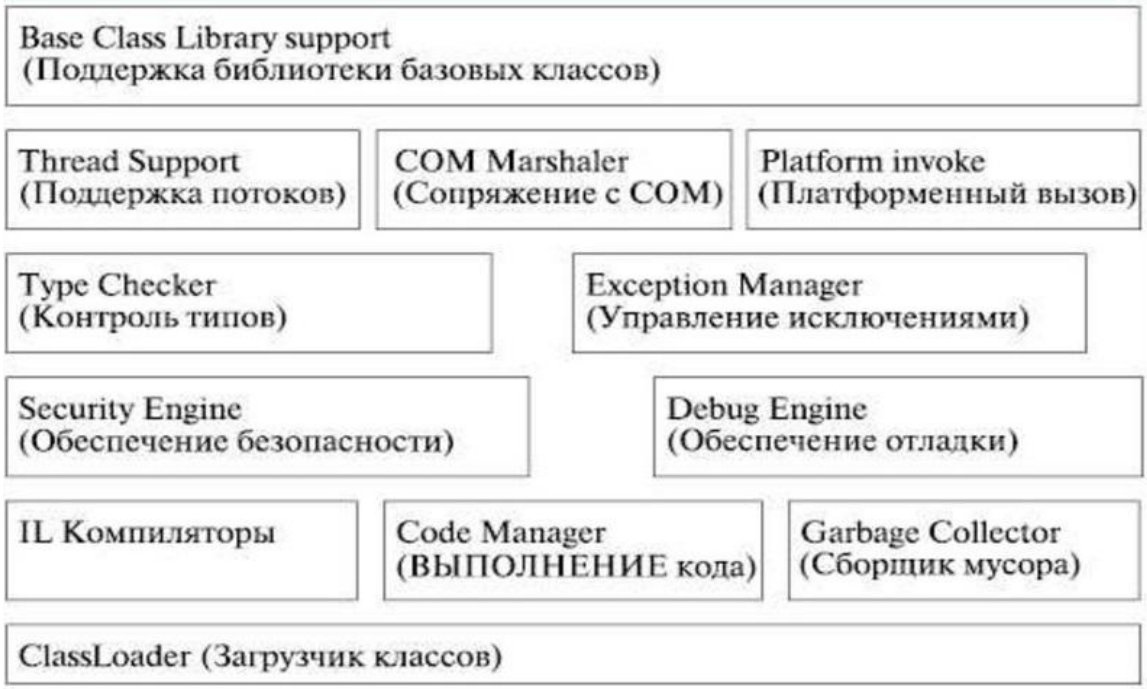


Рисунок 2.7 – Структура среды выполнения CLR (основные функциональные элементы среды)

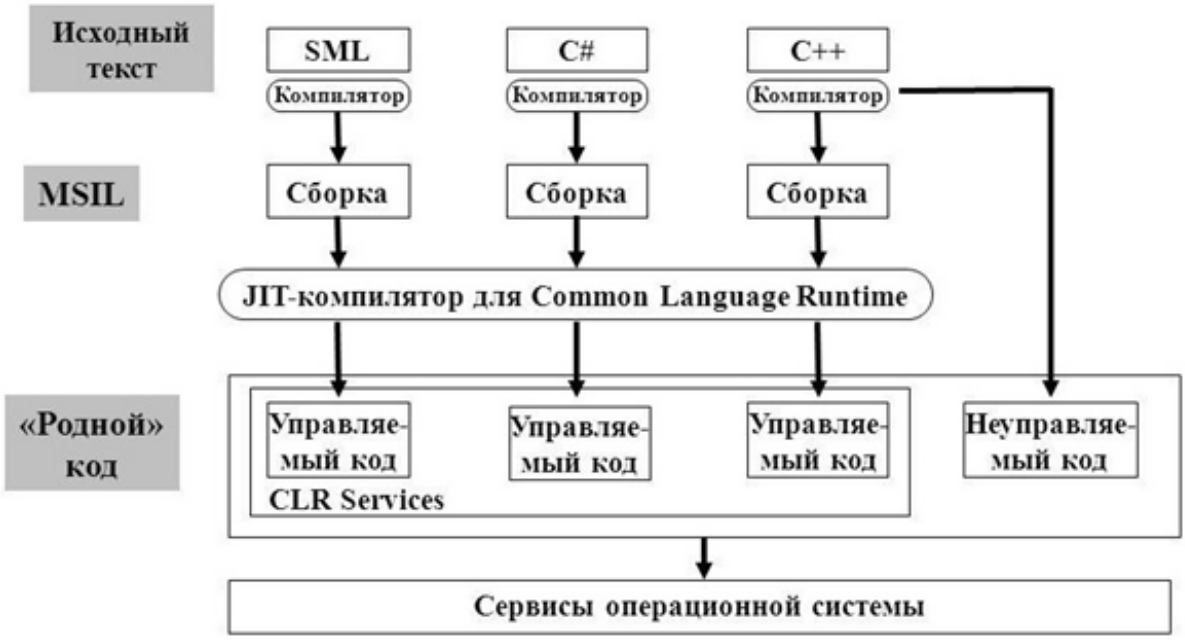


Рисунок 2.8 – Схема выполнения CLR

CLR (Common Language Runtime) поддерживает C++/CLI, C#, F#, Visual Basic, Iron Python, Iron Ruby, Ассемблер Intermediate Language (IL).

Система типизации Microsoft .NET – это множество, упорядоченное частично, на качественном уровне представляет собой ISA-иерархию. Система типов Microsoft .NET представляет собой иерархию с возрастанием общности снизу-вверх рисунок 2.8:

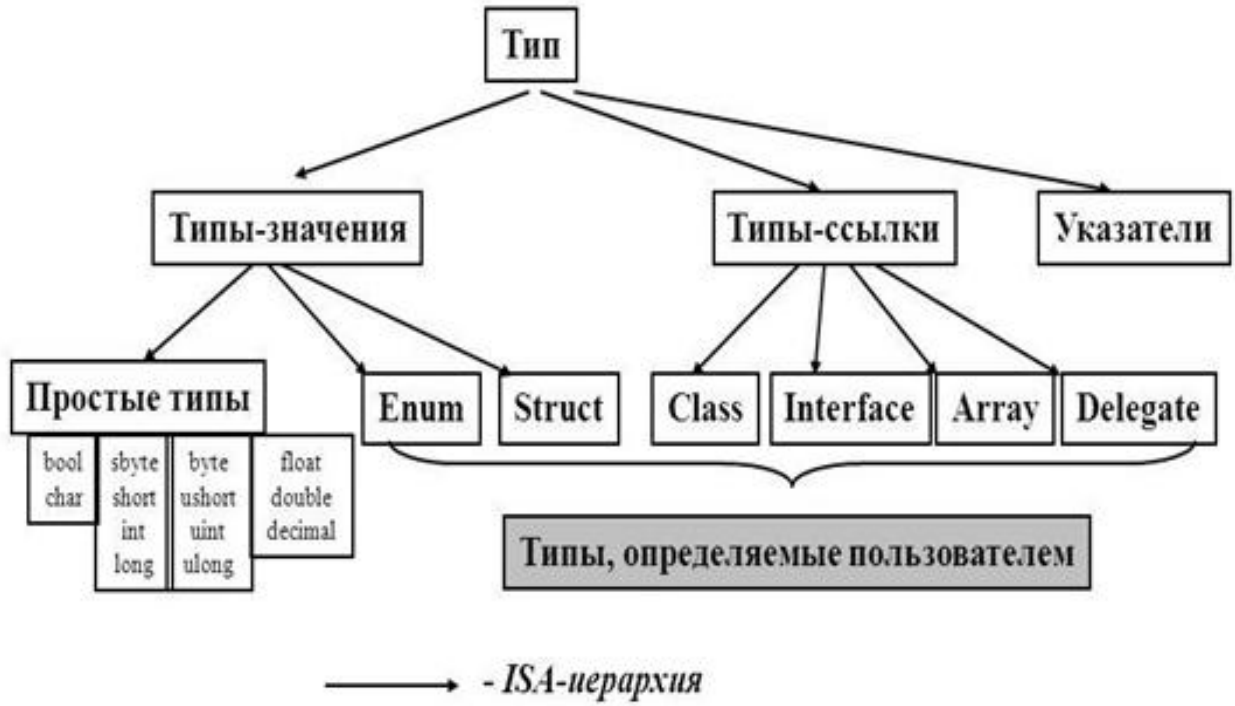


Рисунок 2.9 – Универсальная система типизации (UTS)

На рисунке представлены две большие группы типов, такие как типы-ссылки. Различие между типами-значениями выражаются в особенностях их вызова в процедурах: по значению или по ссылке (call-by-reference, CBR) и по имени (call-by-name, CBN). Система типизации Microsoft .NET, как дополнение к развитой иерархии предопределенных типов, даёт возможность создания пользователю собственных типов, таких как типы-значения и типы-ссылки, на основе раннее созданных.

Обобщенная структура программы на языке программирования C# представлена на рисунке 2.10.

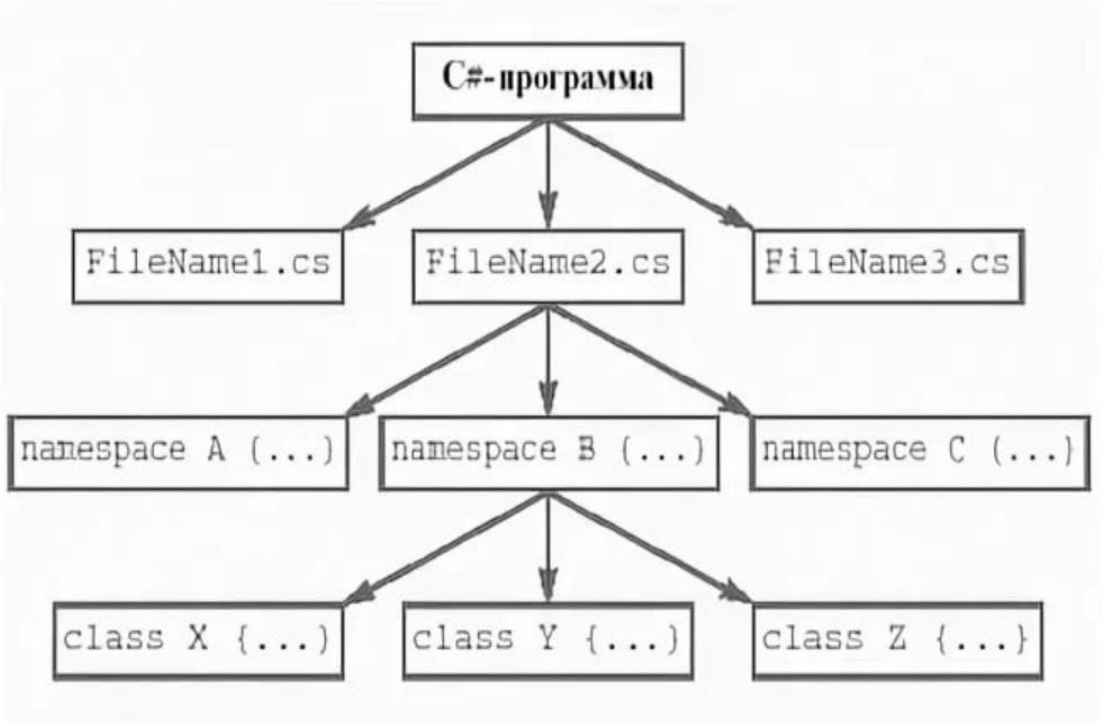


Рисунок 2.10 – Структура программы на языке C#

Стоит отметить, что программа на языке C# может быть представлена одним или несколькими файлами, в которых находится исходный текст. Каждый файл расширением .CS имеет имея, например, имена файлов FileName1.cs, FileName2.cs и FileName3.cs. Любой файл с исходным текстом на языке C# может содержать пространства имен или нет. Так в приведённом примере файл FileName2.cs состоит из трех пространства имен A, B, C, а FileName1.cs и FileName3.cs не имеют пространств имен. Каждое пространство имен может содержать описание классов, одного или нескольких, в приведенном примере пространство имен B содержит три описания трех классов X, Y, Z, а пространства имен A и C не описывают ни один из классов.

2.6 Microsoft Visual Studio 2012

Microsoft Visual Studio 2012 представляет собой среду разработки и является средой программного обеспечения, принадлежащей компании Microsoft.

Достоинствами данного продукта являются:

- оптимальная производительность для разработчика. поддержка всех языков программирования, интуитивная оптимизация рабочего процесса, высокий уровень автоматизации;
- единая модель для разработки всех приложений. поддержка всех известных платформ;
- полная поддержка жизненного цикла разработки, от планирования и тестирования программного продукта до развёртывания и последующего управления. возможность расширения и адаптации среды для всесторонних поставленных задач.

2.7 Разработка методики связи Visual Studio и NX

После завершения установки Siemens NX 9.0 и Microsoft Visual Studio 2012 появится шаблон NX9_Open C# Wizard, при необходимости шаблоны подгружаются вручную (рисунок 2.10).

Далее выбирается шаблон NX9_Open, после чего откроется окно Wizard’a, при помощи которого происходит настройка параметров создаваемого нами проекта. Первоочередным для любого разработчика является решение о том, каким будет его приложение: отдельное приложение EXE формата, запускаемое двумя кликами мыши, или внутреннего типа – DLL файл, для запуска которого потребуется непосредственно то приложение, для которого и было написано это приложение, в данном случае это Siemens NX 9.0.

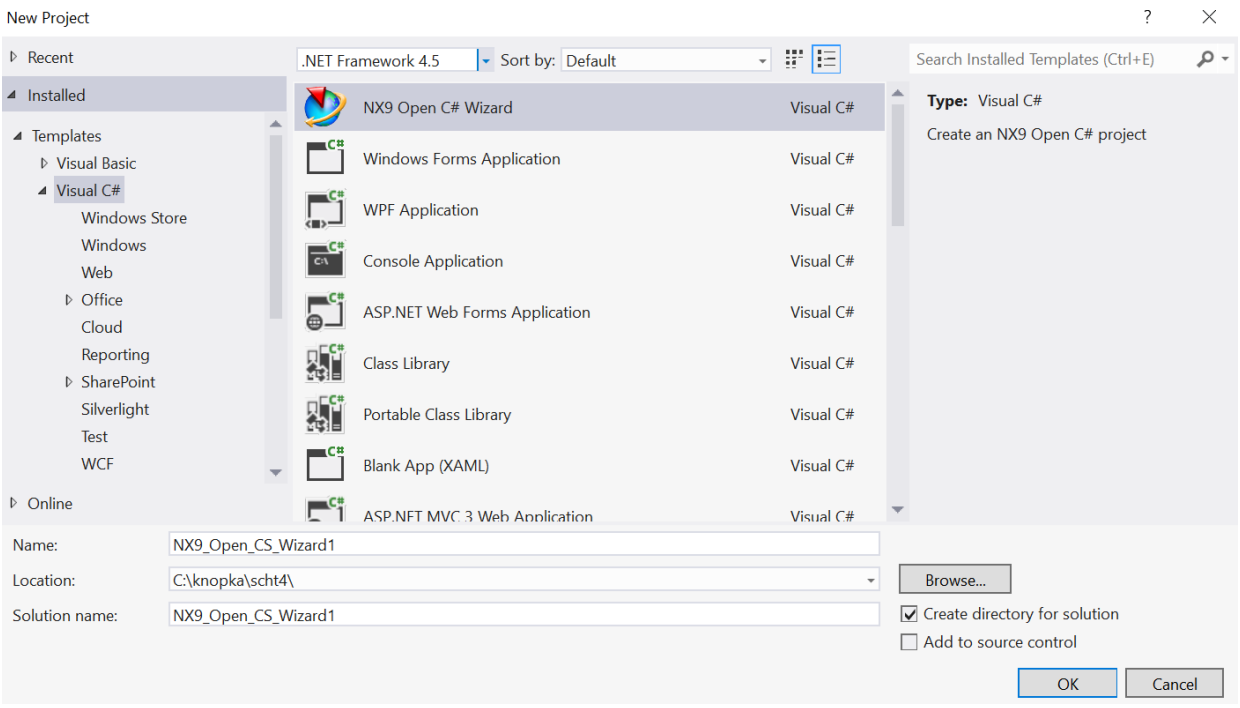


Рисунок 2.11 – Шаблон NX9_Open C# Wizard в Visual Studio 2012

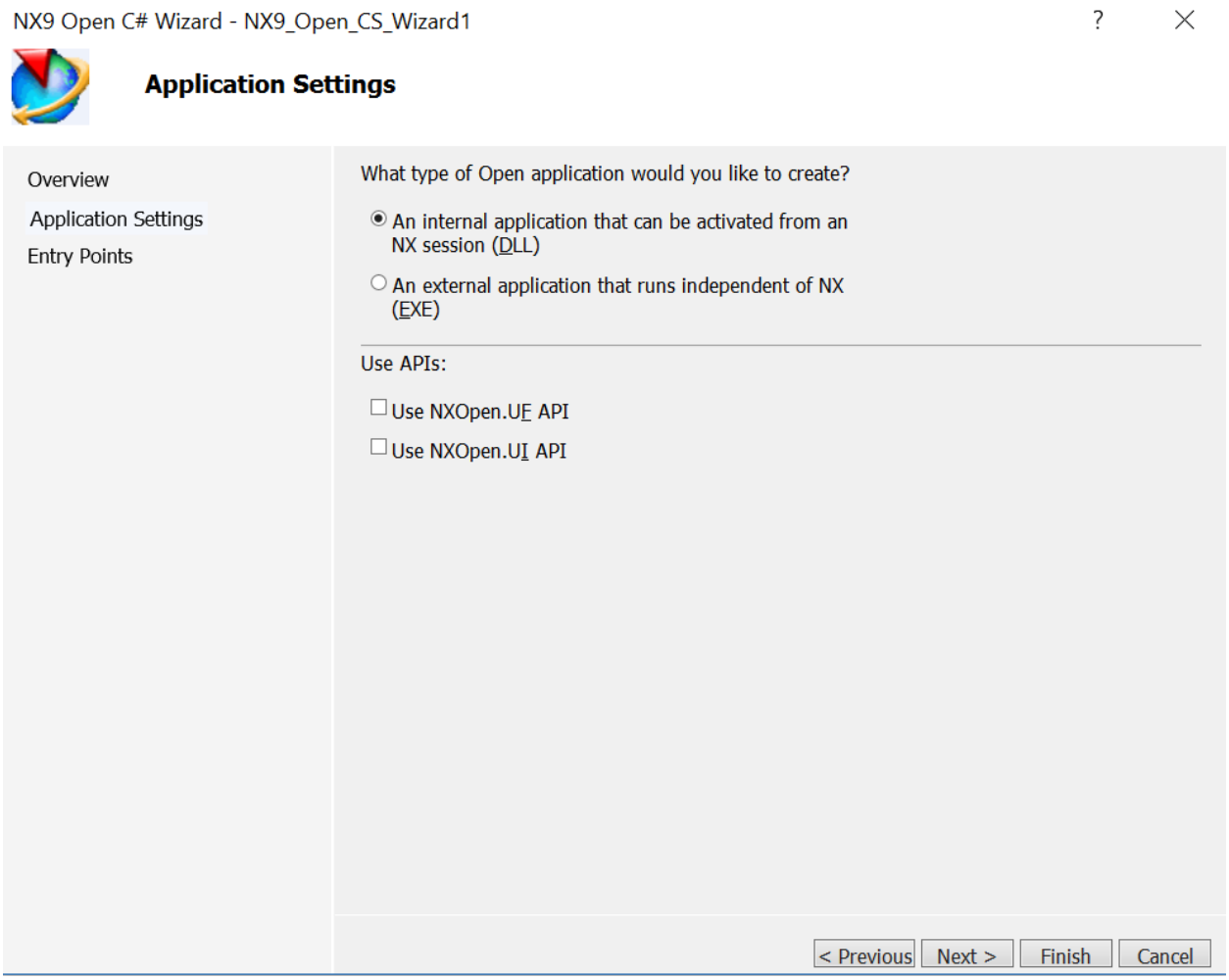


Рисунок 2.12 – Настройка параметров проекта

Затем настраиваются точки входа. Wizard предоставляет выбрать из представленных вариантов, запустить системой NX приложение автоматически при инициализации или пользователем вручную заданной одной или нескальными им точками входа. Так же существуют точки входа во внутренний модуль NX 9.0, которые выполняются при выборе пункта From a User Exit. Главная точка входа в прикладной модуль – ufusr (), вызывается в момент, когда пользователь запускает приложение. Далее выбирается процесс выгрузки приложения, автоматически выгружать при завершении работы приложения или же всей сессии NX, а также явно через диалог выгрузки. Настройка точек входа и выгрузки показана на рисунке 7.

По завершению всех настроек создастся проект, который содержит заголовочный файл, в который далее будут записываться файлы всех необходимых библиотек, и файл, непосредственно содержащий код программы, реализуемой разработанным приложением.

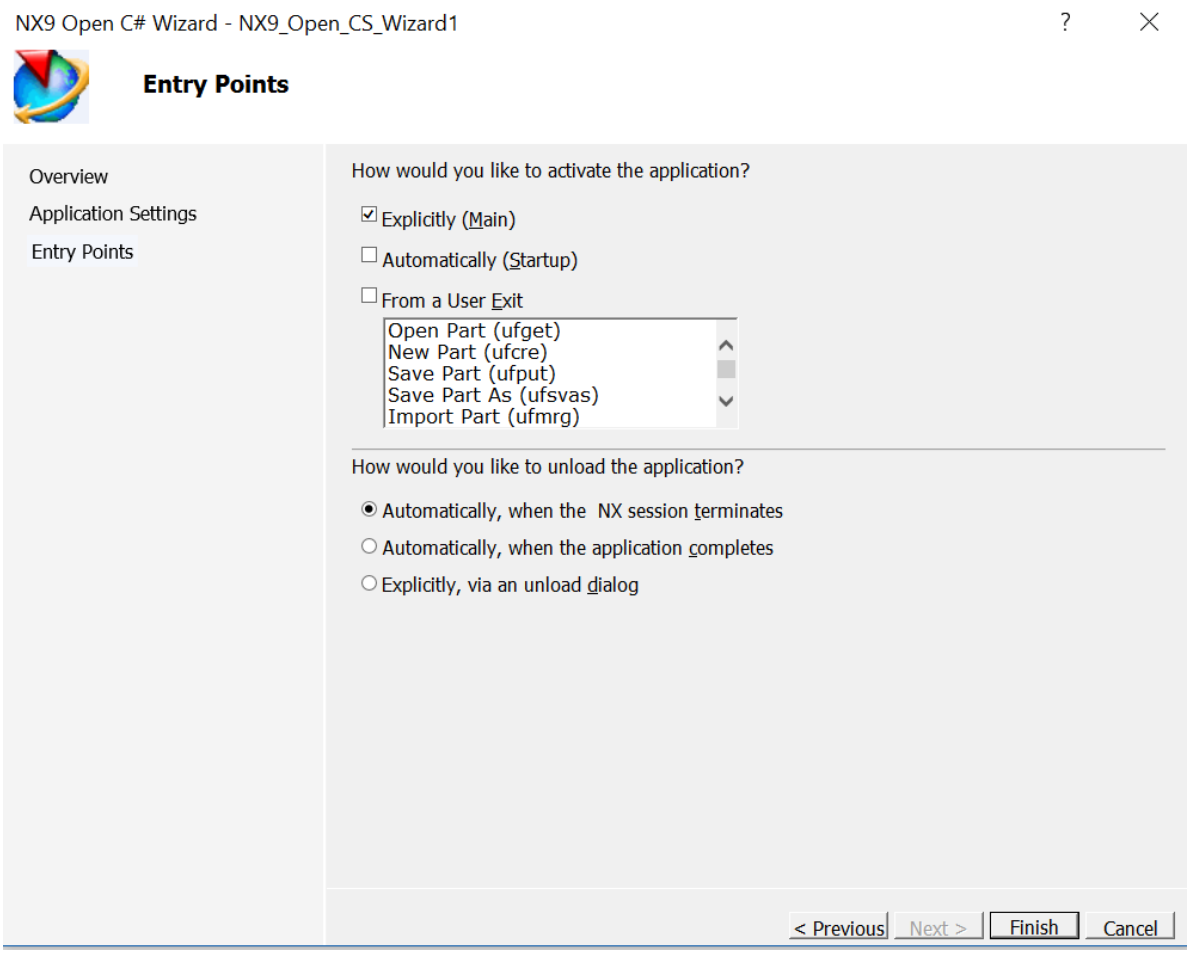


Рисунок 2.13 – Выбор точек входа и выгрузки приложения

2.8 Выводы раздела

Во второй части магистерской диссертации рассмотрен процесс написания исходного кода для загрузки выше описанной модели и создание связей между ними. В частности необходимо было решить задачи загрузки концептуальной модели штампа и деталей в проект, проверки имен выбора элементов в рабочем окне их имен, параметров при вводе числа строковой переменной, выбора из списка. Так же стояли задачи автоматической обрезки тел другими деталями применением булевых операций различными частям одного файла, размещенного пользователем изделий.

3. Разработка приложения методами NX

3.1 Особенности программирования в NX и его модулях

В настоящее время как и у большинства программных пакетов САПР, у Siemens NX 9.0 открытая система. Особенности архитектуры позволяют создать программные модули в дополнение к существующим, в последствии используя их работая над проектами. Нестандартные задачи, решаемые пользователем на предприятии, вынуждают оптимизировать, расширять и дополнять стандартные возможности чертежно-графического редактора NX.

Модули для программирования в NX представлены в виде функциональных законченных фрагментов программы, в виде приложений, содержащих исходный код, являясь непосредственной её части. Это файлы программы, которые в случае вызова пользователем будут исполняться компьютером.

Программный продукт NX – это передовое приложение для Windows, имеющее интерфейс прикладных разработок, именуемый API. NX Open API является совокупностью технологий и инструментов, при помощи которых приложения извне могут использовать возможности NX. NX Open посредством программного способа, который основывается на рассчитанных параметрах, позволяет проектировать сборки и детали, при этом давая возможность выпуска документации. Не смотря на то что с помощью NX Open открывается доступ практически ко всем возможностям NX, существует класс объектов, создаваемых исключительно программным способом. По сути NX Open – инструментальные средства для разработки приложений, например, прикладных САПР, библиотек типовых конструктивов и прочего.

Пользователями для создания приложений NX Open используется среда разработки Microsoft Visual Studio 2012. Для создания приложений для

NX 9.0 используют .NET Framework 4.5. Зачастую внутреннее приложение NX Open – это построенная по определённому алгоритму динамическая библиотека DLL, которая подгружается к процессу NX. С помощью нее производится поэтапное создание пользовательской библиотеки для NX 9.0, разработанной средствами API функций NX 9.0 на языке C# в среде разработки Visual Studio 2012.

Компилирование и редактирование кода происходит в Microsoft Visual Studio, куда программный продукт NX встраивает шаблон проекта, при этом необходимо соблюсти соответствие версий NX и Visual Studio, так, для NX 9.0 понадобиться Visual Studio 2012. Библиотека NX Open состоит из классов и функции пространства имен NXOpen, определенных в большом количестве заголовков файлов в директории UGOPEN\NXOpen программного продукта NX 9.0. Функционал приложению осуществляют за счет подключения к нему соответствующих заголовочных файлов.

3.2 Создание папок и файлов для работы приложения

В текстовом редакторе по типу Notepad++ или Блокнот создается файл, содержащий ссылку на разрабатываемое приложение. Прописывается путь к файлам будущего приложения в котором так же будет храниться и вся информация о нем. Путь задается по типу Диск:\папка, вся запись не должна иметь пробелов и писаться латиницей. Расположение папки не имеет значение, главное, чтобы весь путь к ней и файлам, содержащимся в ней, не имел пробелы, поэтому часто использующаяся папка Program Files не подойдет для размещения собственного приложения. Файл сохраняется в папку C:\Program Files\Siemens\NX 9.0\UGII под именем ugii_env.master с изменением разрешения .txt файла на .master.

На диске, создается папка C:\primer, в которой создаются еще две папки: application и startup. Папка application отвечает за хранения файлов геометрии моделей, файлов диалога и исполняемой библиотеки (файлы. dlx и

.dll). Пака startup будет прописан путь к вызову пользовательского меню, опционально, в текстовом редакторе по типу Notepad++ или Блокнот.

Далее происходит процесс создания пользовательского интерфейса. Запускается NX и открывается или создается с нуля любая модель.

3.3 Разработка метода создания графического интерфейса приложений для Siemens NX 9

Неотъемлемой частью любого приложения является его пользовательский интерфейс. В NX Open API разработка интерфейса представлена в виде:

1. Задействование особых функций при создании интерфейса.
2. Создание окон диалога при помощи визуального конструктора User Styler Interface.
3. Создание окон диалога при помощи визуального конструктора Block Styler.
4. Задействование системных библиотек Windows API или библиотек аналогов других издателей.

Оптимальным визуальным конструктором для создания окон диалога является Block Styler. Панель блоков и основное окно диалога, из которого состоит интерфейс Block Styler, представлен на рисунке 1. Данный модуль вызывается из главного меню NX при помощи команды «Начало -> Все приложения -> Разработка интерфейса пользователя».

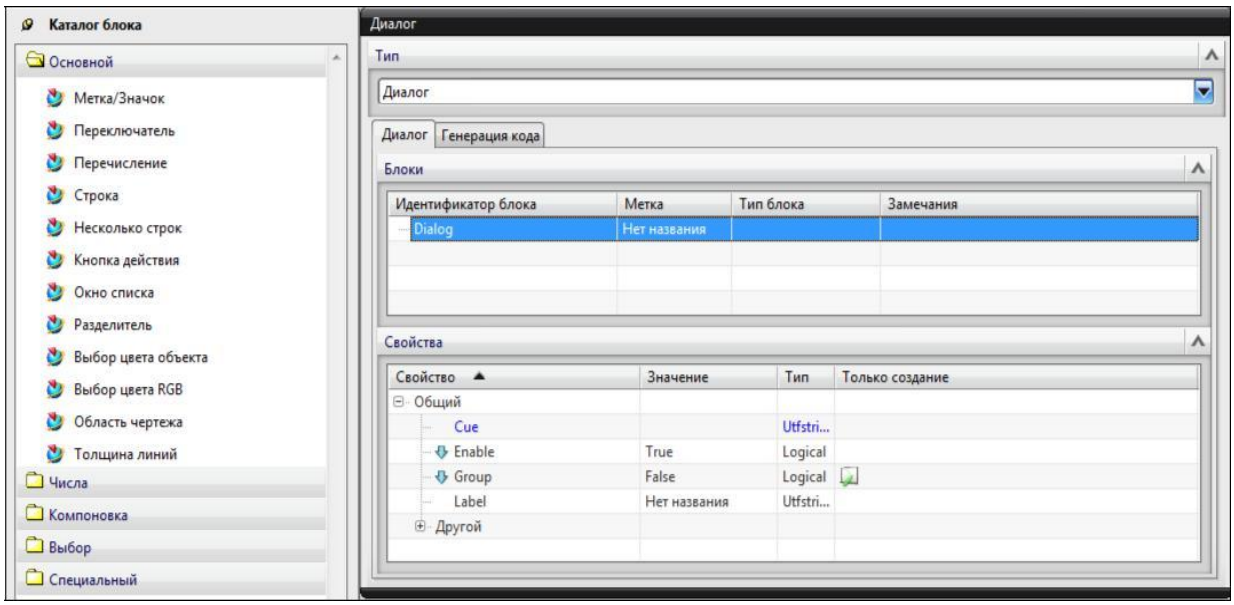


Рисунок 3.1 – Интерфейс Block Styler

В окне Dialog во вкладке Code General в строке Language (язык записи кода) выбирается в контекстом меню C#, рисунок 3.3.

Так же при выборе языка необходимо выставить аналогичные значения и для журнала, в противном случае файл не будет компилирован.

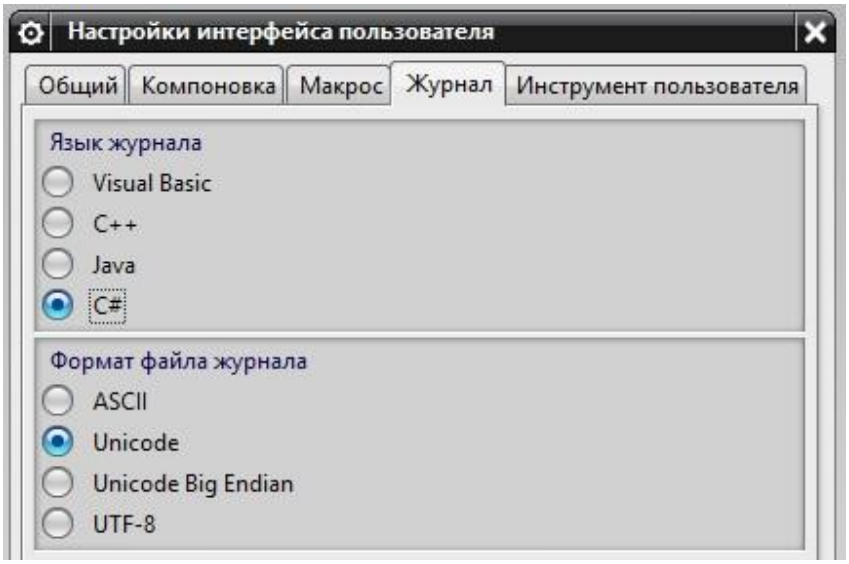


Рисунок 3.2 – Язык журнала

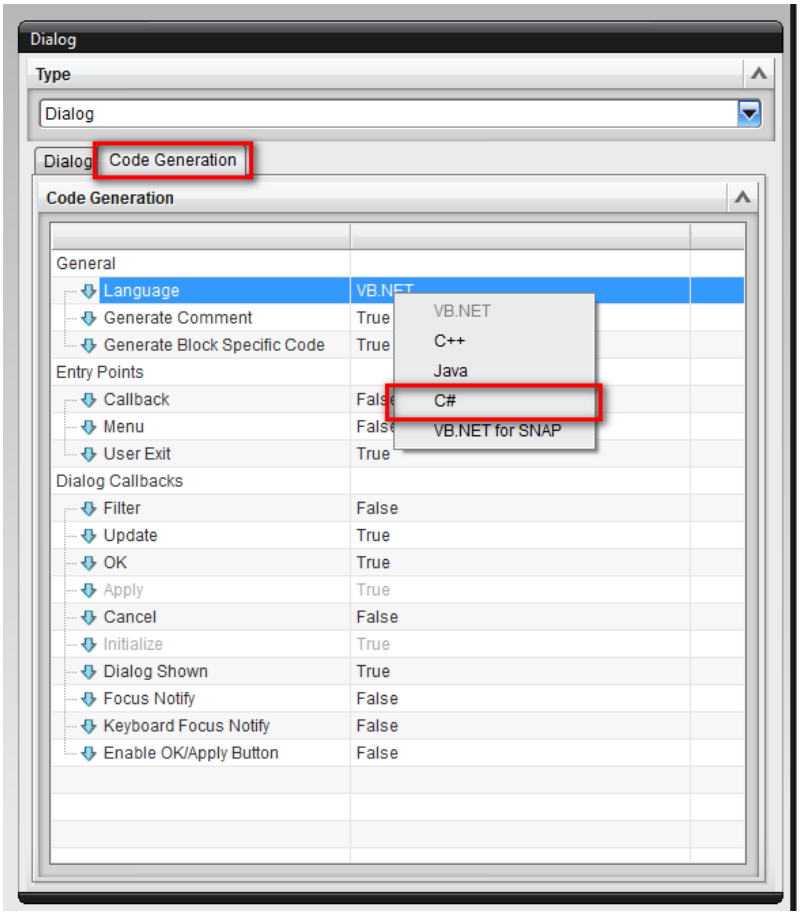


Рисунок 3.3 – Выбор языка кода

В окне "Диалог" структура диалогового окна представлена в виде дерева. Далее для выбранного пользователем блока представлены перечисляются свойства, с возможностью изменения при необходимости. Во вкладке "Генерация кода" располагаются настройки для шаблона программы, представленные в виде используемых кнопок внизу диалогового окна, с заданием таких параметров как язык программирования, отображение комментариев, используемые точки входа и так далее.

Панель "Каталог блока" содержит в себе все доступные блоки, которые при необходимости вставляют в диалоговое окно. К ним относятся метки, таблицы, поля ввода данных, списки, блоки для выбора объектов в трехмерном пространстве, блоки для группировки других блоков и так далее. При добавлении пользователем первого блока происходит вызов заготовки

диалогового окна согласно используемым настройкам (рисунок 3.4), так же соответствующим блоком дополняется дерево структуры.

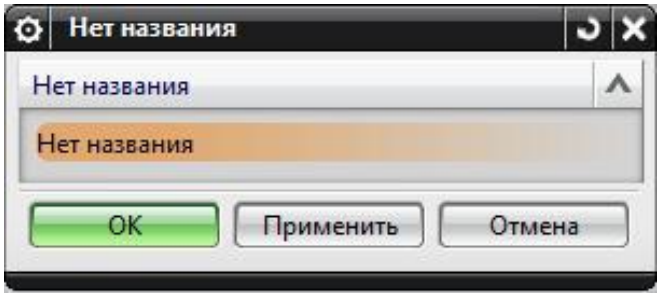


Рисунок 3.4 – Диалоговое окно с блоком типа "Метка/Значок"

Каждый блок имеет общие и специальные настройки, например, относительное расположение блока по отношению к другим элементам в диалоговом окне группа "Присоединения". Сюда же относятся параметры группы "Общие", наиболее значимым является свойство "BlockID", ведь по этому имени к блоку происходит обращение в программном коде. Свойства такие как "Enable" и "Show» отвечают за отображение и активность блока в момент пуска программы. Свойство "Label" используют для задания текста, отображаемого рядом с блоком. Свойство "Group" отвечает за стиль отображения блока, словно он помещен в группу использования блока "Группа", но без изменения структуры дерева.

Запуск программы осуществляется с помощью командой «Файл -> Выполнить -> NX функция пользователя» после чего выбирается созданный dll-файл.

По завершению создания интерфейса нашего окна диалога этот проект сохраняется в папку application под именем без пробелов и латинскими буквами. В папке создаются два файла: файл диалога. dlx и файл кода .cs.

В папке C:\Program Files\Siemens\NX 9.0\UGII\menus располагаются файлы формата *.men, отвечающие за построение меню UG. В файле ug_main.men прописывается путь к вызову кнопки в виде, представленном на рисунке 3.5.

```
! *************** File Menu ***************

MENU UG_FILE

  BUTTON primer
  LABEL Please work
  MESSAGE Create something
  SYNONYMS PDW, punch, start, wizard
  BITMAP mw_assembly
  ACTIONS C:\\primer\\application\\primer.dll
```

Рисунок 3.5 – Синтаксис кода

На рисунке 3.5 BUTTON – это имя созданной копки, которая должна совпадать с именем файла dlx, LABEL – это описание созданной кнопки, которую разработчик добавляет, как комментарий, MESSAGE – это слова, которые всплывают при наведении курсора на строку меню, SYNONYMS – это синонимы для поиска функций, BITMAP – файл иконки, которой будет отображаться в меню и ACTIONS – это ссылка на файл dll.

Создается код, который запуститься при нажатии на эту кнопку, после чего запускается Visual Studio, выбирается шаблон NX Open C# Wizard. Имя проекта должно совпадать с окна диалога, а папка для кода application, тип файла – dll.

3.4 Создание динамической библиотеки

В обозревателе решений Visual Studio 2012 при помощи контекстного меню из дерева удаляется файл кода Program.cs, который был создан по умолчанию и необходимо его заменить на код, созданный нами ранее. Код будет находиться в папке application и иметь название primer.cs. В дальнейшем код будет дополнен необходимыми операциями и компонентами.

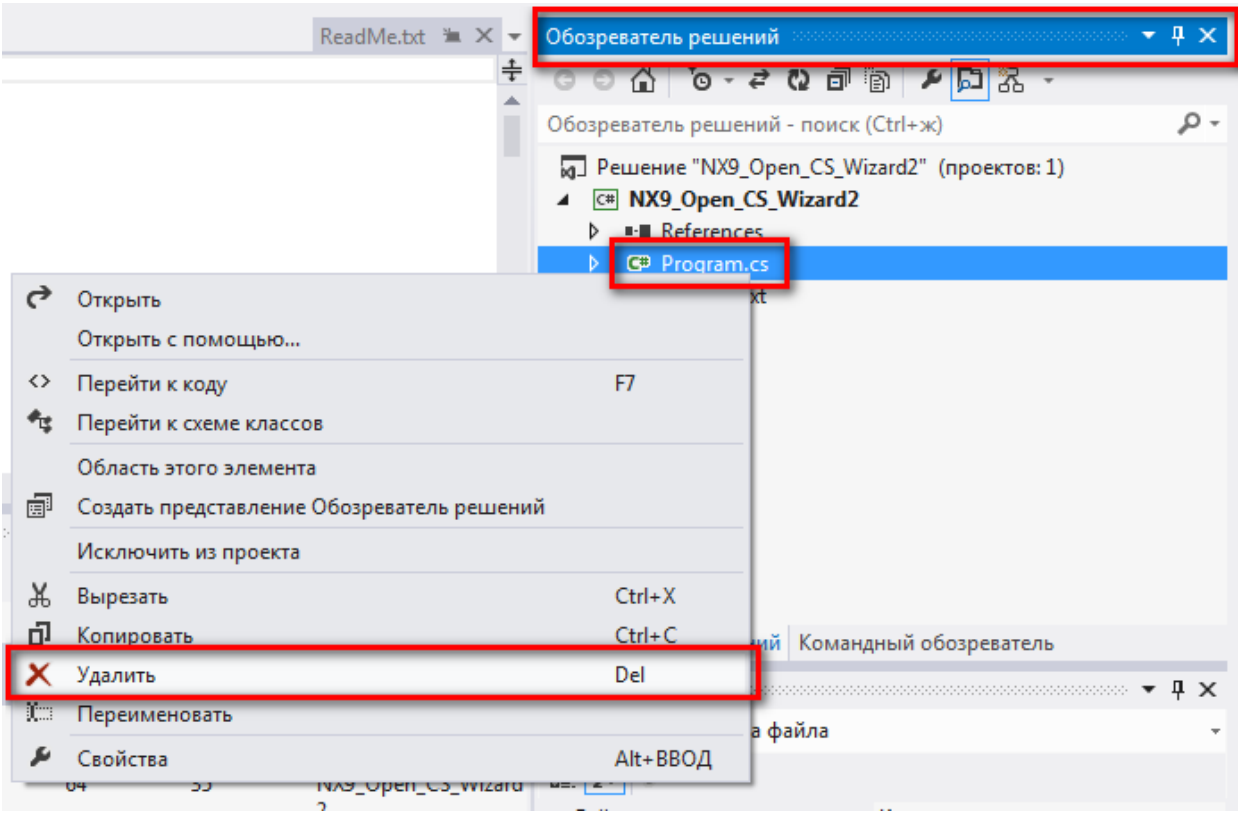


Рисунок 3.6 –Удаление .cs файла

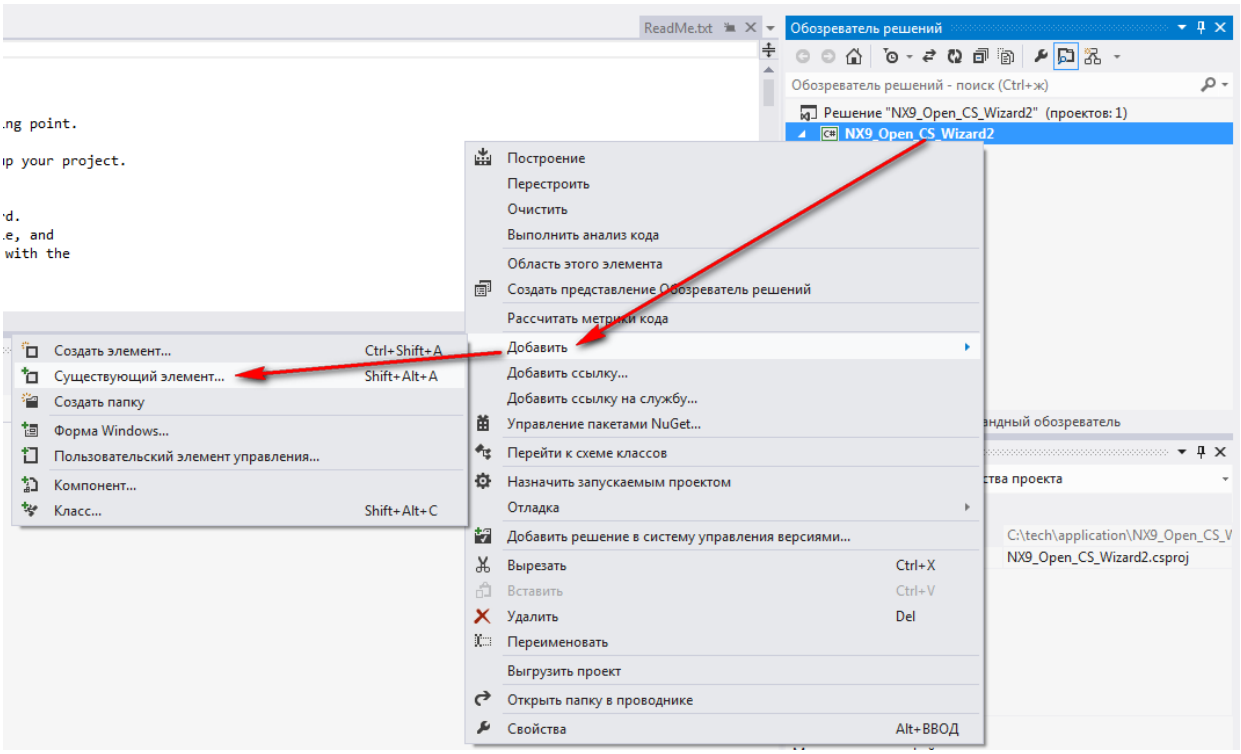


Рисунок 3.7 – Замена .cs файла

Инициализируется подгруздка библиотек NXOpen в созданный проект. Выбирается обозреватель решений строка Ссылок (references) и проверяется наличие подключенных библиотек. Загрузка библиотек производится вызовом контекстно меню, далее необходимо добавить ссылку и в открывшемся окне в нижней части выбирается кнопка «Обзор». Теперь необходимо прописать путь C:\Program Files\Siemens\NX 9.0\UGII\managed, где подключаются все библиотеки. При сообщении от системы о том, что библиотека подгружена следует выбирать все, кроме нее, до тех пор, пока не загрузятся все библиотеки. Когда код написан, запускается проект, во время чего происходит отладка, то есть проверяется синтаксис кода, ищутся ошибки, проверяется наличия библиотек и совместимость.

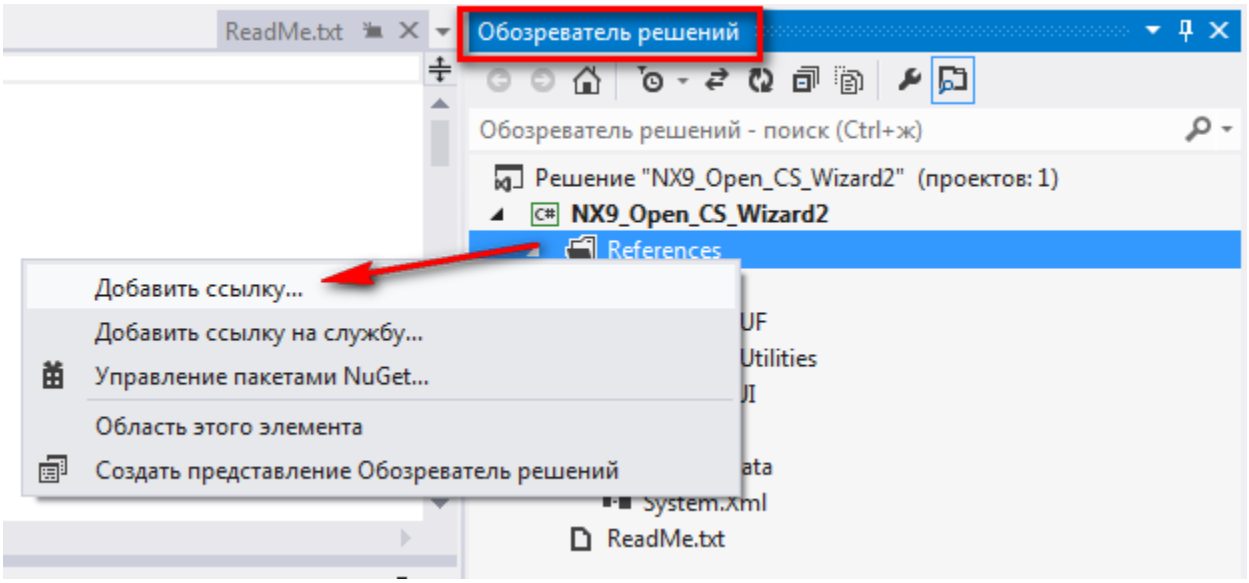


Рисунок 3.8 – Подгруздка библиотек

В директории имени файла формата dlx следует указать путь по типу Диск:\папка, так как созданный код по умолчанию содержит непосредственно пространство имени подгружаемого файла, а не путь на его расположение.

```
public primer()
{
    try
    {
        theSession = Session.GetSession();
        theUI = UI.GetUI();
        theDlxFileName = "C:\\primer\\application\\primer.dlx";
        theDialog = theUI.CreateDialog(theDlxFileName);
        theDialog.AddApplyHandler(new NXOpen.BlockStyler.BlockDialog.Apply(apply_cb));
        theDialog.AddOkHandler(new NXOpen.BlockStyler.BlockDialog.Ok(ok_cb));
        theDialog.AddUpdateHandler(new NXOpen.BlockStyler.BlockDialog.Update(update_cb));
        theDialog.AddInitializeHandler(new NXOpen.BlockStyler.BlockDialog.Initialize(initialize_cb));
        theDialog.AddDialogShownHandler(new NXOpen.BlockStyler.BlockDialog.DialogShown(dialogShown_cb));
    }
    catch (Exception ex)
    {

        //---- Enter your exception handling code here -----
        throw  ex;
```

Рисунок 3.9 – Ссылка на .dlx файл

Нажатием кнопки F6 происходит построение решения или кнопкой решить.

Среди файлов, которые скомпилировались по итогу в папке проекта C:\primer\application\primer\primer\bin\Debug, создастся динамическая библиотека .dll с именем папки проекта. Библиотеку копировать в папку application. Эта динамическая библиотека будет основой нашей программы, загружаемой в операционную систему.

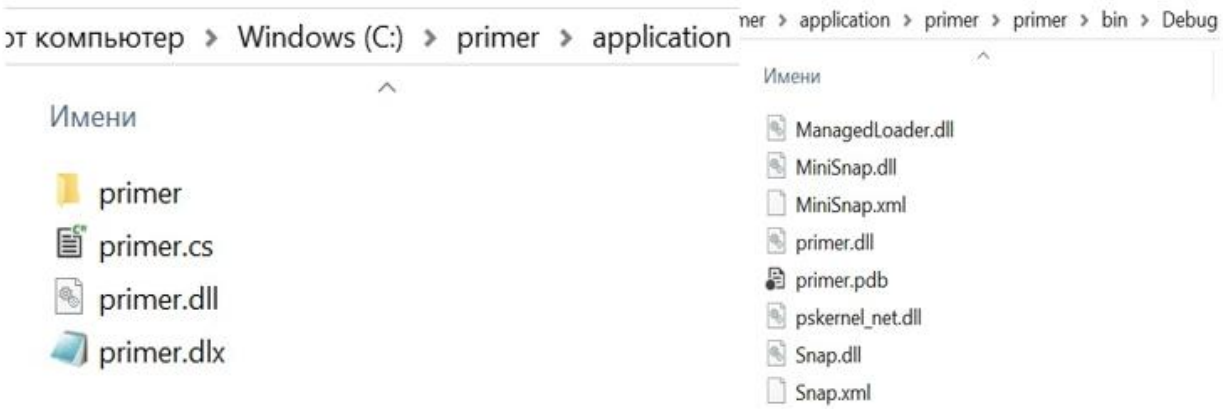


Рисунок 3.10 – Копирование динамической библиотека .dll

3.5 Создание журнала

Разрабатываемый в журнале код создаётся в одном файле и используется при запуске. Требуется создавать дополнительные классы, располагающиеся в том же файле журнала. Может требоваться замена системы .Net к примеру Linq. В этом случае к проекту в Visual Studio добавляется ссылка на dll. В проектах, где классы находятся в разных файлах, приложение используют в виде ссылок, что не практично, поэтому используют скомпилированное приложение. Процедура запуска осуществляется двумя способами:

- Запуск журнала Alt+F8 или Меню – Инструменты журнал – Воспроизведение, выбор типа файла, запуск dll Ctrl+U.
- Файл – Выполнить – NXOpen.

Одним из самых оптимальных способов автоматизации написания кода в NX 9 является запись действий в журнал, с последующим его редактированием или копирования кода, пример приведен на рисунке 3.10.

```
4  using System;
5  using NXOpen;
6
7  public class NXJournal
8  {
9    public static void Main(string[] args)
10   {
11     Session theSession = Session.GetSession();
12     Part workPart = theSession.Parts.Work;
13     Part displayPart = theSession.Parts.Display;
```

Рисунок 3.11 – Структура журнала

Пространства имен System и NXOpen, а также начальные фрагменты нашего журнала NXJournal объявляет, как класс и переменные для сессии, отображаемой и рабочей детали в методе Main.

Что бы записать журнал необходимо перейти в Меню – Инструменты – Журнал – Запись.

Во время записи журнала интерфейс NX приобретает зеленые и желтые маркеры, но записи подлежат не все действия. Возможность полной записи отображается зелеными маркерами. Необходим выбор языка записи в данном случае язык журнала C# и формат файла Unicode Меню – Настройки – Интерфейс пользователя – Инструменты – Журнал. Файл журнала – это текстовый файл открывается текстовом редакторе по типу Notepad++ или Блокнот, оптимально если есть подсветка синтаксиса.

3.6 Объектная модель

TaggedObject или Теговый объект – это базовый класс, описывающий элементы модели NX, такие как линия, вытягивание, рабочая деталь и другие. Класс persistent entities – это объекты, сохраняемые при сохранении измененной модели или завершении работы с NX. Класс TransientObject – это не сохраняемые объекты. Классы содержат свойства и методы и для чтения и установки атрибутов и имен для элементов модели NX. TaggedObjectCollection – это класс для классов в виде коллекции объектов TaggedObject. Пример выбора рабочей детали и детали, отображаемой из коллекции деталей в сессии:

```
NXOpen.Part workPart = theSession.Parts.Work;
NXOpen.Part displayPart = theSession.Parts.Display;
```

Part – это класс детали NX. Элементы модели в NX – это feature их коллекции есть в каждом TaggedObjectCollection внутри объекта класса Part. Коллекции нужны для инициализации объектов класса FeatureBuilder при изменении или создании элементов построения модели. Элементы и объекты редактируются и создаются объектами класса Builder, методом Commit() создается объект. Метод Destroy() следует после завершении работы объекта Builder.

3.7 Создание собственного приложения

Для создания собственного приложения используется модуль разработчика пользовательского интерфейса. Рассмотрение каталога блока и набора функций которые в дальнейшем будут использованы для создания собственного интерфейса. Каталог блока состоит из набора функций конструктора, которые сгруппированы в пять разделов: основной, числа, компоновка, выбор и специальный.

Для начала обзор основного радела конструктора, в него входят раздел, показанные на рисунке 3.12.

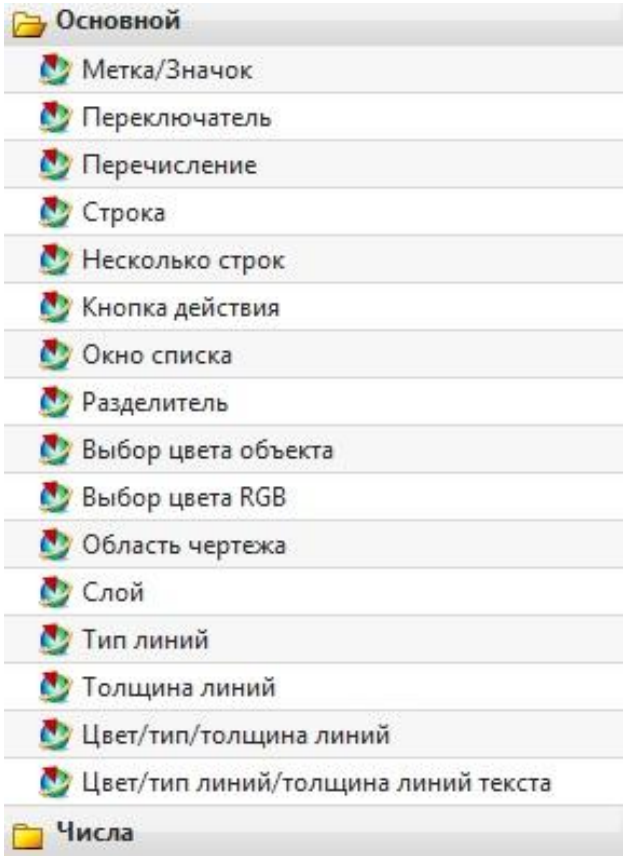


Рисунок 3.12 – Раздел «Основной»

Блок «Кнопка действия» предоставляет элемент управления для инициирования действия для команды. Блок «Область чертежа» предоставляет элемент управления для отображения области в диалоговом окне команды, которая может использоваться для отображения изображений

JPG, BMP, TIFF и PNG. Блок «Перечисление» предоставляет элемент управления для представления перечисляемого списка параметров для команды. Блок «Метка/Значок» предоставляет способ поместить метку или растровое изображение в диалог команды. Он не собирает входные данные для команды. Блок «Строка» многоканальный обеспечивает управление входом на массив строк. Блок «Выбора цвета объекта» предоставляет элемент управления для задания цветов. Этот блок поддерживает 216 цветов и позволяет выбрать более одного цвета. Блок «Выбора цвета RGB» предоставляет элемент управления для ввода цвета RGB. Блок «Разделитель» не собирает входные данные для команды - он позволяет разместить горизонтальный разделитель в диалоговом окне. Это больше связано с внешним видом окна диалога, а не с функциональностью. Блок «Строка» предоставляет элемент управления для ввода одной строки в команду. Блок «Переключатель» предоставляет элемент управления для ввода логического значения (вкл / выкл, да / нет) в команду. Блок «Список» предоставляет элемент управления для ввода выбора из нуля, одного или нескольких элементов в команду. Он может дополнительно отображать следующие кнопки рядом со списком: добавить, удалить, вверх и вниз. Используйте блок «Слой» чтобы указать слой объекта для команды. Блок «Толщина линий» предоставляет элемент управления для представления списка параметров линии ширины для команды. Используйте блок «Тип линий» чтобы указать шрифт строки объекта для команды. Используйте блок «Цвет/тип/толщина линий» чтобы указать цвет линии, шрифт и ширину объекта для команды.

Далее обзор раздела конструктора «Числа» (рис. 3.13).

Рисунок 3.13 – Радел «Числа»

Блок «Угловой размер» предоставляет элемент управления для ввода выражения, размерность которого является угловой. Как таковой, он является частным случаем блока выражения. Блок «Двойной точности» обеспечивает управление входом двойной номер к команде. Блок предоставляет сообщения об ошибках проверки ввода (если пользователь вводит значение вне допустимого диапазона, указанного в свойствах блока Минимальное и Максимальное значение). Блок «Таблица чисел двойной точности» предоставляет элемент управления для размещения нескольких полей с двойной записью в конфигурации таблицы. Поля ввода принимают двойной номер. Этот блок используется, когда есть два или более связанных, но разных цифровых ввода для одного или нескольких параметров. Блок «Выражение» позволяет собирать, изменять и ссылаться на значения и формулы. Выражения позволяют создавать и управлять параметрическими отношениями. Блок «Целое» обеспечивает управление вводом целого числа в команду. Блок предоставляет сообщения об ошибках проверки ввода (если пользователь вводит значение вне допустимого диапазона, указанного в свойствах блока Минимальное и Максимальное значение). Блок «Таблица целых чисел» предоставляет элемент управления для размещения ряда полей ввода в конфигурации таблицы. Поля ввода принимают целое число. Блок «Линейный размер» предоставляет элемент управления для ввода выражения

в команду, размерность которой всегда равна длине линейного расстояния. Блок «Размер по трассе» предоставляет четыре метода для определения местоположения вдоль цепочки кривых, длина дуги, % длина дуги, сквозная точка (необязательно), % параметр. Доступность указанных выше параметров для пользователя определяется свойством «OptionMask». Что бы блок работал правильно, в нем должна быть предусмотрена кривая, которая будет действовать как путь. Этот путь может быть установлен только с помощью автоматизации. Блок «Размер радиуса» предоставляет элемент управления для ввода выражения в команду, размерность которой всегда является радиусом. Помимо отображения в графическом окне, этот блок идентичен блоку линейного измерения.

Далее обзор раздела компоновка конструктора (рис. 3.14)

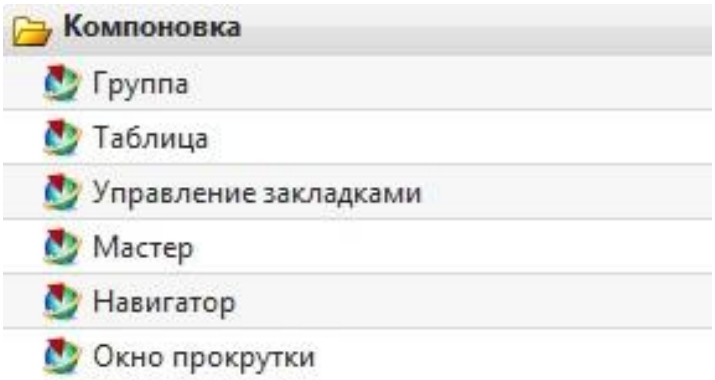


Рисунок 3.14 – Раздел «Компоновка»

Блок компоновки «Группа» обеспечивает элемент управления для организации других блоков в логические группы. В блоке «Окно прокрутки» используются для создания прокручивать область с другим блоком внутри него. Блок «Управление закладками» размещает другие блоки на отдельных страницах с вкладками. Блок «Таблица» предоставляет элемент управления для организации других блоков в диалоге в виде таблицы. Используйте блок «Навигатор» чтобы организовать большой набор входных данных в диалоговом окне для удобной навигации. Также возможность поместить узел навигатора блок в блоке навигатор. В каждом блоке узла навигатора

возможно разместить любой другой блок, доступный в каталоге блоков. Блок макета «Мастер» предоставляет элемент управления для организации других блоков в виде шагов на страницах мастера. Возможно добавить шаги в мастер, используя пункт шаг мастера из каталога блоков. Используйте блок Wizard для команды, если применимы один или несколько из следующих случаев: команда имеет большое количество входов, команда следует за различными рабочими процессами в зависимости от выбора пользователя и настроек, команда создает один или несколько типов объектов в рабочем процессе. Блок мастера также предоставляет навигатор задач для навигации по страницам.

Далее обзор раздела выбора конструктора, в него входят:

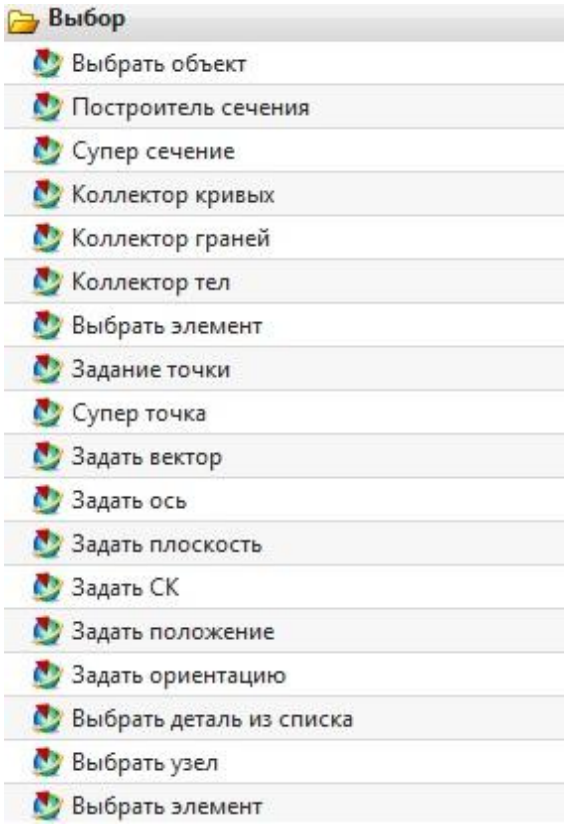


Рисунок 3.15 – Раздел «Выбор»

Блок «Коллектор кривых» используется для выбора одной или нескольких кривых, в том числе цепочки кривых, связанных с концом, с помощью намерения выбора. В этом блоке используется метод выбора кривой, но он не поддерживает параметры правила SI «Остановиться на

пересечении», «Следовать по уклону», «Цепочка между» и «Цепь в элементе». Блок «Коллектор граней» используется для выбора одной или нескольких граней, в том числе цепочки граней, соединенных с ребром, через намерение выбора. Блок «Коллектор тел» используется в блоках, которые требуют ввода одного или нескольких тел. Тела могут быть твердыми телами, или листовыми телами, или обоими. Этот блок поддерживается намерением выбора. Правила намерения выбора, применимые к этим областям блока, следующие: одно тело, особые тела, тела в группе. Блок «Построитель сечения» предоставляет элемент управления для ввода выбора кривой в команду. В этом блоке используется намерение выбора кривой (SI), и оно также включает опции правила SI «Остановиться на пересечении», «Следовать по уклону», «Цепочка между» и «Цепочка внутри объекта». Блок «Выбрать элемента» используется для выбора одной или нескольких функций. Намерение выбора не применимо к этому блоку. Блок «Выбрать объект» предоставляет элемент управления для ввода выбора объекта любого типа в команду. Таким образом, этот блок служит блоком выбора «поймать все» для различных типов объектов. Выбор объект блок поддерживает наложение указать точки блока для размещения с указанием точек, в частности оснастке точки. Этот блок также поддерживает фильтрацию типов объектов через фильтр типов на панели инструментов глобального выбора. По умолчанию все типы объектов установлены в этом блоке. Блок «Задать ось» предоставляет элемент управления для указания оси. Доступ к конструктору векторов и конструктору точек предоставляется. Блок «Задать СК» предоставляет элемент управления для указания СК. Доступ к конструктору СК предоставляется. Блок «Задать положение» предоставляет элемент управления для ввода положения экрана (определяемого расположением курсора) в команду. Указанное местоположение не является ассоциативным, и точечный объект не создается. Привязка точек и конструктор точек не используются. Блок «Задать ориентацию» предоставляет элемент управления для указания ориентации. Блок

отображает триаду осей X, Y, Z в графическом окне и обрабатывает, чтобы изменить начало триады и направление осей. Блок используется в команде для вставки базовых СК с использованием типа динамика. Блок «Задать плоскость» предоставляет элемент управления для определения плоскости. Доступ к конструктору самолетов предоставляется. Этот блок используется для выбора или определения точки. Существующие точки могут быть выбраны или местоположение точки может быть определено с помощью точки привязки. Доступ к конструктору точек также предоставляется. Блок «Задать вектор» предоставляет элемент управления для указания вектора. Доступ к конструктору векторов предоставляется. Блок «Супер сечение» предоставляет элемент управления для ввода выбора кривой в команду или для создания эскиза на лету. В этом блоке используется метод выбора кривой, и он также включает в себя параметры правила SI «Остановиться на пересечении», «Следовать по уклону», «Цепочка между» и «Цепочка внутри объекта». Другими словами, супер сечение - это комбинация блока создание сечения и создание эскиза на лету. Часть этого блока, которая рисует на лету, такая же, как команда эскиз. Команда эскиз получает контроль при использовании эскиза на лету. Блок «Выбрать узел» используется для выбора одного или нескольких узлов. Выбор узла можно сделать в графическом окне или указав метки узла. Блок «Выбрать элемент» используется для выбора одного или нескольких элементов. Выбор элемента может быть сделан в графическом окне или путем указания меток узла.

В заключении обзор на специальный раздел конструктора (рис. 3.16).

Блок «Сменит направление» это блок, который синхронизирует кнопку обратного направления с экранной ручкой. Направление можно изменить, выбрав кнопку в диалоговом окне или дважды щелкнув ручку на сцене. Он может быть использован для обращения к нормали поверхности, изменения направления вектора, изменения направления измерения, изменения направления плоскости и т. д.

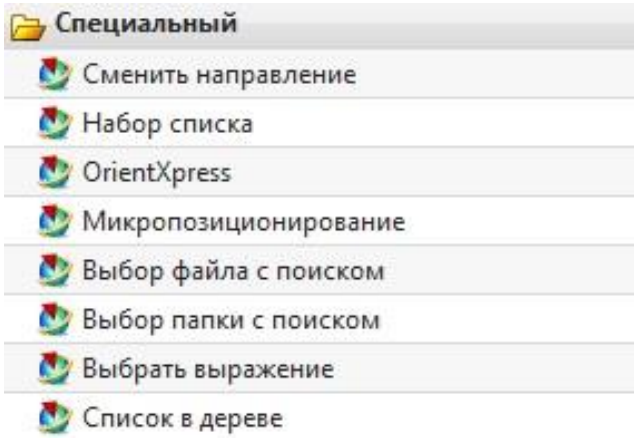


Рисунок 3.16 – Раздел «Специальный»

Блок «Набор списка» является элементом управления для списка наборов блоков. Чтобы использовать набор списка, необходимо указать начальный блок. Начальный блок - это блок, который содержит другие блоки. Начальный блок указывается с использованием файла dlx, который называется SeedDlxFile. Каждый элемент в списке является клоном начального блока, но с различными настройками свойств блоков внутри каждого элемента списка. Каждый элемент в этом блоке соответствует экземпляру начального блока. Когда элемент выбран или отменен, экземпляр начального блока, соответствующий этому элементу, получает или теряет фокус, вызывая обратный вызов FocusNotify. Экземпляр начального блока, полученный в этом обратном вызове, может быть приведен к объекту класса CompositeBlock. Блок «выбрать выражение» предоставляет элемент управления для ввода выражения в команды. Выражение может быть тем, которое уже существует в детали, или оно может быть создано на лету. OrientXpress - это вспомогательный помощник для выбора оси или плоскости, параллельной оси или плоскости системы координат. Взаимодействие с OrientXpress в основном происходит через элементы управления в графическом окне. Блок «Микропозиционирование» предоставляет элемент управления для определения чувствительности или тонкости движений при перетаскивании маркера в графическом окне. На все дескрипторы, включенные приложением, влияет использование микро

позиции. Блок обеспечивает шкалу, значение которой уменьшает общую скорость перемещения ручки. Микро позиция игнорируется, когда ручка перетаскивается к месту во время перетаскивания. Блок «Выбор файла с поиском» предоставляет элемент управления для ввода только имени файла детали или просмотра его с помощью стандартного диалогового окна выбора собственных файлов. Блок «Выбор папки с поиском» предоставляет элемент управления для ввода имени папки или просмотра его с помощью стандартного диалогового окна выбора папки. Блок «Список в дереве» создает древовидную структуру, представляющую иерархию узлов. Возможно добавить узлы и столбцы в список деревьев и назначить действия обратного вызова для событий дерева и узлов. Используйте этот блок, когда требуется древовидная структура, такая как в навигаторе деталей. Следующие классы помогают в создании блока:
Класс дерева - NXOpen.BlockStyler.Tree.
Класс узла - NXOpen.BlockStyler.Node.
Класс меню древовидного списка - NXOpen.BlockStyler.TreeListMenu.
Класс формы данных - NXOpen.DataContainer.

3.8 Выводы раздела

Все поставленные задачи решались с помощью виртуальной среды, разработанной Visual Studio с подключенными библиотеками NX, поскольку использовался язык программирования C#, а соответствующей литературы о связи NX и Visual Studio на русском языке практически нет, то были разработаны методические указания о совместимости работы NX и Visual Studio.

4. Применение разработанного приложения для построения штампа первой вытяжки изделия

4.1 Интерфейс приложения

В данном разделе рассматривается интерфейс разработанного приложения, его функциональные особенности и нюансы написания исходного кода для осуществления поставленных задач.

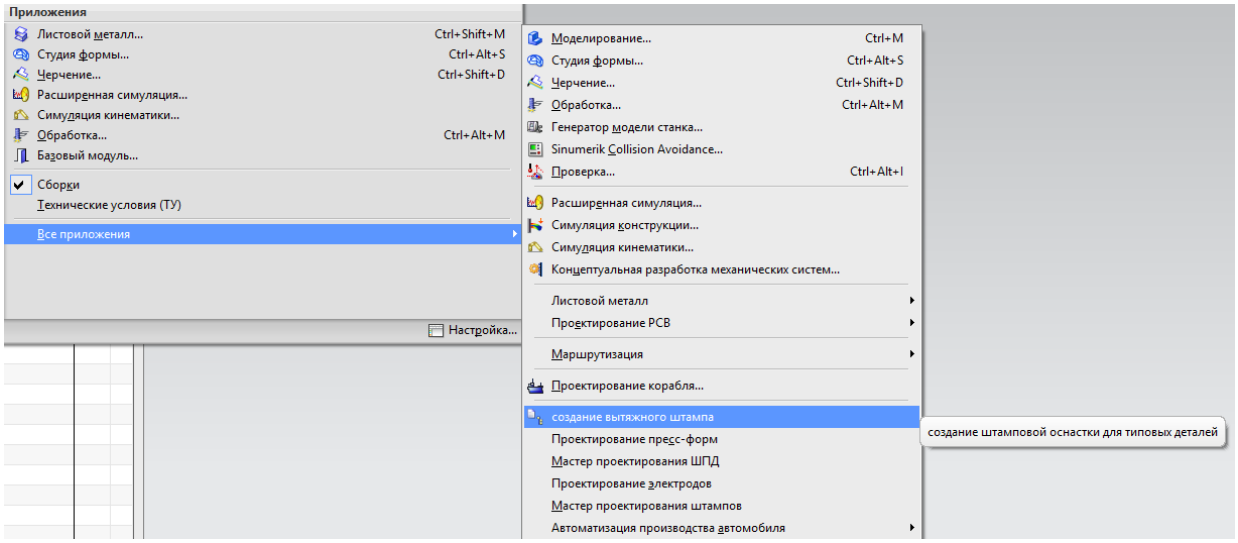


Рисунок 4.1 – Кнопка разработанного приложения

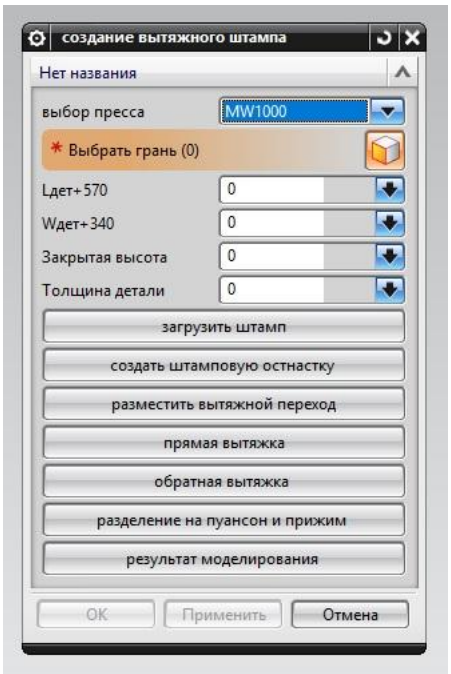


Рисунок 4.2 – Интерфейс разработанного приложения

Программный продукт носит название «Создание вытяжного штампа», реализация происходит на основе вытяжного перехода детали, загружаемой пользователем.

Для начала рассматривается выпадающая строка выбор пресса. В данном выпадающем списке несколько вытяжных штампов MW отличающиеся усилиями.

После выбора пресса следует элемент интерфейса выбрать грань, который позволяет выбрать деталь, загруженную пользователем.

```
group0 = (NXOpen.BlockStyler.Group)theDialog.TopBlock.FindBlock("group0");
enum0 = (NXOpen.BlockStyler.Enumeration)theDialog.TopBlock.FindBlock("enum0");
face_select0 = (NXOpen.BlockStyler.FaceCollector)theDialog.TopBlock.FindBlock("face_select0");
```

Рисунок 4.3 – Фрагмент кода, отображение строк названия, выбора пресса, выбора грани интерфейса разработанного приложения в коде C#

Далее производится параметризация штамповой оснастки согласно габаритам детали, загруженной пользователей, и данными необходимыми под конкретный техпроцесс. Длина штампа соответствует длине вытяжного перехода с прибавлением величины минимального зазора от него до крайней грани компонента «plate» равной 570 мм, ширина штампа соответствует длине вытяжного перехода и величины минимального зазора от него до крайней грани компонента «holder», равной 340 мм. Превышение данных размеров не допустимо, так как иначе вытяжной переход будет превышать размеры оснастки. Деталь изначальна не параметризована выражениями. Данная параметризация происходит за счёт изменения параметров выражения в файле «Conceptual». Было принято решение, что размер будет задаваться в файле концептуальной модели и наследоваться деталями штампа.

```
expression0 = (NXOpen.BlockStyler.ExpressionBlock)theDialog.TopBlock.FindBlock("expression0");
expression01 = (NXOpen.BlockStyler.ExpressionBlock)theDialog.TopBlock.FindBlock("expression01");
expression02 = (NXOpen.BlockStyler.ExpressionBlock)theDialog.TopBlock.FindBlock("expression02");
expression03 = (NXOpen.BlockStyler.ExpressionBlock)theDialog.TopBlock.FindBlock("expression03");
```

Рисунок 4.4 – Фрагмент кода, отображение строк выражения интерфейса разработанного приложения в коде C#

```
double a;
a = (double)expression0.Value;
string x = Convert.ToString(a);

Expression expression1 = (Expression)workPart.Expressions.FindObject("l_pr");
Unit unit1 = (Unit)workPart.UnitCollection.FindObject("MilliMeter");
workPart.Expressions.EditWithUnits(expression1, unit1, x);

double b;
b = (double)expression01.Value;
string y = Convert.ToString(b);

Expression expression2 = (Expression)workPart.Expressions.FindObject("w_pr");
workPart.Expressions.EditWithUnits(expression2, unit1, y);

double c;
c = (double)expression02.Value;
string z = Convert.ToString(c);

Expression expression3 = (Expression)workPart.Expressions.FindObject("H_r");
workPart.Expressions.EditWithUnits(expression3, unit1, z);
```

Рисунок 4.5 – Фрагмент кода, параметризованные строки выражения интерфейса разработанного приложения в коде C#

4.2 Операция «загрузить штамп»

Далее рассматривается процесс написания исходного кода для загрузки штамповой оснастки сборки и последующего назначения рабочей деталью концептуальную модели штампа в проекте. Для начала добавляется компонент в существующую сборку модели загруженной ранее пользователем, а именно головой файла сборки штампа «Stamp_assy».

```
NXOpen.Session.UndoMarkId markId1;
markId1 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Visible, "Add Component");
```

Рисунок 4.6 – Фрагмент кода, добавление компонента в сборку

```
BasePart basePart1;
PartLoadStatus partLoadStatus1;
basePart1 = theSession.Parts.OpenBase("C:\\knopka\\scht4\\Stamp_assy.prt", out partLoadStatus1);
```

Рисунок 4.7 – Фрагмент кода, загрузка файла сборки штампа «Stamp_assy»

После чего необходимо сделать рабочим файл «Conceptual». Параметризованные выражения будут изменяться сначала в нем, а потом ссылаться в головной файл сборки, который в свою очередь ссылается на файл конкретной модели.

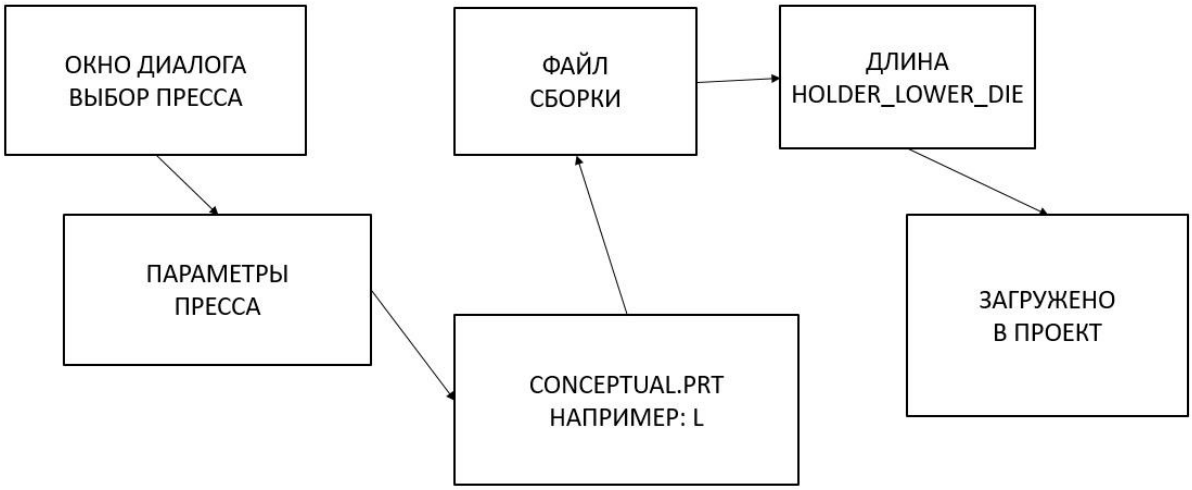


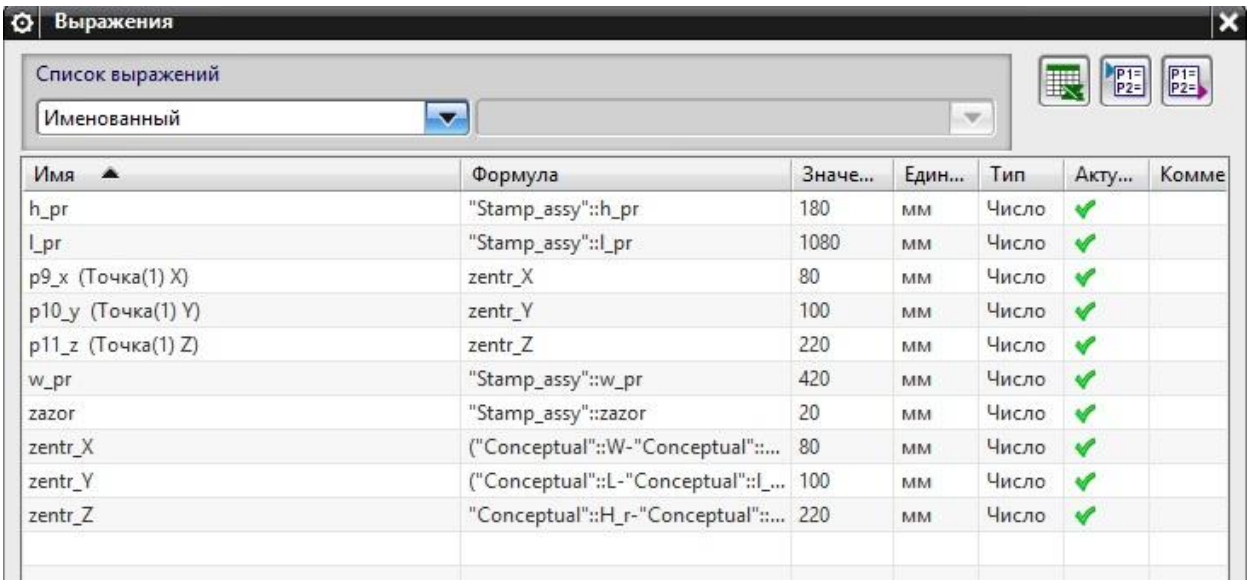
Рисунок 4.8 – Параметрическая схема штампа

```
NXOpen.Assemblies.Component component2 = (NXOpen.Assemblies.Component)component1.FindObject("COMPONENT Conceptual 1");
PartLoadStatus partLoadStatus3;
theSession.Parts.SetWorkComponent(component2, out partLoadStatus3);

workPart = theSession.Parts.Work;
partLoadStatus3.Dispose();
theSession.SetUndoMarkName(markId9, "Make Work Part");
```

Рисунок 4.9 – Фрагмент кода, назначение «Conceptual» рабочим компонентом

Так же что бы предотвратить смещения компонентов относительно центра по осям x, y, z были созданы точки привязки «zentr». Это один из примеров как дорабатывалась высоко параметризованная модель вытяжного штампа 2017 года, на основе которой производилась автоматизация процесса моделирования элементов штампов.



Имя	Формула	Значе...	Един...	Тип	Акту...	Комме
h_pr	"Stamp_assy"::h_pr	180	мм	Число	✔	
l_pr	"Stamp_assy"::l_pr	1080	мм	Число	✔	
p9_x (Точка(1) X)	zentr_X	80	мм	Число	✔	
p10_y (Точка(1) Y)	zentr_Y	100	мм	Число	✔	
p11_z (Точка(1) Z)	zentr_Z	220	мм	Число	✔	
w_pr	"Stamp_assy"::w_pr	420	мм	Число	✔	
zazor	"Stamp_assy"::zazor	20	мм	Число	✔	
zentr_X	("Conceptual"::W-"Conceptual"::...	80	мм	Число	✔	
zentr_Y	("Conceptual"::L-"Conceptual"::l_...	100	мм	Число	✔	
zentr_Z	"Conceptual"::H_r-"Conceptual"::...	220	мм	Число	✔	

Рисунок 4.10 – Выражения для дополнительная параметризация модели штампа

Далее происходит изменение относительно введённых пользователем данных в строках выражений и обновление всех взаимосвязанных элементов, по завершению всех изменений головной файл сборки снова становиться рабочим. Далее результатом действия первой кнопки является параметризованный и смоделированный компонент сборки «Conceptual».

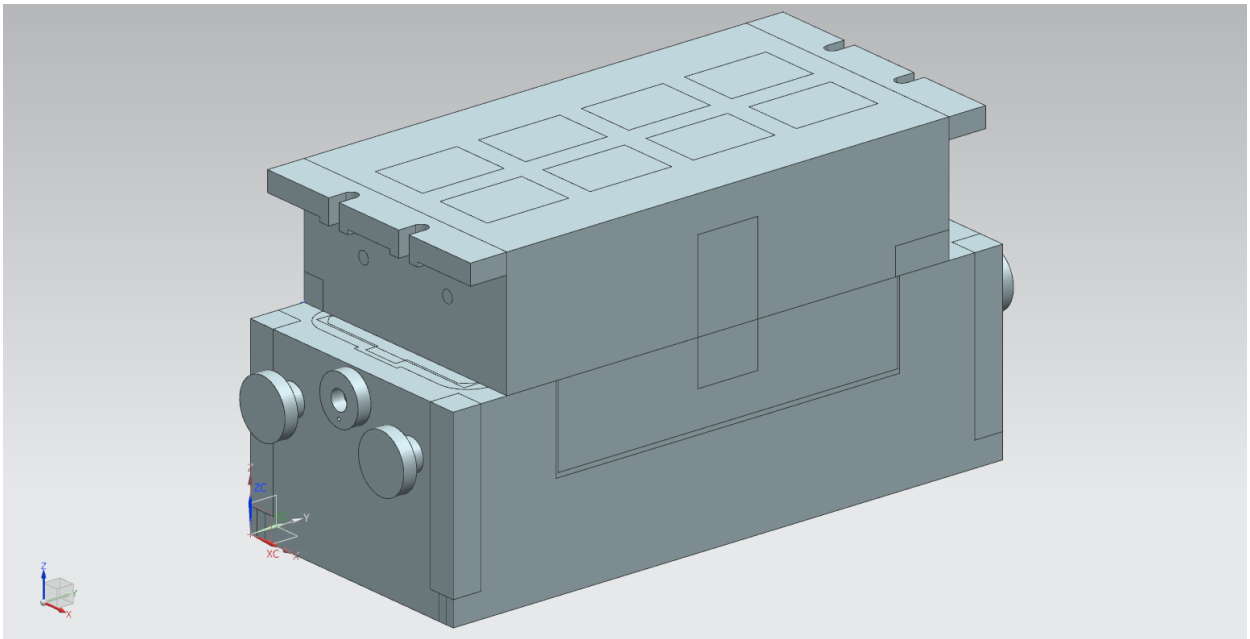


Рисунок 4.11 – Результат нажатия кнопки «загрузить штамп»

4.3 Операция «создать штамповую оснастку»

При нажатии данной кнопки происходит объединение и вычитание компонентов в ходящих в состав нижней плиты, пуансона-прижима и матрицы.

Для того что бы создать полость для установки в плиту прижима, компонент детали делается рабочим и производится ассоциативное копирование элемента «Листовое тело», которое является ограничением объема для блока прижима, при помощи операции «Редактор геометрических связей WAVE».

```
theSession.SetUndoMarkName(markId4, "Диалоговое окно Редактор геометрических связей WAVE");
extractFaceBuilder1.Associative = true;
extractFaceBuilder1.MakePositionIndependent = false;
extractFaceBuilder1.FixAtCurrentTimestamp = false;
extractFaceBuilder1.HideOriginal = false;
extractFaceBuilder1.InheritDisplayProperties = false;
ScCollector scCollector1;
scCollector1 = extractFaceBuilder1.ExtractBodyCollector;
extractFaceBuilder1.CopyThreads = true;
Body[] bodies1 = new Body[1];
```

Рисунок 4.12 – Фрагмент кода, сохранение позиции и размеров копируемого элемента

```
NXOpen.Assemblies.Component component3 =
(NXOpen.Assemblies.Component)component1.FindObject("COMPONENT Conceptual 1");
Body body1 = (Body)component3.FindObject("PROTO#.Bodies|NSIDED_SURFACE(241)");
bodies1[0] = body1;
BodyDumbRule bodyDumbRule1;
bodyDumbRule1 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies1, true);
SelectionIntentRule[] rules1 = new SelectionIntentRule[1];
rules1[0] = bodyDumbRule1;
scCollector1.ReplaceRules(rules1, false);
```

Рисунок 4.13 – Фрагмент кода, выбор в сборке компонента «Conceptual» и тела для ассоциативного копирования данного листового тела в компонент «plate»

Данное листовое тело создавалось в сборке с той целью что геометрия пуансона-прижима в результате вычитания не может сформировать

окончательный вид компонента «нижняя плита» даже с учетом всех последующих булевых операций, производимых над данной моделью.

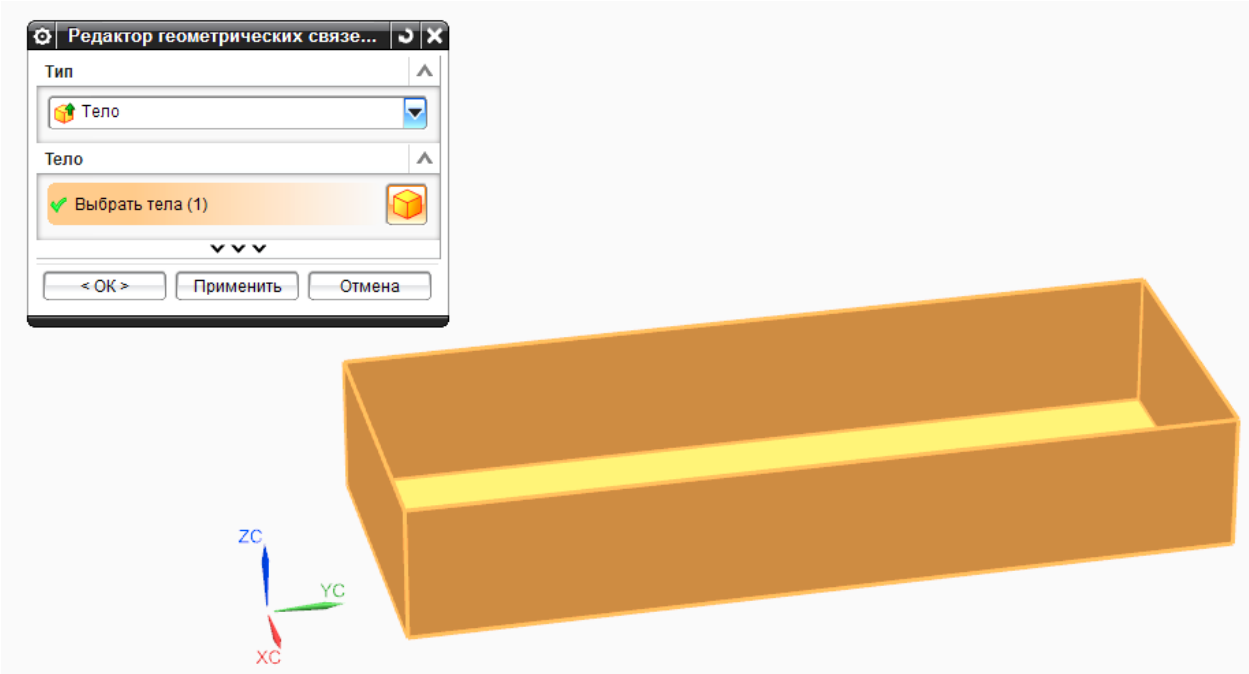


Рисунок 4.14 – Ассоциативное копирование элемента «листовое тело» в файл компонента «plate»

Далее относительно блока используется операцию «Обрезка тела», в качестве инструмента использована «ассоциативная копия» листового тела.

```
theSession.SetUndoMarkName(markId7, "Диалоговое окно Обрезка тела");
ScCollector scCollector2;
scCollector2 = workPart.ScCollectors.CreateCollector();
Body[] bodies2 = new Body[1];
Body body2 = (Body)workPart.Bodies.FindObject("BLOCK(1)");
bodies2[0] = body2;
BodyDumbRule bodyDumbRule2;
bodyDumbRule2 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies2, true);
SelectionIntentRule[] rules2 = new SelectionIntentRule[1];
rules2[0] = bodyDumbRule2;
scCollector2.ReplaceRules(rules2, false);
trimBody2Builder1.TargetBodyCollector = scCollector2;
Body body3 = (Body)workPart.Bodies.FindObject("LINKED_BODY(2)");
FaceBodyRule faceBodyRule1;
theSession.SetUndoMarkName(markId7, "Обрезка тела");
trimBody2Builder1.Destroy();
```

Рисунок 4.15 – Фрагмент кода, блок компонента «plate» обрезается листовым ассоциативно скопированным листовым телом.

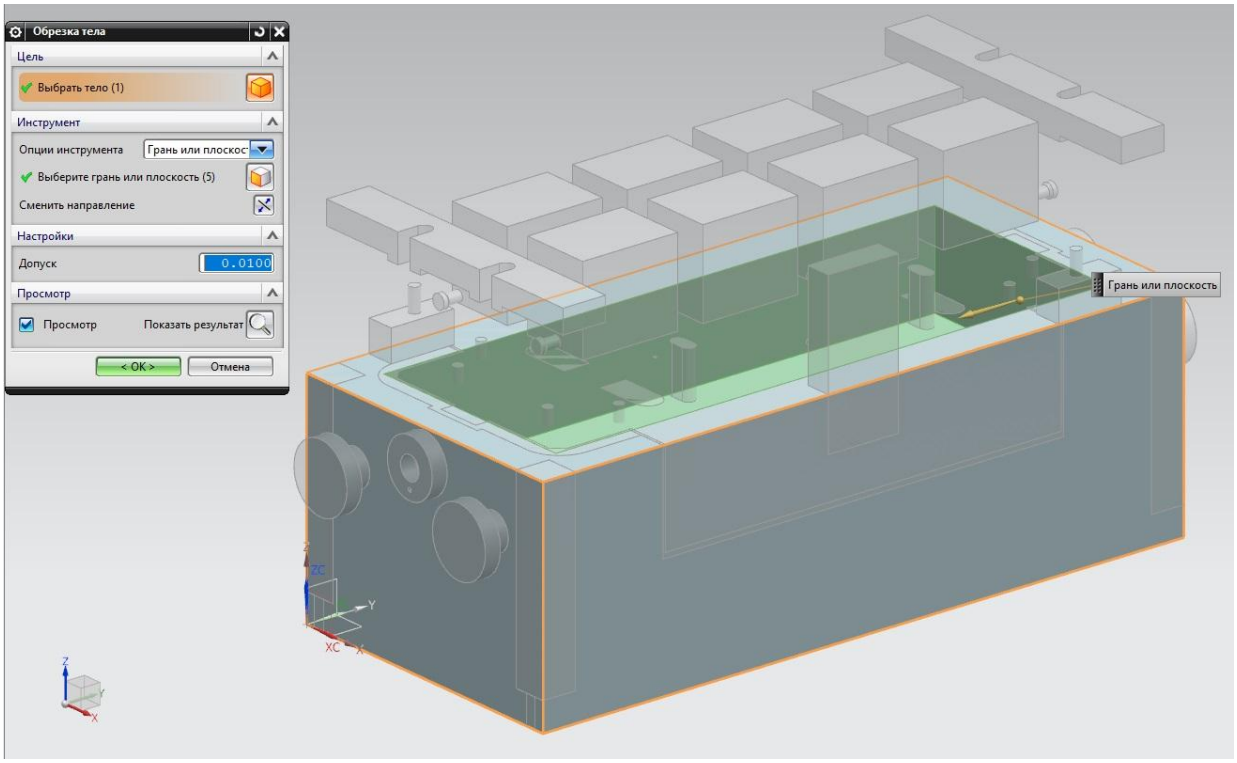


Рисунок 4.16 – операция «Обрезка тела» относительно модели нижней

Следующем шагом производится операция «вычитание». Для этого в файл модели выбираются инструменты, которыми будет произведено вычитание. К ним относятся элементы, входящие в состав концептуальной модели, например, элементы, служащие направляющими для прижима, которые были объединены с зазорами, находящимися между площадкой паза ограничителя хода и плитой. Выбирается так же площадка для паза ограничителя хода, необходимые отверстия служащие для установки ограничителей закрытой высоты и соответственно отверстия для установки данных ограничителей хода, элементы необходимые для удаления изделия загруженного пользователем с поверхности прижима, элементы крепящие нижнюю плиту, а также и элементы, приводящие прижим в движение, боковины данного прижима, конструктивно предусмотренные зазоры между прижимом и нижней плитой. Так же эта операция показана в виде кода, но только объявление данной операции «вычитание» в начале и присоединение последнего из компонентов, входящих в состав данной булевой операции.

```
theSession.SetUndoMarkName(markId10, "Диалоговое окно Вычитание");
bool added1;
added1 = booleanBuilder1.Targets.Add(body2);
TaggedObject[] targets1 = new TaggedObject[1];
targets1[0] = body2;
```

Рисунок 4.17 – Фрагмент кода, инициализации булевой операции «вычитание».

```
SelectionIntentRule[] rules101 = new SelectionIntentRule[1];
rules101[0] = bodyDumbRule100;
scCollector5.ReplaceRules(rules101, false);
Body[] bodies101 = new Body[33];
bodies101[0] = body5;
bodies101[1] = body7;
bodies101[2] = body9;
bodies101[3] = body11;
bodies101[4] = body13;
bodies101[5] = body15;
bodies101[6] = body17;
bodies101[7] = body19;
bodies101[8] = body21;
bodies101[9] = body23;
bodies101[10] = body25;
bodies101[11] = body27;
bodies101[12] = body29;
bodies101[13] = body31;
bodies101[14] = body33;
bodies101[15] = body35;
bodies101[16] = body37;
bodies101[17] = body39;
bodies101[18] = body41;
bodies101[19] = body43;
bodies101[20] = body45;
bodies101[21] = body47;
bodies101[22] = body49;
bodies101[23] = body51;
bodies101[24] = body53;
bodies101[25] = body55;
bodies101[26] = body57;
bodies101[27] = body59;
bodies101[28] = body61;
bodies101[29] = body63;
bodies101[30] = body65;
bodies101[31] = body67;
bodies101[32] = body69;
BodyDumbRule bodyDumbRule101;
bodyDumbRule101 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies101, true);
```

Рисунок 4.18 – Фрагмент кода, последовательного выбора тел для булевой операции «вычитание».

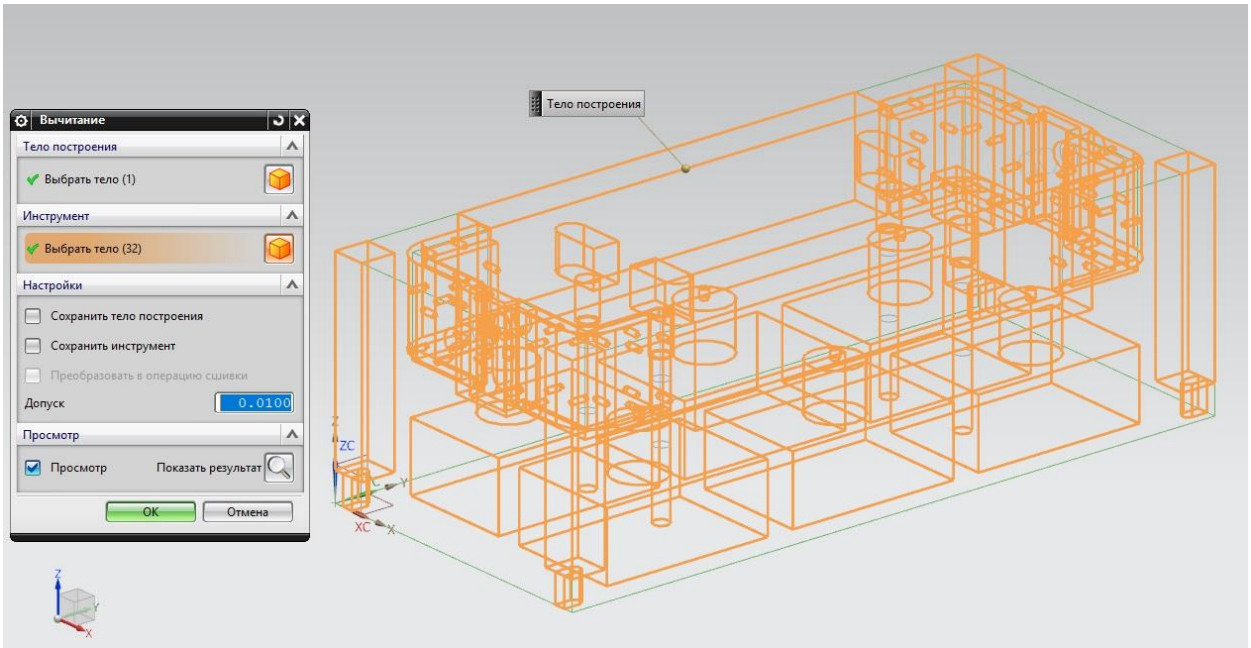


Рисунок 4.19 – Элементы для выполнения операции «вычитание» компонента «plate»

Для следующей операции модель нижней плиты необходимо дополнить приливами с отверстиями под ограничители хода, а также транспортными приливами. Объединение элементов с телом плиты. Так же эта операция показана в виде кода, но только объявление данной операции «объединение» в начале и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId46, "Диалоговое окно Объединение");
bool added2;
added2 = booleanBuilder2.Targets.Add(body2);
TaggedObject[] targets35 = new TaggedObject[1];
targets35[0] = body2;
```

Рисунок 4.20 – Фрагмент кода, инициализация операции «объединение»

Так как ранее были рассмотрены объявления операций «объединение» и «вычитание», то в дальнейшем они не будут рассмотрены в виде кода, являясь идентичными по структуре, записанной инструментом разработки, таким как журнал.

```
SelectionIntentRule[] rules119 = new SelectionIntentRule[1];
rules119[0] = bodyDumbRule118;
scCollector40.ReplaceRules(rules119, false);
Body[] bodies119 = new Body[6];
bodies119[0] = body71;
bodies119[1] = body73;
bodies119[2] = body75;
bodies119[3] = body77;
bodies119[4] = body79;
bodies119[5] = body81;
BodyDumbRule bodyDumbRule119;
bodyDumbRule119 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies119, true);
```

Рисунок 4.21 – Фрагмент кода, последовательный выбора тел для операции «объединение»

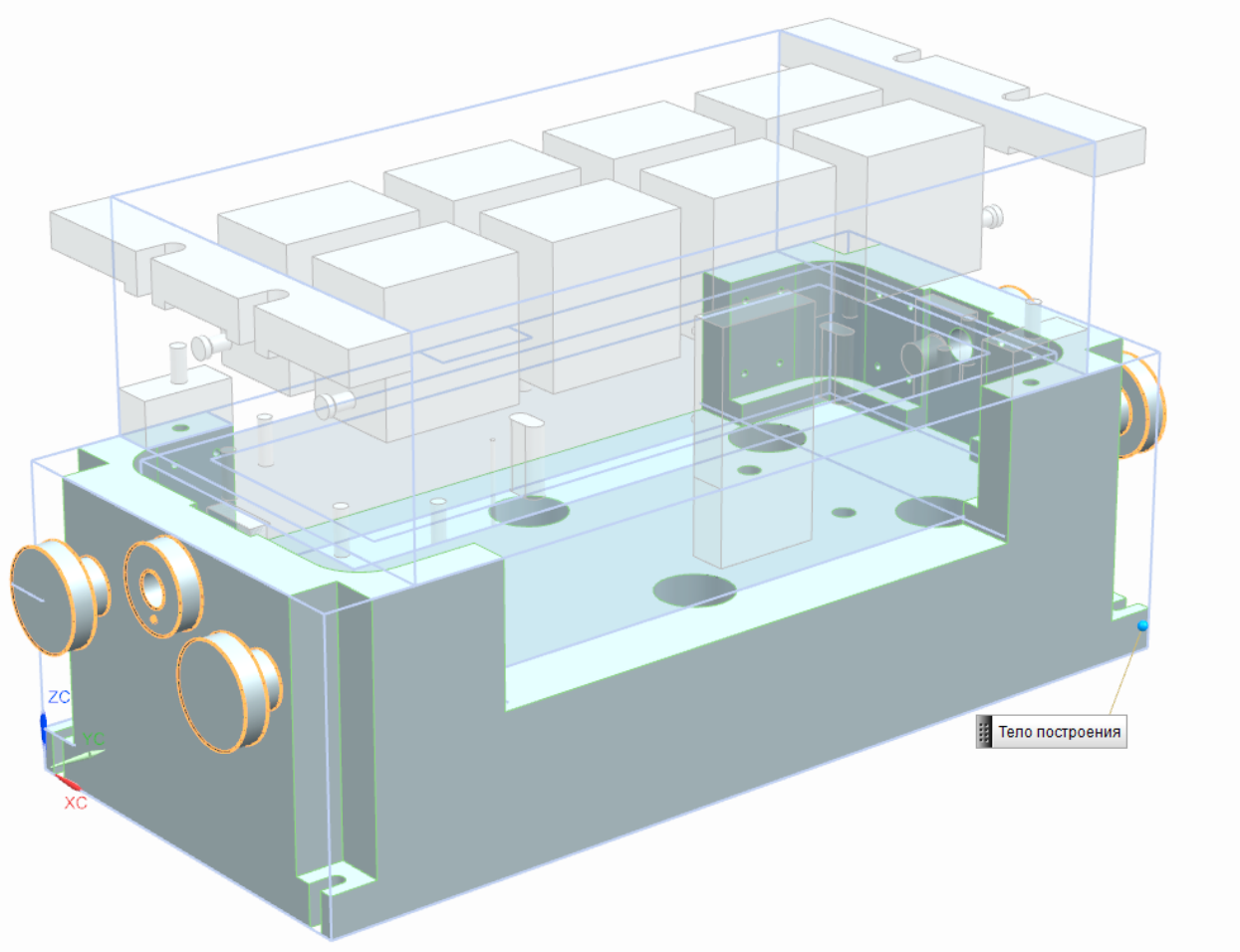


Рисунок 4.22 – операция «Объединение» относительно модели нижней плиты

По итогу четырех последовательных операций таких как связанное тело, обрезка тела, вычитание и объединение, получать готовую

твердотельную нижнюю плиту, носящую имя компонента «plate» в сборке. Показан результат моделирования нижней плиты на рисунке 4.23.

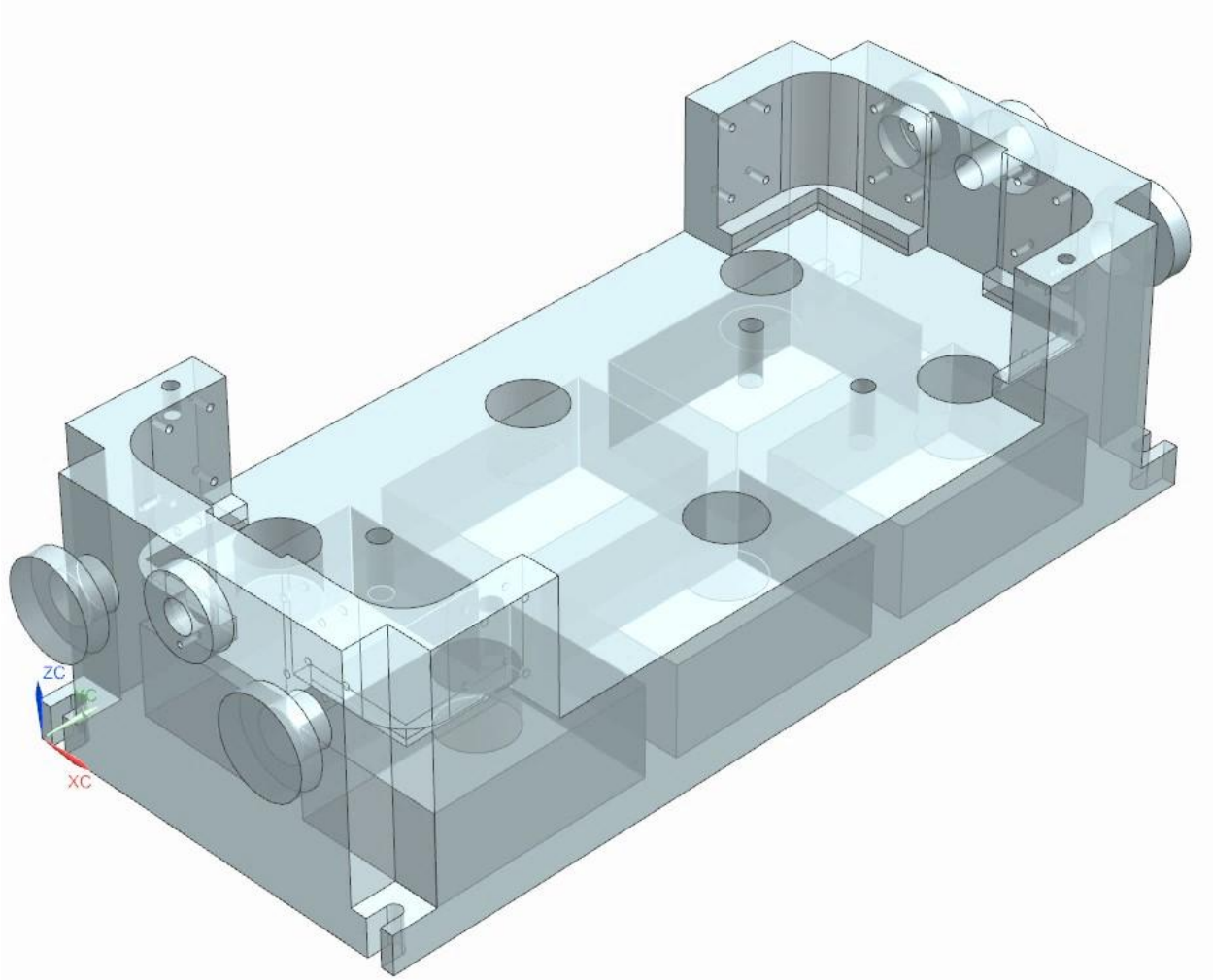


Рисунок 4.23 – Результат моделирования нижней плиты

Далее для построения модели прижима-пуансона осуществляется операция «вычитание». Для этого в файл модели выбираются инструменты, которыми будет произведено вычитание. Это элементы, входящие в состав концептуальной модели как элементы, служащие направляющими для прижима, которые были объединены с зазорами между площадкой паза ограничителя хода и плитой, карманы, созданные под ловители, находящиеся у заготовки по наружному контуру, элементы, задействованные для удаления с поверхности прижима изделия, элементы прижима, приводящие его в движение, выходные отверстия под воздух. Так же эта операция показана в виде кода, но только присоединение последнего из компонентов, входящих в

состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
SelectionIntentRule[] rules177 = new SelectionIntentRule[1];
rules177[0] = bodyDumbRule176;
scCollector48.ReplaceRules(rules177, false);
Body[] bodies177 = new Body[18];
bodies177[0] = body83;
bodies177[1] = body84;
bodies177[2] = body85;
bodies177[3] = body87;
bodies177[4] = body89;
bodies177[5] = body90;
bodies177[6] = body92;
bodies177[7] = body96;
bodies177[8] = body98;
bodies177[9] = body99;
bodies177[10] = body101;
bodies177[11] = body103;
bodies177[12] = body104;
bodies177[13] = body106;
bodies177[14] = body108;
bodies177[15] = body110;
bodies177[16] = body112;
bodies177[17] = body114;
BodyDumbRule bodyDumbRule177;
bodyDumbRule177 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies177, true);
```

Рисунок 4.24 – Фрагмент кода, последовательный выбора тел для булевой операции «вычитание»

Производится поиск функций и методов класса типа объект NX для автоматического выбора указанных элементов для булевой операции «вычитание». Журнал дополняет код приложения соответствующими строками и обеспечивает связь между геометрией инструмента и выбранного тела на уровне сквозного моделирования в сборке.

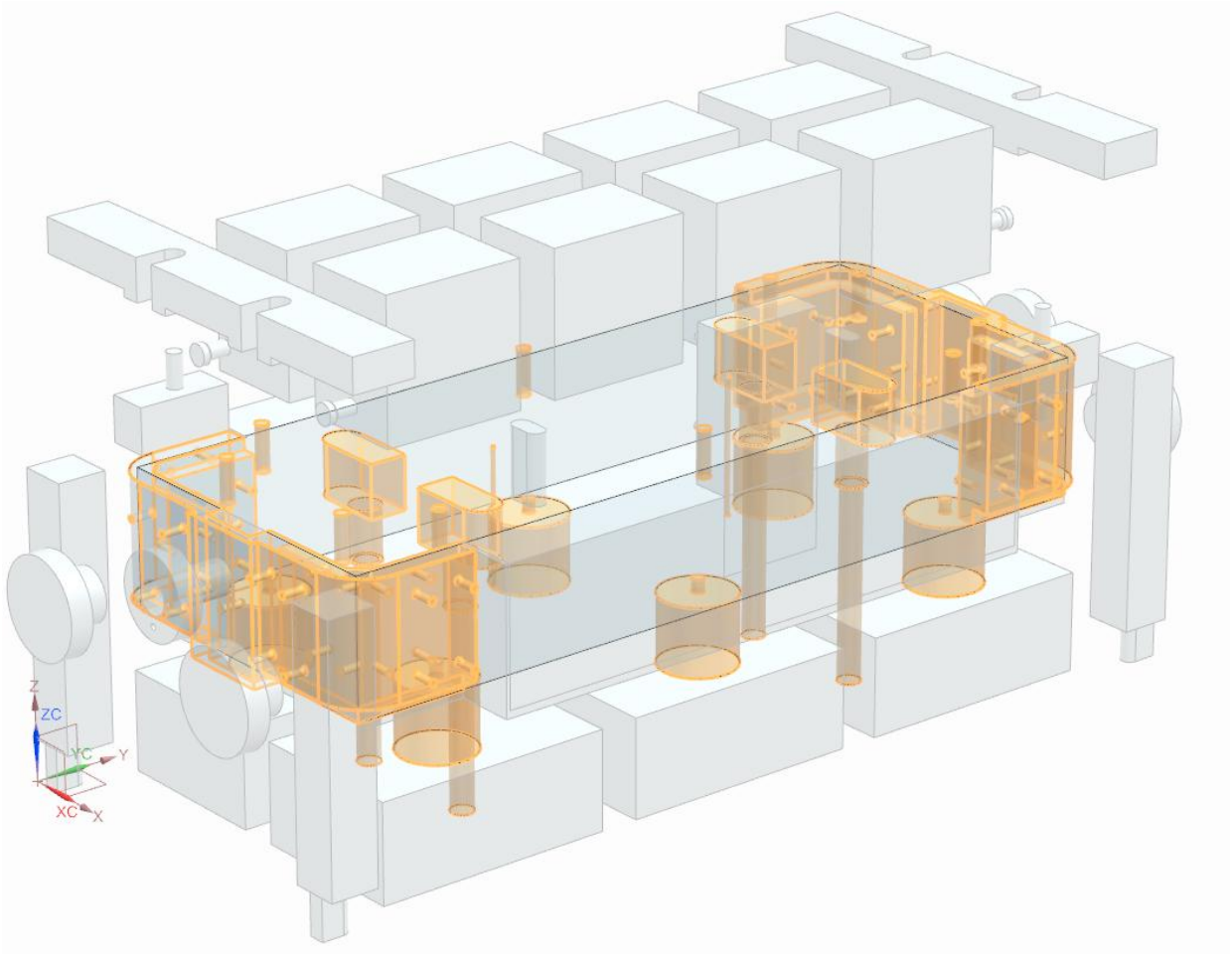


Рисунок 4.25 – Первая операция «вычитание» для компонента «holder_lower_die»

Следующим шагом будет выбор инструментов, которыми будет произведено объединение. Это элементы, входящие в состав концептуальной модели как элементы выступы, в состав которых входят приливы-площадки служащие для направляющего паза ограничителя хода, а также боковины прижима. Так же эта операция показана в виде кода, но только присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
SelectionIntentRule[] rules189 = new SelectionIntentRule[1];
rules189[0] = bodyDumbRule188;
scCollector69.ReplaceRules(rules189, false);
Body[] bodies189 = new Body[4];
bodies189[0] = body115;
bodies189[1] = body116;
bodies189[2] = body117;
bodies189[3] = body118;
BodyDumbRule bodyDumbRule189;
bodyDumbRule189 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies189, true);
```

Рисунок 4.26 – Фрагмент кода, последовательный выбора тел для булевой операции «объединение»

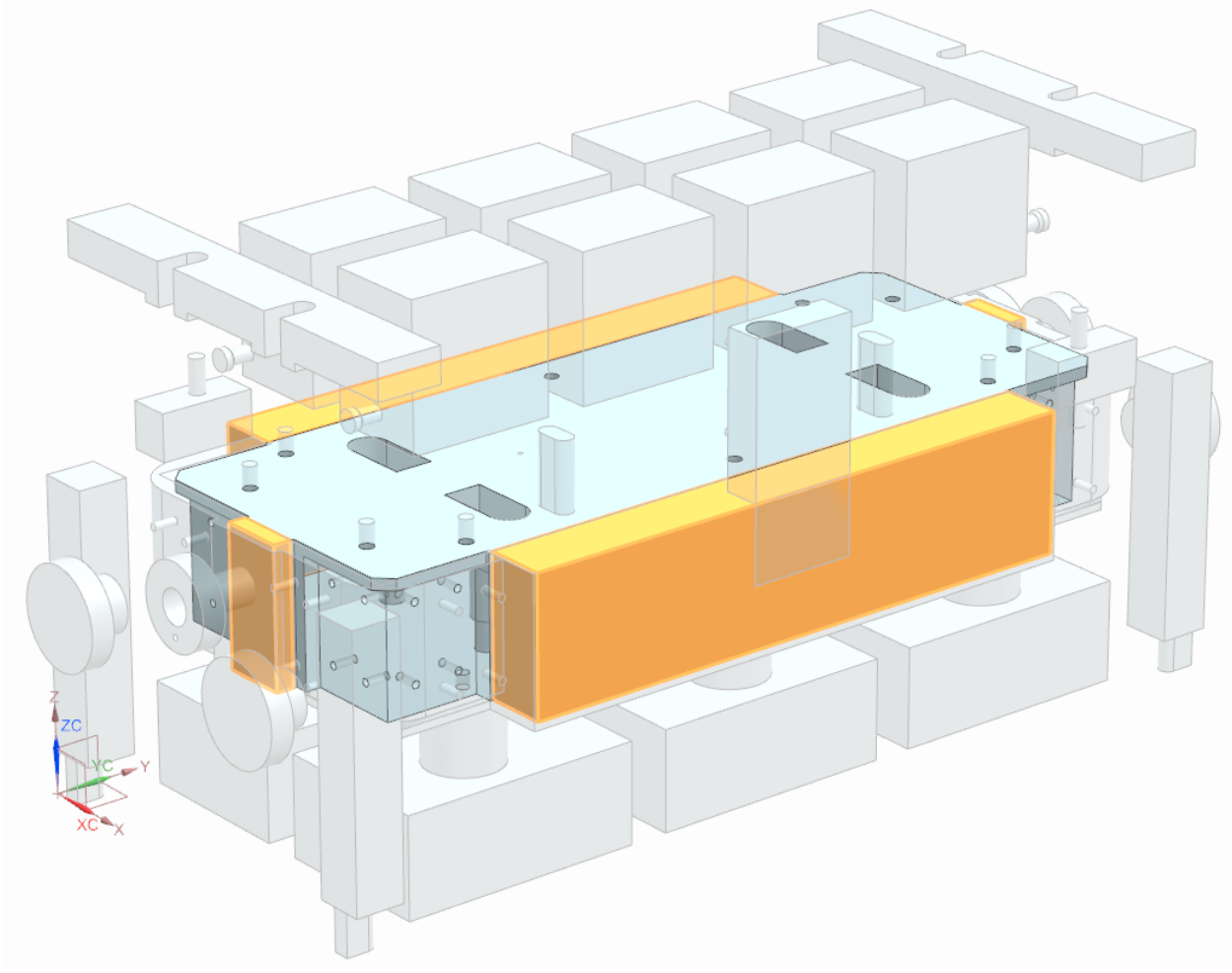


Рисунок 4.27 - Операция «объединение» для компонента «holder_lower_die»

Заключительным шагом для модели пуансона-прижима будет выбор инструментов, которыми будет произведено вычитание. Это элементы, входящие в состав концептуальной модели как верха штампа карманы направляющих, направляющие пазы созданные под ограничитель хода частей штампа подвижных, имеющие отверстия для запирания прижима. Так же эта операция показана в виде кода, но только присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
SelectionIntentRule[] rules207 = new SelectionIntentRule[1];
rules207[0] = bodyDumbRule206;
scCollector75.ReplaceRules(rules207, false);
Body[] bodies207 = new Body[6];
bodies207[0] = body120;
bodies207[1] = body121;
bodies207[2] = body123;
bodies207[3] = body124;
bodies207[4] = body125;
bodies207[5] = body127;
BodyDumbRule bodyDumbRule207;
bodyDumbRule207 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies207, true);
```

Рисунок 4.28 – Фрагмент кода последовательного выбора тел для булевой операции «вычитание»

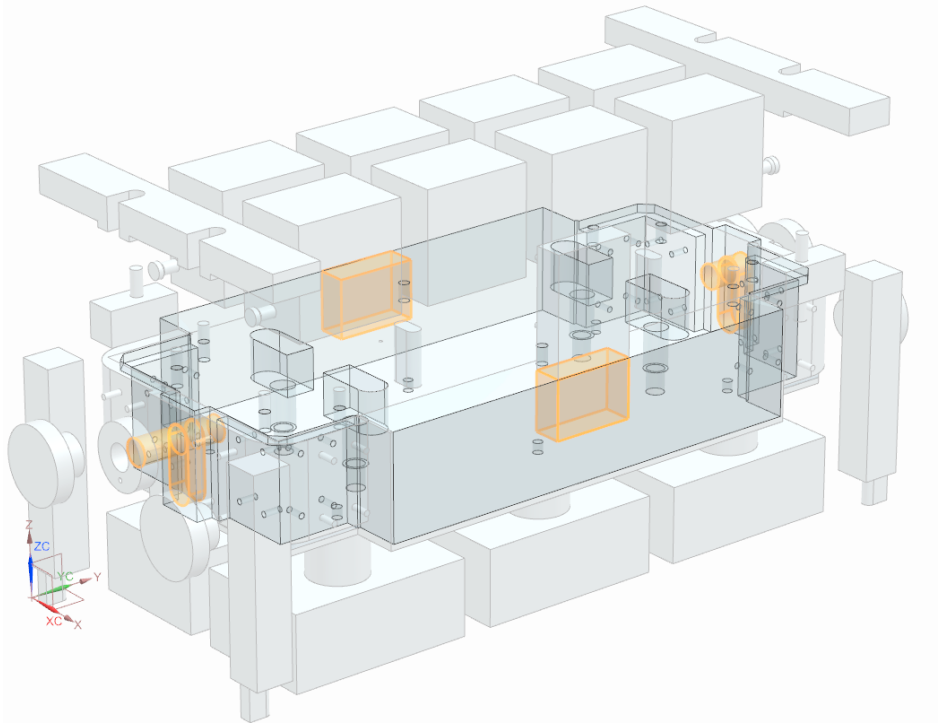


Рисунок 4.29 – Операция «вычитания» для компонента «holder_lower_die»

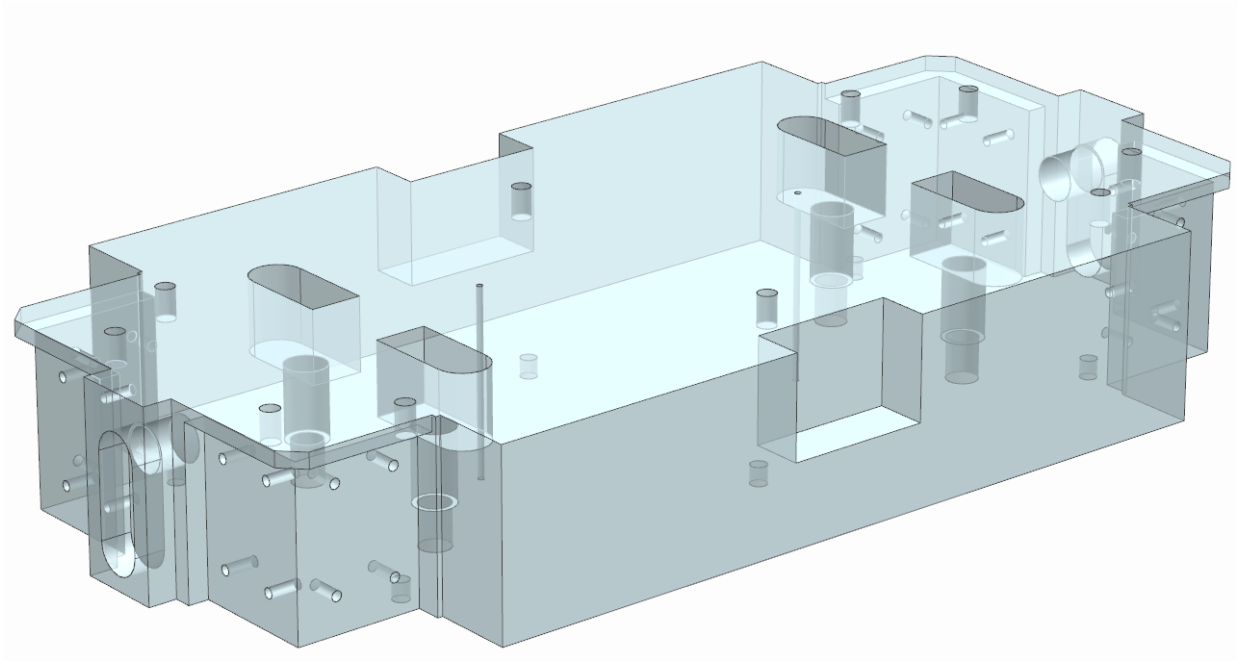


Рисунок 4.30 – Модель прижима-пуансона

Далее для построения модели матрицы осуществляется операция «вычитание». Для этого в файл модели выбираются инструменты, которыми будет произведено вычитание. Это элементы, входящие в состав концептуальной модели как карманы находящиеся в основании матрицы, карманы матрицы для направляющих, элементы, созданные для ограничения закрытой высоты в штампе, карманы, спроектированные для выталкивателей с поверхности матрицы, отверстия для транспортных винтов и по наружному контуру карманы под ловители. Так же эта операция показана в виде кода, но только присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении. Производится поиск функций и методов класса типа объект NX для автоматического выбора указанных элементов для булевой операции «вычитания». Журнал дополняет код приложения соответствующими строками и обеспечивает связь между геометрией инструмента и выбранного тела на уровне сквозного моделирования в сборке.

```
SelectionIntentRule[] rules303 = new SelectionIntentRule[1];
rules303[0] = bodyDumbRule302;
scCollector83.ReplaceRules(rules303, false);
Body[] bodies303 = new Body[32];
bodies303[0] = body130;
bodies303[1] = body132;
bodies303[2] = body134;
bodies303[3] = body136;
bodies303[4] = body138;
bodies303[5] = body140;
bodies303[6] = body142;
bodies303[7] = body144;
bodies303[8] = body146;
bodies303[9] = body148;
bodies303[10] = body150;
bodies303[11] = body152;
bodies303[12] = body154;
bodies303[13] = body156;
bodies303[14] = body158;
bodies303[15] = body160;
bodies303[16] = body162;
bodies303[17] = body164;
bodies303[18] = body165;
bodies303[19] = body167;
bodies303[20] = body169;
bodies303[21] = body171;
bodies303[22] = body172;
bodies303[23] = body173;
bodies303[24] = body174;
bodies303[25] = body175;
bodies303[26] = body176;
bodies303[27] = body177;
bodies303[28] = body178;
bodies303[29] = body179;
bodies303[30] = body180;
bodies303[31] = body181;
BodyDumbRule bodyDumbRule303;
bodyDumbRule303 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies303, true);
```

Рисунок 4.31 – Фрагмент кода, последовательный выбора тел для булевой операции «вычитание»

Результат операция «вычитание» показан на рисунке 4.32, тело построения выбрано для компонента «upper_die» и выбранные 32 связанных тела для непосредственно осуществления данной операции.

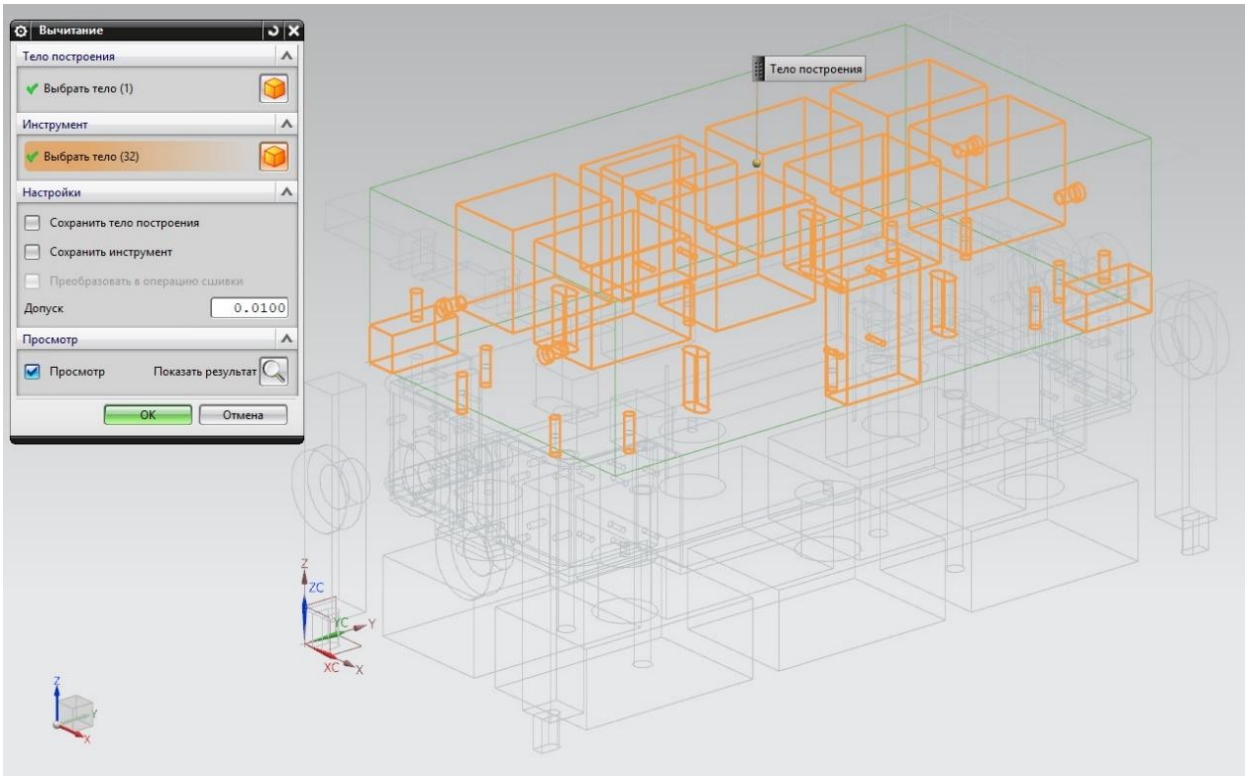


Рисунок 4.32 – Операция «вычитание» для компонента «upper_die»

Заключительным шагом для модели матрица и кнопки создать штамповую оснастку будет выбор инструментов, которыми будет произведено объединение. Это элементы, входящие в состав концептуальной модели как элементы крепления матрицы. Так же эта операция показана в виде кода, но только объявление данной операции вычитания в начале, ее завершающее объявление в конце и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
SelectionIntentRule[] rules309 = new SelectionIntentRule[1];
rules309[0] = bodyDumbRule308;
scCollector117.ReplaceRules(rules309, false);
Body[] bodies309 = new Body[2];
bodies309[0] = body183;
bodies309[1] = body185;
BodyDumbRule bodyDumbRule309;
bodyDumbRule309 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies309, true);
```

Рисунок 4.33 – Фрагмент кода, последовательный выбора тел для булевой операции «объединение»

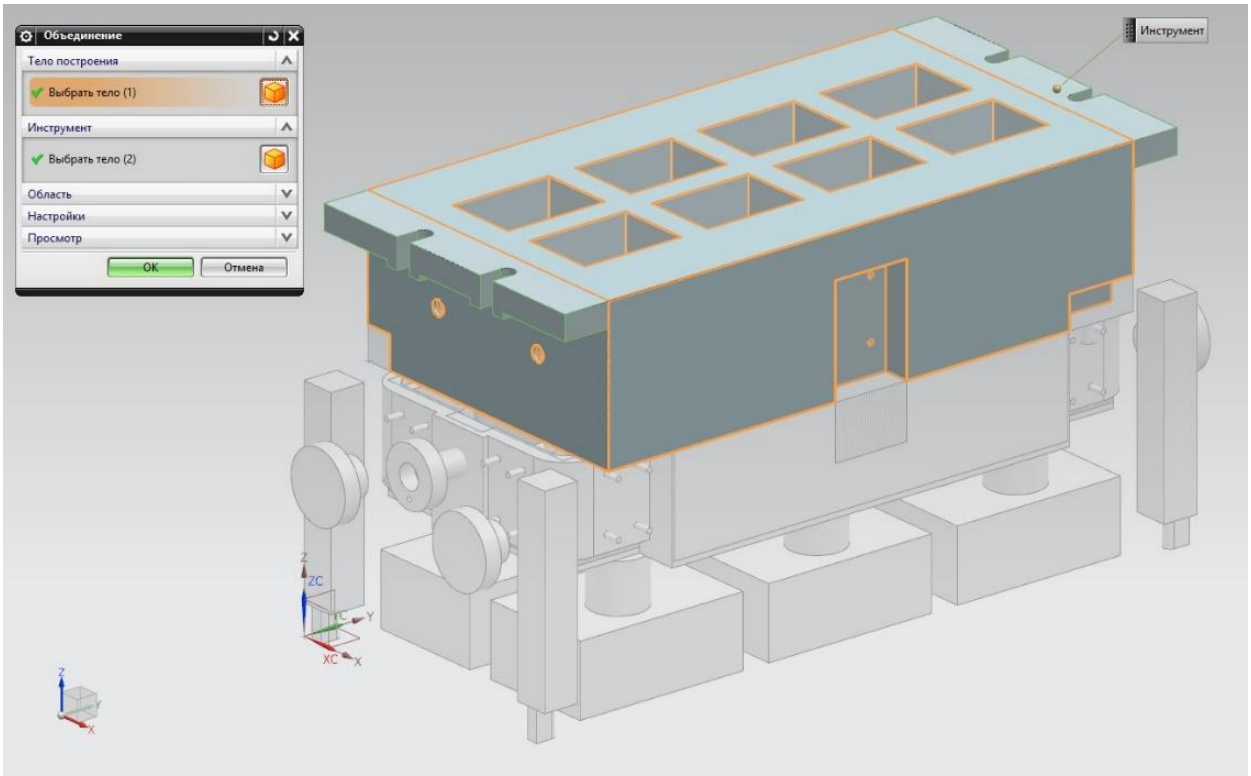


Рисунок 4.34 – Операция «объединение» для компонента «upper_die»

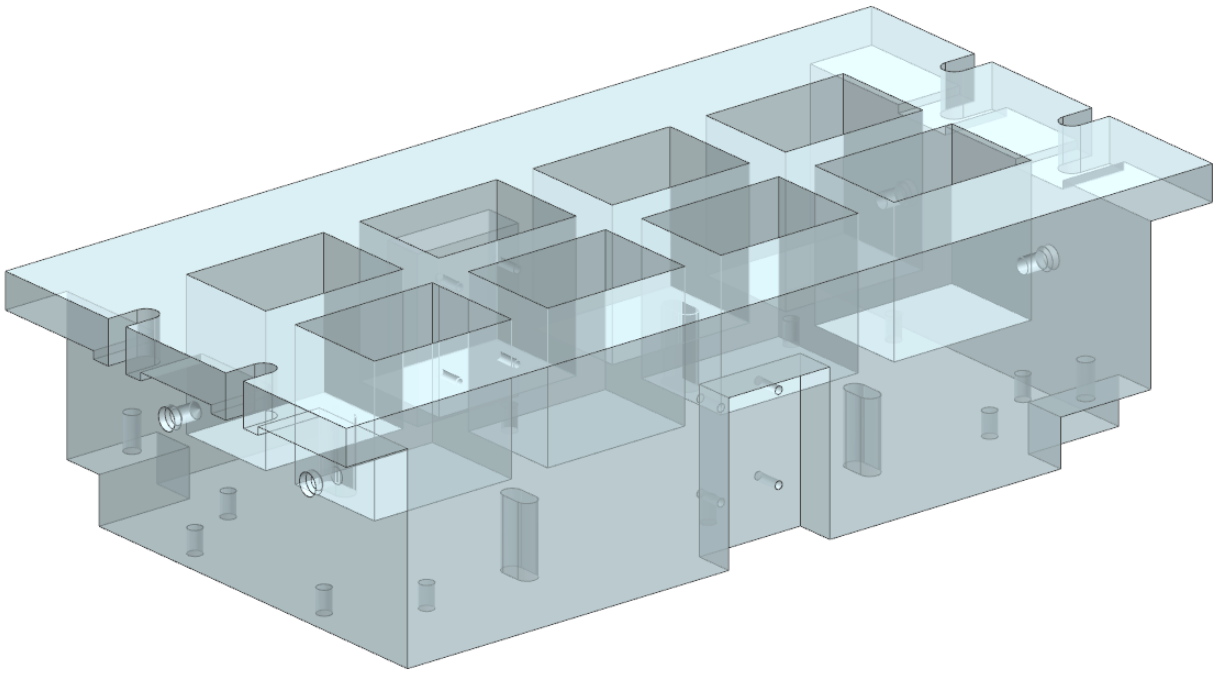


Рисунок 4.35 – Модель матрицы

4.4 Операция «разместить вытяжной переход»

Для того что бы сориентировать вытяжной переход относительно центра загруженного и построенного ранее штампа необходимо совместить их центры друг относительно друга. Для этого необходимо воспользоваться операцией «точка». В выпадающим окне выбирать между двумя точками и поочередно выбирать серединные точки боковых нижних граней вытяжного перехода для обратной вытяжки, в случае с прямой вытяжкой верхних. Так же эта операция показана в виде кода полностью в конце раздела.

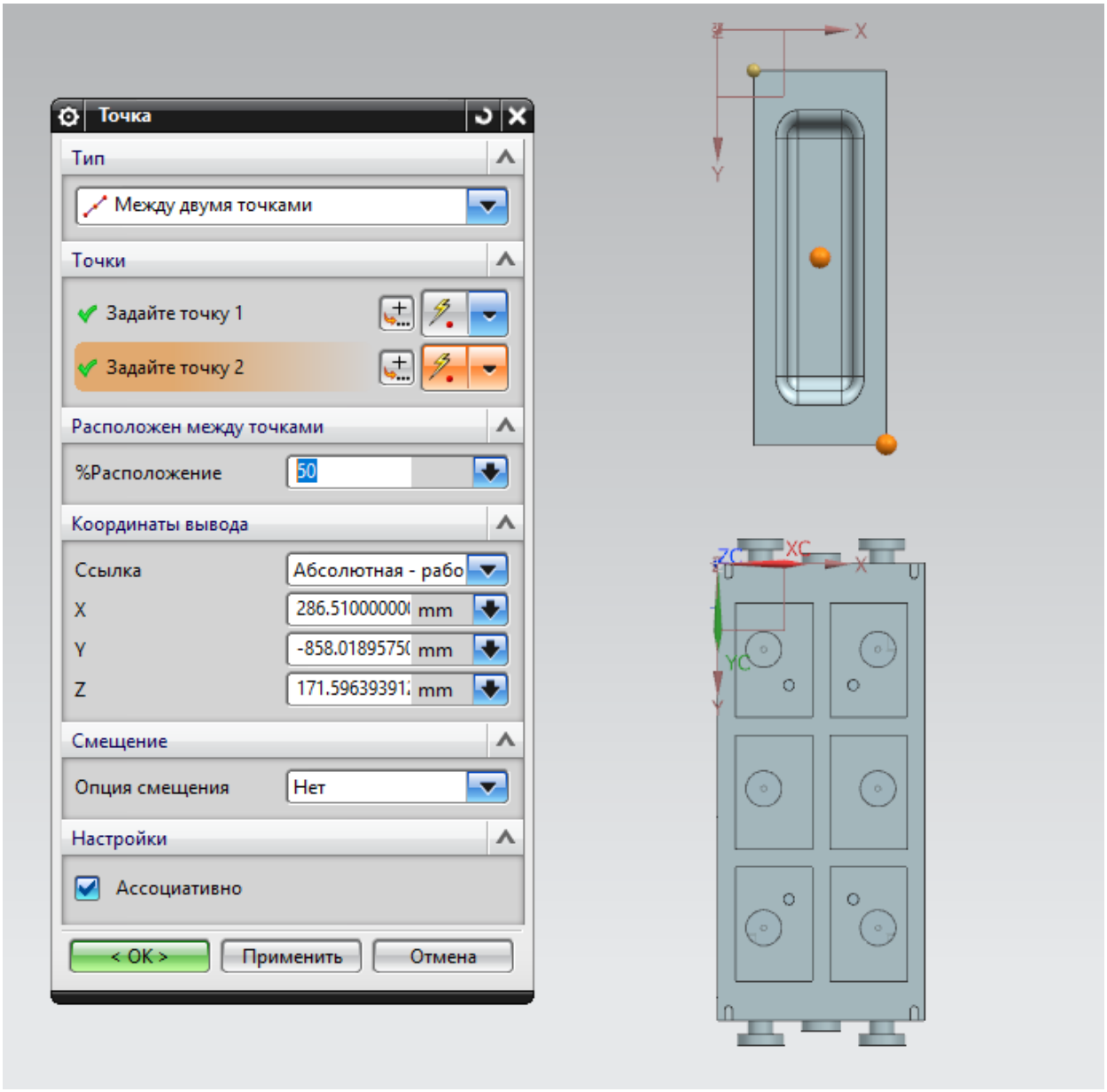


Рисунок 4.36 – Операция «точка» для компонента «заготовка»

Далее создать контекстную точку по середине поверхности прижима-пуансона. Центр заранее указан в параметризованной модели компонента «holder_lower_die».

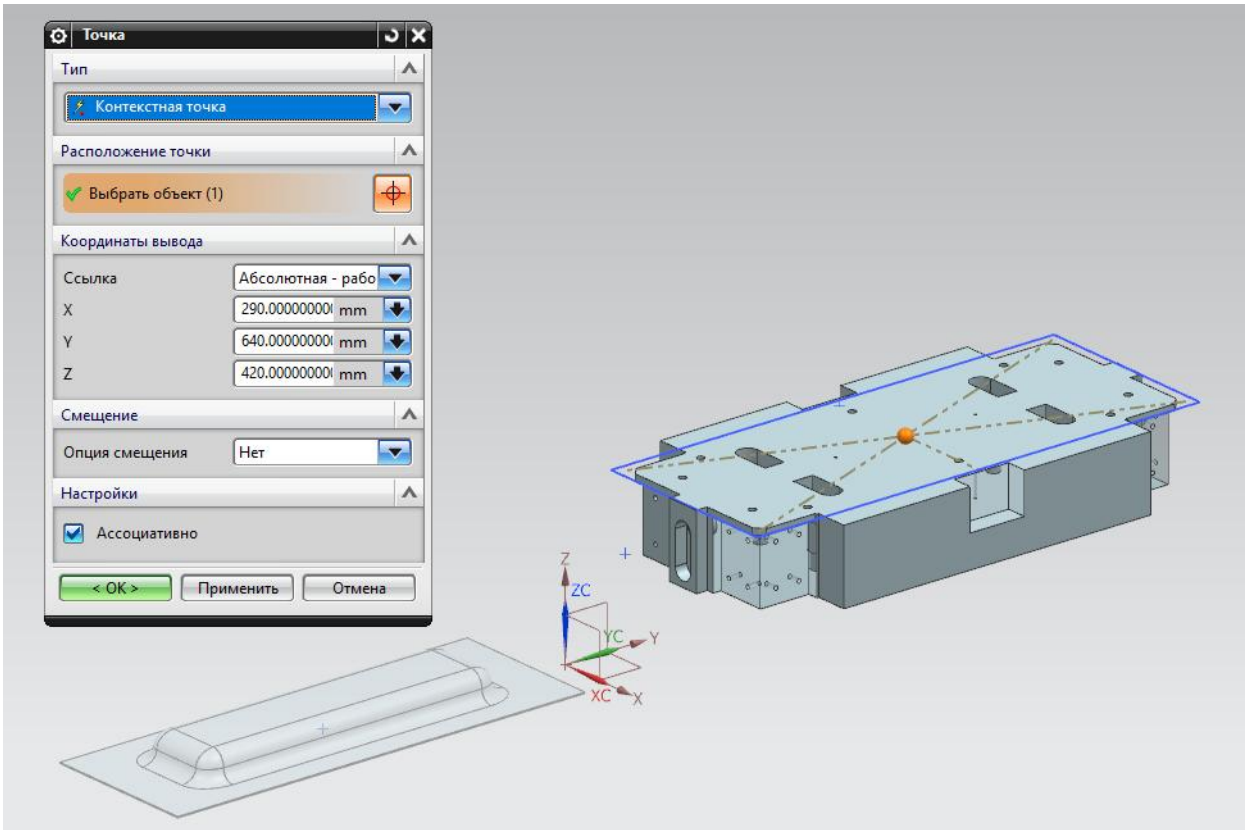


Рисунок 4.37 – Операция «точка» для компонента «holder_lower_die»

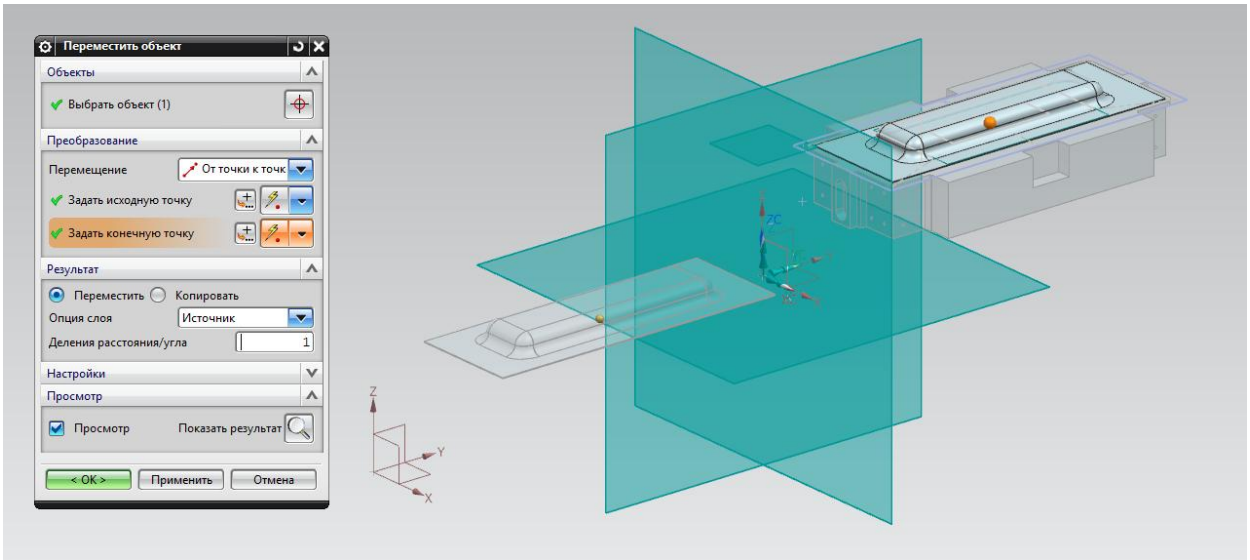


Рисунок 4.38 – Операция «переместить объект» для компонента вытяжного перехода и «holder_lower_die»

Следующим шагом необходимо совместить точки центров заготовки и прижима-пуансона. Для этого операцией «переместить объект» выбирать вытяжной переход, в виде исходной точки выбирать центр заготовки, в виде конечной точки середину компонента «holder_lower_die».

Результат данной кнопки показан на рисунке 4.42 и полный код для выполнения вышеприведенных операций.

```
theSession.SetUndoMarkName(markId1, "Диалоговое окно Точка");
(NXOpen.Features.Extrude)workPart.Features.FindObject("EXTRUDE(4)");
Edge    edge1    =    (Edge)extrude1.FindObject("EDGE    *    120    *    170    {(410.0000000000001,-
1116.1889575070945,172.1463939129912)(290.0000000000001,-
1116.1889575070945,172.1463939129912)(170.0000000000001,-1116.1889575070945,172.1463939129912)
EXTRUDE(4)}");
```

Рисунок 4.39 – Фрагмент кода, выбор вытяжного перехода и его координаты относительно базовой оси координат для операции создания «серединная точка»

```
NXOpen.SmartObject.UpdateOption.WithinModeling);
workPart.Points.RemoveParameters(point5);
point2.SetVisibility(NXOpen.SmartObject.VisibilityOption.Invisible);
point4.SetVisibility(NXOpen.SmartObject.VisibilityOption.Invisible);
expression2.RightHandSide = "290";
expression3.RightHandSide = "-856.188957507095";
expression4.RightHandSide = "172.146393912991";
expression2.RightHandSide = "290.00000000000";
expression3.RightHandSide = "-856.18895750709";
expression4.RightHandSide = "172.14639391299";
```

Рисунок 4.40 – Фрагмент кода, координаты серединной точки на нижней грани вытяжного перехода

```
theSession.SetUndoMarkName(markId11, "Диалоговое окно Переместить объект");
Body body1 = (Body)workPart.Bodies.FindObject("EXTRUDE(4)");
bool added1;
added1 = moveObjectBuilder1.ObjectToMoveObject.Add(body1);
moveObjectBuilder1.TransformMotion.Option =
NXOpen.GeometricUtilities.ModlMotion.Options.PointToPoint;
```

Рисунок 4.41 – Фрагмент кода, переместить объект от точки к тоске, для размещения вытяжного перехода по центру компонента «holder_lower_die»

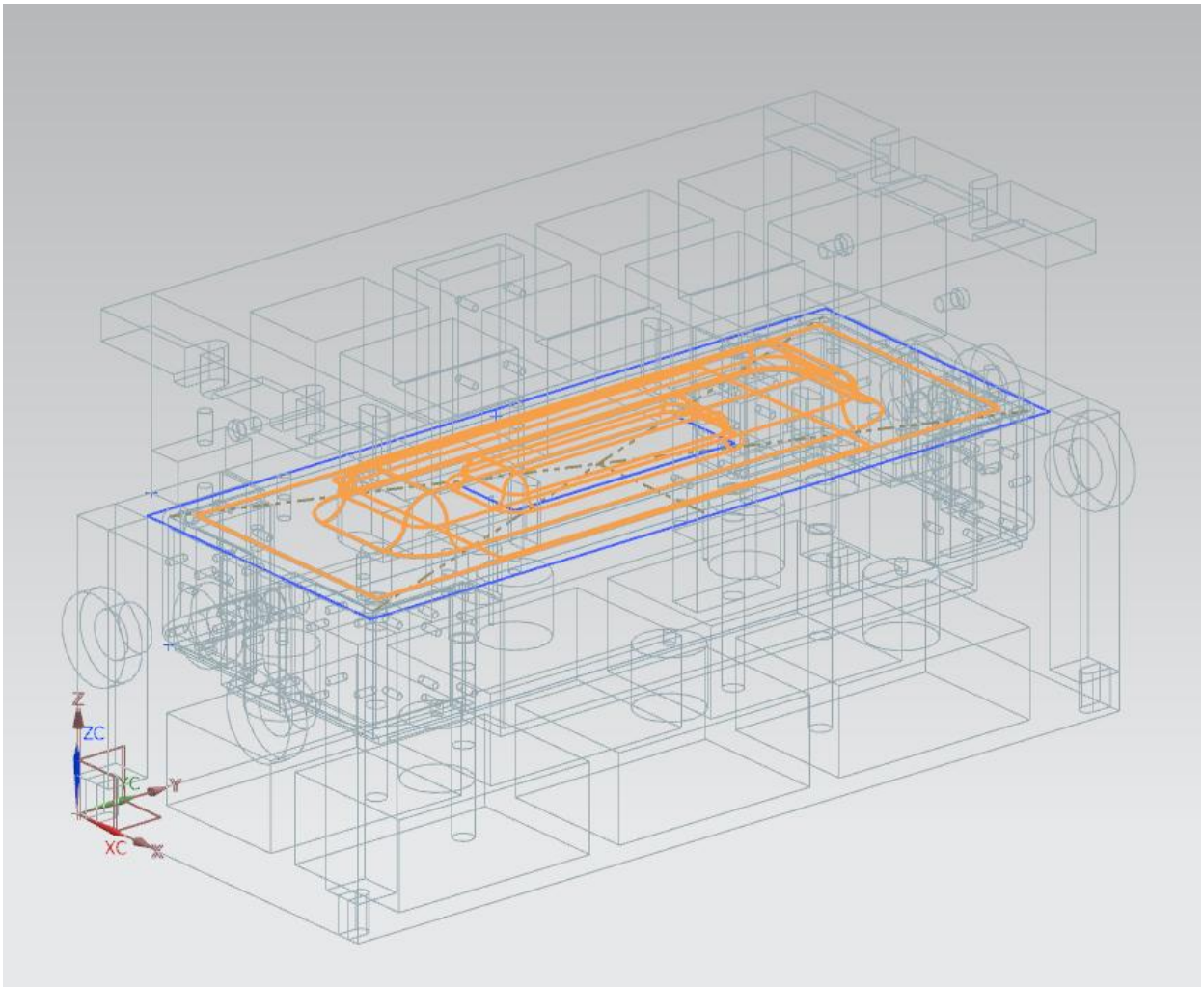


Рисунок 4.42 – Результат кнопки «разместить вытяжной переход»

4.5 Операция «прямая вытяжка»

При нажатии данной кнопки происходит объединение, вычитание и удаление грани компонентов в ходящих в состав пуансона-прижима и матрицы. При прямой вытяжке заготовку в матрицу втягивает пуансон.

Для начала, чтобы создать законченную модель матрицы, для прямой вытяжки осуществляется операция «объединение». Для этого в файле модели выбрать инструменты, которыми будет произведено объединение. Этот элемент – вытяжной переход детали, загруженный пользователем. Так же эта операция показана в виде кода, но только объявление данной операции вычитания в начале, ее завершающее объявление в конце и присоединение

последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId6, "Диалоговое окно Объединение");
Body body1 = (Body)workPart.Bodies.FindObject("BLOCK(2)");
bool added1;
added1 = booleanBuilder1.Targets.Add(body1);
TaggedObject[] targets1 = new TaggedObject[1];
targets1[0] = body1;
extractFace2.SetName("Тело");
Body[] bodies2 = new Body[1];
Body body3 = (Body)workPart.Bodies.FindObject("LINKED_BODY(37)");
bodies2[0] = body3;
BodyDumbRule bodyDumbRule2;
bodyDumbRule2 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies2, true);
```

Рисунок 4.43 – Фрагмент кода последовательного выбора тел для булевой операции «объединение»

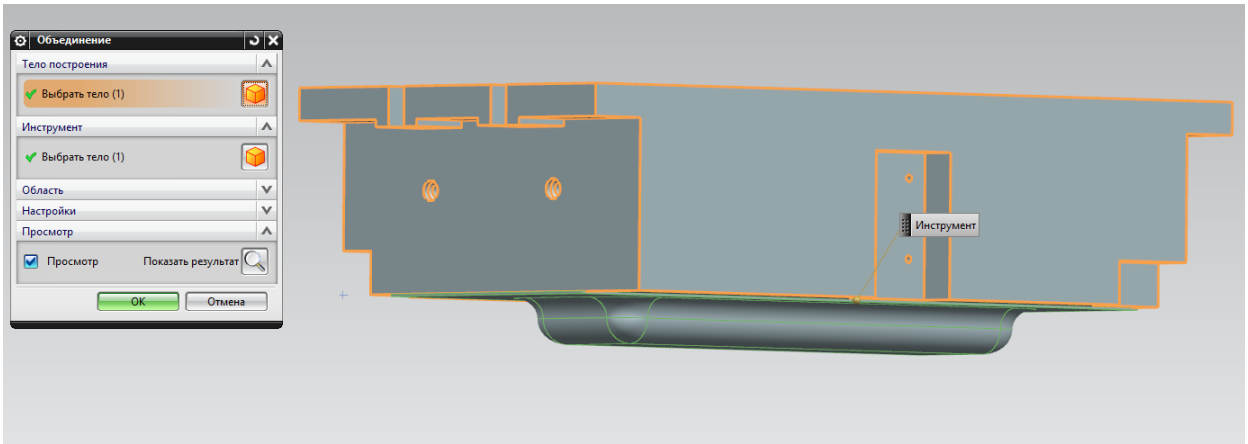


Рисунок 4.44 – Операция «объединение» для компонента «upper_die»

Далее, чтобы доработать модель прижим-пуансон для прямой вытяжки, осуществляется операция «вычитание». Для этого в файле модели выбрать инструменты, которыми будет произведено вычитание. Этот элемент – матрица с вытяжным переходом. Так же эта операция показана в виде кода, но только объявление данной операции «вычитания» в начале и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId13, "Диалоговое окно Вычитание");
Body body4 = (Body)workPart.Bodies.FindObject("BLOCK(2)");
bool added2;
added2 = booleanBuilder2.Targets.Add(body4);
TaggedObject[] targets3 = new TaggedObject[1];
targets3[0] = body4;
extractFace4.SetName("Тело");
Body[] bodies5 = new Body[1];
Body body6 = (Body)workPart.Bodies.FindObject("LINKED_BODY(33)");
bodies5[0] = body6;
BodyDumbRule bodyDumbRule5;
bodyDumbRule5 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies5, true);
```

Рисунок 4.45 – Фрагмент кода, последовательный выбор тел для булевой операции «вычитание»

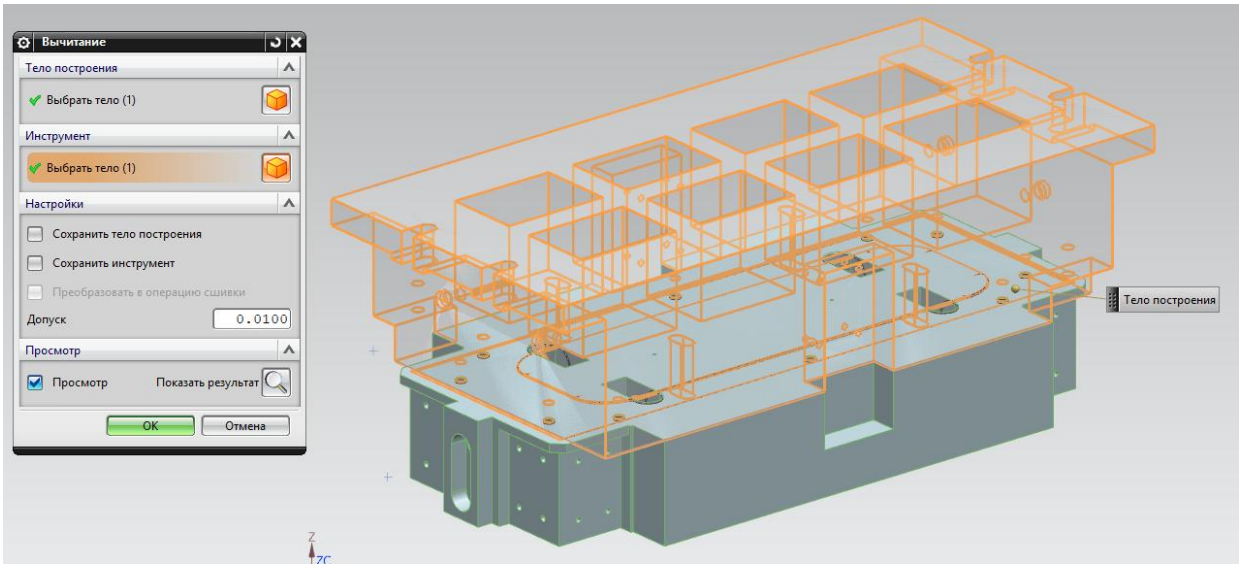


Рисунок 4.46 – Финишная операция «вычитание» для компонента «holder_lower_die»

Заключительным шагом для модели пуансона-прижима и кнопки прямая вытяжка будет выбор инструмента, которыми будет произведено удаление грани, так как вытяжной переход является листовой деталью.

```
Face[] faces1 = new Face[1];
NXOpen.Features.Block block1 = (NXOpen.Features.Block)workPart.Features.FindObject("BLOCK(2)");
Face face1 = (Face)block1.FindObject("FACE 1 1 {(284.9999999999997,645.0000000000003,420)
SUBTRACT(33)1}");
faces1[0] = face1;
FaceDumbRule faceDumbRule1;
faceDumbRule1 = workPart.ScRuleFactory.CreateRuleFaceDumb(faces1);
```

Рисунок 4.47 – Фрагмент кода «удалить грань» и координаты геометрии относительно которой происходит операция удаления

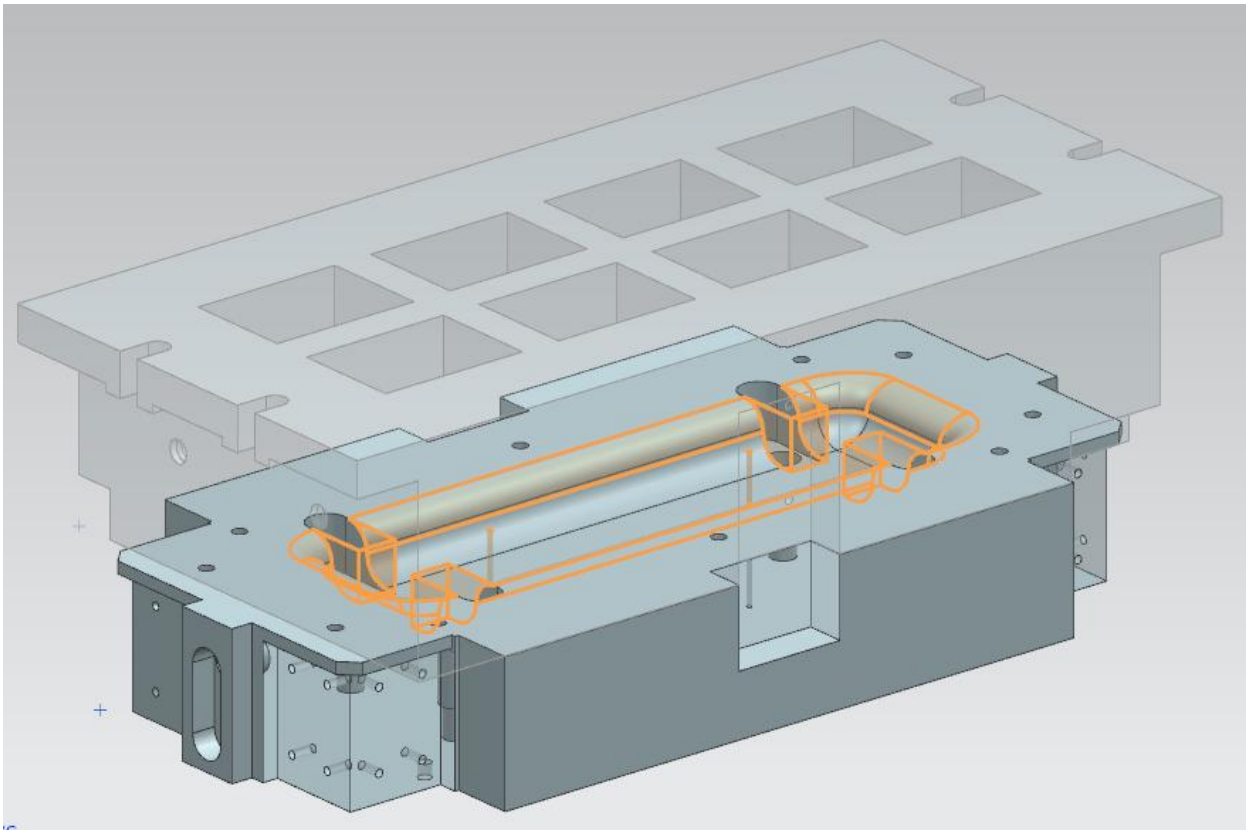


Рисунок 4.48 – Операция «удалить грань» для компонента «holder_lower_die»

Так как прямая вытяжка осуществляется для другого типа оборудования, но концептуальная возможность данной операции рассмотрена далее, будет показана оптимизация поверхностей прижима и матрицы до финального вида, но с помощью кода данные операции осуществлены не будут.

Для того что бы задать зазор между прижимом и матрицей на величину толщины вытяжного перехода необходимо сместить грани на величину этих толщин.

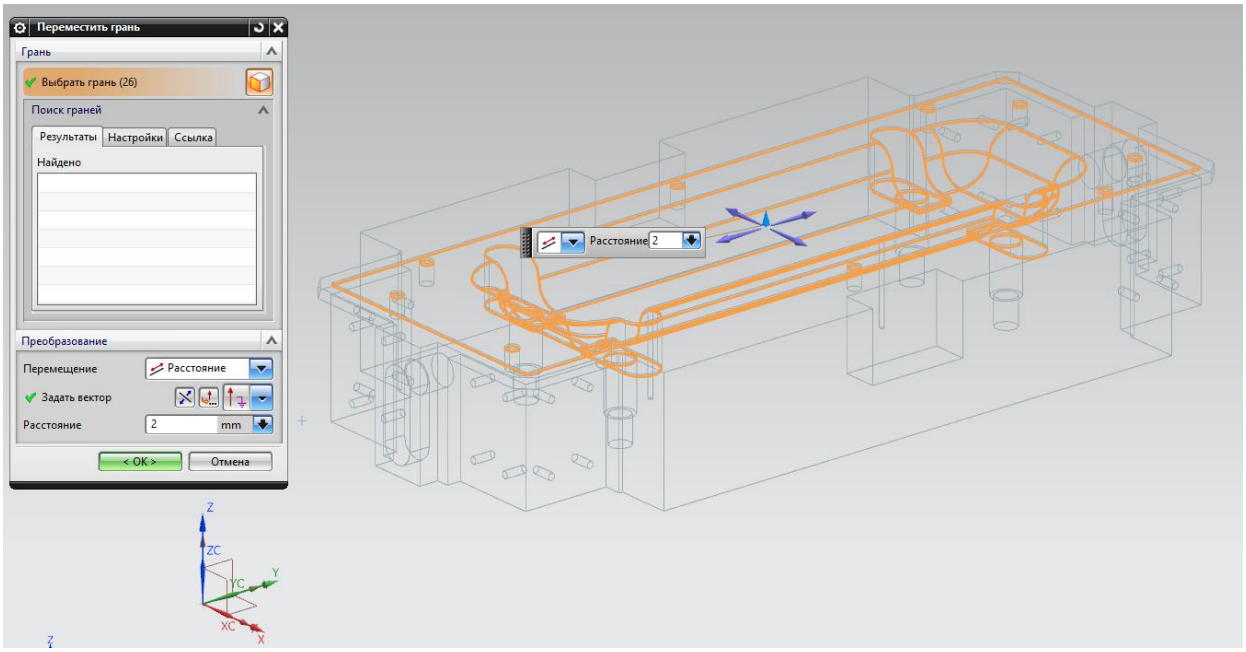


Рисунок 4.49 – Операция «переместить грань» для компонента «holder_lower_die»

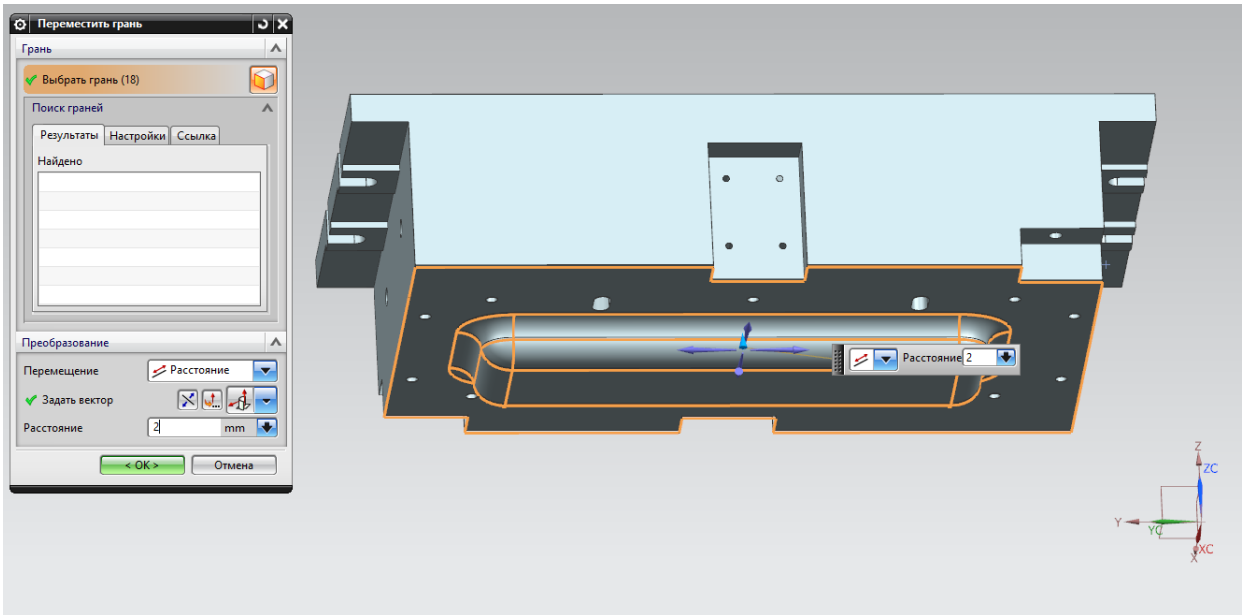


Рисунок 4.50 – Операция «переместить грань» для компонента «upper_die»

На рисунке 4.49 показано смещение грани на прижиме, на рисунке 4.50 аналогичная операция только для матрицы. Результат операции кнопки «прямая вытяжка» представлен на рисунке 4.51.

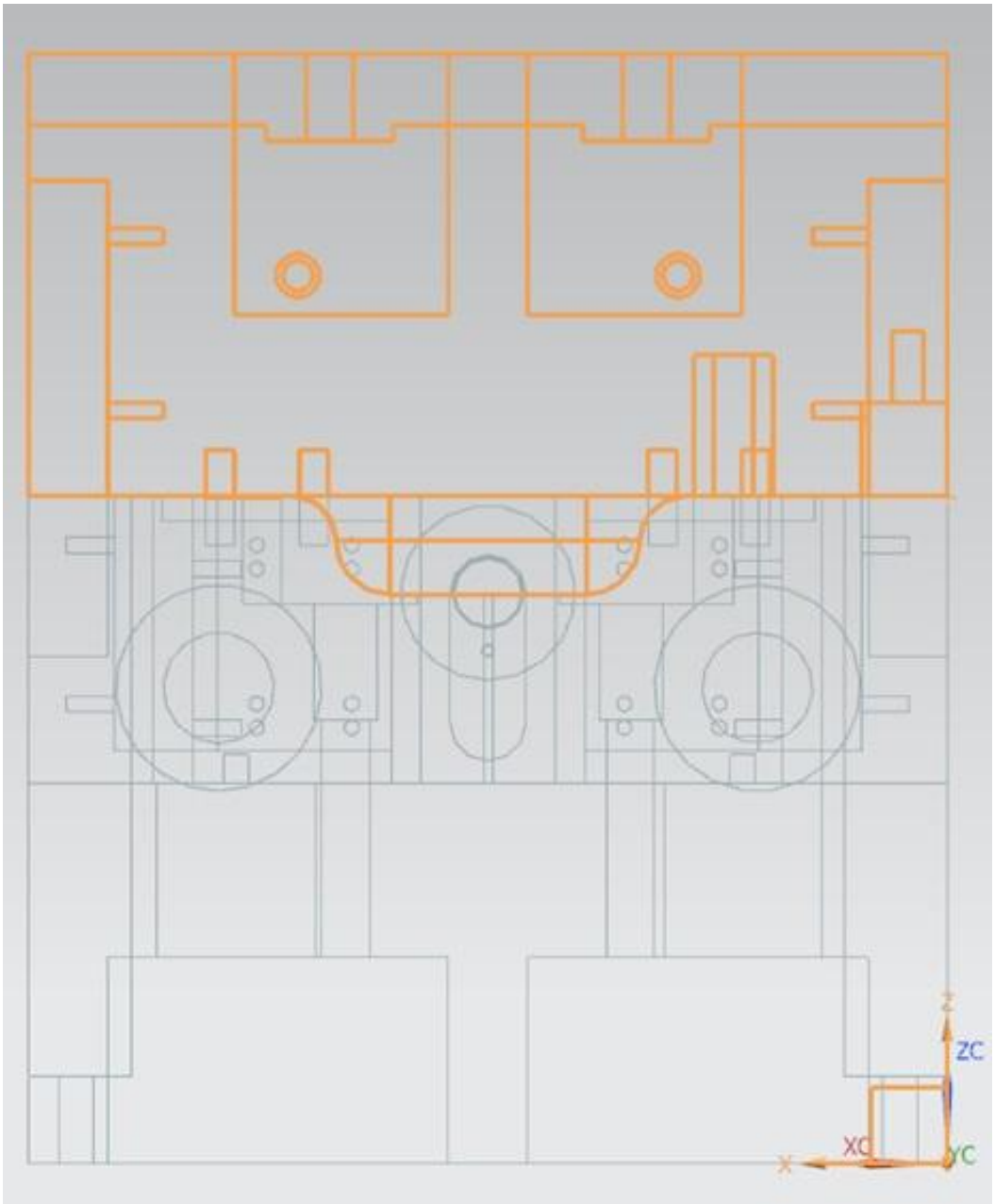


Рисунок 4.51 – Результат операции «прямая вытяжка»

4.6 Операция «обратная вытяжка»

При нажатии данной кнопки происходит объединение, вычитание и удаление грани компонентов в ходящих в состав пуансона-прижима и матрицы. При обратной вытяжке заготовку в пуансон-прижим втягивает матрица.

Для начала что бы создать законченную модель пуансона-прижима для прямой вытяжки осуществляется операция «объединение». Для этого в файле

модели выбрать инструменты, которыми будет произведено объединение. Этот элемент – вытяжной переход детали, загруженный пользователем. Так же эта операция показана в виде кода, но только объявление данной операции вычитания в начале и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId5, "Диалоговое окно Объединение");
Body body1 = (Body)workPart.Bodies.FindObject("BLOCK(2)");
bool added1;
added1 = booleanBuilder1.Targets.Add(body1);
TaggedObject[] targets1 = new TaggedObject[1];
targets1[0] = body1;
Body[] bodies1 = new Body[1];
Body body2 = (Body)displayPart.Bodies.FindObject("EXTRUDE(4)");
bodies1[0] = body2;
extractFace2.SetName("Тело");
extractFaceBuilder1.Destroy();
Body[] bodies2 = new Body[1];
Body body3 = (Body)workPart.Bodies.FindObject("LINKED_BODY(33)");
bodies2[0] = body3;
BodyDumbRule bodyDumbRule2;
bodyDumbRule2 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies2, true);
```

Рисунок 4.52 – Фрагмент кода последовательного выбора тел для булевой операции «объединение»

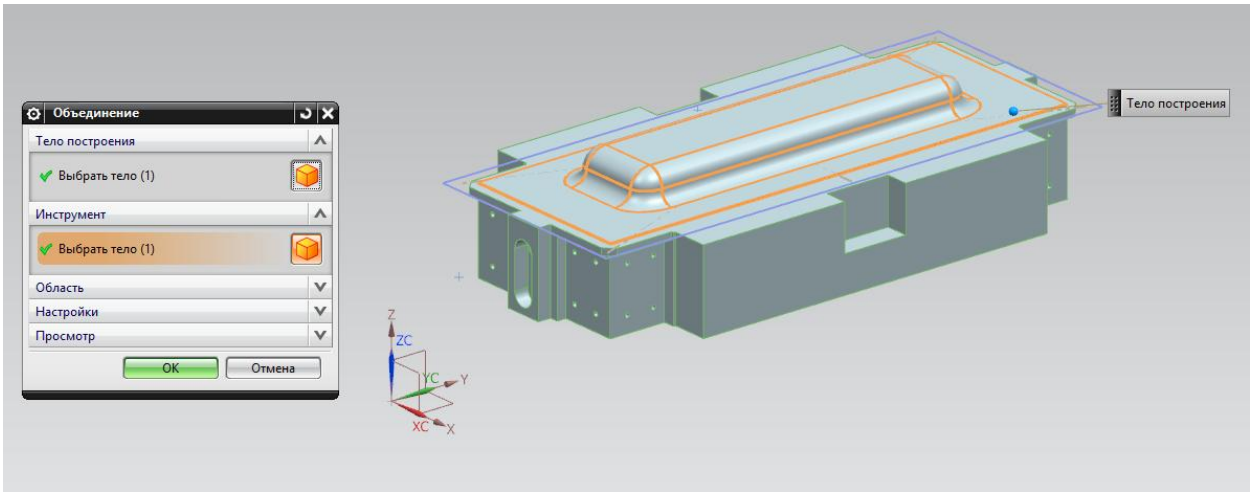


Рисунок 4.53 – Операция «объединение» для компонента «holder_lower_die» и вытяжного перехода

Далее, чтобы доработать модель матрицы для обратной вытяжки, осуществляется операция «вычитание». Для этого в файле модели выбрать инструменты, которыми будет произведено вычитание. Этот элемент – прижим-пуансон с вытяжным переходом. Так же эта операция показана в виде кода, но только объявление данной операции «вычитания» в начале, ее завершающее объявление в конце и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId12, "Диалоговое окно Вычитание");
Body body4 = (Body)workPart.Bodies.FindObject("BLOCK(2)");
bool added2;
added2 = booleanBuilder2.Targets.Add(body4);
TaggedObject[] targets3 = new TaggedObject[1];
targets3[0] = body4;
extractFace4.SetName("Тело");
extractFaceBuilder3.Destroy();
Body[] bodies5 = new Body[1];
Body body6 = (Body)workPart.Bodies.FindObject("LINKED_BODY(37)");
```

Рисунок 4.54 – Фрагмент кода, последовательный выбор тел для булевой операции «вычитание»

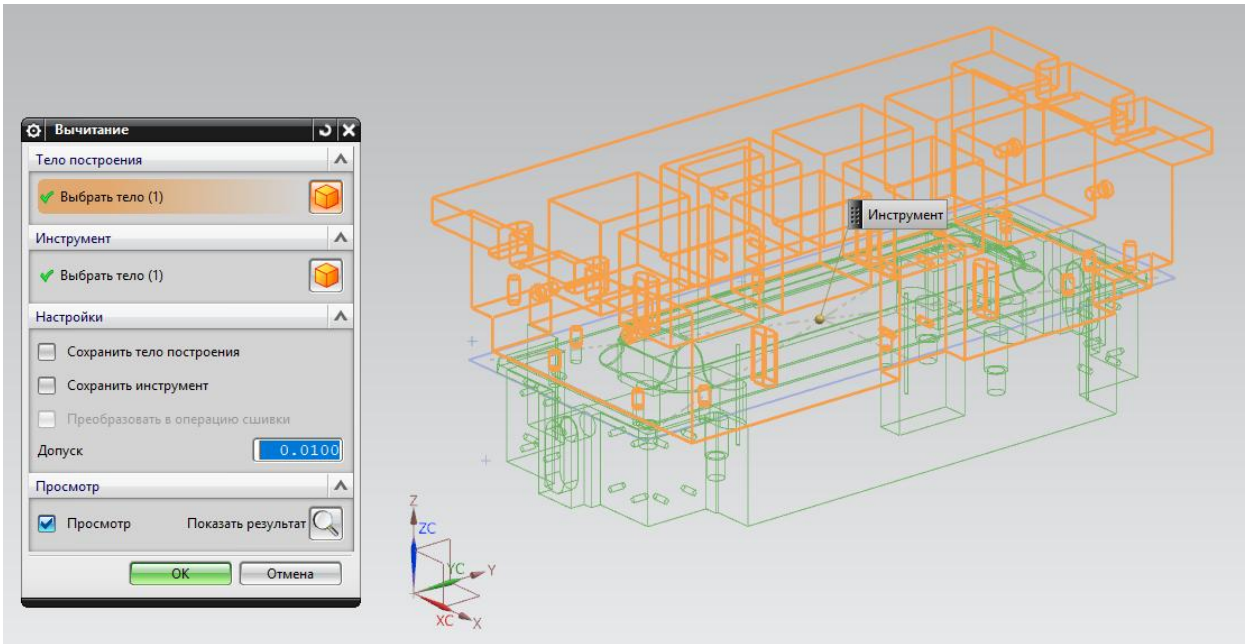


Рисунок 4.55 – Финишная операция «вычитание» для компонента «upper_die»

Заключительным шагом для модели матрицы и кнопки обратная вытяжка будет выбор инструмента, которыми будет произведено удаление грани, так как вытяжной переход является листовой деталью.

```
theSession.SetUndoMarkName(markId19, "Диалоговое окно Удалить грань");
Face[] faces1 = new Face[1];
NXOpen.Features.Block block1 = (NXOpen.Features.Block)workPart.Features.FindObject("BLOCK(2)");
Face face1 = (Face)block1.FindObject("FACE 5 1 {(284.9999999999997,635.0000000000007,420)
SUBTRACT(37)1}");
faces1[0] = face1;
FaceDumbRule faceDumbRule1;
faceDumbRule1 = workPart.ScRuleFactory.CreateRuleFaceDumb(faces1);
```

Рисунок 4.56 – Фрагмент кода «удалить грань» и координаты геометрии относительно которой происходит операция удаления

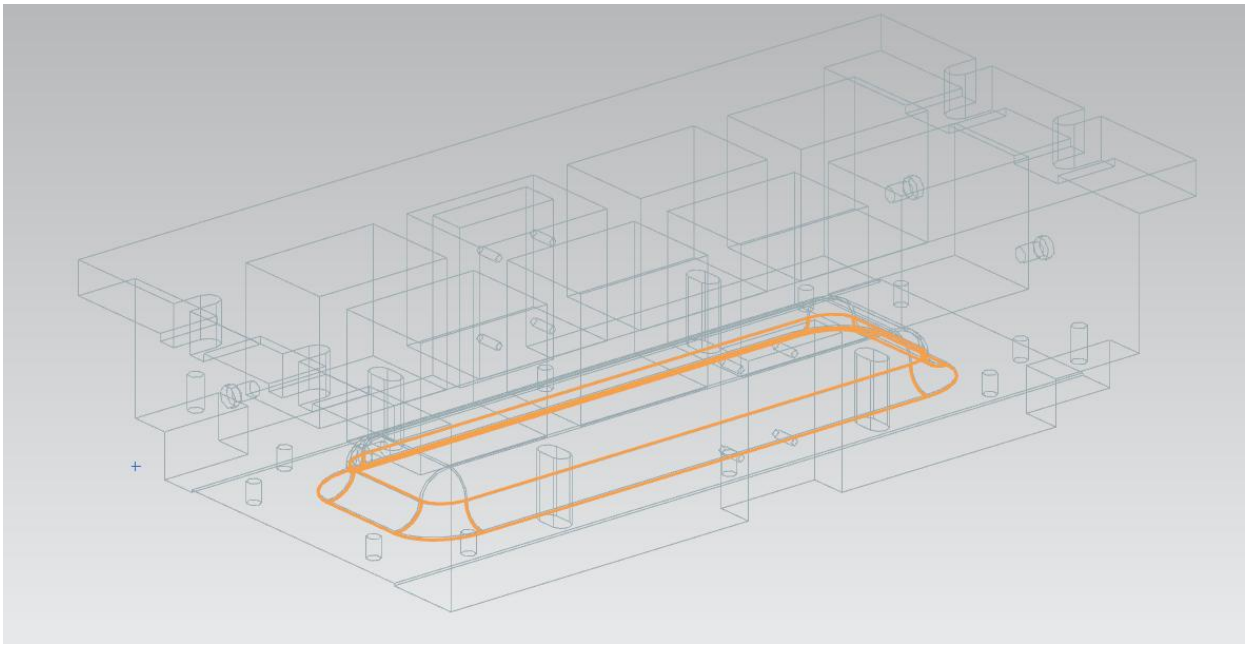


Рисунок 4.57 – Операция «удалить грань» для компонента «upper_die»

Производится поиск функций и методов класса типа объект NX для автоматического выбора указанных элементов для операции «удалить грань». Журнал дополняет код приложения соответствующими строками и обеспечивает связь между геометрией инструмента и выбранного тела на уровне сквозного моделирования в сборке.

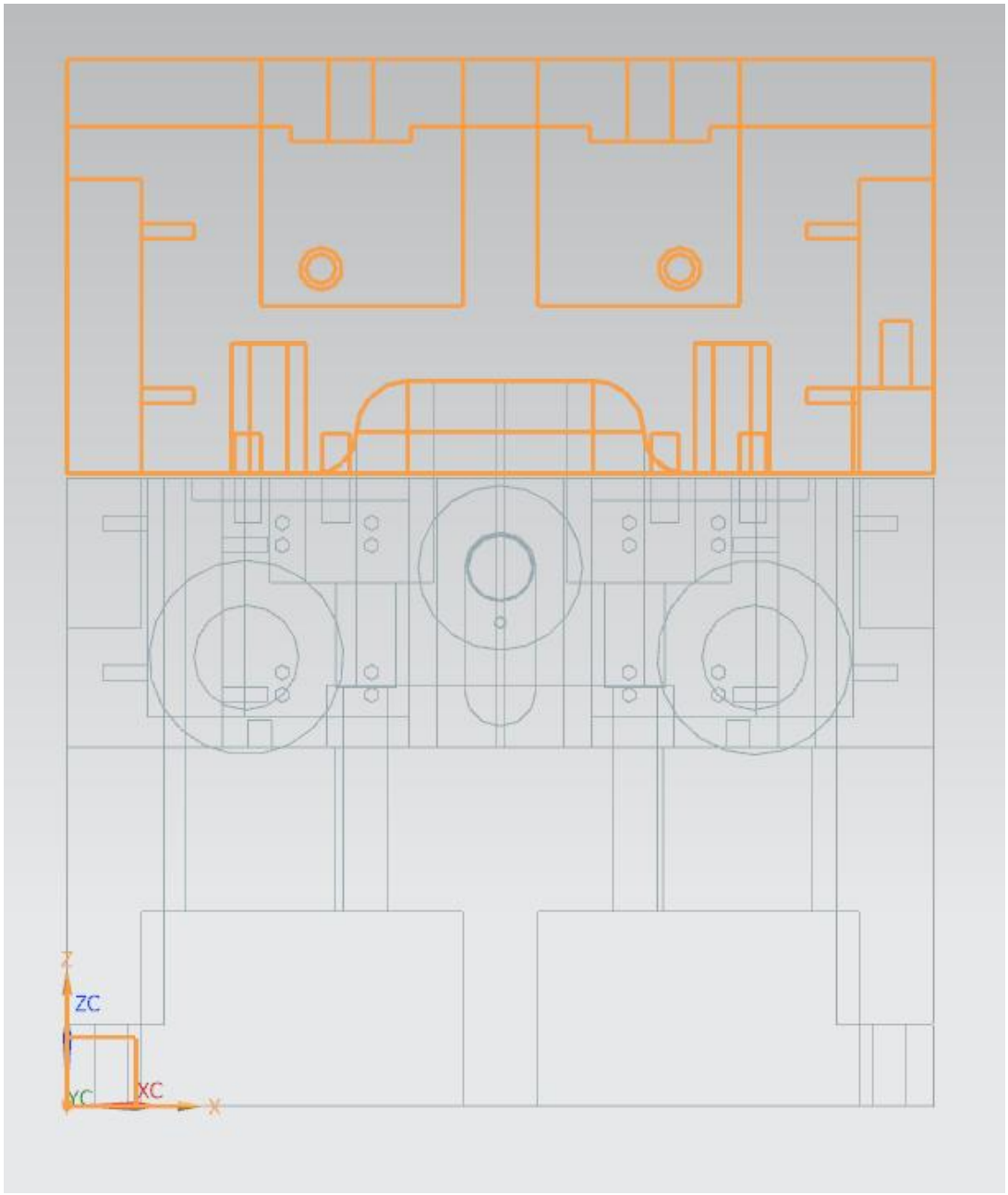


Рисунок 4.58 – Результат операции «обратная вытяжка»

4.7 Операция «разделение на пуансон и прижим»

После завершения операции «вытяжка» следует разделить пуансон-прижим на две отдельные детали и создать в дереве сборки два компонента «holder» и «punch». Для этого компонент «holder_lower_die» делается рабочим и вызывается операция «объединить кривые» для создания пуансона по вытяжному переходу.

```
theSession.SetUndoMarkName(markId3, "Диалоговое окно Объединить кривые");
NXOpen.Features.ExtractFace extractFace1 =
(NXOpen.Features.ExtractFace)workPart.Features.FindObject("UNITE(33:1B)");
Edge edge2 = (Edge)extractFace1.FindObject("EDGE * 28 * 33
{(347.0000000000001,800.0000000000001,440)(347.0000000000001,635.0000000000001,440)(347.000000000
0001,470.0000000000001,440) BLOCK(2)}");
NXObject nullNXObject = null;
theSession.SetUndoMarkName(markId3, "Объединить кривые");
joinCurvesBuilder1.Destroy();
```

Рисунок 4.59 – Фрагмент кода, «объединить кривые» и координаты геометрии относительно которой происходит операция «объединение»

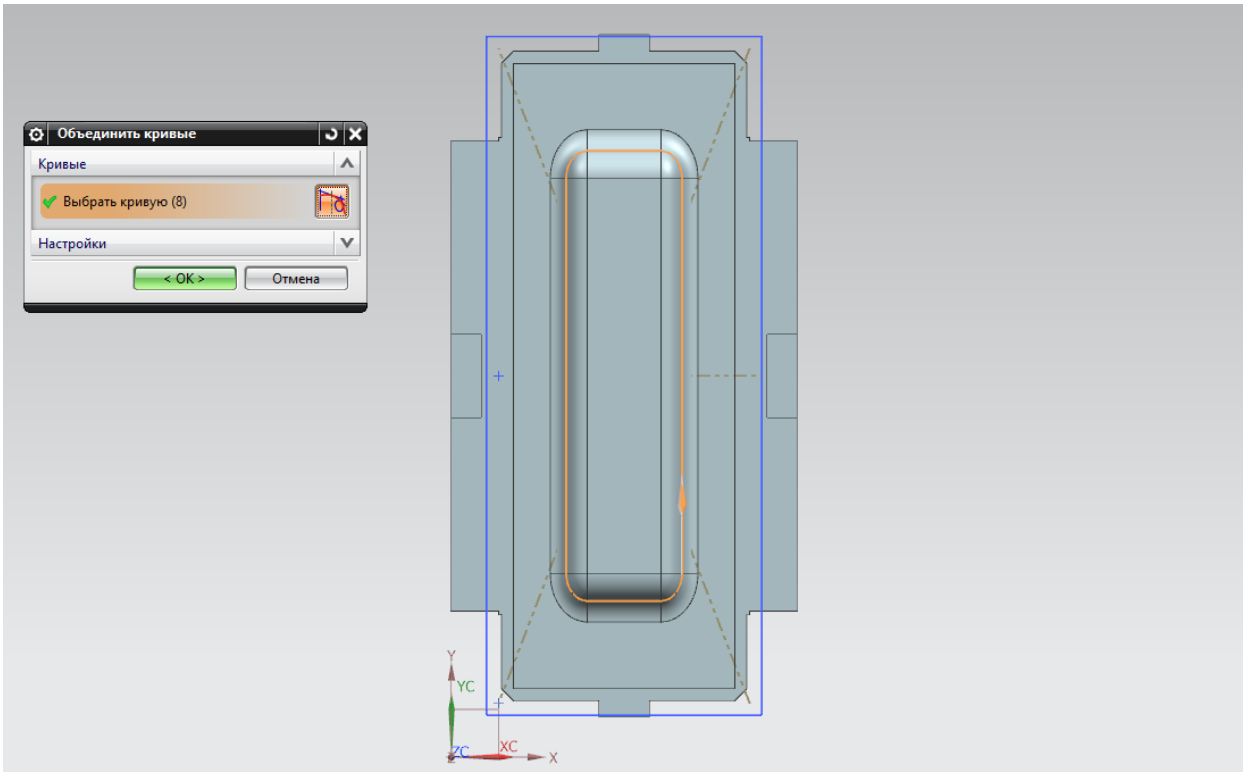


Рисунок 4.60 – Результат операции «объединить кривые»

Следующей операцией «вытягивание» заполняется пустота внутри между геометрией прижимом-пуансоном и вытяжным переходом после операции «объединения». Производится поиск функций и методов класса типа объект NX для автоматического выбора указанных элементов для операции «вытягивание». Журнал дополняет код приложения соответствующими строками и обеспечивает связь между геометрией инструмента и выбранного тела на уровне сквозного моделирования в сборке.

```
theSession.SetUndoMarkName(markId24, "Диалоговое окно Вытягивание");
NXOpen.Features.ExtractFace extractFace2 =
(NXOpen.Features.ExtractFace)workPart.Features.FindObject("UNITE(33:1B)");
Face face1 = (Face)extractFace1.FindObject("FACE 33
{(340.5563491861041,635.0000000000001,455.556349186104) BLOCK(2)}");
NXOpen.Session.UndoMarkId markId28;
markId28 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible, null);
Edge edge4 = (Edge)extractFace1.FindObject("EDGE * 28 * 33
{(347.0000000000001,800.0000000000001,440)(347.0000000000001,635.0000000000001,440)(347.000000000
0001,470.0000000000001,440) BLOCK(2)}");
section1.RemoveRules(edge4, nullNXObject, nullNXObject, NXOpen.Section.Mode.Create);
```

Рисунок 4.61 – Фрагмент кода, вытягивания для заполнения пустот в теле пуансона и координаты геометрии относительно которой происходит операция «вытягивание»

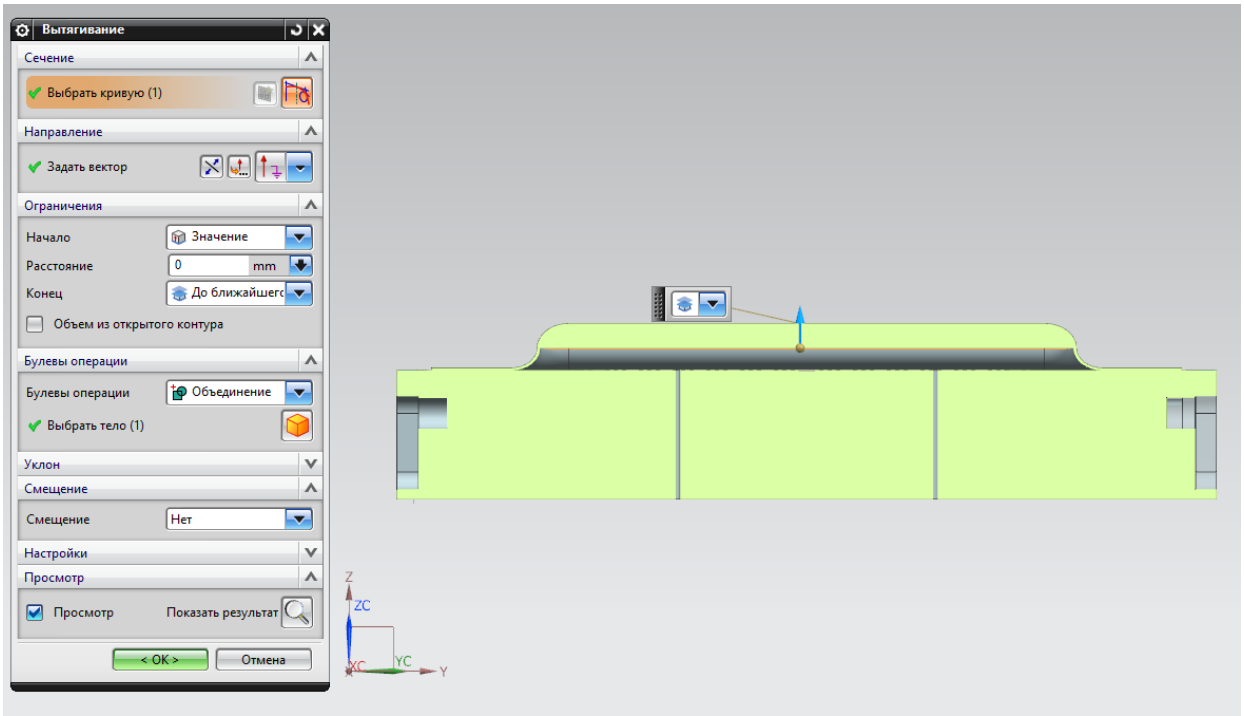


Рисунок 4.62 – Результат операций «вытягивание»

Первая операция задается ограничением до ближайшего, и начинается от объединённых ранее кривых. Это сделано для того что бы в одну операцию относительно геометрии вытяжного перехода детали сделать твердотельную основу для будущего пуансона, при этом не используя дополнительных операций для удаления лишнего, если бы эта операция была вытягивание по умолчанию, что облегчает моделирование деталей.

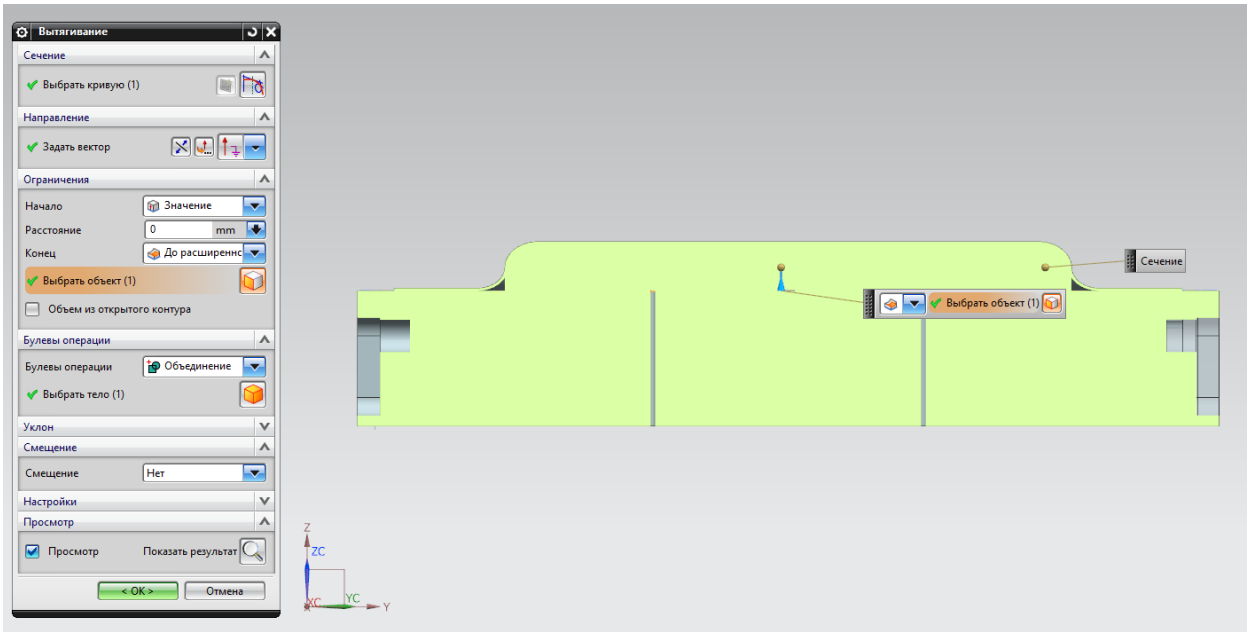


Рисунок 4.63 – Результат операций «вытягивание»

Вторая операция задается ограничением на расстоянии, и начинается от объединённых ранее кривых. Это сделано для того, чтобы в одну операцию относительно геометрии вытяжного перехода детали сделать твердотельную основу для будущего пуансона. На рисунке 4.63 показан результат двух операций вытягивания, которые полностью заполнили пустоты между прижимом-пуансоном и геометрией вытяжного перехода детали, тем самым создав твердотельный пуансон.

Далее операцией «разделить тело» создается разделение компонента прижим-пуансон на два отдельных тела прижим и пуансон.

```
theSession.SetUndoMarkName(markId33, "Диалоговое окно Разделить тело");
Spline spline2 = (Spline)component2.FindObject("PROTO#.Features|JOIN_CURVE(34)|CURVE 1");
NXObject nXObject3;
nXObject3 = splitBodyBuilder2.Commit();
splitBodyBuilder2.BooleanTool.ExtrudeRevolveTool.ToolSection.CleanMappingData();
splitBodyBuilder2.ShowResults();
theSession.SetUndoMarkName(markId39, "Разделить тело");
splitBodyBuilder2.BooleanTool.ExtrudeRevolveTool.ToolSection.CleanMappingData();
splitBodyBuilder2.Destroy();
```

Рисунок 4.64 – Фрагмент кода, разделить тело, для создания пуансона и прижима

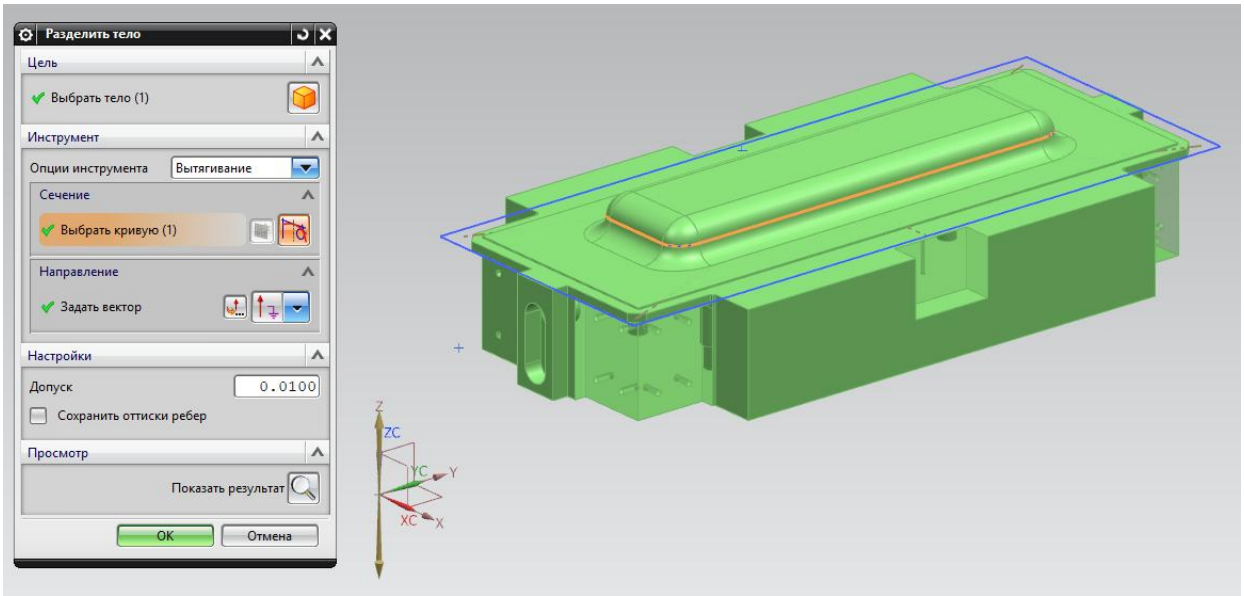


Рисунок 4.65 – Результат операции «разделить тело»

Далее операцией «вытягивание» со смещением и так же объединением в основании пуансона закончить проектирование его оснастки.

```
theSession.SetUndoMarkName(markId41, "Диалоговое окно Вытягивание");
NXOpen.Features.Block block1 =
(NXOpen.Features.Block)workPart.Features.FindObject("BLOCK(2)");
Face face2 = (Face)block1.FindObject("FACE 5 {(290,640.0000000000001,240) BLOCK(2)}");
Body[] targetBodies6 = new Body[1];
Body body2 = (Body)workPart.Bodies.FindObject("SPLIT BODY(36)1");
extrudeBuilder2.Offset.Option = NXOpen.GeometricUtilities.Type.SingleOffset;
extrudeBuilder2.Limits.EndExtend.Value.RightHandSide = "-170";
```

Рисунок 4.66 – Фрагмент кода, «вытягивание» для завершения создания основания пуансона

Следующей операцией «вытягивание» создается основание пуансона, представленный теперь в виде отдельной детали. Производится поиск функций и методов класса типа объект NX для автоматического выбора указанных элементов для операции «вытягивание». Журнал дополняет код приложения соответствующими строками и обеспечивает связь между геометрией инструмента и выбранного тела на уровне сквозного моделирования в сборке.

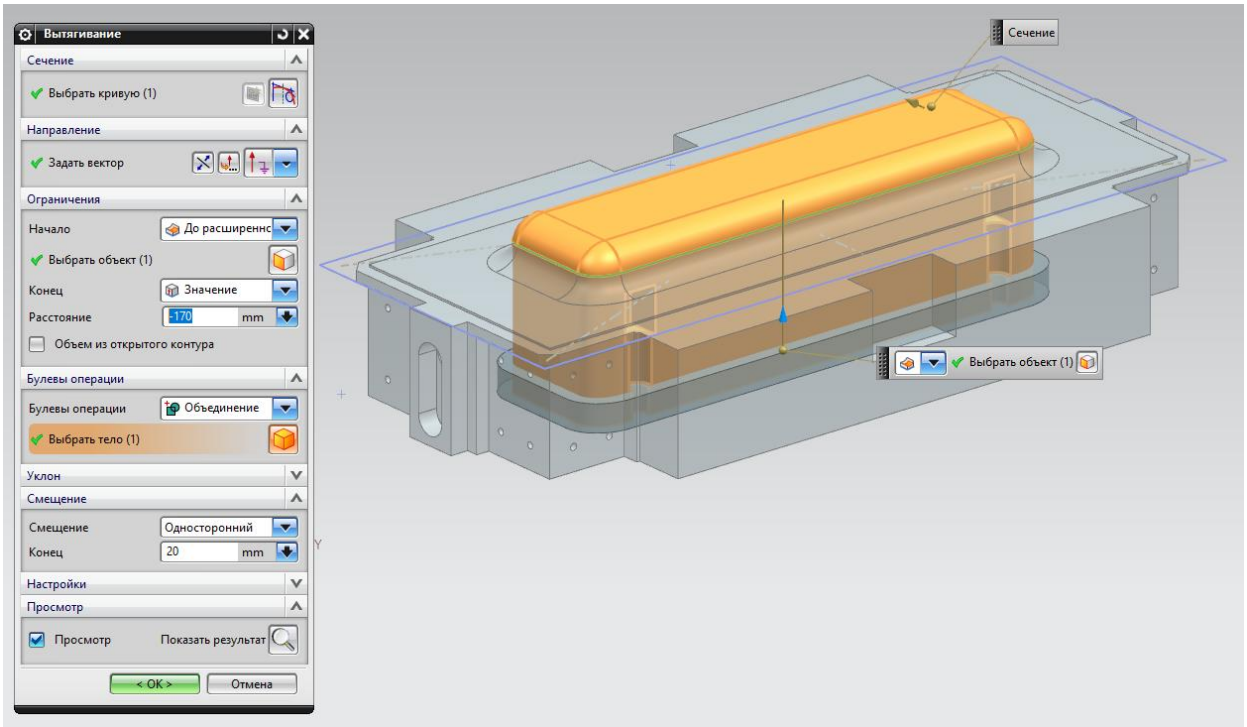


Рисунок 4.67 – Результат операции «вытягивания»

Для детали прижим операцией «вытягивание со смещением» и булевой операцией «вычитание» создается отверстие в прижиме относительно геометрии созданного ранее пуансона. Так же, как и на предыдущих операциях за основу берутся объединённые кривых, относительно которых в данной кнопке и производятся большинство операций. Данные кривые, как описывалось ранее, необходимы для выбора ограничения в данном случае «до расширенного».

```
theSession.SetUndoMarkName(markId46, "Диалоговое окно Вытягивание");
extrudeBuilder3.Limits.EndExtend.Value.RightHandSide = "-170";
extrudeBuilder3.BooleanOperation.Type =
NXOpen.GeometricUtilities.BooleanOperation.BooleanType.Unite;
Body[] targetBodies8 = new Body[1];
targetBodies8[0] = nullBody;
extrudeBuilder3.BooleanOperation.SetTargetBodies(targetBodies8);
extrudeBuilder3.Offset.EndOffset.RightHandSide = "20";
```

Рисунок 4.68 – Фрагмент кода «вытягивание» для создания отверстия в теле прижима под основание тела пуансона

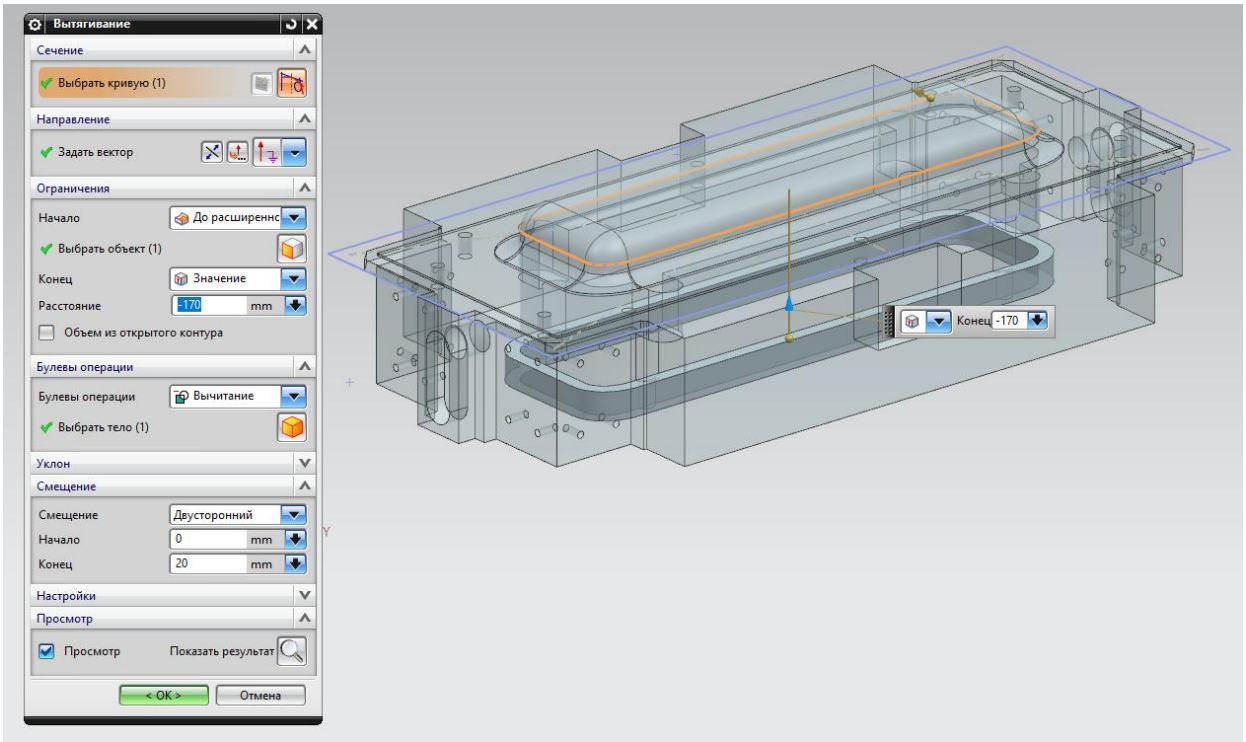


Рисунок 4.69 – Результат операции «вытягивания»

Для того что прижим получил свой окончательный вид, необходимо удалить все грани, оставшиеся после разделения на пуансон и прижим. Для этого использовать операцию «заменить грань».

```
theSession.SetUndoMarkName(markId33, "Диалоговое окно Заменить грань");
Face[] faces1 = new Face[1];
Face face3 = (Face)extractFace1.FindObject("FACE 24 {(290.0000000000001,639.9999999999999,423.1) BLOCK(2)}");
SelectionIntentRule[] rules15 = new SelectionIntentRule[1];
rules15[0] = faceDumbRule8;
replaceFaceBuilder1.FaceToReplace.ReplaceRules(rules15, false);
Face[] faces9 = new Face[9];
faces9[0] = face3;
faces9[1] = face4;
faces9[2] = face5;
faces9[3] = face6;
faces9[4] = face7;
faces9[5] = face8;
faces9[6] = face9;
faces9[7] = face10;
Face face11 = (Face)extractFace1.FindObject("FACE 37 {(188.5059999999999,993.3503999999998,434.26)
BLOCK(2)}");
faces9[8] = face11;
FaceDumbRule faceDumbRule9;
faceDumbRule9 = workPart.ScRuleFactory.CreateRuleFaceDumb(faces9);
SelectionIntentRule[] rules16 = new SelectionIntentRule[1];
rules16[0] = faceDumbRule9;
replaceFaceBuilder1.FaceToReplace.ReplaceRules(rules16, false);
Face[] faces10 = new Face[1];
Face face12 = (Face)block1.FindObject("FACE 1 {(290,640.0000000000001,420) BLOCK(2)}");
faces10[0] = face12;
FaceDumbRule faceDumbRule10;
faceDumbRule10 = workPart.ScRuleFactory.CreateRuleFaceDumb(faces10);
```

Рисунок 4.70 – Фрагмент кода, операция «заменить грань», выбранные грани, координаты граничащей поверхности и поверхность для замены

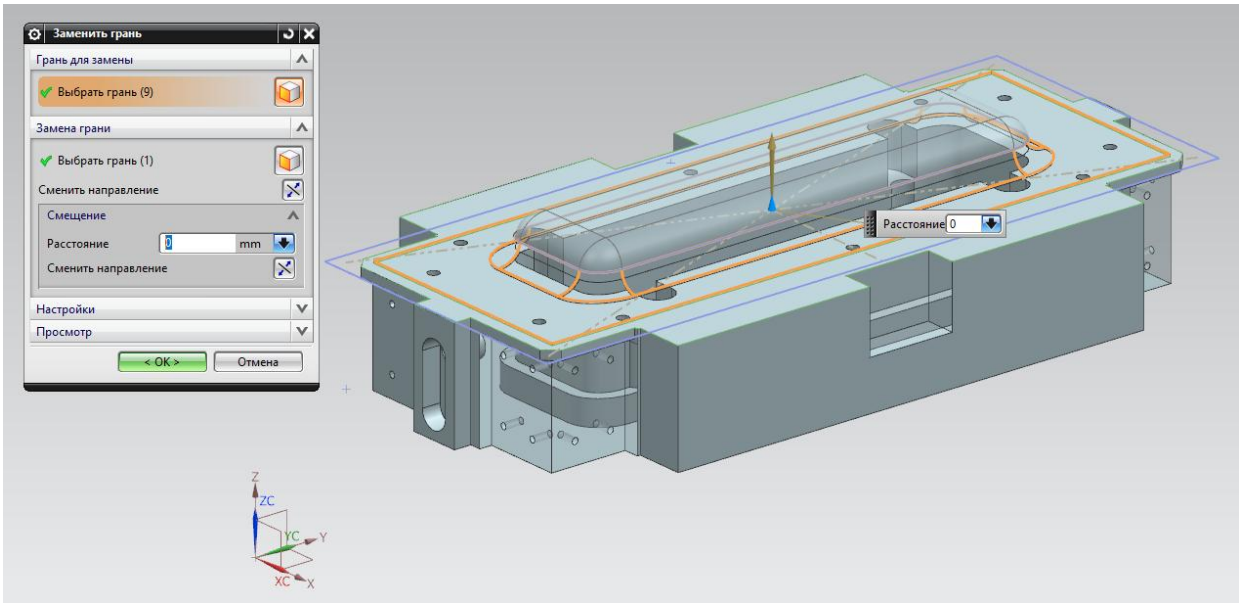


Рисунок 4.71 – Результат операции «заменить грань»

Для пуансона производим операцию «вычитание», для создания отверстия под маркетные выталкиватели. Так же эта операция показана в виде кода, но только объявление данной операции «вычитания» в начале, ее завершающее объявление в конце и присоединение последнего из компонентов, входящих в состав данной булевой операции. Весь код для данной операции представлен в приложении.

```
theSession.SetUndoMarkName(markId42, "Диалоговое окно Вычитание");
SelectionIntentRule[] rules34 = new SelectionIntentRule[1];
rules34[0] = bodyDumbRule18;
scCollector5.ReplaceRules(rules34, false);
Body[] bodies19 = new Body[6];
bodies19[0] = body8;
bodies19[1] = body10;
bodies19[2] = body12;
bodies19[3] = body14;
bodies19[4] = body16;
bodies19[5] = body18;
BodyDumbRule bodyDumbRule19;
bodyDumbRule19 = workPart.ScRuleFactory.CreateRuleBodyDumb(bodies19, true);
```

Рисунок 4.72 – Фрагмент кода, операция «вычитание», инициализация операции, выбранные тела из компонента «Conceptual» для вычитания из созданного компонента «punch»

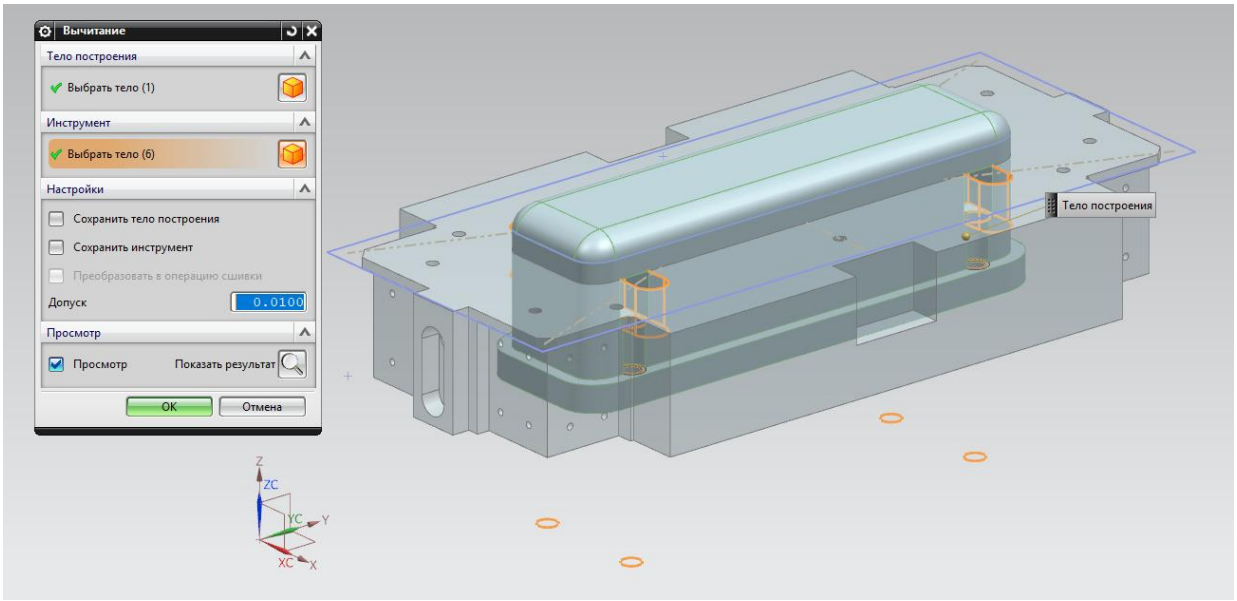


Рисунок 4.73 – Результат операции «вычитание»

Так как при объединении геометрии пуансона с вытяжным переходом детали сквозные отверстия, такие как отверстия для выпуска в пуансоне воздуха при штамповке и отверстия под маркетные выталкиватели не являются сквозным, применяется операция «переместить грань» для данных 6 элементов. Весь код данной операции представлен в приложении.

```
SelectionIntentRule[] rules46 = new SelectionIntentRule[0];
scCollector11.ReplaceRules(rules46, false);
Face[] faces16 = new Face[6];
faces16[0] = face13;
faces16[1] = face14;
faces16[2] = face15;
faces16[3] = face16;
faces16[4] = face17;
Face face18 = (Face)extrude1.FindObject("FACE 130 2 {(289.9999999999999,459.9999999999999,420)
SPLIT BODY(37)1}");
nErrs1 = theSession.UpdateManager.AddToDeleteList(direction10);
admMoveFaceBuilder4.Motion.DistanceValue.RightHandSide = "15";
```

Рисунок 4.74 – Фрагмент кода, операция «переместить грань», выбранные грани и координаты их перемещения для сквозного формирования отверстий соответствующей геометрии

Данная операция представлена на рисунке 4.75. Последовательно выбраны шесть граней для сквозного удаления из тела пуансона для завершения его окончательной геометрии.

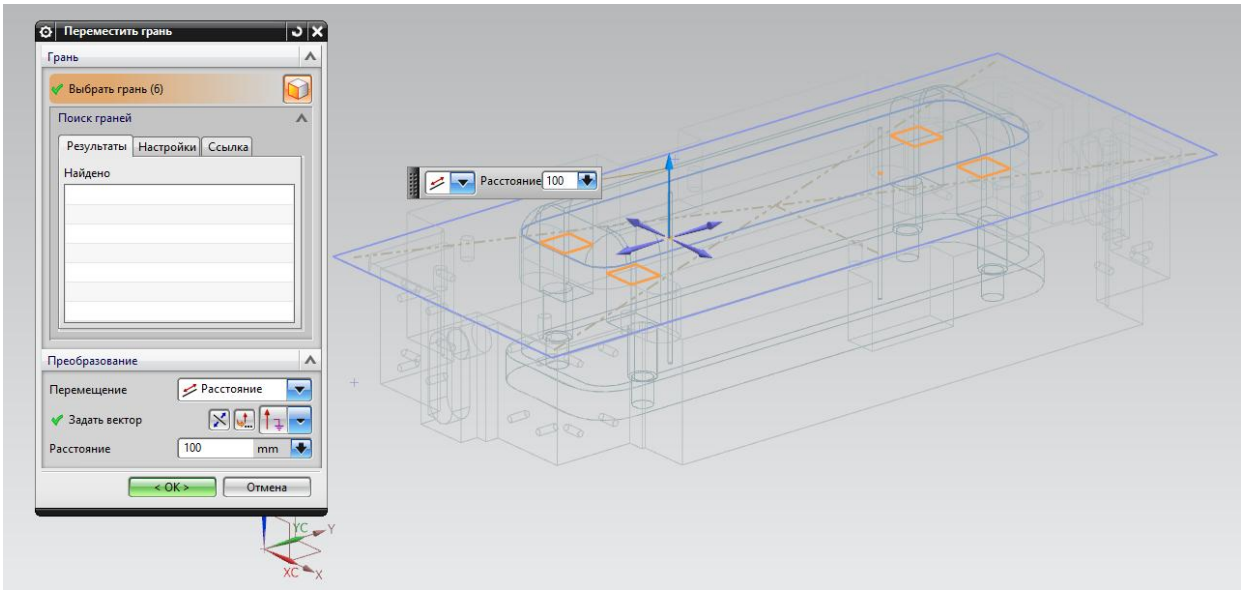


Рисунок 4.75 – Результат операции «переместить грань»

Что бы получить необходимый зазор производится операция «переместить грань» для матрицы.

```
theSession.SetUndoMarkName(markId52, "Диалоговое окно Переместить грань");
admMoveFaceBuilder1.Motion.DistanceAngle.Distance.RightHandSide = "2";
```

Рисунок 4.76 – Фрагмент кода переместить грань и расстояние для перемещения

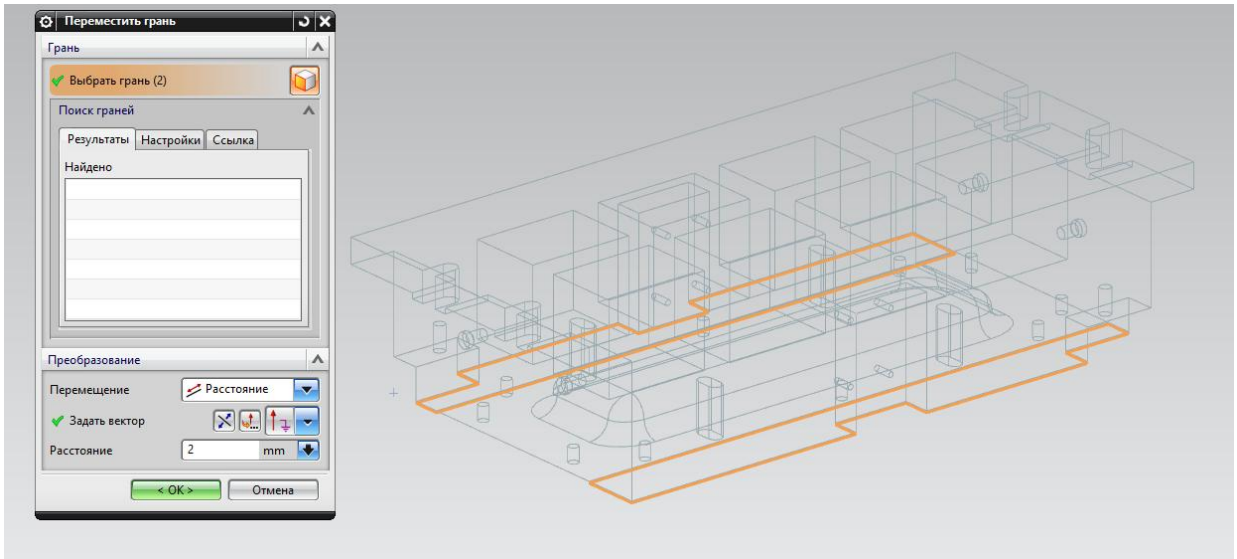


Рисунок 4.77 – Результат операции «переместить грань»

В результате получается необходимый зазор под вытяжной переход в закрытом состоянии штампа.

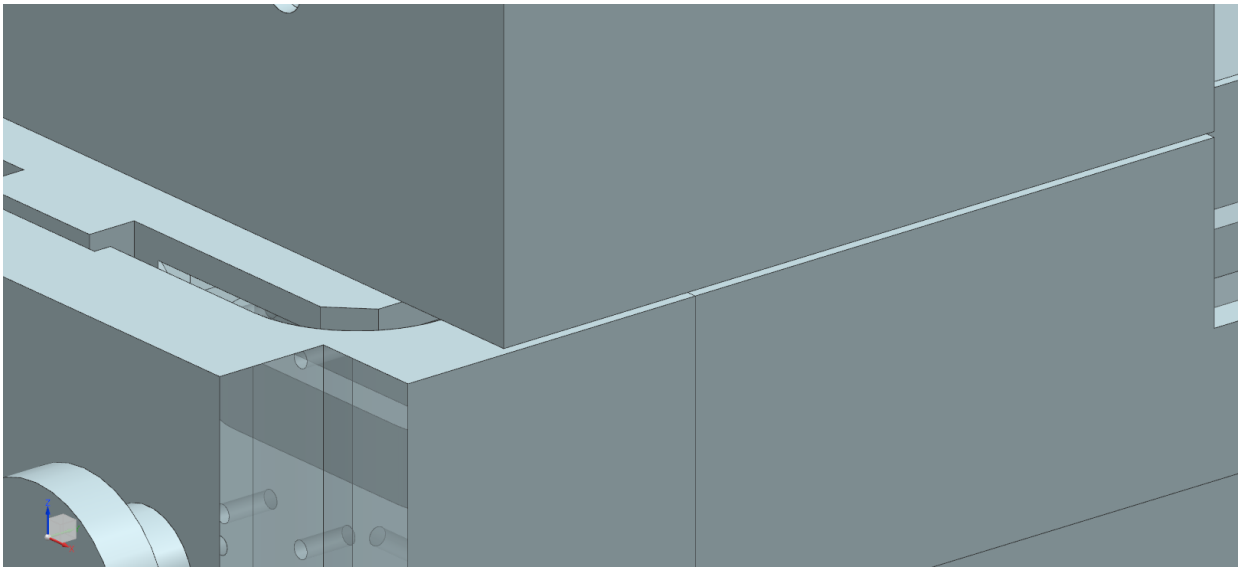


Рисунок 4.78 – Зазор под вытяжной переход

Заключительным для этой кнопки будет создание в сборке двух новых компонентов «punch» и «holder», после чего каждый из этих компонентов делается рабочим и с помощью редактора геометрических связей wave в них копируются тела пуансона и прижима.

```
theSession.SetUndoMarkName(markId53, "Диалоговое окно Файл нового компонента");
fileNew1.TemplateFileName = "model-plain-1-mm-template.prt";
fileNew1.NewFileName = "C:\\knopka\\scht4\\punch.prt";
theSession.SetUndoMarkName(markId53, "Файл нового компонента");
createNewComponentBuilder1.NewComponentName = "MODEL4";
createNewComponentBuilder1.NewComponentName = "PUNCH";
createNewComponentBuilder2.NewComponentName = "HOLDER";
theSession.SetUndoMarkName(markId69, "Диалоговое окно Редактор геометрических связей WAVE");
NXOpen.Assemblies.Component component4 =
(NXOpen.Assemblies.Component)component1.FindObject("COMPONENT holder 1");
```

Рисунок 4.79 – Фрагмент кода создания компонентов «punch» и «holder » в дереве сборки как отдельные детали и их ассоциативное копирование в данные компоненты с переименованием

Результат операции «разделение на пуансон и прижим» представлен в виде двух твердотельных моделей на рисунке 4.80 и на рисунке 4.81.

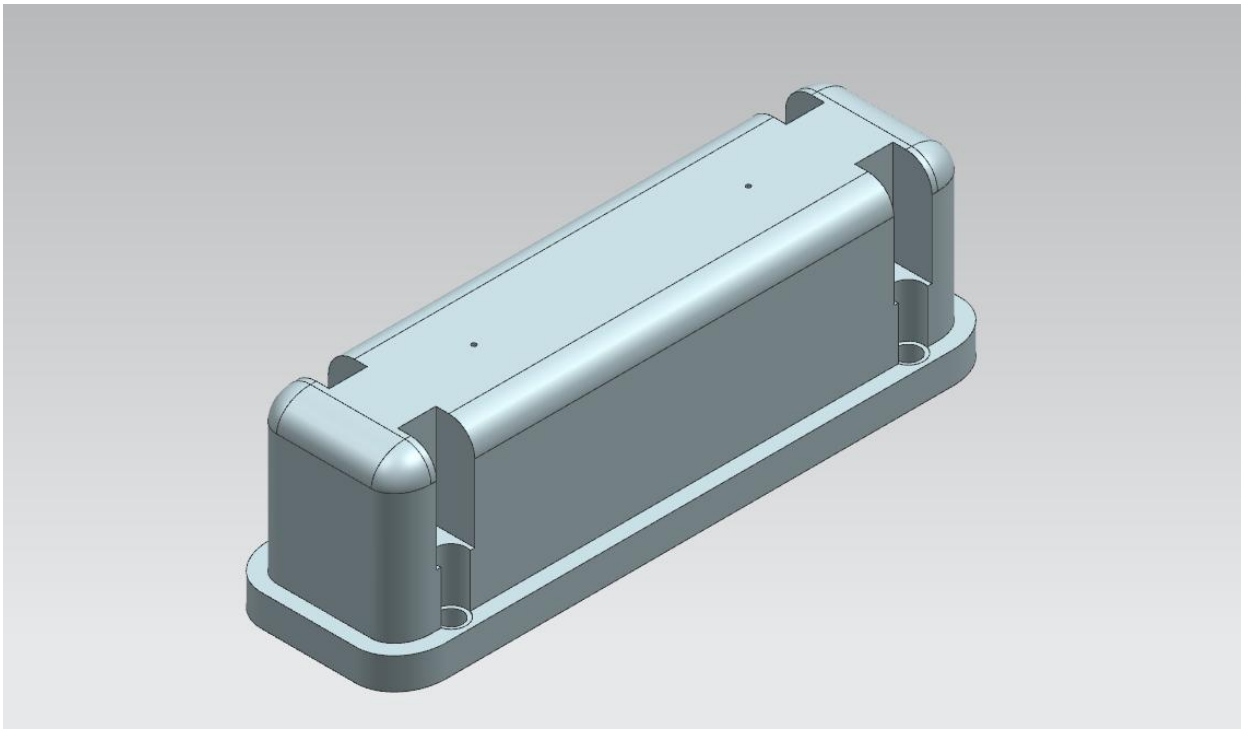


Рисунок 4.80 – Модель «пуансон»

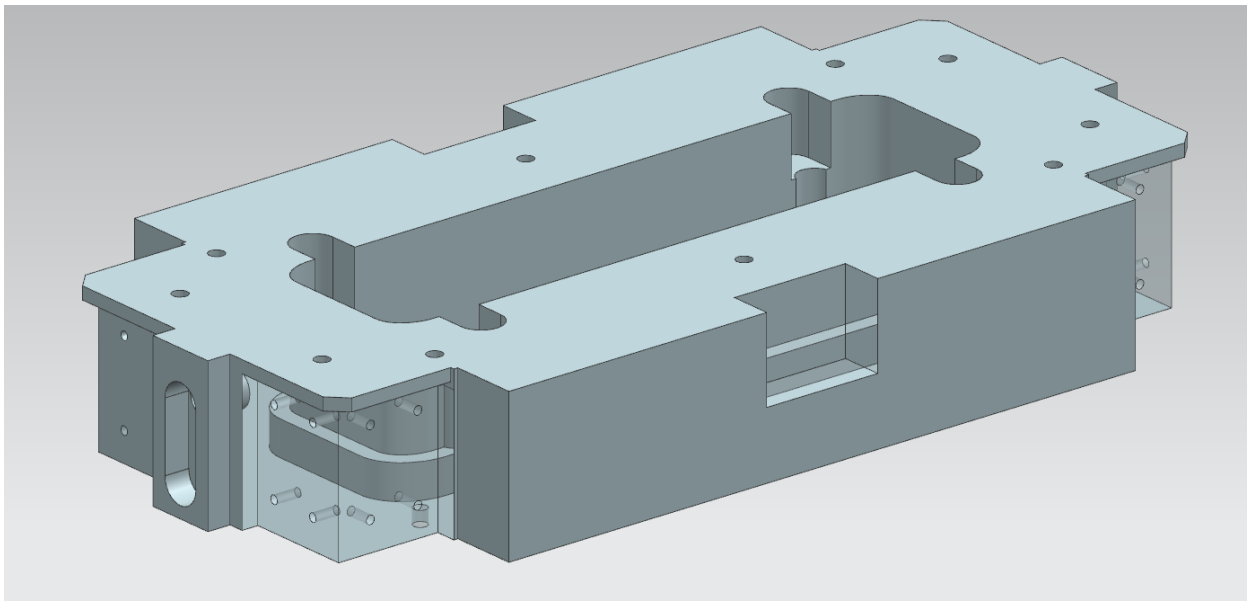


Рисунок 4.81 – Модель «прижим»

4.8 Операция «результат моделирования»

Для того чтобы облегчить последующую работу с проектом, посредством команды «Подавление» из сборки был выгружен компонент

«Conceptual» и «holder_lower_die». Данную команду возможно обратить при необходимости, можно восстановить компонент в сборке.

```
NXOpen.Session.UndoMarkId markId7;
markId7 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Visible, "Edit
Arrangement Suppression");
NXOpen.Assemblies.Component component2 =
(NXOpen.Assemblies.Component)component1.FindObject("COMPONENT Conceptual 1");
NXOpen.Assemblies.Component component3 =
(NXOpen.Assemblies.Component)component1.FindObject("COMPONENT holder_lower_die 1");
components2[0] = component3;
```

Рисунок 4.82 – Фрагмент кода, подавить компонент сборки для компонентов «Conceptual» и «holder_lower_die»

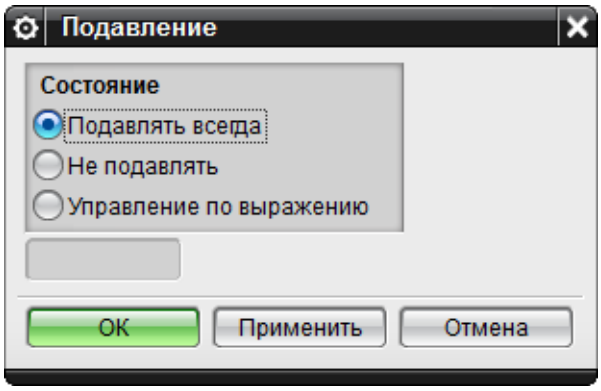


Рисунок 4.83 – Операция «Подавление» по отношению к компоненту «Conceptual» и «holder_lower_die»

Заключительной операцией будет изменение цвета компонентов сборки для улучшения интуитивного визуального восприятия электронной модели сборки.

```
NXOpen.Assemblies.Component component400 =
(NXOpen.Assemblies.Component)component1.FindObject("COMPONENT plate 1");
objects300[0] = component400;
displayModification3.Apply(objects300);
displayModification4.NewColor = 12;
// ----------------------------------------------
//   Меню: Ориентация вида->Трехмерный
// ----------------------------------------------
displayPart.ModelingViews.WorkView.Orient(NXOpen.View.Canned.Trimetric,
NXOpen.View.ScaleAdjustment.Fit);
```

Рисунок 4.84 – Фрагмент кода, «изменить цвет» для деталей штампа и ориентация вида трёхмерный для презентации результата работы приложения

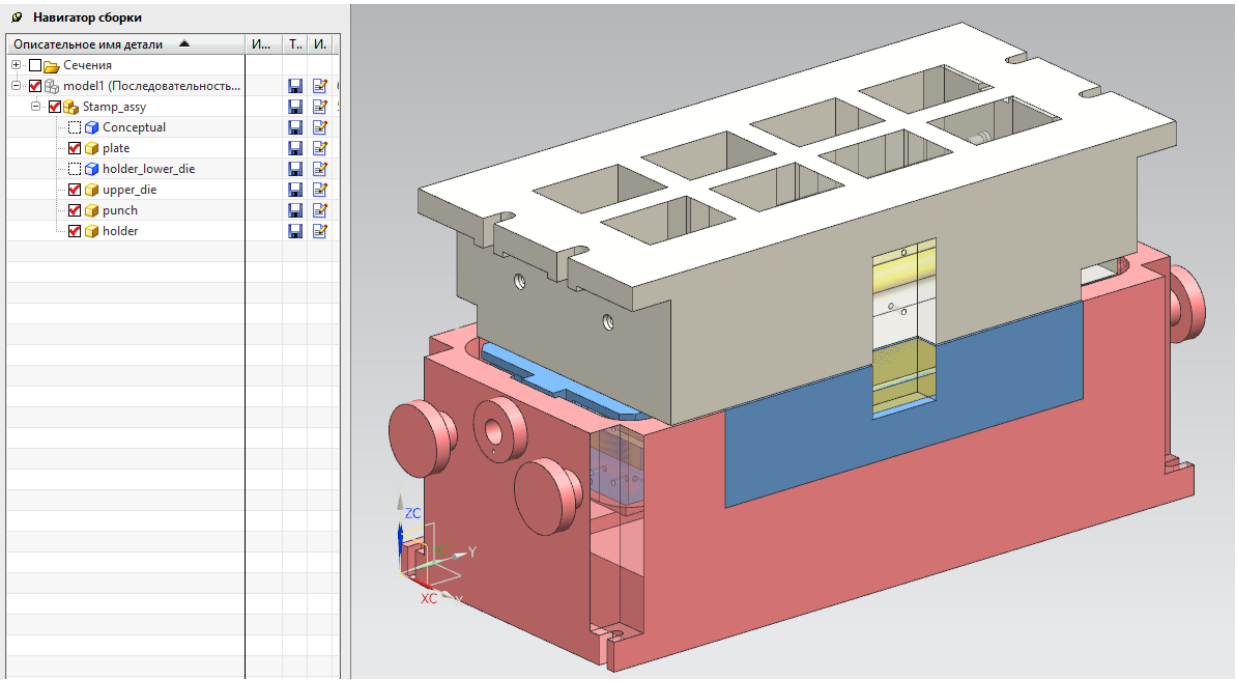


Рисунок 4.85 – Результат работы приложения

4.9 Инструмент «журнал»

Ранее были рассмотрены основные положения, касающиеся журнала, далее будут рассмотрены конкретные его доработки под необходимые задачи.

Частая проблема, связанная с записью и воспроизведением журнала, состоит в том, что все вносимые значения не могут быть изменены в поле ввода создаваемого интерфейса. Это касается высоты, ширины и длины создаваемого блока или расстояния используемого функцией вытягивания, что очень сильно ограничивает функционал разрабатываемого приложения. Наша задача добавить вариативности в поля ввода чисел, для последующего исполнения команд под конкретные задачи. Ниже приведён пример использования операторов double и string для создания блока и тела вытягивания:

```
double a;
a = (double)double0.Value;
string x = Convert.ToString(a);
extrudeBuilder1.Limits.EndExtend.Value.RightHandSide = x;
```

Рисунок 4.86 – Параметризация операции «вытягивание»

```
double a,b,c;
a = (double)double0.Value;
b = (double)double01.Value;
c = (double)double02.Value;

string x = Convert.ToString(a);
string y = Convert.ToString(b);
string z = Convert.ToString(c);
blockFeatureBuilder1.SetOriginAndLengths(originPoint1, x, y, z);
```

Рисунок 4.87 – Параметризация операции «создать блок»

Double – это простой тип для хранения значений 64-разрядных с плавающей запятой. В одном выражении возможно использовать типы с плавающей запятой и числовые целочисленные типы. В таком варианте целочисленные типы становятся типами с плавающей запятой. Выражения с плавающей запятой могут принимать значения: положительная и отрицательная бесконечность, положительный и отрицательный нуль, конечный набор ненулевых значений, нечисловое значение (NaN).

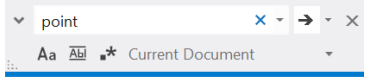
String – это последовательность символов. В C # string представляет собой последовательность символов Unicode. Это тип данных, который хранит последовательность значений данных, обычно байтов, в которых элементы обычно обозначают символы в соответствии с кодировкой символов. Когда строка появляется буквально в исходном коде, она называется строковым литералом. Строки – это объекты, существует два основных класса для работы со строками: System.String, System.Text.StringBuilder.

String неизменная последовательность символов. StringBuilder изменчивая последовательность символов. В C # string это псевдоним для System.String. Это string ключевое слово языка, а System.Stringтип .NET.

4.10 Удаление «мусора» из кода

После того как код из журнала вносится в строку, соответствующей кнопки которая отвечает за данное действие, необходимо произвести ручное редактирование. Так как при записи журнала все происходящее на экране, за редкими исключениями, записывается в виде кода, то при его воспроизведении, воспроизведутся и вспомогательные действия не влияющие на работу программы, но усложняющие код и влияющие на демонстрацию работы как отдельной функции, так и всего разработанного приложения.

Одним из таких самых распространенных примеров является запись изменения ориентации модели. При этом во время исполнения команды будет воспроизводится все ориентированные манипуляции, произведенные над моделью во время записи журнала. Что бы этого избежать необходимо после проверки работоспособности и соответствия необходимому функционалу разработанного приложения подвергнуть ручном редактированию. Для этого находимо открыть скомпилированный проект приложения и при помощи сочетания клавиш «Ctrl+F» произвести поиски программного мусора, для оптимизации работы приложения и минимизации разработанного кода программного продукта.




```
Matrix3x3 rotMatrix2;
rotMatrix2.Xx = 0.928165271137107;
rotMatrix2.Xy = 0.178814582886179;
rotMatrix2.Xz = 0.326396345571183;
rotMatrix2.Yx = -0.332118930812124;
rotMatrix2.Yy = 0.00219676414058814;
rotMatrix2.Yz = 0.943234960136403;
rotMatrix2.Zx = 0.167947150172883;
rotMatrix2.Zy = -0.983880337833089;
rotMatrix2.Zz = 0.0614266682659469;
Point3d translation2 = new Point3d(-596.452694014831, -215.716282268844, 751.23463439699);
displayPart.ModelingViews.WorkView.SetRotationTranslationScale(rotMatrix2, translation2, 0.207640959078287);
```

Рисунок 4.88 – Поиски программного «мусора»

Теперь при помощи вызванной поисковой строки ищутся все упоминания «Matrix3x3» и далее удалить данный блок, отвечающий за изменение ориентации положение рабочей детали при совершении над ней изменений.

```
Matrix3x3 rotMatrix6;
rotMatrix6.Xx = -0.18173283399262;
rotMatrix6.Xy = -0.977893459620251;
rotMatrix6.Xz = 0.103429003093655;
rotMatrix6.Yx = 0.545781849784398;
rotMatrix6.Yy = -0.0128139015783121;
rotMatrix6.Yz = 0.837829324130077;
rotMatrix6.Zx = -0.817982487278873;
rotMatrix6.Zy = 0.208710770106092;
rotMatrix6.Zz = 0.536045207931934;
Point3d translation6 = new Point3d(532.058747850765, -429.335386502565, 132.664847671508);
displayPart.ModelingViews.WorkView.SetRotationTranslationScale(rotMatrix6, translation6, 0.207640959078287);
```

Рисунок 4.89 – Фрагмент кода, отвечающий за изменение положения ориентации

Далее необходим о так же удалить все привязки изменения положения перемещения относительно оси координат, так как это на прямую влияет на положение рабочей детали при совершении над ней изменений. Так как все не нужные перемещения на экране могут существенно исказить представление пользователя данного программного модуля излишнем нагромождение сопутствующих манипуляций, что не как не влияют на работу приложения, но портят впечатление от программного продукта, а также ведут к снижению интуитивного понимания работы данного продукта. Теперь при помощи вызванной поисковой строки искать все упоминания «Point3d scaleAboutPoint» и удалить данный блок изменения положения относительно оси координат.

```
Point3d scaleAboutPoint1 = new Point3d(-580.413945631481, -27.3960479277218, 0.0);
Point3d viewCenter1 = new Point3d(580.413945631483, 27.39604792772, 0.0);
displayPart.ModelingViews.WorkView.ZoomAboutPoint(1.25, scaleAboutPoint1, viewCenter1);

Point3d scaleAboutPoint2 = new Point3d(-464.331156505184, -21.9168383421776, 0.0);
Point3d viewCenter2 = new Point3d(464.331156505186, 21.9168383421757, 0.0);
displayPart.ModelingViews.WorkView.ZoomAboutPoint(1.25, scaleAboutPoint2, viewCenter2);
```

Рисунок 4.90 – Фрагмент кода, отвечающий за изменение положения относительно оси координат

Далее необходимо так же удалить все ориентации вида, которые были вызваны из контекстного меню нажатием на соответствующую кнопку при записи журнала. Аналогично все не нужные перемещения на экране могут существенно исказить представление пользователя данного программного модуля излишнем нагромождение сопутствующих манипуляций, что не как не влияют на работу приложения. Теперь при помощи вызванной поисковой строки искать все упоминания Ориентация вида и удалить показанный фрагмент.

```
// ----------------------------------------------
//   Меню: Ориентация вида->Трехмерный
// ----------------------------------------------
workPart.ModelingViews.WorkView.Orient(NXOpen.View.Canned.Trimetric, NXOpen.View.ScaleAdjustment.Fit);
```

Рисунок 4.91 – Фрагмент кода, отвечающий за изменение ориентации вида

Следующим шагом необходимо так же удалить все стили закраски. При записи кода они необходимы что бы выделить элементы рабочей модели или всей сборки, которые являются скрытыми прилегающими или каркасными элементами тли деталями так же используют для этих целей функцию выбрать из списка что не отображается на экране во время работы приложения, но записывается в код данного погромного продукта. Теперь при помощи вызванной поисковой строки искать все упоминания стиль закраски и удалить показанный фрагмент.

```
// ----------------------------------------------
//   Меню: Стиль закраски->Статический каркасный
// ----------------------------------------------
workPart.ModelingViews.WorkView.RenderingStyle = NXOpen.View.RenderingStyleType.StaticWireframe;

// ----------------------------------------------
//   Меню: Стиль закраски->Закраска с ребрами
// ----------------------------------------------
workPart.ModelingViews.WorkView.RenderingStyle = NXOpen.View.RenderingStyleType.ShadedWithEdges;
```

Рисунок 4.92 – Фрагмент кода, отвечающий за изменение стиля закраски

Так же при помощи вызванной поисковой строки искать все упоминания выбрать из списка и удалить показанный фрагмент.

```
// ------------------------------------------------
//   Меню: Выберите из списка...
// ------------------------------------------------
NXOpen.Features.ExtractFaceBuilder extractFaceBuilder158;
extractFaceBuilder158 = workPart.Features.CreateExtractFaceBuilder(nullFeatures_Feature);
```

Рисунок 4.93 – Фрагмент кода, отвечающий за выбор объекта из списка

4.11 Блок-схема приложения

Блок-схема отражает структуру разработанного приложения. На рисунке 4.94 показаны элементы интерфейса приложения в виде названия операций и операторов методов приложения, отвечающих за их исполнение.

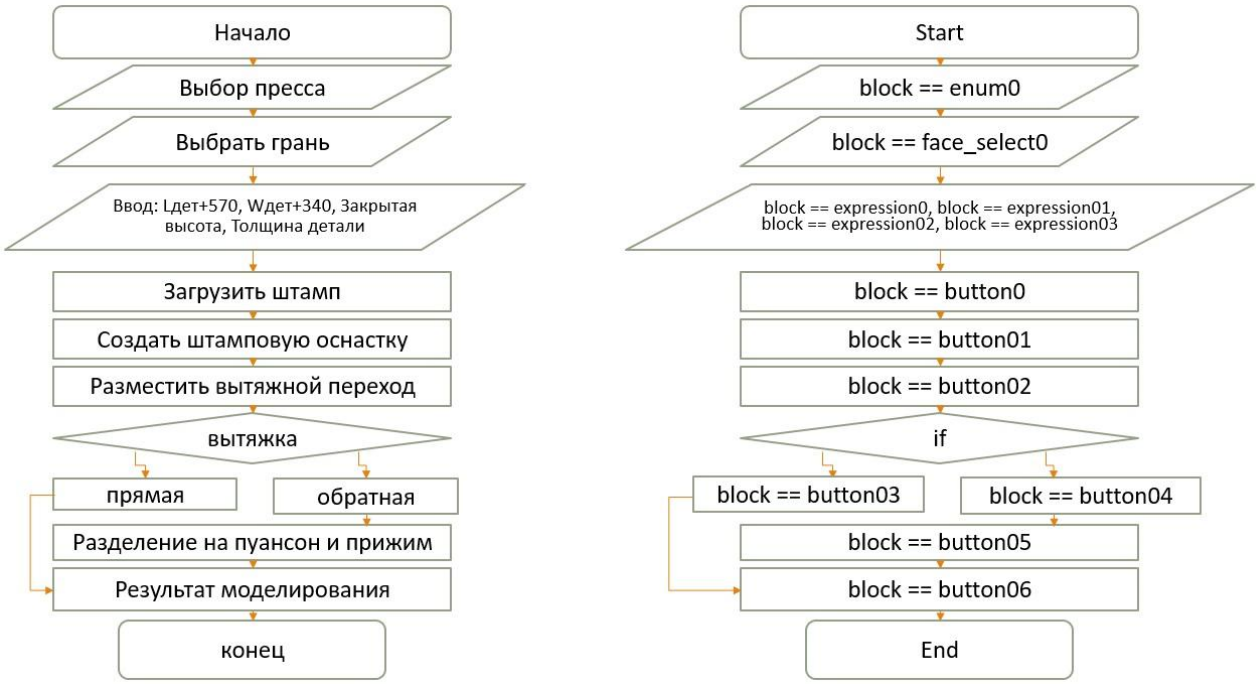


Рисунок 4.94 – Блок-схема приложения

4.12 Выводы раздела

1. Была разработана параметрическая схема.
2. Были внесены изменения в параметрическую модель штампа для пресса MullerWeinGarten-1000.
3. Был разработан интерфейс приложения для прямой и боратной вытяжки.

4. Был написан код на языке объектно-ориентированного программирования C# в среде VisaulStudio с подгруженными библиотеками разработки NX-приложений.
5. Была разработана методика разработки приложений проектирования вытяжных штампов с определением функций и методов для NX.

ЗАКЛЮЧЕНИЕ

В данной магистерской диссертации рассмотрен процесс создания приложения для автоматической загрузки типовых конструкций штампа, их дальнейшее позиционирования относительно вытяжного перехода детали, объединение, вычитание, смещение, удаление, подавление и позиционирование элементов штампов для создания конечных ассоциативных копий готовых электронных моделей.

Были объектно-ориентированно описаны конструкции штампа на языке программирования C#. Разработано приложение для проектирования элементов штампов многопозиционной штамповки на базе типовых решений, существующего программного обеспечения для создания конструкций штамповой оснастки для существующих процессов.

Апробация метода для вытяжного штампа абстрактной листовой детали лонжерона показали эффективность и работоспособность приложения. Была получен проект штамповой оснастки, готовый для доработки в существующих производствах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Siemens PLM Software. NXOpen API Help [Электронный ресурс]: URL.: http://www.siemensplm.com.
2. Shah, J.J. Parametric and feature-based CAD/CAM: concepts, techniques, and applications.: Wiley, New York. 2001. – 195с.
3. Wang, Y.H. Geometry-based semantic IDfor persistent and interoperable reference in feature-based parametric modeling.: Comput Aided Des 2005. – 254с.
4. Zha X.F., Knowledge-based approach and system for assembly oriented design, Part I: the approach.: Eng Appl Artif Intell 2001. – 103с.
5. Zha, X.F. Knowledge-based approach and system for assembly oriented design, Part II: the system implementation.: Eng Appl Artif Intell, 2001. – 254с.
6. Аверкиев Ю.А. Технология холодной штамповки: Учебник для вузов по специальности «Машины и технология обработки металлов давлением» и «Обработка металлов давлением». – М.: Машиностроение, 1989. – 304 с.
7. Банкетов, А.Н. Кузнечно–штамповочное оборудование. – М.: Машиностроение, 1982. – 576 с.
8. Владимиров, В.М. Изготовление штампов и пресс–форм. – М.: Машиностроение, 1981. – 431 с.
9. Данилов Ю.А. Практическое использование NX. – М.: ДМК Пресс, 2011. – 332 с.
10. Малов, А.Н. Технология холодной штамповки – М.: Машиностроение, 1969. – 568 стр.
11. Нефедов, А.П. Конструирование и изготовление штампов: из опыта Горьковского автомобильного. – Москва: Машиностроение, 1973. – 408 с.
12. Почекуев, Е.Н. Проектирование в Siemens NX технологических процессов изготовления деталей листовой штамповкой: электронное учеб. – метод. пособие. – Тольятти: изд–во ТГУ, 2014. – 1 электрон. опт. диск.

13. Романовский, В.П. Справочник по холодной штамповке. – Л.: Машиностроение, 1979. – 568 с.

14. Аникин, В.М. Справочник конструктора штампов для холодной штамповки. М.: Машиностроение, 1960. – 296 с.

15. Схиртладзе, А. Г. Автоматизированное проектирование штампов: учебное пособие.: Изд-во Владим. Гос. Ун-та, 2007. – 284 с.

16. Руководство пользователя UGS NX v9, 2013. – 205 с.

17. Зубцов, М.Е. Листовая штамповка.: Машиностроение, 1980. – 432 с.

18. Скворцов, Г.Д. Основы конструирования штампов для холодной листовой штамповки. – М.: Машиностроение, 1974 – 318 с.

19. Скрипачев, А.В. Технологичность листовых штампованных деталей. Методические указания по технологии листовой штамповки. Тольятти: ТолПИ, 1992. – 102 с.

20. Смолин, Е.Л. Основы конструирования штамповой оснастки: учебное пособие. – Тольятти: ТГУ, 2007. – 72 с.

21. Якуничев, Е.В. Технология холодной штамповки. Сборник задач. – Тольятти: ТолПИ, 1991. – 85 с.

22. Тихомиров, В.А. Разработка приложений для Unigraphics на языке C.: Дальнаука, 2011. – 422 с.

23. Гончаров, П.С. NX для конструктора-машиностроителя. М.: ДМК Пресс, 2010. – 504с.

24. Зиборов, В.В. Visual C# 2012 на примерах. – М.: БХВ-Петербург, 2013. – 480 с.

25. Смоленцев, Н. К. MATLAB. Программирование на Visual C#. – М.: ДМК Пресс, 2011. - 456 с.

26. Ишкова, Э. А. Самоучитель C#. Начала программирования. – М.: Наука и техника, 2013. - 496 с.

27. Албахари, Д.А. C# 3.0. Справочник. – М.: БХВ-Петербург, 2015. - 499 с.

28. Фленов, М.Е. Библия C#. – М.: БХВ-Петербург, 2015. - 532 с.

29. Фримен, А.Г. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов. – М.: Вильямс, 2016. - 688 с.
30. Троелсен, Э.К. Язык программирования C# 5.0 и платформа .NET 4.5. – М.: Вильямс, 2015. - 486 с.

Приложение